necessary assumption for most of private-key cryptography was shown in [93]. The proof of Proposition 7.29 is from [72].

Our presentation is heavily influenced by Goldreich's book [75], which is highly recommended for those interested in exploring the topics of this chapter in greater detail.

Exercises

7.1 Prove that if there exists a one-way function, then there exists a one-way function f such that $f(0^n) = 0^n$ for every n. Note that for infinitely many values y, it is easy to compute $f^{-1}(y)$. Why does this not contradict one-wayness?

7.2 Prove that if f is a one-way function, then the function g defined by $g(x_1, x_2) \stackrel{\text{def}}{=} (f(x_1), x_2)$, where $|x_1| = |x_2|$, is also a one-way function. Observe that g reveals half of its input, but is nevertheless one-way.

7.3 Prove that if there exists a one-way function, then there exists a length-preserving one-way function.

> **Hint:** Let f be a one-way function and let $p(\cdot)$ be a polynomial such that $|f(x)| \leq p(|x|)$. (Justify the existence of such a p.) Define $f'(x) \stackrel{\text{def}}{=} f(x) \| 10^{p(|x|)-|f(x)|}$. Further modify f' to get a length-preserving function that remains one-way.

7.4 Let (Gen, H) be a collision-resistant hash function, where H maps strings of length $2n$ to strings of length n. Prove that the function family $(\mathsf{Gen}, \mathsf{Samp}, H)$ is one-way (cf. Definition 7.3), where Samp is the trivial algorithm that samples a uniform string of length $2n$.

> **Hint:** Choosing uniform $x \in \{0,1\}^{2n}$ and finding an inverse of $y = H^s(x)$ does not guarantee a collision. But it does yield a collision most of the time...

7.5 Let F be a (length-preserving) pseudorandom permutation.

(a) Show that the function $f(x, y) = F_x(y)$ is not one-way.

(b) Show that the function $f(y) = F_{0^n}(y)$ (where $n = |y|$) is not one-way.

(c) Prove that the function $f(x) = F_x(0^n)$ (where $n = |x|$) is one-way.

7.6 Let f be a length-preserving one-way *function*, and let hc be a hard-core predicate of f. Define G as $G(x) = f(x) \| \mathsf{hc}(x)$. Is G necessarily a pseudorandom generator? Prove your answer.

7.7 Prove that there exist one-way functions if and only if there exist one-way function families. Discuss why your proof does not carry over to the case of one-way permutations.

7.8 Let f be a one-way function. Is $g(x) \stackrel{\text{def}}{=} f(f(x))$ necessarily a one-way function? What about $g'(x) \stackrel{\text{def}}{=} f(x)\|f(f(x))$? Prove your answers.

7.9 Let $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ be a function family. A function $\mathsf{hc} : \{0,1\}^* \to \{0,1\}$ is a *hard-core predicate of* Π if it is efficiently computable and if for every PPT algorithm $\mathcal{A}$ there is a negligible function negl such that

$$\Pr_{I \leftarrow \mathsf{Gen}(1^n),\, x \leftarrow \mathsf{Samp}(I)}[\mathcal{A}(I, f_I(x)) = \mathsf{hc}(I, x)] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Prove a version of the Goldreich–Levin theorem for this setting, namely, if a one-way function (resp., permutation) family Π exists, then there exists a one-way function (resp., permutation) family Π' and a hard-core predicate hc of Π'.

7.10 Show a construction of a pseudorandom generator from any one-way permutation family. You may use the result of the previous exercise.

7.11 This exercise is for students who have taken a course in complexity theory or are otherwise familiar with $\mathcal{NP}$ completeness.

(a) Show that the existence of one-way functions implies $\mathcal{P} \neq \mathcal{NP}$.

(b) Assume that $\mathcal{P} \neq \mathcal{NP}$. Show that there exists a function f that is: (1) computable in polynomial time, (2) hard to invert *in the worst case* (i.e., for all probabilistic polynomial-time $\mathcal{A}$, $\Pr_{x \leftarrow \{0,1\}^n}[f(\mathcal{A}(f(x))) = f(x)] \neq 1$), but (3) is *not* one-way.

7.12 Let $x \in \{0,1\}^n$ and denote $x = x_1 \cdots x_n$. Prove that if there exists a one-way function, then there exists a one-way function f such that for every i there is an algorithm A_i such that

$$\Pr_{x \leftarrow \{0,1\}^n}[A_i(f(x)) = x_i] \geq \frac{1}{2} + \frac{1}{2n}.$$

(This exercise demonstrates that it is not possible to claim that every one-way function hides at least one *specific* bit of the input.)

7.13 Show that if a one-to-one function f has a hard-core predicate, then f is one-way.

7.14 Show that if Construction 7.21 is modified in the natural way so that $F_k(x)$ is defined for every nonempty string x of length at most n, then the construction is no longer a pseudorandom function.

7.15 Prove that if there exists a pseudorandom function that, using a key of length n, maps n-bit inputs to single-bit outputs, then there exists a pseudorandom function that maps n-bit inputs to n-bit outputs.

> **Hint:** Use a key of length n^2, and prove your construction secure using a hybrid argument.

7.16 Prove that a two-round Feistel network using pseudorandom round functions (as in Equation (7.15)) is not pseudorandom.

7.17 Prove that a three-round Feistel network using pseudorandom round functions (as in Equation (7.16)) is not *strongly* pseudorandom.

> **Hint:** This is significantly more difficult than the previous exercise. Use a distinguisher that makes two queries to the permutation and one query to its inverse.

7.18 Consider the keyed permutation F^* defined by

$$F_k^*(x) \stackrel{\text{def}}{=} \mathsf{Feistel}_{F_k,F_k,F_k}(x).$$

(Note that the same key is used in each round.) Show that F^* is not pseudorandom.

7.19 Let G be a pseudorandom generator with expansion factor $\ell(n) = n+1$. Prove that G is a one-way function.

7.20 Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be probability ensembles. Prove that if $\mathcal{X} \stackrel{\text{c}}{\equiv} \mathcal{Y}$ and $\mathcal{Y} \stackrel{\text{c}}{\equiv} \mathcal{Z}$, then $\mathcal{X} \stackrel{\text{c}}{\equiv} \mathcal{Z}$.

7.21 Prove Theorem 7.32.

7.22 Let $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_n\}_{n\in\mathbb{N}}$ be computationally indistinguishable probability ensembles. Prove that for any probabilistic polynomial-time algorithm $\mathcal{A}$, the ensembles $\{\mathcal{A}(X_n)\}_{n\in\mathbb{N}}$ and $\{\mathcal{A}(Y_n)\}_{n\in\mathbb{N}}$ are computationally indistinguishable.

Part III

Public-Key (Asymmetric) Cryptography

Chapter 8

Number Theory and Cryptographic Hardness Assumptions

Modern cryptosystems are invariably based on an assumption that *some* problem is hard. In Chapters 3 and 4, for example, we saw that private-key cryptography—both encryption schemes and message authentication codes—can be based on the assumption that pseudorandom permutations (a.k.a. block ciphers) exist. Recall, roughly, this means that there exists some keyed permutation F for which it is hard to distinguish in polynomial time between interactions with F_k (for a uniform, unknown key k) and interactions with a truly random permutation.

On the face of it, the assumption that pseudorandom permutations exist seems quite strong and unnatural, and it is reasonable to ask whether this assumption is true or whether there is any evidence to support it. In Chapter 6 we explored how block ciphers are constructed in practice. The fact that existing constructions are resistant to attack serves as an indication that the existence of pseudorandom permutations is plausible. Still, it may be difficult to believe that there are *no* efficient distinguishing attacks on existing block ciphers. Moreover, the current state of our theory is such that we do not know how to prove the pseudorandomness of any of the existing practical constructions relative to any "simpler" or "more reasonable" assumption. All in all, this is not entirely a satisfying state of affairs.

In contrast, as mentioned in Chapter 3 (and investigated in detail in Chapter 7) it is possible to *prove* that pseudorandom permutations exist based on the much milder assumption that one-way functions exist. (Informally, a function is *one-way* if it is easy to compute but hard to invert; see Section 8.4.1.) Apart from a brief discussion in Section 7.1.2, however, we have not seen concrete examples of functions believed to be one-way.

One goal of this chapter is to introduce various problems believed to be "hard," and to present conjectured one-way functions based on those problems.[1] This chapter can thus be viewed as a culmination of a "top down" approach to private-key cryptography. (See Figure 8.1.) That is, in Chapters 3 and 4 we have shown that private-key cryptography can be based on pseudorandom functions and permutations. We have then seen that the latter

[1] Recall we currently do not know how to *prove* that one-way functions exist, so the best we can do is base one-way functions on assumptions regarding the hardness of certain problems.

can be instantiated in practice using block ciphers, as explored in Chapter 6, or can be constructed in a rigorous fashion from any one-way function, as shown in Chapter 7. Here, we take this one step further and show how one-way functions can be based on certain hard mathematical problems.

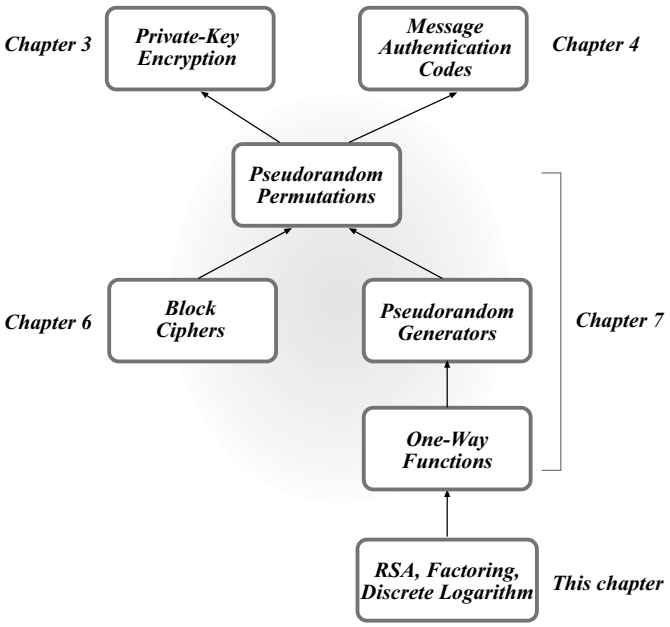

FIGURE 8.1: Private-key cryptography: a top-down approach.

The examples we explore are *number theoretic* in nature, and we therefore begin with a short introduction to number theory and group theory. Because we are also interested in problems that can be solved efficiently (even a one-way function needs to be easy to compute in one direction, and a cryptographic scheme must admit efficient algorithms for the honest parties), we also initiate a study of *algorithmic* number theory. Even the reader who is familiar with number theory or group theory is encouraged to read this chapter, since algorithmic aspects are typically ignored in a purely mathematical treatment of these topics.

A second goal of this chapter is to develop the material needed for *public-key cryptography*, whose study we will begin in Chapter 10. Strikingly, although in the private-key setting there exist efficient constructions of the necessary primitives (block ciphers and hash functions) without invoking any number theory, in the public-key setting *all known constructions rely on hard number-theoretic problems*. The material in this chapter thus serves both as a culmination of what we have studied so far with regard to private-key cryptography, as well as the foundation for public-key cryptography.

8.1 Preliminaries and Basic Group Theory

We begin with a review of prime numbers and basic modular arithmetic. Even the reader who has seen these topics before should skim the next two sections since some of the material may be new and we include proofs for most of the stated results.

8.1.1 Primes and Divisibility

The set of integers is denoted by $\mathbb{Z}$. For $a, b \in \mathbb{Z}$, we say that a *divides* b, written $a \mid b$, if there exists an integer c such that $ac = b$. If a does not divide b, we write $a \nmid b$. (We are primarily interested in the case where a, b, and c are all positive, although the definition makes sense even when one or more of these is negative or zero.) A simple observation is that if $a \mid b$ and $a \mid c$ then $a \mid (Xb + Yc)$ for any $X, Y \in \mathbb{Z}$.

If $a \mid b$ and a is positive, we call a a *divisor* of b. If in addition $a \notin \{1, b\}$ then a is called a *nontrivial* divisor, or a *factor*, of b. A positive integer $p > 1$ is *prime* if it has no factors; i.e., it has only two divisors: 1 and itself. A positive integer greater than 1 that is not prime is called *composite*. By convention, the number 1 is neither prime nor composite.

A fundamental theorem of arithmetic is that every integer greater than 1 can be expressed *uniquely* (up to ordering) as a product of primes. That is, any positive integer $N > 1$ can be written as $N = \prod_i p_i^{e_i}$, where the $\{p_i\}$ are distinct primes and $e_i \geq 1$ for all i; furthermore, the $\{p_i\}$ (and $\{e_i\}$) are uniquely determined up to ordering.

We are familiar with the process of *division with remainder* from elementary school. The following proposition formalizes this notion.

PROPOSITION 8.1 *Let a be an integer and let b be a positive integer. Then there exist unique integers q, r for which $a = qb + r$ and $0 \leq r < b$.*

Furthermore, given integers a and b as in the proposition it is possible to compute q and r in polynomial time; see Appendix B.1. (An algorithm's running time is measured as a function of the length(s) of its input(s). An important point in the context of algorithmic number theory is that integer inputs are always assumed to be represented in binary. The running time of an algorithm taking as input an integer N is therefore measured in terms of $\|N\|$, the *length of the binary representation of* N. Note that $\|N\| = \lfloor \log N \rfloor + 1$.)

The *greatest common divisor* of two integers a, b, written $\gcd(a, b)$, is the largest integer c such that $c \mid a$ and $c \mid b$. (We leave $\gcd(0, 0)$ undefined.) The notion of greatest common divisor makes sense when either or both of a, b are negative but we will typically have $a, b \geq 1$; anyway, $\gcd(a, b) = \gcd(|a|, |b|)$.

Note that $\gcd(b, 0) = \gcd(0, b) = b$; also, if p is prime then $\gcd(a, p)$ is either equal to 1 or p. If $\gcd(a, b) = 1$ we say that a and b are *relatively prime*.

The following is a useful result:

PROPOSITION 8.2 *Let a, b be positive integers. Then there exist integers X, Y such that $Xa + Yb = \gcd(a, b)$. Furthermore, $\gcd(a, b)$ is the smallest positive integer that can be expressed in this way.*

PROOF Consider the set $I \stackrel{\text{def}}{=} \{\hat{X}a + \hat{Y}b \mid \hat{X}, \hat{Y} \in \mathbb{Z}\}$. Note that $a, b \in I$, and so I certainly contains some positive integers. Let d be the smallest positive integer in I. We show that $d = \gcd(a, b)$; since d can be written as $d = Xa + Yb$ for some $X, Y \in \mathbb{Z}$ (because $d \in I$), this proves the theorem.

To show this, we must prove that $d \mid a$ and $d \mid b$, and that d is the largest integer with this property. In fact, we can show that d divides every element in I. To see this, take an arbitrary $c \in I$ and write $c = X'a + Y'b$ with $X', Y' \in \mathbb{Z}$. Using division with remainder (Proposition 8.1) we have that $c = qd + r$ with q, r integers and $0 \leq r < d$. Then

$$r = c - qd = X'a + Y'b - q(Xa + Yb) = (X' - qX)a + (Y' - qY)b \in I.$$

If $r \neq 0$, this contradicts our choice of d as the *smallest* positive integer in I (because $r < d$). So, $r = 0$ and hence $d \mid c$. This shows that d divides every element of I.

Since $a \in I$ and $b \in I$, the above shows that $d \mid a$ and $d \mid b$ and so d is a common divisor of a and b. It remains to show that it is the largest common divisor. Assume there is an integer $d' > d$ such that $d' \mid a$ and $d' \mid b$. Then by the observation made earlier, $d' \mid Xa + Yb$. Since the latter is equal to d, this means $d' \mid d$. But this is impossible if d' is larger than d. We conclude that d is the largest integer dividing both a and b, and hence $d = \gcd(a, b)$. ■

Given a and b, the *Euclidean algorithm* can be used to compute $\gcd(a, b)$ in polynomial time. The *extended Euclidean algorithm* can be used to compute X, Y (as in the above proposition) in polynomial time as well. See Appendix B.1.2 for details.

The preceding proposition is very useful in proving additional results about divisibility. We show two examples now.

PROPOSITION 8.3 *If $c \mid ab$ and $\gcd(a, c) = 1$, then $c \mid b$. Thus, if p is prime and $p \mid ab$ then either $p \mid a$ or $p \mid b$.*

PROOF Since $c \mid ab$ we have $\gamma c = ab$ for some integer γ. If $\gcd(a, c) = 1$ then, by the previous proposition, we know there exist integers X, Y such that

$1 = Xa + Yc$. Multiplying both sides by b, we obtain

$$b = Xab + Ycb = X\gamma c + Ycb = c \cdot (X\gamma + Yb).$$

Since $(X\gamma + Yb)$ is an integer, it follows that $c \mid b$.

The second part of the proposition follows from the fact that if $p \not| a$ then $\gcd(a, p) = 1$. ■

PROPOSITION 8.4 *If $a \mid N$, $b \mid N$, and $\gcd(a, b) = 1$, then $ab \mid N$.*

PROOF Write $ac = N$, $bd = N$, and (using Proposition 8.2) $1 = Xa + Yb$, where c, d, X, Y are all integers. Multiplying both sides of the last equation by N we obtain

$$N = XaN + YbN = Xabd + Ybac = ab(Xd + Yc),$$

showing that $ab \mid N$. ■

8.1.2 Modular Arithmetic

Let $a, b, N \in \mathbb{Z}$ with $N > 1$. We use the notation $[a \bmod N]$ to denote the remainder of a upon division by N. In more detail: by Proposition 8.1 there exist unique q, r with $a = qN + r$ and $0 \leq r < N$, and we define $[a \bmod N]$ to be equal to this r. Note therefore that $0 \leq [a \bmod N] < N$. We refer to the process of mapping a to $[a \bmod N]$ as *reduction modulo N*.

We say that a and b are *congruent modulo N*, written $a = b \bmod N$, if $[a \bmod N] = [b \bmod N]$, i.e., if the remainder when a is divided by N is the same as the remainder when b is divided by N. Note that $a = b \bmod N$ if and only if $N \mid (a - b)$. By way of notation, in an expression such as

$$a = b = c = \cdots = z \bmod N,$$

the understanding is that *every* equal sign in this sequence (and not just the last) refers to congruence modulo N.

Note that $a = [b \bmod N]$ implies $a = b \bmod N$, but not vice versa. For example, $36 = 21 \bmod 15$ but $36 \neq [21 \bmod 15] = 6$. (On the other hand, $[a \bmod N] = [b \bmod N]$ if and only if $a = b \bmod N$.)

Congruence modulo N is an equivalence relation: i.e., it is reflexive ($a = a \bmod N$ for all a), symmetric ($a = b \bmod N$ implies $b = a \bmod N$), and transitive (if $a = b \bmod N$ and $b = c \bmod N$, then $a = c \bmod N$). Congruence modulo N also obeys the standard rules of arithmetic with respect to addition, subtraction, and multiplication; so, for example, if $a = a' \bmod N$ and $b = b' \bmod N$ then $(a + b) = (a' + b') \bmod N$ and $ab = a'b' \bmod N$. A consequence is that we can "reduce and then add/multiply" instead of having to "add/multiply and then reduce," which can often simplify calculations.

Example 8.5
Let us compute $[1093028 \cdot 190301 \bmod 100]$. Since $1093028 = 28 \bmod 100$ and $190301 = 1 \bmod 100$, we have

$$\begin{aligned} 1093028 \cdot 190301 &= [1093028 \bmod 100] \cdot [190301 \bmod 100] \bmod 100 \\ &= 28 \cdot 1 = 28 \bmod 100. \end{aligned}$$

The alternate way of calculating the answer (i.e., computing the product $1093028 \cdot 190301$ and then reducing the result modulo 100) is less efficient. ♢

Congruence modulo N does *not* (in general) respect division. That is, if $a = a' \bmod N$ and $b = b' \bmod N$ then it is not necessarily true that $a/b = a'/b' \bmod N$; in fact, the expression "$a/b \bmod N$" is not always well-defined. As a specific example that often causes confusion, $ab = cb \bmod N$ does *not* necessarily imply that $a = c \bmod N$.

Example 8.6
Take $N = 24$. Then $3 \cdot 2 = 6 = 15 \cdot 2 \bmod 24$, but $3 \neq 15 \bmod 24$. ♢

In certain cases, however, we can define a meaningful notion of division. If for a given integer b there exists an integer c such that $bc = 1 \bmod N$, we say that b is *invertible* modulo N and call c a (multiplicative) *inverse* of b modulo N. Clearly, 0 is never invertible. It is also not difficult to show that if c is a multiplicative inverse of b modulo N then so is $[c \bmod N]$. Furthermore, if c' is another multiplicative inverse of b then $[c \bmod N] = [c' \bmod N]$. When b is invertible we can therefore simply let b^{-1} denote the *unique* multiplicative inverse of b that lies in the range $\{1, \ldots, N-1\}$.

When b is invertible modulo N, we define division by b modulo N as multiplication by b^{-1} (i.e., we define $[a/b \bmod N] \stackrel{\text{def}}{=} [ab^{-1} \bmod N]$). We stress that division by b is *only defined* when b is invertible. If $ab = cb \bmod N$ and b is invertible, then we may divide each side of the equation by b (or, really, multiply each side by b^{-1}) to obtain

$$(ab) \cdot b^{-1} = (cb) \cdot b^{-1} \bmod N \quad \Rightarrow \quad a = c \bmod N.$$

We see that in this case, division works "as expected." Invertible integers modulo N are therefore "nicer" to work with, in some sense.

The natural question is: which integers are invertible modulo a given modulus N? We can fully answer this question using Proposition 8.2:

PROPOSITION 8.7 *Let b, N be integers, with $b \geq 1$ and $N > 1$. Then b is invertible modulo N if and only if* $\gcd(b, N) = 1$.

PROOF Assume b is invertible modulo N, and let c denote its inverse. Since $bc = 1 \bmod N$, this implies that $bc - 1 = \gamma N$ for some $\gamma \in \mathbb{Z}$. Equiv-

alently, $bc - \gamma N = 1$. Since, by Proposition 8.2, $\gcd(b, N)$ is the smallest positive integer that can be expressed in this way, and there is no positive integer smaller than 1, this implies that $\gcd(b, N) = 1$.

Conversely, if $\gcd(b, N) = 1$ then by Proposition 8.2 there exist integers X, Y such that $Xb + YN = 1$. Reducing each side of this equation modulo N gives $Xb = 1 \bmod N$, and we see that X is a multiplicative inverse of b. (In fact, this gives an efficient algorithm to compute inverses.) ■

Example 8.8
Let $b = 11$ and $N = 17$. Then $(-3) \cdot 11 + 2 \cdot 17 = 1$, and so $14 = [-3 \bmod 17]$ is the inverse of 11. One can verify that $14 \cdot 11 = 1 \bmod 17$. ♢

Addition, subtraction, multiplication, and computation of inverses (when they exist) modulo N can all be carried out in polynomial time; see Appendix B.2. Exponentiation (i.e., computing $[a^b \bmod N]$ for $b > 0$ an integer) can also be computed in polynomial time; see Appendix B.2.3.

8.1.3 Groups

Let $\mathbb{G}$ be a set. A *binary operation* $\circ$ on $\mathbb{G}$ is simply a function $\circ(\cdot, \cdot)$ that takes as input two elements of $\mathbb{G}$. If $g, h \in \mathbb{G}$ then instead of using the cumbersome notation $\circ(g, h)$, we write $g \circ h$.

We now introduce the important notion of a *group*.

DEFINITION 8.9 *A* group *is a set $\mathbb{G}$ along with a binary operation $\circ$ for which the following conditions hold:*

- **(Closure:)** *For all $g, h \in \mathbb{G}$, $g \circ h \in \mathbb{G}$.*
- **(Existence of an identity:)** *There exists an* identity *$e \in \mathbb{G}$ such that for all $g \in \mathbb{G}$, $e \circ g = g = g \circ e$.*
- **(Existence of inverses:)** *For all $g \in \mathbb{G}$ there exists an element $h \in \mathbb{G}$ such that $g \circ h = e = h \circ g$. Such an h is called an* inverse *of g.*
- **(Associativity:)** *For all $g_1, g_2, g_3 \in \mathbb{G}$, $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.*

When $\mathbb{G}$ has a finite number of elements, we say $\mathbb{G}$ is finite *and let $|\mathbb{G}|$ denote the* order *of the group (that is, the number of elements in $\mathbb{G}$).*

A group $\mathbb{G}$ with operation $\circ$ is abelian *if the following holds:*

- **(Commutativity:)** *For all $g, h \in \mathbb{G}$, $g \circ h = h \circ g$.*

When the binary operation is understood, we simply call the set $\mathbb{G}$ a group.

We will always deal with finite, abelian groups. We will be careful to specify, however, when a result requires these assumptions.

Associativity implies that we do not need to include parentheses when writing long expressions; that is, the notation $g_1 \circ g_2 \circ \cdots \circ g_n$ is unambiguous since it does not matter in what order we evaluate the operation $\circ$.

One can show that the identity element in a group $\mathbb{G}$ is *unique*, and so we can therefore refer to *the* identity of a group. One can also show that each element g of a group has a *unique* inverse. See Exercise 8.1.

If $\mathbb{G}$ is a group, a set $\mathbb{H} \subseteq \mathbb{G}$ is a *subgroup of* $\mathbb{G}$ if $\mathbb{H}$ itself forms a group under the same operation associated with $\mathbb{G}$. To check that $\mathbb{H}$ is a subgroup, we need to verify closure, existence of identity and inverses, and associativity as per Definition 8.9. (In fact, associativity—as well as commutativity if $\mathbb{G}$ is abelian—is inherited automatically from $\mathbb{G}$.) Every group $\mathbb{G}$ always has the trivial subgroups $\mathbb{G}$ and $\{1\}$. We call $\mathbb{H}$ a *strict* subgroup of $\mathbb{G}$ if $\mathbb{H} \neq \mathbb{G}$.

In general, we will not use the notation $\circ$ to denote the group operation. Instead, we will use either *additive* notation or *multiplicative* notation depending on the group under discussion. *This does not imply that the group operation corresponds to integer addition or multiplication; it is merely useful notation.* When using additive notation, the group operation applied to two elements g, h is denoted $g + h$; the identity is denoted by 0; the inverse of an element g is denoted by $-g$; and we write $h - g$ in place of $h + (-g)$. When using multiplicative notation, the group operation applied to g, h is denoted by $g \cdot h$ or simply gh; the identity is denoted by 1; the inverse of an element g is denoted by g^{-1}; and we sometimes write h/g in place of hg^{-1}.

At this point, it may be helpful to see some examples.

Example 8.10

A set may be a group under one operation, but not another. For example, the set of integers $\mathbb{Z}$ is an abelian group under addition: the identity is the element 0, and every integer g has inverse $-g$. On the other hand, it is not a group under multiplication since, for example, the integer 2 does not have a multiplicative inverse in the integers. ♢

Example 8.11

The set of real numbers $\mathbb{R}$ is not a group under multiplication, since 0 does not have a multiplicative inverse. The set of *nonzero* real numbers, however, is an abelian group under multiplication with identity 1. ♢

The following example introduces the group $\mathbb{Z}_N$ that we will use frequently.

Example 8.12

Let $N > 1$ be an integer. The set $\{0, \ldots, N-1\}$ with respect to addition modulo N (i.e., where $a + b \stackrel{\text{def}}{=} [a + b \bmod N]$) is an abelian group of order N. Closure is obvious; associativity and commutativity follow from the fact that the integers satisfy these properties; the identity is 0; and, since $a + (N - a) = 0 \bmod N$, it follows that the inverse of any element a is $[(N - a) \bmod N]$. We

denote this group by $\mathbb{Z}_N$. (We will also sometimes use $\mathbb{Z}_N$ to denote the set $\{0, \ldots, N-1\}$ without regard to any particular group operation.) $\diamondsuit$

We end this section with an easy lemma that formalizes a "cancelation law" for groups.

LEMMA 8.13 *Let $\mathbb{G}$ be a group and $a, b, c \in \mathbb{G}$. If $ac = bc$, then $a = b$. In particular, if $ac = c$ then a is the identity in $\mathbb{G}$.*

PROOF We know $ac = bc$. Multiplying both sides by the unique inverse c^{-1} of c, we obtain $a = b$. In detail:

$$ac = bc \;\Rightarrow\; (ac)c^{-1} = (bc)\cdot c^{-1} \;\Rightarrow\; a(cc^{-1}) = b(cc^{-1}) \;\Rightarrow\; a \cdot 1 = b \cdot 1 \;\Rightarrow\; a = b.$$

■

Compare the above proof to the discussion (preceding Proposition 8.7) regarding a cancelation law for division modulo N. As indicated by the similarity, the *invertible* elements modulo N form a group under multiplication modulo N. We will return to this example in more detail shortly.

Group Exponentiation

It is often useful to be able to describe the group operation applied m times to a fixed element g, where m is a positive integer. When using additive notation, we express this as $m \cdot g$ or mg; that is,

$$mg = m \cdot g \stackrel{\text{def}}{=} \underbrace{g + \cdots + g}_{m \text{ times}}.$$

Note that m is an *integer*, while g is a *group element*. So mg does *not* represent the group operation applied to m and g (indeed, we are working in a group where the group operation is written additively). Thankfully, however, the notation "behaves as it should"; so, for example, if $g \in \mathbb{G}$ and m, m' are integers then $(mg) + (m'g) = (m + m')g$, $m(m'g) = (mm')g$, and $1 \cdot g = g$. In an abelian group $\mathbb{G}$ with $g, h \in \mathbb{G}$, $(mg) + (mh) = m(g + h)$.

When using multiplicative notation, we express application of the group operation m times to an element g by g^m. That is,

$$g^m \stackrel{\text{def}}{=} \underbrace{g \cdots g}_{m \text{ times}}.$$

The familiar rules of exponentiation hold: $g^m \cdot g^{m'} = g^{m+m'}$, $(g^m)^{m'} = g^{mm'}$, and $g^1 = g$. Also, if $\mathbb{G}$ is an abelian group and $g, h \in \mathbb{G}$ then $g^m \cdot h^m = (gh)^m$.

All these are simply "translations" of the results from the previous paragraph to the setting of groups written multiplicatively rather than additively.

The above notation is extended in the natural way to the case when m is zero or a negative integer. When using additive notation we define $0 \cdot g \stackrel{\text{def}}{=} 0$ and $(-m) \cdot g \stackrel{\text{def}}{=} m \cdot (-g)$ for m a positive integer. (Note that in the equation '$0 \cdot g = 0$' the 0 on the left-hand side is the integer 0 while the 0 on the right-hand side is the identity element in the group.) Observe that $-g$ is the inverse of g and, as one would expect, $(-m) \cdot g = -(mg)$. When using multiplicative notation, $g^0 \stackrel{\text{def}}{=} 1$ and $g^{-m} \stackrel{\text{def}}{=} (g^{-1})^m$. Again, g^{-1} is the inverse of g, and we have $g^{-m} = (g^m)^{-1}$.

Let $g \in \mathbb{G}$ and $b \geq 0$ be an integer. Then the exponentiation g^b can be computed using a polynomial number of underlying group operations in $\mathbb{G}$. Thus, if the group operation can be computed in polynomial time then so can exponentiation. This is discussed in Appendix B.2.3.

We now know enough to prove the following remarkable result:

THEOREM 8.14 *Let $\mathbb{G}$ be a finite group with $m = |\mathbb{G}|$, the order of the group. Then for any element $g \in \mathbb{G}$, $g^m = 1$.*

PROOF We prove the theorem only when $\mathbb{G}$ is abelian (although it holds for any finite group). Fix arbitrary $g \in \mathbb{G}$, and let $g_1, \ldots, g_m$ be the elements of $\mathbb{G}$. We claim that

$$g_1 \cdot g_2 \cdots g_m = (gg_1) \cdot (gg_2) \cdots (gg_m).$$

To see this, note that $gg_i = gg_j$ implies $g_i = g_j$ by Lemma 8.13. So each of the m elements in parentheses on the right-hand side is distinct. Because there are exactly m elements in $\mathbb{G}$, the m elements being multiplied together on the right-hand side are simply all elements of $\mathbb{G}$ in some permuted order. Since $\mathbb{G}$ is abelian, the order in which elements are multiplied does not matter, and so the right-hand side is equal to the left-hand side.

Again using the fact that $\mathbb{G}$ is abelian, we can "pull out" all occurrences of g and obtain

$$g_1 \cdot g_2 \cdots g_m = (gg_1) \cdot (gg_2) \cdots (gg_m) = g^m \cdot (g_1 \cdot g_2 \cdots g_m).$$

Appealing once again to Lemma 8.13, this implies $g^m = 1$. ■

An important corollary of the above is that we can work "modulo the group order" in the exponent:

COROLLARY 8.15 *Let $\mathbb{G}$ be a finite group with $m = |\mathbb{G}| > 1$. Then for any $g \in \mathbb{G}$ and any integer x, we have $g^x = g^{[x \bmod m]}$.*

PROOF Say $x = qm + r$, where q, r are integers and $r = [x \bmod m]$. Then

$$g^x = g^{qm+r} = g^{qm} \cdot g^r = (g^m)^q \cdot g^r = 1^q \cdot g^r = g^r$$

(using Theorem 8.14), as claimed. ■

Example 8.16
Written additively, the above corollary says that if g is an element in a group of order m, then $x \cdot g = [x \bmod m] \cdot g$. As an example, consider the group $\mathbb{Z}_{15}$ of order $m = 15$, and take $g = 11$. The corollary says that

$$152 \cdot 11 = [152 \bmod 15] \cdot 11 = 2 \cdot 11 = 11 + 11 = 22 = 7 \bmod 15.$$

The above agrees with the fact (cf. Example 8.5) that we can "reduce and then multiply" rather than having to "multiply and then reduce." ♢

Another corollary that will be extremely useful for cryptographic applications is the following:

COROLLARY 8.17 *Let $\mathbb{G}$ be a finite group with $m = |\mathbb{G}| > 1$. Let $e > 0$ be an integer, and define the function $f_e : \mathbb{G} \to \mathbb{G}$ by $f_e(g) = g^e$. If $\gcd(e, m) = 1$, then f_e is a permutation (i.e., a bijection). Moreover, if $d = e^{-1} \bmod m$ then f_d is the inverse of f_e. (Note by Proposition 8.7, $\gcd(e, m) = 1$ implies e is invertible modulo m.)*

PROOF Since $\mathbb{G}$ is finite, the second part of the claim implies the first; thus, we need only show that f_d is the inverse of f_e. This is true because for any $g \in \mathbb{G}$ we have

$$f_d(f_e(g)) = f_d(g^e) = (g^e)^d = g^{ed} = g^{[ed \bmod m]} = g^1 = g,$$

where the fourth equality follows from Corollary 8.15. ■

8.1.4 The Group $\mathbb{Z}_N^*$

As discussed in Example 8.12, the set $\mathbb{Z}_N = \{0, \ldots, N-1\}$ is a group under addition modulo N. Can we define a group with respect to *multiplication* modulo N? In doing so, we will have to eliminate those elements in $\mathbb{Z}_N$ that are not invertible; e.g., we will have to eliminate 0 since it has no multiplicative inverse. Nonzero elements may also fail to be invertible (cf. Proposition 8.7).

Which elements $b \in \{1, \ldots, N-1\}$ are invertible modulo N? Proposition 8.7 says that these are exactly those elements b for which $\gcd(b, N) = 1$. We have

also seen in Section 8.1.2 that whenever b is invertible, it has an inverse lying in the range $\{1, \ldots, N-1\}$. This leads us to define, for any $N > 1$, the set

$$\mathbb{Z}_N^* \stackrel{\text{def}}{=} \{b \in \{1, \ldots, N-1\} \mid \gcd(b, N) = 1\};$$

i.e., $\mathbb{Z}_N^*$ consists of integers in the set $\{1, \ldots, N-1\}$ that are relatively prime to N. The group operation is multiplication modulo N; i.e., $ab \stackrel{\text{def}}{=} [ab \bmod N]$.

We claim that $\mathbb{Z}_N^*$ is an abelian group with respect to this operation. Since 1 is always in $\mathbb{Z}_N^*$, the set clearly contains an identity element. The discussion above shows that each element in $\mathbb{Z}_N^*$ has a multiplicative inverse in the same set. Commutativity and associativity follow from the fact that these properties hold over the integers. To show that closure holds, let $a, b \in \mathbb{Z}_N^*$; then $[ab \bmod N]$ has inverse $[b^{-1}a^{-1} \bmod N]$, which means that $\gcd([ab \bmod N], N) = 1$ and so $ab \in \mathbb{Z}_N^*$. Summarizing:

PROPOSITION 8.18 *Let $N > 1$ be an integer. Then $\mathbb{Z}_N^*$ is an abelian group under multiplication modulo N.*

Define $\phi(N) \stackrel{\text{def}}{=} |\mathbb{Z}_N^*|$, the order of the group $\mathbb{Z}_N^*$. (ϕ is called the *Euler phi function*.) What is the value of $\phi(N)$? First consider the case when $N = p$ is prime. Then *all* elements in $\{1, \ldots, p-1\}$ are relatively prime to p, and so $\phi(p) = |\mathbb{Z}_p^*| = p - 1$. Next consider the case that $N = pq$, where p, q are distinct primes. If an integer $a \in \{1, \ldots, N-1\}$ is not relatively prime to N, then either $p \mid a$ or $q \mid a$ (a cannot be divisible by both p and q since this would imply $pq \mid a$ but $a < N = pq$). The elements in $\{1, \ldots, N-1\}$ divisible by p are exactly the $(q-1)$ elements $p, 2p, 3p, \ldots, (q-1)p$, and the elements divisible by q are exactly the $(p-1)$ elements $q, 2q, \ldots, (p-1)q$. The number of elements remaining (i.e., those that are neither divisible by p nor q) is therefore given by

$$(N-1) - (q-1) - (p-1) = pq - p - q + 1 = (p-1)(q-1).$$

We have thus proved that $\phi(N) = (p-1)(q-1)$ when N is the product of two distinct primes p and q.

You are asked to prove the following general result (used only rarely in the rest of the book) in Exercise 8.4:

THEOREM 8.19 *Let $N = \prod_i p_i^{e_i}$, where the $\{p_i\}$ are distinct primes and $e_i \geq 1$. Then $\phi(N) = \prod_i p_i^{e_i - 1}(p_i - 1)$.*

Example 8.20
Take $N = 15 = 5 \cdot 3$. Then $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ and $|\mathbb{Z}_{15}^*| = 8 = 4 \cdot 2 = \phi(15)$. The inverse of 8 in $\mathbb{Z}_{15}^*$ is 2, since $8 \cdot 2 = 16 = 1 \bmod 15$. ♢

We have shown that $\mathbb{Z}_N^*$ is a group of order $\phi(N)$. The following are now easy corollaries of Theorem 8.14 and Corollary 8.17:

COROLLARY 8.21 *Take arbitrary integer $N > 1$ and $a \in \mathbb{Z}_N^*$. Then*

$$a^{\phi(N)} = 1 \bmod N.$$

For the specific case that $N = p$ is prime and $a \in \{1, \ldots, p-1\}$, we have

$$a^{p-1} = 1 \bmod p.$$

COROLLARY 8.22 *Fix $N > 1$. For integer $e > 0$ define $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ by $f_e(x) = [x^e \bmod N]$. If e is relatively prime to $\phi(N)$ then f_e is a permutation. Moreover, if $d = e^{-1} \bmod \phi(N)$ then f_d is the inverse of f_e.*

8.1.5 *Isomorphisms and the Chinese Remainder Theorem

Two groups are *isomorphic* if they have the same underlying structure. From a mathematical point of view, an isomorphism of a group $\mathbb{G}$ provides an alternate, but equivalent, way of thinking about $\mathbb{G}$. From a computational perspective, an isomorphism provides a different way to *represent* elements in $\mathbb{G}$, which can often have a significant impact on algorithmic efficiency.

DEFINITION 8.23 *Let $\mathbb{G}, \mathbb{H}$ be groups with respect to the operations $\circ_{\mathbb{G}}, \circ_{\mathbb{H}}$, respectively. A function $f : \mathbb{G} \to \mathbb{H}$ is an* isomorphism from $\mathbb{G}$ to $\mathbb{H}$ *if:*

1. *f is a bijection, and*
2. *For all $g_1, g_2 \in \mathbb{G}$ we have $f(g_1 \circ_{\mathbb{G}} g_2) = f(g_1) \circ_{\mathbb{H}} f(g_2)$.*

If there exists an isomorphism from $\mathbb{G}$ to $\mathbb{H}$ then we say that these groups are isomorphic *and write $\mathbb{G} \simeq \mathbb{H}$.*

In essence, an isomorphism from $\mathbb{G}$ to $\mathbb{H}$ is just a *renaming* of elements of $\mathbb{G}$ as elements of $\mathbb{H}$. Note that if $\mathbb{G}$ is finite and $\mathbb{G} \simeq \mathbb{H}$, then $\mathbb{H}$ must be finite and of the same size as $\mathbb{G}$. Also, if there exists an isomorphism f from $\mathbb{G}$ to $\mathbb{H}$ then f^{-1} is an isomorphism from $\mathbb{H}$ to $\mathbb{G}$. It is possible, however, that f is efficiently computable while f^{-1} is not (or vice versa).

The aim of this section is to use the language of isomorphisms to better understand the group structure of $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$ when $N = pq$ is a product of two distinct primes. We first need to introduce the notion of a *direct product* of groups. Given groups $\mathbb{G}, \mathbb{H}$ with group operations $\circ_{\mathbb{G}}, \circ_{\mathbb{H}}$, respectively, we define a new group $\mathbb{G} \times \mathbb{H}$ (the *direct product of* $\mathbb{G}$ *and* $\mathbb{H}$) as follows. The elements of $\mathbb{G} \times \mathbb{H}$ are ordered pairs (g, h) with $g \in \mathbb{G}$ and $h \in \mathbb{H}$; thus, if $\mathbb{G}$

has n elements and $\mathbb{H}$ has n' elements, $\mathbb{G} \times \mathbb{H}$ has $n \cdot n'$ elements. The group operation $\circ$ on $\mathbb{G} \times \mathbb{H}$ is applied component-wise; that is:

$$(g, h) \circ (g', h') \stackrel{\text{def}}{=} (g \circ_{\mathbb{G}} g',\ h \circ_{\mathbb{H}} h').$$

We leave it to Exercise 8.8 to verify that $\mathbb{G} \times \mathbb{H}$ is indeed a group. The above notation can be extended to direct products of more than two groups in the natural way, although we will not need this for what follows.

We may now state and prove the *Chinese remainder theorem*.

THEOREM 8.24 (Chinese remainder theorem) *Let $N = pq$ where $p, q > 1$ are relatively prime. Then*

$$\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad \textit{and} \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*.$$

Moreover, let f be the function mapping elements $x \in \{0, \ldots, N-1\}$ to pairs (x_p, x_q) with $x_p \in \{0, \ldots, p-1\}$ and $x_q \in \{0, \ldots, q-1\}$ defined by

$$f(x) \stackrel{\text{def}}{=} ([x \bmod p], [x \bmod q]).$$

Then f is an isomorphism from $\mathbb{Z}_N$ to $\mathbb{Z}_p \times \mathbb{Z}_q$, and the restriction of f to $\mathbb{Z}_N^*$ is an isomorphism from $\mathbb{Z}_N^*$ to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.

PROOF For any $x \in \mathbb{Z}_N$ the output $f(x)$ is a pair of elements (x_p, x_q) with $x_p \in \mathbb{Z}_p$ and $x_q \in \mathbb{Z}_q$. We claim that if $x \in \mathbb{Z}_N^*$, then $(x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$. Indeed, if $x_p \notin \mathbb{Z}_p^*$ then this means that $\gcd([x \bmod p],\, p) \neq 1$. But then $\gcd(x, p) \neq 1$. This implies $\gcd(x, N) \neq 1$, contradicting the assumption that $x \in \mathbb{Z}_N^*$. (An analogous argument holds if $x_q \notin \mathbb{Z}_q^*$.)

We now show that f is an isomorphism from $\mathbb{Z}_N$ to $\mathbb{Z}_p \times \mathbb{Z}_q$. (The proof that it is an isomorphism from $\mathbb{Z}_N^*$ to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$ is similar.) Let us start by proving that f is one-to-one. Say $f(x) = (x_p, x_q) = f(x')$. Then $x = x_p = x' \bmod p$ and $x = x_q = x' \bmod q$. This in turn implies that $(x - x')$ is divisible by both p and q. Since $\gcd(p, q) = 1$, Proposition 8.4 says that $pq = N$ divides $(x - x')$. But then $x = x' \bmod N$. For $x, x' \in \mathbb{Z}_N$, this means that $x = x'$ and so f is indeed one-to-one. Since $|\mathbb{Z}_N| = N = p \cdot q = |\mathbb{Z}_p| \cdot |\mathbb{Z}_q|$, the sizes of $\mathbb{Z}_N$ and $\mathbb{Z}_p \times \mathbb{Z}_q$ are the same. This in combination with the fact that f is one-to-one implies that f is bijective.

In the following paragraph, let $+_N$ denote addition modulo N, and let $\boxplus$ denote the group operation in $\mathbb{Z}_p \times \mathbb{Z}_q$ (i.e., addition modulo p in the first component and addition modulo q in the second component). To conclude the proof that f is an isomorphism from $\mathbb{Z}_N$ to $\mathbb{Z}_p \times \mathbb{Z}_q$, we need to show that for all $a, b \in \mathbb{Z}_N$ it holds that $f(a +_N b) = f(a) \boxplus f(b)$.

To see that this is true, note that

$$\begin{aligned} f(a +_N b) &= \Big([(a +_N b) \bmod p],\, [(a +_N b) \bmod q]\Big) \\ &= \Big([(a + b) \bmod p],\, [(a + b) \bmod q]\Big) \\ &= \Big([a \bmod p],\, [a \bmod q]\Big) \boxplus \Big([b \bmod p],\, [b \bmod q]\Big) = f(a) \boxplus f(b). \end{aligned}$$

(For the second equality, above, we use the fact that $[[X \bmod N] \bmod p\,] = [[X \bmod p\,] \bmod p\,]$ when $p \mid N$; see Exercise 8.9.) ■

An extension of the Chinese remainder theorem says that if $p_1, p_2, \ldots, p_\ell$ are pairwise relatively prime (i.e., $\gcd(p_i, p_j) = 1$ for all $i \neq j$) and $N \stackrel{\text{def}}{=} \prod_{i=1}^{\ell} p_i$, then

$$\mathbb{Z}_N \simeq \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\ell} \quad \text{and} \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_{p_1}^* \times \cdots \times \mathbb{Z}_{p_\ell}^*.$$

An isomorphism in each case is obtained by a natural extension of the one used in the theorem above.

By way of notation, with N understood and $x \in \{0, 1, \ldots, N-1\}$ we write $x \leftrightarrow (x_p, x_q)$ for $x_p = [x \bmod p]$ and $x_q = [x \bmod q]$. That is, $x \leftrightarrow (x_p, x_q)$ if and only if $f(x) = (x_p, x_q)$, where f is as in the theorem above. One way to think about this notation is that it means "x (in $\mathbb{Z}_N$) *corresponds to* (x_p, x_q) (in $\mathbb{Z}_p \times \mathbb{Z}_q$)." The same notation is used when dealing with $x \in \mathbb{Z}_N^*$.

Example 8.25
Take $15 = 5 \cdot 3$, and consider $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. The Chinese remainder theorem says this group is isomorphic to $\mathbb{Z}_5^* \times \mathbb{Z}_3^*$. We can compute

$$\begin{array}{cccc} 1 \leftrightarrow (1,1) & 2 \leftrightarrow (2,2) & 4 \leftrightarrow (4,1) & 7 \leftrightarrow (2,1) \\ 8 \leftrightarrow (3,2) & 11 \leftrightarrow (1,2) & 13 \leftrightarrow (3,1) & 14 \leftrightarrow (4,2)\,, \end{array}$$

where each pair (a, b) with $a \in \mathbb{Z}_5^*$ and $b \in \mathbb{Z}_3^*$ appears exactly once. ♢

Using the Chinese Remainder Theorem

If two groups are isomorphic, then they both serve as representations of the same underlying "algebraic structure." Nevertheless, the choice of which representation to use can affect the *computational efficiency* of group operations. We discuss this abstractly, and then in the specific context of $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$.

Let $\mathbb{G}, \mathbb{H}$ be groups with operations $\circ_{\mathbb{G}}, \circ_{\mathbb{H}}$, respectively, and say f is an isomorphism from $\mathbb{G}$ to $\mathbb{H}$ where both f and f^{-1} can be computed efficiently. Then for $g_1, g_2 \in \mathbb{G}$ we can compute $g = g_1 \circ_{\mathbb{G}} g_2$ in two ways: either by directly computing the group operation in $\mathbb{G}$, or via the following steps:

1. Compute $h_1 = f(g_1)$ and $h_2 = f(g_2)$;

2. Compute $h = h_1 \circ_{\mathbb{H}} h_2$ using the group operation in $\mathbb{H}$;
3. Compute $g = f^{-1}(h)$.

The above extends in the natural way when we want to compute multiple group operations in $\mathbb{G}$ (e.g., to compute g^x for some integer x). Which method is better depends on the relative efficiency of computing the group operation in each group, as well as the efficiency of computing f and f^{-1}.

We now turn to the specific case of computations modulo N, when $N = pq$ is a product of distinct primes. The Chinese remainder theorem shows that addition, multiplication, or exponentiation (which is just repeated multiplication) modulo N can be "transformed" to analogous operations modulo p and q. Building on Example 8.25, we show some simple examples with $N = 15$.

Example 8.26
Say we want to compute the product $14 \cdot 13$ modulo 15 (i.e., in $\mathbb{Z}_{15}^*$). Example 8.25 gives $14 \leftrightarrow (4, 2)$ and $13 \leftrightarrow (3, 1)$. In $\mathbb{Z}_5^* \times \mathbb{Z}_3^*$, we have

$$(4, 2) \cdot (3, 1) = ([4 \cdot 3 \bmod 5],\ [2 \cdot 1 \bmod 3]) = (2, 2).$$

Note $(2, 2) \leftrightarrow 2$, which is the correct answer since $14 \cdot 13 = 2 \bmod 15$. ♢

Example 8.27
Say we want to compute $11^{53} \bmod 15$. Example 8.25 gives $11 \leftrightarrow (1, 2)$. Notice that $2 = -1 \bmod 3$ and so

$$(1, 2)^{53} = ([1^{53} \bmod 5], [(-1)^{53} \bmod 3]) = (1, [-1 \bmod 3]) = (1, 2).$$

Thus, $11^{53} \bmod 15 = 11$. ♢

Example 8.28
Say we want to compute $[29^{100} \bmod 35]$. We first compute the correspondence $29 \leftrightarrow ([29 \bmod 5],\ \ [29 \bmod 7]) = ([-1 \bmod 5], 1)$. Using the Chinese remainder theorem, we have

$$([-1 \bmod 5], 1)^{100} = ([(-1)^{100} \bmod 5],\ [1^{100} \bmod 7]) = (1, 1),$$

and it is immediate that $(1, 1) \leftrightarrow 1$. We conclude that $[29^{100} \bmod 35] = 1$. ♢

Example 8.29
Say we want to compute $[18^{25} \bmod 35]$. We have $18 \leftrightarrow (3, 4)$ and so

$$18^{25} \bmod 35 \leftrightarrow (3, 4)^{25} = ([3^{25} \bmod 5],\ [4^{25} \bmod 7]).$$

Since $\mathbb{Z}_5^*$ is a group of order 4, we can "work modulo 4 in the exponent" (cf. Corollary 8.15) and see that

$$3^{25} = 3^{[25 \bmod 4]} = 3^1 = 3 \bmod 5.$$

Similarly,

$$4^{25} = 4^{[25 \bmod 6]} = 4^1 = 4 \bmod 7.$$

Thus, $([3^{25} \bmod 5], [4^{25} \bmod 7]) = (3, 4) \leftrightarrow 18$ and so $[18^{25} \bmod 35] = 18$. $\diamondsuit$

One thing we have not yet discussed is how to convert back and forth between the representation of an element modulo N and its representation modulo p and q. The conversion can be carried out efficiently provided the factorization of N is known. Assuming p and q are known, it is easy to map an element x modulo N to its corresponding representation modulo p and q: the element x corresponds to $([x \bmod p], [x \bmod q])$, and both the modular reductions can be carried out efficiently (cf. Appendix B.2).

For the other direction, we make use of the following observation: an element with representation (x_p, x_q) can be written as

$$(x_p, x_q) = x_p \cdot (1, 0) + x_q \cdot (0, 1).$$

So, if we can find elements $1_p, 1_q \in \{0, \ldots, N-1\}$ such that $1_p \leftrightarrow (1, 0)$ and $1_q \leftrightarrow (0, 1)$, then (appealing to the Chinese remainder theorem) we know that

$$(x_p, x_q) \leftrightarrow [(x_p \cdot 1_p + x_q \cdot 1_q) \bmod N].$$

Since p, q are distinct primes, $\gcd(p, q) = 1$. We can use the extended Euclidean algorithm (cf. Appendix B.1.2) to find integers X, Y such that

$$Xp + Yq = 1.$$

Note that $Yq = 0 \bmod q$ and $Yq = 1 - Xp = 1 \bmod p$. This means that $[Yq \bmod N] \leftrightarrow (1, 0)$; i.e., $[Yq \bmod N] = 1_p$. Similarly, $[Xp \bmod N] = 1_q$.

In summary, we can convert an element represented as (x_p, x_q) to its representation modulo N in the following way (assuming p and q are known):

1. Compute X, Y such that $Xp + Yq = 1$.
2. Set $1_p := [Yq \bmod N]$ and $1_q := [Xp \bmod N]$.
3. Compute $x := [(x_p \cdot 1_p + x_q \cdot 1_q) \bmod N]$.

If many such conversions will be performed, then $1_p, 1_q$ can be computed once-and-for-all in a preprocessing phase.

Example 8.30

Take $p = 5$, $q = 7$, and $N = 5 \cdot 7 = 35$. Say we are given the representation $(4, 3)$ and want to convert this to the corresponding element of $\mathbb{Z}_{35}$. Using the extended Euclidean algorithm, we compute

$$3 \cdot 5 - 2 \cdot 7 = 1.$$

Thus, $1_p = [-2 \cdot 7 \bmod 35] = 21$ and $1_q = [3 \cdot 5 \bmod 35] = 15$. (We can check that these are correct: e.g., for $1_p = 21$ we can verify that $[21 \bmod 5] = 1$ and $[21 \bmod 7] = 0$.) Using these values, we can then compute

$$\begin{aligned}(4,3) &= 4 \cdot (1,0) + 3 \cdot (0,1) \\ &\leftrightarrow [4 \cdot 1_p + 3 \cdot 1_q \bmod 35] \\ &= [4 \cdot 21 + 3 \cdot 15 \bmod 35] \;\; = \;\; 24.\end{aligned}$$

Since $24 = 4 \bmod 5$ and $24 = 3 \bmod 7$, this is indeed the correct result. ♢

8.2 Primes, Factoring, and RSA

In this section, we show the first examples of number-theoretic problems that are conjectured to be "hard." We begin with a discussion of one of the oldest problems: *integer factorization* or just *factoring*.

Given a composite integer N, the factoring problem is to find integers $p, q > 1$ such that $pq = N$. Factoring is a classic example of a hard problem, both because it is so simple to describe and since it has been recognized as a hard computational problem for a long time (even before its use in cryptography). The problem can be solved in *exponential* time $\mathcal{O}(\sqrt{N} \cdot \mathsf{polylog}(N))$ using *trial division*: that is, by exhaustively checking whether p divides N for $p = 2, \ldots, \lfloor \sqrt{N} \rfloor$. (This method requires $\sqrt{N}$ divisions, each one taking $\mathsf{polylog}(N) = \|N\|^c$ time for some constant c.) This always succeeds because although the *largest* prime factor of N may be as large as $N/2$, the *smallest* prime factor of N can be at most $\lfloor \sqrt{N} \rfloor$. Although algorithms with better running time are known (see Chapter 9), no *polynomial-time* algorithm for factoring has been demonstrated despite many years of effort.

Consider the following experiment for a given algorithm $\mathcal{A}$ and parameter n:

The weak factoring experiment $\mathsf{w\text{-}Factor}_{\mathcal{A}}(n)$**:**

1. *Choose two uniform n-bit integers x_1, x_2.*
2. *Compute $N := x_1 \cdot x_2$.*
3. *$\mathcal{A}$ is given N, and outputs $x_1', x_2' > 1$.*
4. *The output of the experiment is defined to be 1 if $x_1' \cdot x_2' = N$, and 0 otherwise.*

We have just said that the factoring problem is believed to be hard. Does this mean that

$$\Pr[\mathsf{w\text{-}Factor}_{\mathcal{A}}(n) = 1] \leq \mathsf{negl}(n)$$

is negligible for every PPT algorithm $\mathcal{A}$? Not at all. For starters, the number N in the above experiment is *even* with probability 3/4 (this occurs when

either x_1 or x_2 is even); it is, of course, easy for $\mathcal{A}$ to factor N in this case. While we can make $\mathcal{A}$'s job more difficult by requiring $\mathcal{A}$ to output integers x'_1, x'_2 of length n, it remains the case that x_1 or x_2 (and hence N) might have small prime factors that can still be easily found. For cryptographic applications, we will need to prevent this.

As this discussion indicates, the "hardest" numbers to factor are those having only large prime factors. This suggests redefining the above experiment so that x_1, x_2 are random n-bit *primes* rather than random n-bit *integers*, and in fact such an experiment will be used when we formally define the factoring assumption in Section 8.2.3. For this experiment to be useful in a cryptographic setting, however, it is necessary to be able to generate random n-bit primes *efficiently*. This is the topic of the next two sections.

8.2.1 Generating Random Primes

A natural approach to generating a random n-bit prime is to repeatedly choose random n-bit integers until we find one that is prime; we repeat this at most t times or until we are successful. See Algorithm 8.31 for a high-level description of the process.

ALGORITHM 8.31
Generating a random prime – high-level outline

Input: Length n; parameter t
Output: A uniform n-bit prime

for $i = 1$ to t:
  $p' \leftarrow \{0,1\}^{n-1}$
  $p := 1 \| p'$
  if p is prime **return** p
return fail

Note that the algorithm forces the output to be an integer of length *exactly* n (rather than length *at most* n) by fixing the high-order bit of p to "1." Our convention throughout this book is that an "integer of length n" means an integer whose binary representation *with most significant bit equal to* 1 is exactly n bits long.

Given a way to determine whether or not a given integer p is prime, the above algorithm outputs a *uniform* n-bit prime conditioned on the event that it does not output fail. The probability that the algorithm outputs fail depends on t, and for our purposes we will want to set t so as to obtain a failure probability that is negligible in n. To show that Algorithm 8.31 leads to an *efficient* (i.e., polynomial-time in n) algorithm for generating primes, we need a better understanding of two issues: (1) the probability that a uniform n-bit integer is prime and (2) how to efficiently test whether a given integer

p is prime. We discuss these issues briefly now, and defer a more in-depth exploration of the second topic to the following section.

The distribution of primes. The *prime number theorem*, an important result in mathematics, gives fairly precise bounds on the fraction of integers of a given length that are prime. For our purposes, we need only a weak, one-sided version of that result that we do not prove here:

THEOREM 8.32 (Bertrand's postulate) *For any $n > 1$, the fraction of n-bit integers that are prime is at least $1/3n$.*

Returning to the approach for generating primes described above, this implies that if we set $t = 3n^2$ then the probability that a prime is *not* chosen in all t iterations of the algorithm is at most

$$\left(1 - \frac{1}{3n}\right)^t = \left(\left(1 - \frac{1}{3n}\right)^{3n}\right)^n \leq \left(e^{-1}\right)^n = e^{-n}$$

(using Inequality A.2), which is negligible in n. Thus, using $\mathsf{poly}(n)$ iterations we obtain an algorithm for which the probability of outputting fail is negligible in n. (Tighter results than Theorem 8.32 are known, and so in practice even fewer iterations are needed.)

Testing primality. The problem of efficiently determining whether a given number is prime has a long history. In the 1970s the first efficient algorithms for testing primality were developed. These algorithms were *probabilistic* and had the following guarantee: if the input p were a prime number, the algorithm would always output "prime." On the other hand, if p were composite, then the algorithm would almost always output "composite," but might output the wrong answer ("prime") with probability negligible in the length of p. Put differently, if the algorithm outputs "composite" then p is definitely composite, but if the output is "prime" then it is very likely that p is prime but it is also possible that a mistake has occurred (and p is really composite).[2]

When using a randomized primality test of this sort in Algorithm 8.31 (the prime-generation algorithm shown earlier), the output of the algorithm is a uniform prime of the desired length so long as the algorithm does not output fail *and* the randomized primality test did not err during the execution of the algorithm. This means that an additional source of error (besides the possibility of outputting fail) is introduced, and the algorithm may now output a composite number by mistake. Since we can ensure that this happens with only negligible probability, this remote possibility is of no practical concern and we can safely ignore it.

[2]There also exist probabilistic primality tests that work in the opposite way: they always correctly identify composite numbers but sometimes make a mistake when given a prime as input. We will not consider algorithms of this type.

A *deterministic* polynomial-time algorithm for testing primality was demonstrated in a breakthrough result in 2002. That algorithm, although running in polynomial time, is slower than the probabilistic tests mentioned above. For this reason, probabilistic primality tests are still used exclusively in practice for generating large prime numbers.

In Section 8.2.2 we describe and analyze one of the most commonly used probabilistic primality tests: the *Miller–Rabin* algorithm. This algorithm takes two inputs: an integer p and a parameter t (in unary) that determines the error probability. The Miller–Rabin algorithm runs in time polynomial in $\|p\|$ and t, and satisfies:

THEOREM 8.33 *If p is prime, then the Miller–Rabin test always outputs "prime." If p is composite, the algorithm outputs "composite" except with probability at most 2^{-t}.*

Putting it all together. Given the preceding discussion, we can now describe a polynomial-time prime-generation algorithm that, on input n, outputs an n-bit prime except with probability negligible in n; moreover, conditioned on the output p being prime, p is a uniformly distributed n-bit prime. The full procedure is described in Algorithm 8.34.

ALGORITHM 8.34
Generating a random prime

Input: Length n
Output: A uniform n-bit prime

for $i = 1$ to $3n^2$:
  $p' \leftarrow \{0,1\}^{n-1}$
  $p := 1 \| p'$
  run the Miller–Rabin test on input p and parameter 1^n
  if the output is "prime," **return** p
return fail

Generating primes of a particular form. It is sometimes desirable to generate a random n-bit prime p of a particular form, for example, satisfying $p = 3 \bmod 4$ or such that $p = 2q + 1$ where q is also prime (p of the latter type are called *strong primes*). In this case, appropriate modifications of the prime-generation algorithm shown above can be used. (For example, in order to obtain a prime of the form $p = 2q + 1$, modify the algorithm to generate a random prime q, compute $p := 2q + 1$, and then output p if it too is prime.) While these modified algorithms work well in practice, rigorous proofs that they run in polynomial time and fail with only negligible probability are more complex (and, in some cases, rely on unproven number-theoretic conjectures

regarding the density of primes of a particular form). A detailed exploration of these issues is beyond the scope of this book, and we will simply assume the existence of appropriate prime-generation algorithms when needed.

8.2.2 *Primality Testing

We now describe the Miller–Rabin primality test and prove Theorem 8.33. (We rely on the material presented in Section 8.1.5.) This material is not used directly in the rest of the book.

The key to the Miller–Rabin algorithm is to find a property that distinguishes primes and composites. Let N denote the input number to be tested. We start with the following observation: if N is prime then $|\mathbb{Z}_N^*| = N - 1$, and so for any $a \in \{1, \ldots, N-1\}$ we have $a^{N-1} = 1 \bmod N$ by Theorem 8.14. This suggests testing whether N is prime by choosing a uniform element a and checking whether $a^{N-1} \stackrel{?}{=} 1 \bmod N$. If $a^{N-1} \neq 1 \bmod N$, then N cannot be prime. Conversely, we might hope that if N is not prime then there is a reasonable chance that we will pick a with $a^{N-1} \neq 1 \bmod N$, and so by repeating this test many times we can determine whether N is prime or not with high confidence. The above approach is shown as Algorithm 8.35. (Recall that exponentiation modulo N and computation of greatest common divisors can be carried out in polynomial time. Choosing a uniform element of $\{1, \ldots, N-1\}$ can also be done in polynomial time. See Appendix B.2.)

ALGORITHM 8.35
Primality testing – first attempt

Input: Integer N and parameter 1^t
Output: A decision as to whether N is prime or composite

for $i = 1$ to t:
  $a \leftarrow \{1, \ldots, N-1\}$
  if $a^{N-1} \neq 1 \bmod N$ **return** "composite"
return "prime"

If N is prime the algorithm always outputs "prime." If N is composite, the algorithm outputs "composite" if in any iteration it finds an $a \in \{1, \ldots, N-1\}$ such that $a^{N-1} \neq 1 \bmod N$. Observe that if $a \notin \mathbb{Z}_N^*$ then $a^{N-1} \neq 1 \bmod N$. (If $\gcd(a, N) \neq 1$ then $\gcd(a^{N-1}, N) \neq 1$ and so $[a^{N-1} \bmod N]$ cannot equal 1.) For now, we therefore restrict our attention to $a \in \mathbb{Z}_N^*$. We refer to any such a with $a^{N-1} \neq 1 \bmod N$ as a *witness that N is composite*, or simply a *witness*. We might hope that when N is composite there are *many* witnesses, and thus the algorithm finds such a witness with "high" probability. This intuition is correct *provided there is at least one witness*. Before proving this, we need two group-theoretic lemmas.

PROPOSITION 8.36 *Let $\mathbb{G}$ be a finite group, and $\mathbb{H} \subseteq \mathbb{G}$. Assume $\mathbb{H}$ is nonempty, and for all $a, b \in \mathbb{H}$ we have $ab \in \mathbb{H}$. Then $\mathbb{H}$ is a subgroup of $\mathbb{G}$.*

PROOF We need to verify that $\mathbb{H}$ satisfies all the conditions of Definition 8.9. By assumption, $\mathbb{H}$ is closed under the group operation. Associativity in $\mathbb{H}$ is inherited automatically from $\mathbb{G}$. Let $m = |\mathbb{G}|$ (here is where we use the fact that $\mathbb{G}$ is finite), and consider an arbitrary element $a \in \mathbb{H}$. Closure of $\mathbb{H}$ means that $\mathbb{H}$ contains $a^{m-1} = a^{-1}$ as well as $a^m = 1$. Thus, $\mathbb{H}$ contains the inverse of each of its elements, as well as the identity. ■

LEMMA 8.37 *Let $\mathbb{H}$ be a strict subgroup of a finite group $\mathbb{G}$ (i.e., $\mathbb{H} \neq \mathbb{G}$). Then $|\mathbb{H}| \leq |\mathbb{G}|/2$.*

PROOF Let $\bar{h}$ be an element of $\mathbb{G}$ that is *not* in $\mathbb{H}$; since $\mathbb{H} \neq \mathbb{G}$, we know such an $\bar{h}$ exists. Consider the set $\bar{\mathbb{H}} \stackrel{\text{def}}{=} \{\bar{h}h \mid h \in \mathbb{H}\}$. We show that (1) $|\bar{\mathbb{H}}| = |\mathbb{H}|$, and (2) every element of $\bar{\mathbb{H}}$ lies outside of $\mathbb{H}$; i.e., the intersection of $\mathbb{H}$ and $\bar{\mathbb{H}}$ is empty. Since both $\mathbb{H}$ and $\bar{\mathbb{H}}$ are subsets of $\mathbb{G}$, these imply $|\mathbb{G}| \geq |\mathbb{H}| + |\bar{\mathbb{H}}| = 2|\mathbb{H}|$, proving the lemma.

For any $h_1, h_2 \in \mathbb{H}$, if $\bar{h}h_1 = \bar{h}h_2$ then, multiplying by $\bar{h}^{-1}$ on each side, we have $h_1 = h_2$. This shows that every distinct element $h \in \mathbb{H}$ corresponds to a distinct element $\bar{h}h \in \bar{\mathbb{H}}$, proving (1).

Assume toward a contradiction that $\bar{h}h \in \mathbb{H}$ for some h. This means $\bar{h}h = h'$ for some $h' \in \mathbb{H}$, and so $\bar{h} = h'h^{-1}$. Now, $h'h^{-1} \in \mathbb{H}$ since $\mathbb{H}$ is a subgroup and $h', h^{-1} \in \mathbb{H}$. But this means that $\bar{h} \in \mathbb{H}$, in contradiction to the way $\bar{h}$ was chosen. This proves (2) and completes the proof of the lemma. ■

The following theorem will enable us to analyze the algorithm given earlier.

THEOREM 8.38 *Fix N. Say there exists a witness that N is composite. Then at least half the elements of $\mathbb{Z}_N^*$ are witnesses that N is composite.*

PROOF Let Bad be the set of elements in $\mathbb{Z}_N^*$ that are *not* witnesses; that is, $a \in \mathsf{Bad}$ means $a^{N-1} = 1 \bmod N$. Clearly, $1 \in \mathsf{Bad}$. If $a, b \in \mathsf{Bad}$, then $(ab)^{N-1} = a^{N-1} \cdot b^{N-1} = 1 \cdot 1 = 1 \bmod N$ and hence $ab \in \mathsf{Bad}$. By Lemma 8.36, we conclude that Bad is a subgroup of $\mathbb{Z}_N^*$. Since (by assumption) there is at least one witness, Bad is a *strict* subgroup of $\mathbb{Z}_N^*$. Lemma 8.37 then shows that $|\mathsf{Bad}| \leq |\mathbb{Z}_N^*|/2$, showing that at least half the elements of $\mathbb{Z}_N^*$ are *not* in Bad (and hence are witnesses). ■

Let N be composite. If there exists a witness that N is composite, then there are at least $|\mathbb{Z}_N^*|/2$ witnesses. The probability that we find either a witness or an element not in $\mathbb{Z}_N^*$ in any given iteration of the algorithm is

thus at least 1/2, and so the probability that the algorithm does not find a witness in any of the t iterations (and hence the probability that the algorithm mistakenly outputs "prime") is at most 2^{-t}.

The above, unfortunately, does not give a complete solution since there are infinitely many composite numbers N that do not have *any* witnesses that they are composite! Such values N are known as *Carmichael numbers*; a detailed discussion is beyond the scope of this book.

Happily, a refinement of the above test can be shown to work for all N. Let $N-1 = 2^r u$, where u is odd and $r \geq 1$. (It is easy to compute r and u given N. Also, restricting to $r \geq 1$ means that N is odd, but testing primality is easy when N is even!) The algorithm shown previously tests only whether $a^{N-1} = a^{2^r u} = 1 \bmod N$. A more refined algorithm looks at the *sequence* of $r+1$ values $a^u, a^{2u}, \ldots, a^{2^r u}$ (all modulo N). Each term in this sequence is the square of the preceding term; thus, if some value is equal to ± 1 then all subsequent values will be equal to 1.

Say that $a \in \mathbb{Z}_N^*$ is a *strong witness that N is composite* (or simply a *strong witness*) if (1) $a^u \neq \pm 1 \bmod N$ and (2) $a^{2^i u} \neq -1 \bmod N$ for all $i \in \{1, \ldots, r-1\}$. Note that when an element a is *not* a strong witness then the sequence $(a^u, a^{2u}, \ldots, a^{2^r u})$ (all taken modulo N) takes one of the following forms:

$$(\pm 1, 1, \ldots, 1) \quad \text{or} \quad (\star, \ldots, \star, -1, 1, \ldots, 1),$$

where $\star$ is an arbitrary term. If a is *not* a strong witness then we have $a^{2^{r-1} u} = \pm 1 \bmod N$ and

$$a^{N-1} = a^{2^r u} = \left(a^{2^{r-1} u}\right)^2 = 1 \bmod N,$$

and so a is not a witness that N is composite, either. Put differently, if a is a witness then it is also a strong witness and so there can only possibly be *more* strong witnesses than witnesses.

We first show that if N is prime then there does not exist a strong witness that N is composite. In doing so, we rely on the following easy lemma (which is a special case of Proposition 13.16 proved subsequently in Chapter 13):

LEMMA 8.39 *Say $x \in \mathbb{Z}_N^*$ is a* square root of 1 modulo N *if $x^2 = 1 \bmod N$. If N is an odd prime then the only square roots of 1 modulo N are $[\pm 1 \bmod N]$.*

PROOF Say $x^2 = 1 \bmod N$ with $x \in \{1, \ldots, N-1\}$. Then $0 = x^2 - 1 = (x+1)(x-1) \bmod N$, implying that $N \mid (x+1)$ or $N \mid (x-1)$ by Proposition 8.3. This can only possibly occur if $x = [\pm 1 \bmod N]$. ∎

Let N be an odd prime and fix arbitrary $a \in \mathbb{Z}_N^*$. Let $i \geq 0$ be the minimum value for which $a^{2^i u} = 1 \bmod N$; since $a^{2^r u} = a^{N-1} = 1 \bmod N$ we know that

some such $i \leq r$ exists. If $i = 0$ then $a^u = 1 \bmod N$ and a is not a strong witness. Otherwise,

$$\left(a^{2^{i-1}u}\right)^2 = a^{2^i u} = 1 \bmod N$$

and $a^{2^{i-1}u}$ is a square root of 1. If N is an odd prime, the only square roots of 1 are ± 1; by choice of i, however, $a^{2^{i-1}u} \neq 1 \bmod N$. So $a^{2^{i-1}u} = -1 \bmod N$, and a is not a strong witness. We conclude that when N is an odd prime there is no strong witness that N is composite.

A composite integer N is a *prime power* if $N = p^r$ for some prime p and integer $r \geq 1$. We now show that every odd, composite N that is not a prime power has many strong witnesses.

THEOREM 8.40 *Let N be an odd number that is not a prime power. Then at least half the elements of $\mathbb{Z}_N^*$ are strong witnesses that N is composite.*

PROOF Let $\mathsf{Bad} \subseteq \mathbb{Z}_N^*$ denote the set of elements that are not strong witnesses. We define a set Bad' and show that: (1) Bad is a subset of Bad', and (2) Bad' is a strict subgroup of $\mathbb{Z}_N^*$. This suffices because by combining (2) and Lemma 8.37 we have that $|\mathsf{Bad}'| \leq |\mathbb{Z}_N^*|/2$. Furthermore, by (1) it holds that $\mathsf{Bad} \subseteq \mathsf{Bad}'$, and so $|\mathsf{Bad}| \leq |\mathsf{Bad}'| \leq |\mathbb{Z}_N^*|/2$ as in Theorem 8.38. Thus, at least half the elements of $\mathbb{Z}_N^*$ are strong witnesses. (We stress that we do not claim that Bad is a subgroup of $\mathbb{Z}_N^*$.)

Note first that $-1 \in \mathsf{Bad}$ since $(-1)^u = -1 \bmod N$ (recall u is odd). Let $i \in \{0, \ldots, r-1\}$ be the largest integer for which there exists an $a \in \mathsf{Bad}$ with $a^{2^i u} = -1 \bmod N$; alternatively, i is the largest integer for which there exists an $a \in \mathsf{Bad}$ with

$$(a^u, a^{2u}, \ldots, a^{2^r u}) = (\underbrace{\star, \ldots, \star, -1}_{i+1 \text{ terms}}, 1, \ldots, 1).$$

Since $-1 \in \mathsf{Bad}$ and $(-1)^{2^0 u} = -1 \bmod N$, some such i exists.

Fix i as above, and define

$$\mathsf{Bad}' \stackrel{\text{def}}{=} \{a \mid a^{2^i u} = \pm 1 \bmod N\}.$$

We now prove what we claimed above.

CLAIM 8.41 $\mathsf{Bad} \subseteq \mathsf{Bad}'$.

Let $a \in \mathsf{Bad}$. Then either $a^u = 1 \bmod N$ or $a^{2^j u} = -1 \bmod N$ for some $j \in \{0, \ldots, r-1\}$. In the first case, $a^{2^i u} = (a^u)^{2^i} = 1 \bmod N$ and so $a \in \mathsf{Bad}'$. In the second case, we have $j \leq i$ by choice of i. If $j = i$ then clearly $a \in \mathsf{Bad}'$.

If $j < i$ then $a^{2^i u} = (a^{2^j u})^{2^{i-j}} = 1 \bmod N$ and $a \in \mathsf{Bad}'$. Since a was arbitrary, this shows $\mathsf{Bad} \subseteq \mathsf{Bad}'$.

CLAIM 8.42 Bad' *is a subgroup of* $\mathbb{Z}_N^*$.

Clearly $1 \in \mathsf{Bad}'$. Furthermore, if $a, b \in \mathsf{Bad}'$ then

$$(ab)^{2^i u} = a^{2^i u} b^{2^i u} = (\pm 1)(\pm 1) = \pm 1 \bmod N$$

and so $ab \in \mathsf{Bad}'$. By Lemma 8.36, Bad' is a subgroup.

CLAIM 8.43 Bad' *is a strict subgroup of* $\mathbb{Z}_N^*$.

If N is an odd, composite integer that is not a prime power, then N can be written as $N = N_1 N_2$ with $N_1, N_2 > 1$ odd and $\gcd(N_1, N_2) = 1$. Appealing to the Chinese remainder theorem, let $a \leftrightarrow (a_1, a_2)$ denote the representation of $a \in \mathbb{Z}_N^*$ as an element of $\mathbb{Z}_{N_1}^* \times \mathbb{Z}_{N_2}^*$; that is, $a_1 = [a \bmod N_1]$ and $a_2 = [a \bmod N_2]$. Take $a \in \mathsf{Bad}'$ such that $a^{2^i u} = -1 \bmod N$ (such an a must exist by the way we defined i), and say $a \leftrightarrow (a_1, a_2)$. Since $-1 \leftrightarrow (-1, -1)$ we have

$$(a_1, a_2)^{2^i u} = (a_1^{2^i u},\, a_2^{2^i u}) = (-1, -1),$$

and so

$$a_1^{2^i u} = -1 \bmod N_1 \quad \text{and} \quad a_2^{2^i u} = -1 \bmod N_2.$$

Consider the element $b \in \mathbb{Z}_N^*$ with $b \leftrightarrow (a_1, 1)$. Then

$$b^{2^i u} \leftrightarrow (a_1, 1)^{2^i u} = ([a_1^{2^i u} \bmod N_1],\, 1) = (-1, 1) \not\leftrightarrow \pm 1.$$

That is, $b^{2^i u} \neq \pm 1 \bmod N$ and so we have found an element $b \notin \mathsf{Bad}'$. This proves that Bad' is a *strict* subgroup of $\mathbb{Z}_N^*$ and so, by Lemma 8.37, the size of Bad' (and thus the size of Bad) is at most half the size of $\mathbb{Z}_N^*$. ■

An integer N is a *perfect power* if $N = \hat{N}^e$ for integers $\hat{N}$ and $e \geq 2$ (here it is not required for $\hat{N}$ to be prime, although of course any prime power is also a perfect power). Algorithm 8.44 gives the Miller–Rabin primality test. Exercises 8.12 and 8.13 ask you to show that testing whether N is a perfect power, and testing whether a particular a is a strong witness, can be done in polynomial time. Given these results, the algorithm clearly runs in time polynomial in $\|N\|$ and t. We can now complete the proof of Theorem 8.33:

PROOF If N is an odd prime, there are no strong witnesses and so the Miller–Rabin algorithm always outputs "prime." If N is even or a prime power, the algorithm always outputs "composite." The interesting case is when N is an odd, composite integer that is not a prime power. Consider any

ALGORITHM 8.44
The Miller–Rabin primality test

Input: Integer $N > 2$ and parameter 1^t
Output: A decision as to whether N is prime or composite

if N is even, **return** "composite"
if N is a perfect power, **return** "composite"
compute $r \geq 1$ and u odd such that $N - 1 = 2^r u$
for $j = 1$ to t:
  $a \leftarrow \{1, \ldots, N-1\}$
  if $a^u \neq \pm 1 \bmod N$ and $a^{2^i u} \neq -1 \bmod N$ for $i \in \{1, \ldots, r-1\}$
    return "composite"
return "prime"

iteration of the inner loop. Note first that if $a \notin \mathbb{Z}_N^*$ then $a^u \neq \pm 1 \bmod N$ and $a^{2^i u} \neq -1 \bmod N$ for $i \in \{1, \ldots, r-1\}$. The probability of finding either a strong witness or an element not in $\mathbb{Z}_N^*$ is at least $1/2$ (invoking Theorem 8.40). Thus, the probability that the algorithm never outputs "composite" in any of the t iterations is at most 2^{-t}. ■

8.2.3 The Factoring Assumption

Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$, and p and q are n-bit primes except with probability negligible in n. (The natural way to do this is to generate two uniform n-bit primes, as discussed previously, and then multiply them to obtain N.) Then consider the following experiment for a given algorithm $\mathcal{A}$ and parameter n:

The factoring experiment $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n)$**:**

1. *Run* $\mathsf{GenModulus}(1^n)$ *to obtain* (N, p, q).
2. $\mathcal{A}$ *is given* N*, and outputs* $p', q' > 1$.
3. *The output of the experiment is defined to be* 1 *if* $p' \cdot q' = N$*, and* 0 *otherwise.*

Note that if the output of the experiment is 1 then $\{p', q'\} = \{p, q\}$, unless p or q are composite (which happens with only negligible probability).

We now formally define the factoring assumption:

DEFINITION 8.45 Factoring is hard relative to GenModulus *if for all probabilistic polynomial-time algorithms* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n).$$

The *factoring assumption* is the assumption that there exists a GenModulus relative to which factoring is hard.

8.2.4 The RSA Assumption

The factoring problem has been studied for hundreds of years without an efficient algorithm being found. Although the factoring assumption does give a one-way function (see Section 8.4.1), it unfortunately does not *directly* yield practical cryptosystems. (In Section 13.5.2, however, we show how to construct efficient cryptosystems based on a problem whose hardness is *equivalent* to that of factoring.) This has motivated a search for other problems whose difficulty is related to the hardness of factoring. The best known of these is a problem introduced in 1978 by Rivest, Shamir, and Adleman and now called the *RSA problem* in their honor.

Given a modulus N and an integer $e > 2$ relatively prime to $\phi(N)$, Corollary 8.22 shows that exponentiation to the eth power modulo N is a *permutation*. We can therefore define $[y^{1/e} \bmod N]$ (for any $y \in \mathbb{Z}_N^*$) as the unique element of $\mathbb{Z}_N^*$ which yields y when raised to the eth power modulo N; that is, $x = y^{1/e} \bmod N$ if and only if $x^e = y \bmod N$. The RSA problem, informally, is to compute $[y^{1/e} \bmod N]$ for a modulus N of unknown factorization.

Formally, let GenRSA be a probabilistic polynomial-time algorithm that, on input 1^n, outputs a modulus N that is the product of two n-bit primes, as well as integers $e, d > 0$ with $\gcd(e, \phi(N)) = 1$ and $ed = 1 \bmod \phi(N)$. (Such a d exists since e is invertible modulo $\phi(N)$. The purpose of d will become clear later.) The algorithm may fail with probability negligible in n. Consider the following experiment for a given algorithm $\mathcal{A}$ and parameter n:

The RSA experiment $\mathsf{RSA\text{-}inv}_{\mathcal{A},\mathsf{GenRSA}}(n)$**:**

1. *Run* $\mathsf{GenRSA}(1^n)$ *to obtain* (N, e, d).
2. *Choose a uniform* $y \in \mathbb{Z}_N^*$.
3. $\mathcal{A}$ *is given* N, e, y, *and outputs* $x \in \mathbb{Z}_N^*$.
4. *The output of the experiment is defined to be* 1 *if* $x^e = y \bmod N$, *and* 0 *otherwise.*

DEFINITION 8.46 The RSA problem is hard relative to GenRSA *if for all probabilistic polynomial-time algorithms* $\mathcal{A}$ *there exists a negligible function* negl *such that* $\Pr[\mathsf{RSA\text{-}inv}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] \leq \mathsf{negl}(n)$.

The *RSA assumption* is that there exists a GenRSA algorithm relative to which the RSA problem is hard. A suitable GenRSA algorithm can be constructed from any algorithm $\mathsf{GenModulus}$ that generates a composite modulus along with its factorization. A high-level outline is provided as Algorithm 8.47, where the only thing left unspecified is how exactly e is chosen. In fact, the RSA problem is believed to be hard for *any* e that is relatively prime to $\phi(N)$. We discuss some typical choices of e below.

ALGORITHM 8.47
GenRSA – **high-level outline**

Input: Security parameter 1^n
Output: N, e, d as described in the text

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return N, e, d

Example 8.48
Say GenModulus outputs $(N, p, q) = (143, 11, 13)$. Then $\phi(N) = 120$. Next, we need to choose an e that is relatively prime to $\phi(N)$; say we take $e = 7$. The next step is to compute d such that $d = [e^{-1} \bmod \phi(N)]$. This can be done as shown in Appendix B.2.2 to obtain $d = 103$. (One can check that $7 \cdot 103 = 721 = 1 \bmod 120$.) Our GenRSA algorithm thus outputs $(143, 7, 103)$.

As an example of the RSA problem relative to these parameters, take $y = 64$ and so the problem is to compute the 7th root of 64 modulo 143 *without* knowledge of d or the factorization of N. ♢

Computing eth roots modulo N becomes easy if d, $\phi(N)$, or the factorization of N is known. (As we show in the next section, any of these can be used to efficiently compute the others.) This follows from Corollary 8.22, which shows that $[y^d \bmod N]$ is the eth root of y modulo N. This asymmetry—namely, that the RSA problem appears to be hard when d or the factorization of N is unknown, but becomes easy when d *is* known—serves as the basis for applications of the RSA problem to public-key cryptography.

Example 8.49
Continuing the previous example, we can compute the 7th root of 64 modulo 143 using the value $d = 103$; the answer is $25 = 64^d = 64^{103} \bmod 143$. We can verify that this is the correct solution since $25^e = 25^7 = 64 \bmod 143$. ♢

On the choice of e. There does not appear to be any difference in the hardness of the RSA problem for different exponents e and, as such, different methods have been suggested for selecting it. One popular choice is to set $e = 3$, since then computing eth powers modulo N requires only two multiplications (see Appendix B.2.3). If e is to be set equal to 3, then p and q must be chosen with $p, q \neq 1 \bmod 3$ so that $\gcd(e, \phi(N)) = 1$. For similar reasons, another popular choice is $e = 2^{16} + 1 = 65537$, a prime number with low Hamming weight (in Appendix B.2.3, we explain why such exponents are preferable). As compared to choosing $e = 3$, this makes exponentiation

slightly more expensive but reduces the constraints on p and q, and avoids some "low-exponent attacks" (described at the end of Section 11.5.1) that can result from poorly implemented cryptosystems based on the RSA problem.

Note that choosing d small (that is, changing GenRSA to choose small d and then compute $e := [d^{-1} \bmod \phi(N)]$) is a bad idea. If d lies in a very small range then a brute-force search for d can be carried out (and, as noted, once d is known the RSA problem can be solved easily). Even if d is chosen so that $d \approx N^{1/4}$, and so brute-force attacks are ruled out, there are known algorithms that can be used to recover d from N and e in this case.

8.2.5 *Relating the RSA and Factoring Assumptions

Say GenRSA is constructed as in Algorithm 8.47. If N can be factored, then we can compute $\phi(N)$ and use this to compute $d := [e^{-1} \bmod \phi(N)]$ for any given e (using Algorithm B.11). So for the RSA problem to be hard relative to GenRSA, the factoring problem must be hard relative to GenModulus. Put differently, the RSA problem cannot be *more* difficult than factoring; hardness of factoring (relative to GenModulus) can only potentially be a *weaker* assumption than hardness of the RSA problem (relative to GenRSA).

What about the other direction? That is, is hardness of the RSA problem implied by hardness of factoring? This remains an open question. The best we can show is that *computing d from N and e is as hard as factoring.*

THEOREM 8.50 *There is a probabilistic polynomial-time algorithm that, given as input a composite integer N and integers e, d with $ed = 1 \bmod \phi(N)$, outputs a factor of N except with probability negligible in $\|N\|$.*

PROOF The theorem holds for any N, but for simplicity—and because it is the case most relevant to cryptography—we focus here on the case where N is a product of two distinct (odd) primes. We rely on Proposition 8.36 and Lemma 8.37 as well as the following facts (which follow from more general results proved in Sections 13.4.2 and 13.5.2):

- If N is a product of two distinct, odd primes, then 1 has exactly four square roots modulo N. Two of these are the "trivial" square roots ± 1, and two of these are "nontrivial" square roots.
- Any nontrivial square root of 1 can be used to (efficiently) compute a factor of N. This is by virtue of the fact that $y^2 = 1 \bmod N$ implies

$$0 = y^2 - 1 = (y-1)(y+1) \bmod N,$$

and so $N \mid (y-1)(y+1)$. However, $N \nmid (y-1)$ and $N \nmid (y+1)$ because $y \neq \pm 1 \bmod N$. So it must be the case that $\gcd(y-1, N)$ is equal to one of the prime factors of N.

Let $k = ed - 1$ and note that $\phi(N) \mid k$. Using Corollary 8.21, we have $x^k = 1 \bmod N$ for all $x \in \mathbb{Z}_N^*$. Let $k = 2^r u$ for u an odd integer; note that

$r \geq 1$ since $\phi(N)$ (and hence k) is even. Our strategy for factoring N will be to repeatedly choose a uniform $x \in \mathbb{Z}_N^*$ and compute the sequence

$$x^u,\ x^{2u},\ \ldots,\ x^{2^r u},$$

all modulo N. Each term in this sequence is the square of the preceding term and, as we have just noted, the final term in the sequence is 1. Take the largest i (if any) for which $y \stackrel{\text{def}}{=} [x^{2^i u} \bmod N] \neq 1$. By our choice of i, we have $y^2 = 1 \bmod N$. If $y \neq -1$ we have found a nontrivial square root of N, and can then factor N as discussed above.

All the above steps can be done in polynomial time, and so the only question is to determine the probability, over choice of x, that y is a nontrivial square root of N. Let $i \in \{0, \ldots, r-1\}$ be the largest value of i for which there exists an $x \in \mathbb{Z}_N^*$ such that $x^{2^i u} \neq 1 \bmod N$. (Since u is odd $(-1)^u = -1 \neq 1 \bmod N$, and so the definition is not vacuous.) Then for all $x \in \mathbb{Z}_N^*$, we have $x^{2^{i+1} u} = 1 \bmod N$ and so $[x^{2^i u} \bmod N]$ is a square root of 1. Define

$$\mathsf{Bad} \stackrel{\text{def}}{=} \{x \mid x^{2^i u} = \pm 1 \bmod N\}$$

and observe that if our algorithm chooses $x \notin \mathsf{Bad}$ then it finds a nontrivial square root of 1. We show that Bad is a *strict* subgroup of $\mathbb{Z}_N^*$; by Lemma 8.37, this implies that $|\mathsf{Bad}| \leq |\mathbb{Z}_N^*|/2$. This means that $x \notin \mathsf{Bad}$ (and the algorithm finds a nontrivial square root of 1) with probability at least $1/2$ in each iteration. Using sufficiently many iterations gives the result of the theorem.

We now prove that Bad is a strict subgroup of $\mathbb{Z}_N^*$. First note that Bad is not empty, since $1 \in \mathsf{Bad}$. Furthermore, if $x, x' \in \mathsf{Bad}$ then

$$(xx')^{2^i u} = x^{2^i u}(x')^{2^i u} = (\pm 1) \cdot (\pm 1) = \pm 1 \bmod N,$$

and so $xx' \in \mathsf{Bad}$ and Bad is a subgroup. To see that Bad is a *strict* subgroup, let $x \in \mathbb{Z}_N^*$ be such that $x^{2^i u} \neq 1 \bmod N$ (such an x must exist by our definition of i). If $x^{2^i u} \neq -1 \bmod N$, then $x \notin \mathsf{Bad}$ and we are done. Otherwise, let $N = pq$ with p, q prime, and let $x \leftrightarrow (x_p, x_q)$ be the Chinese remaindering representation of x. Since $x^{2^i u} = -1 \bmod N$, we know that

$$(x_p, x_q)^{2^i u} = (x_p^{2^i u}, x_q^{2^i u}) = (-1, -1) \leftrightarrow -1.$$

But then $(x_p, 1)$ (or rather, the element corresponding to it) is not in Bad since

$$(x_p, 1)^{2^i u} = ([x_p^{2^i u} \bmod p], 1) = (-1, 1) \not\leftrightarrow \pm 1.$$

This completes the proof. ∎

Assuming factoring is hard, the above result rules out the possibility of efficiently solving the RSA problem by first computing d from N and e. However, it does not rule out the possibility that there might be some completely

different way of attacking the RSA problem that does not involve (or imply) factoring N. Thus, based on our current knowledge, the RSA assumption is stronger than the factoring assumption—that is, it may be that the RSA problem can be solved in polynomial time even though factoring cannot. Nevertheless, when GenRSA is constructed based on GenModulus as in Algorithm 8.47, the prevailing conjecture is that the RSA problem is hard relative to GenRSA whenever factoring is hard relative to GenModulus.

8.3 Cryptographic Assumptions in Cyclic Groups

In this section we introduce a class of cryptographic hardness assumptions in *cyclic groups*. We begin with a general discussion of cyclic groups, followed by abstract definitions of the relevant assumptions. We then look at two concrete and widely used examples of cyclic groups in which these assumptions are believed to hold.

8.3.1 Cyclic Groups and Generators

Let $\mathbb{G}$ be a finite group of order m. For arbitrary $g \in \mathbb{G}$, consider the set

$$\langle g \rangle \stackrel{\text{def}}{=} \{g^0, g^1, \ldots\}.$$

(We warn the reader that if $\mathbb{G}$ is an infinite group, $\langle g \rangle$ is defined differently.) By Theorem 8.14, we have $g^m = 1$. Let $i \leq m$ be the smallest positive integer for which $g^i = 1$. Then the above sequence repeats after i terms (i.e., $g^i = g^0$, $g^{i+1} = g^1$, etc.), and so

$$\langle g \rangle = \{g^0, \ldots, g^{i-1}\}.$$

We see that $\langle g \rangle$ contains at most i elements. In fact, it contains exactly i elements since if $g^j = g^k$ with $0 \leq j < k < i$ then $g^{k-j} = 1$ and $0 < k - j < i$, contradicting our choice of i as the smallest positive integer for which $g^i = 1$.

It is not hard to verify that $\langle g \rangle$ is a subgroup of $\mathbb{G}$ for any g (see Exercise 8.3); we call $\langle g \rangle$ the *subgroup generated by* g. If the order of the subgroup $\langle g \rangle$ is i, then i is called the *order of* g; that is:

DEFINITION 8.51 *Let $\mathbb{G}$ be a finite group and $g \in \mathbb{G}$. The* order of g *is the smallest positive integer i with $g^i = 1$.*

The following is a useful analogue of Corollary 8.15 (the proof is identical):

PROPOSITION 8.52 *Let $\mathbb{G}$ be a finite group, and $g \in \mathbb{G}$ an element of order i. Then for any integer x, we have $g^x = g^{[x \bmod i]}$.*

We can prove something stronger:

PROPOSITION 8.53 *Let $\mathbb{G}$ be a finite group, and $g \in \mathbb{G}$ an element of order i. Then $g^x = g^y$ if and only if $x = y \bmod i$.*

PROOF If $x = y \bmod i$ then $[x \bmod i] = [y \bmod i]$ and the previous proposition says that

$$g^x = g^{[x \bmod i]} = g^{[y \bmod i]} = g^y.$$

For the more interesting direction, say $g^x = g^y$. Then $1 = g^{x-y} = g^{[x-y \bmod i]}$ (using the previous proposition). Since $[x - y \bmod i] < i$, but i is the smallest positive integer with $g^i = 1$, we must have $[x - y \bmod i] = 0$. ■

The identity element of any group $\mathbb{G}$ is the only element of order 1, and generates the group $\langle 1 \rangle = \{1\}$. At the other extreme, if there is an element $g \in \mathbb{G}$ that has order m (where m is the order of $\mathbb{G}$), then $\langle g \rangle = \mathbb{G}$. In this case, we call $\mathbb{G}$ a *cyclic group* and say that g is a *generator* of $\mathbb{G}$. (A cyclic group may have multiple generators, and so we cannot speak of *the* generator.) If g is a generator of $\mathbb{G}$ then, by definition, every element $h \in \mathbb{G}$ is equal to g^x for some $x \in \{0, \ldots, m-1\}$, a point we will return to in the next section.

Different elements of the same group $\mathbb{G}$ may have different orders. We can, however, place some restrictions on what these possible orders might be.

PROPOSITION 8.54 *Let $\mathbb{G}$ be a finite group of order m, and say $g \in \mathbb{G}$ has order i. Then $i \mid m$.*

PROOF By Theorem 8.14 we know that $g^m = 1 = g^0$. Proposition 8.53 implies that $m = 0 \bmod i$. ■

The next corollary illustrates the power of this result:

COROLLARY 8.55 *If $\mathbb{G}$ is a group of prime order p, then $\mathbb{G}$ is cyclic. Furthermore, all elements of $\mathbb{G}$ except the identity are generators of $\mathbb{G}$.*

PROOF By Proposition 8.54, the only possible orders of elements in $\mathbb{G}$ are 1 and p. Only the identity has order 1, and so all other elements have order p and generate $\mathbb{G}$. ■

Groups of prime order form one class of cyclic groups. The additive group $\mathbb{Z}_N$, for $N > 1$, gives another example of a cyclic group (the element 1 is always a generator). The next theorem—a special case of Theorem A.21—

gives an important additional class of cyclic groups; a proof is outside the scope of this book, but can be found in any standard abstract algebra text.

THEOREM 8.56 *If p is prime then $\mathbb{Z}_p^*$ is a cyclic group of order $p-1$.*

For $p > 3$ prime, $\mathbb{Z}_p^*$ does not have prime order and so the above does not follow from the preceding corollary.

Example 8.57
Consider the (additive) group $\mathbb{Z}_{15}$. As we have noted, $\mathbb{Z}_{15}$ is cyclic and the element 1 is a generator since $15 \cdot 1 = 0 \bmod 15$ and $i \cdot 1 = i \neq 0 \bmod 15$ for any $0 < i < 15$ (recall that in this group the identity is 0).

$\mathbb{Z}_{15}$ has other generators. For example, $\langle 2 \rangle = \{0, 2, 4, \ldots, 14, 1, 3, \ldots, 13\}$ and so 2 is also a generator.

Not every element generates $\mathbb{Z}_{15}$. For example, the element 3 has order 5 since $5 \cdot 3 = 0 \bmod 15$, and so 3 does not generate $\mathbb{Z}_{15}$. The subgroup $\langle 3 \rangle$ consists of the 5 elements $\{0, 3, 6, 9, 12\}$, and this is indeed a subgroup under addition modulo 15. The element 10 has order 3 since $3 \cdot 10 = 0 \bmod 15$, and the subgroup $\langle 10 \rangle$ consists of the 3 elements $\{0, 5, 10\}$. The orders of the subgroups (i.e., 5 and 3) divide $|\mathbb{Z}_{15}| = 15$ as required by Proposition 8.54. ♢

Example 8.58
Consider the (multiplicative) group $\mathbb{Z}_{15}^*$ of order $(5-1)(3-1) = 8$. We have $\langle 2 \rangle = \{1, 2, 4, 8\}$, and so the order of 2 is 4. As required by Proposition 8.54, 4 divides 8. ♢

Example 8.59
Consider the (additive) group $\mathbb{Z}_p$ of prime order p. We know this group is cyclic, but Corollary 8.55 tells us more: namely, *every* element except 0 is a generator. Indeed, for any $h \in \{1, \ldots, p-1\}$ and integer $i > 0$ we have $ih = 0 \bmod p$ if and only if $p \mid ih$. But then Proposition 8.3 says that either $p \mid h$ or $p \mid i$. The former cannot occur (since $h < p$), and the smallest positive integer for which the latter can occur is $i = p$. We have thus shown that every nonzero element h has order p (and so generates $\mathbb{Z}_p$), in accordance with Corollary 8.55. ♢

Example 8.60
Consider the (multiplicative) group $\mathbb{Z}_7^*$, which is cyclic by Theorem 8.56. We have $\langle 2 \rangle = \{1, 2, 4\}$, and so 2 is *not* a generator. However,

$$\langle 3 \rangle = \{1, 3, 2, 6, 4, 5\} = \mathbb{Z}_7^*,$$

and so 3 is a generator of $\mathbb{Z}_7^*$. ♢

The following example relies on the material of Section 8.1.5.

Example 8.61
Let $\mathbb{G}$ be a cyclic group of order n, and let g be a generator of $\mathbb{G}$. Then the mapping $f : \mathbb{Z}_n \to \mathbb{G}$ given by $f(a) = g^a$ is an isomorphism between $\mathbb{Z}_n$ and $\mathbb{G}$. Indeed, for $a, a' \in \mathbb{Z}_n$ we have

$$f(a + a') = g^{[a+a' \bmod n]} = g^{a+a'} = g^a \cdot g^{a'} = f(a) \cdot f(a').$$

Bijectivity of f can be proved using the fact that n is the order of g. ♢

The previous example shows that all cyclic groups of the same order are isomorphic and thus the same from an *algebraic* point of view. We stress that this is not true in a *computational* sense, and in particular an isomorphism $f^{-1} : \mathbb{G} \to \mathbb{Z}_n$ (which we know must exist) need not be efficiently computable. This point should become clearer from the discussion in the sections below as well as Chapter 9.

8.3.2 The Discrete-Logarithm/Diffie–Hellman Assumptions

We now introduce several computational problems that can be defined for any class of cyclic groups. We will keep the discussion in this section abstract, and consider specific examples of groups in which these problems are believed to be hard in Sections 8.3.3 and 8.3.4.

We let $\mathcal{G}$ denote a generic, polynomial-time, *group-generation algorithm*. This is an algorithm that, on input 1^n, outputs a description of a cyclic group $\mathbb{G}$, its order q (with $\|q\| = n$), and a generator $g \in \mathbb{G}$. The description of a cyclic group specifies how elements of the group are represented as bit-strings; we assume that each group element is represented by a unique bit-string. We require that there are efficient algorithms (namely, algorithms running in time polynomial in n) for computing the group operation in $\mathbb{G}$, as well as for testing whether a given bit-string represents an element of $\mathbb{G}$. Efficient computation of the group operation implies efficient algorithms for exponentiation in $\mathbb{G}$ (see Appendix B.2.3) and for sampling a uniform element $h \in \mathbb{G}$ (simply choose uniform $x \in \mathbb{Z}_q$ and set $h := g^x$).

As discussed at the end of the previous section, although all cyclic groups of a given order are isomorphic, the *representation* of the group determines the computational complexity of mathematical operations in that group.

If $\mathbb{G}$ is a cyclic group of order q with generator g, then $\{g^0, g^1, \ldots, g^{q-1}\}$ is all of $\mathbb{G}$. Equivalently, for every $h \in \mathbb{G}$ there is a *unique* $x \in \mathbb{Z}_q$ such that $g^x = h$. When the underlying group $\mathbb{G}$ is understood from the context, we call this x the *discrete logarithm of h with respect to g* and write $x = \log_g h$. (Logarithms in this case are called "discrete" since they take values in a finite range, as opposed to "standard" logarithms from calculus whose values range over the infinite set of real numbers.) Note that if $g^{x'} = h$ for some arbitrary integer x', then $[x' \bmod q] = \log_g h$.

Discrete logarithms obey many of the same rules as "standard" logarithms. For example, $\log_g 1 = 0$ (where 1 is the identity of $\mathbb{G}$); for any integer r, we have $\log_g h^r = [r \cdot \log_g h \bmod q]$; and $\log_g(h_1 h_2) = [(\log_g h_1 + \log_g h_2) \bmod q]$.

The *discrete-logarithm problem* in a cyclic group $\mathbb{G}$ with generator g is to compute $\log_g h$ for a uniform element $h \in \mathbb{G}$. Consider the following experiment for a group-generation algorithm $\mathcal{G}$, algorithm $\mathcal{A}$, and parameter n:

The discrete-logarithm experiment $\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n)$:

1. *Run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of order q (with $\|q\| = n$), and g is a generator of $\mathbb{G}$.*
2. *Choose a uniform $h \in \mathbb{G}$.*
3. *$\mathcal{A}$ is given $\mathbb{G}, q, g, h$, and outputs $x \in \mathbb{Z}_q$.*
4. *The output of the experiment is defined to be 1 if $g^x = h$, and 0 otherwise.*

DEFINITION 8.62 *We say that* the discrete-logarithm problem is hard relative to $\mathcal{G}$ *if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function* negl *such that* $\Pr[\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1] \leq \mathsf{negl}(n)$.

The discrete-logarithm assumption is simply the assumption that there exists a $\mathcal{G}$ for which the discrete-logarithm problem is hard. The following two sections discuss some candidate group-generation algorithms $\mathcal{G}$ for which this is believed to be the case.

The Diffie–Hellman problems. The so-called *Diffie–Hellman* problems are related, but not known to be equivalent, to the problem of computing discrete logarithms. There are two important variants: the *computational* Diffie–Hellman (CDH) problem and the *decisional* Diffie–Hellman (DDH) problem.

Fix a cyclic group $\mathbb{G}$ and a generator $g \in \mathbb{G}$. Given elements $h_1, h_2 \in \mathbb{G}$, define $\mathsf{DH}_g(h_1, h_2) \stackrel{\text{def}}{=} g^{\log_g h_1 \cdot \log_g h_2}$. That is, if $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$ then

$$\mathsf{DH}_g(h_1, h_2) = g^{x_1 \cdot x_2} = h_1^{x_2} = h_2^{x_1}.$$

The *CDH problem* is to compute $\mathsf{DH}_g(h_1, h_2)$ for uniform h_1 and h_2. Hardness of this problem can be formalized by the natural experiment; we leave the details as an exercise.

If the discrete-logarithm problem relative to some $\mathcal{G}$ is easy, then the CDH problem is, too: given h_1 and h_2, first compute $x_1 := \log_g h_1$ and then output the answer $h_2^{x_1}$. In contrast, it is not clear whether hardness of the discrete-logarithm problem implies that the CDH problem is hard as well.

The *DDH problem*, roughly speaking, is to distinguish $\mathsf{DH}_g(h_1, h_2)$ from a uniform group element when h_1, h_2 are uniform. That is, given uniform h_1, h_2 and a third group element h', the problem is to decide whether $h' = \mathsf{DH}_g(h_1, h_2)$ or whether h' was chosen uniformly from $\mathbb{G}$. Formally:

DEFINITION 8.63 *We say that* the DDH problem is hard relative to $\mathcal{G}$ *if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there is a negligible function* negl *such that*

$$\left|\Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\right| \leq \mathsf{negl}(n),$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^n)$ outputs $(\mathbb{G}, q, g)$, and then uniform $x, y, z \in \mathbb{Z}_q$ are chosen. (Note that when z is uniform in $\mathbb{Z}_q$, then g^z is uniformly distributed in $\mathbb{G}$.)

We have already seen that if the discrete-logarithm problem is easy relative to some $\mathcal{G}$, then the CDH problem is too. Similarly, if the CDH problem is easy relative to $\mathcal{G}$ then so is the DDH problem; you are asked to show this in Exercise 8.15. The converse, however, does not appear to be true, and there are examples of groups in which the discrete-logarithm and CDH problems are believed to be hard even though the DDH problem is easy; see Exercise 13.15.

Using Prime-Order Groups

There are various (classes of) cyclic groups in which the discrete-logarithm and Diffie–Hellman problems are believed to be hard. There is a preference, however, for cyclic groups of *prime order*, for reasons we now explain.

One reason for preferring groups of prime order is because, in a certain sense, the discrete-logarithm problem is hardest in such groups. This is a consequence of the *Pohlig–Hellman algorithm*, described in Chapter 9, which shows that the discrete-logarithm problem in a group of order q becomes easier if q has (small) prime factors. This does not necessarily mean that the discrete-logarithm problem is *easy* in groups of nonprime order; it merely means that the problem becomes *easier*.

Related to the above is the fact that the DDH problem is easy if the group order q has small prime factors. We refer to Exercise 13.15 for one example of this phenomenon.

A second motivation for using prime-order groups is because finding a generator in such groups is trivial. This follows from Corollary 8.55, which says that *every* element of a prime-order group (except the identity) is a generator. In contrast, efficiently finding a generator of an arbitrary cyclic group requires the factorization of the group order to be known (see Appendix B.3).

Proofs of security for some cryptographic constructions require computing multiplicative inverses of certain exponents (we will see an example in Section 8.4.2). When the group order is prime, any nonzero exponent will be invertible, making this computation possible.

A final reason for working with prime-order groups applies in situations when the *decisional* Diffie–Hellman problem should be hard. Fixing a group $\mathbb{G}$ with generator g, the DDH problem boils down to distinguishing between

tuples of the form $(h_1, h_2, \mathsf{DH}_g(h_1, h_2))$ for uniform h_1, h_2, and tuples of the form (h_1, h_2, y), for uniform h_1, h_2, y. A necessary condition for the DDH problem to be hard is that $\mathsf{DH}_g(h_1, h_2)$ *by itself* should be indistinguishable from a uniform group element. One can show that $\mathsf{DH}_g(h_1, h_2)$ is "close" to uniform (in a sense we do not define here) when the group order q is prime, something that is not true otherwise.

8.3.3 Working in (Subgroups of) $\mathbb{Z}_p^*$

Groups of the form $\mathbb{Z}_p^*$, for p prime, give one class of cyclic groups in which the discrete-logarithm problem is believed to be hard. Concretely, let $\mathcal{G}$ be an algorithm that, on input 1^n, chooses a uniform n-bit prime p, and outputs p and the group order $q = p - 1$ along with a generator g of $\mathbb{Z}_p^*$. (Section 8.2.1 discusses efficient algorithms for choosing a random prime, and Appendix B.3 shows how to efficiently find a generator of $\mathbb{Z}_p^*$ given the factorization of $p - 1$.) The representation of $\mathbb{Z}_p^*$ here is the trivial one where elements are represented as integers between 1 and $p - 1$. It is conjectured that the discrete-logarithm problem is hard relative to $\mathcal{G}$ of this sort.

The cyclic group $\mathbb{Z}_p^*$ (for $p > 3$ prime), however, does *not* have prime order. (The preference for groups of prime order was discussed in the previous section.) More problematic, the decisional Diffie–Hellman problem is, in general, *not hard* in such groups (see Exercise 13.15), and they are therefore unacceptable for the cryptographic applications based on the DDH assumption that we will explore in later chapters.

These issues can be addressed by using a prime-order *subgroup* of $\mathbb{Z}_p^*$. Let $p = rq + 1$ where both p and q are prime. We prove that $\mathbb{Z}_p^*$ has a subgroup $\mathbb{G}$ of order q given by the set of *rth residues modulo p*, i.e., the set of elements $\{[h^r \bmod p] \mid h \in \mathbb{Z}_p^*\}$ that are equal to the rth power of some $h \in \mathbb{Z}_p^*$.

THEOREM 8.64 *Let $p = rq + 1$ with p, q prime. Then*

$$\mathbb{G} \stackrel{\text{def}}{=} \left\{[h^r \bmod p] \mid h \in \mathbb{Z}_p^*\right\}$$

is a subgroup of $\mathbb{Z}_p^*$ of order q.

PROOF The proof that $\mathbb{G}$ is a subgroup is straightforward and is omitted. We prove that $\mathbb{G}$ has order q by showing that the function $f_r : \mathbb{Z}_p^* \to \mathbb{G}$ defined by $f_r(g) = [g^r \bmod p]$ is an r-to-1 function. (Since $|\mathbb{Z}_p^*| = p - 1$, this shows that $|\mathbb{G}| = (p - 1)/r = q$.) To see this, let g be a generator of $\mathbb{Z}_p^*$ so that $g^0, \ldots, g^{p-2}$ are all the elements of $\mathbb{Z}_p^*$. By Proposition 8.53 we have $\left(g^i\right)^r = \left(g^j\right)^r$ if and only if $ir = jr \bmod (p-1)$ or, equivalently, $p-1 \mid (i-j)r$. Since $p-1 = rq$, this is equivalent to $q \mid (i-j)$. For any fixed $j \in \{0, \ldots, p-2\}$, this means that the set of values $i \in \{0, \ldots, p-2\}$ for which $\left(g^i\right)^r = \left(g^j\right)^r$ is

exactly the set of r distinct values

$$\{j,\, j+q,\, j+2q,\, \ldots,\, j+(r-1)q\}\,,$$

all reduced modulo $p-1$. (Note that $j + rq = j \bmod (p-1)$.) This proves that f_r is an r-to-1 function. ∎

Besides showing existence of an appropriate subgroup, the theorem also implies that it is easy to generate a uniform element of $\mathbb{G}$ and to test whether a given element of $\mathbb{Z}_p^*$ lies in $\mathbb{G}$. Specifically, choosing a uniform element of $\mathbb{G}$ can be done by choosing uniform $h \in \mathbb{Z}_p^*$ and computing $[h^r \bmod p]$. Since $\mathbb{G}$ has prime order, every element in $\mathbb{G}$ except the identity is a generator of $\mathbb{G}$. Finally, it is possible to determine whether any $h \in \mathbb{Z}_p^*$ is also in $\mathbb{G}$ by checking whether $h^q \stackrel{?}{=} 1 \bmod p$. To see that this works, let $h = g^i$ for g a generator of $\mathbb{Z}_p^*$ and $i \in \{0, \ldots, p-2\}$. Then

$$\begin{aligned} h^q = 1 \bmod p &\iff g^{iq} = 1 \bmod p \\ &\iff iq = 0 \bmod (p-1) \iff rq \mid iq \iff r \mid i, \end{aligned}$$

using Proposition 8.53. So $h = g^i = g^{cr} = (g^c)^r$ for some c, and $h \in \mathbb{G}$.

Algorithm 8.65 encapsulates the above discussion. In the algorithm, we let n denote the length of q, the order of the group, and let ℓ denote the length of p, the modulus being used. The relationship between these parameters is discussed below.

Choosing ℓ. Let $n = \|q\|$ and $\ell = \|p\|$. Two types of algorithms are known for computing discrete logarithms in order-q subgroups of $\mathbb{Z}_p^*$ (see Section 9.2): those that run in time $\mathcal{O}(\sqrt{q}) = \mathcal{O}(2^{n/2})$ and those that run in time $2^{\mathcal{O}((\log p)^{1/3} \cdot (\log \log p)^{2/3})} = 2^{\mathcal{O}(\ell^{1/3} \cdot (\log \ell)^{2/3})}$. Fixing some desired security parameter n, the parameter ℓ should be chosen so as to balance these times. (If ℓ is any smaller, security is reduced; if ℓ is any larger, operations in $\mathbb{G}$ will be less efficient without any gain in security.) See also Section 9.3.

ALGORITHM 8.65
A group-generation algorithm $\mathcal{G}$

Input: Security parameter 1^n, parameter $\ell = \ell(n)$
Output: Cyclic group $\mathbb{G}$, its (prime) order q, and a generator g

generate a uniform n-bit prime q
generate an ℓ-bit prime p such that $q \mid (p-1)$
  // we omit the details of how this is done
choose a uniform $h \in \mathbb{Z}_p^*$ with $h \neq 1$
set $g := [h^{(p-1)/q} \bmod p]$
return p, q, g // $\mathbb{G}$ is the order-q subgroup of $\mathbb{Z}_p^*$

In practice, standardized values (e.g., recommended by NIST) for p, q, and a generator g are used, and there is no need to generate parameters of one's own.

Example 8.66

Consider the group $\mathbb{Z}_{11}^*$ of order 10. Let us try to find a generator of this group. Consider trying 2:

Powers of 2:	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
Values:	1	2	4	8	5	10	9	7	3	6

(All values above are computed modulo 11.) We got lucky the first time—the number 2 is a generator! Let's try 3:

Powers of 3:	3^0	3^1	3^2	3^3	3^4	3^5	3^6	3^7	3^8	3^9
Values:	1	3	9	5	4	1	3	9	5	4

We see that 3 is not a generator of the entire group. Rather, it generates a *subgroup* $\mathbb{G} = \{1, 3, 4, 5, 9\}$ of order 5. Now, let's see what happens with 10:

Powers of 10:	10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
Values:	1	10	1	10	1	10	1	10	1	10

In this case we generate a subgroup of order 2.

For cryptographic purposes we want to work in a prime-order group. Since $11 = 2 \cdot 5 + 1$ we can apply Theorem 8.64 with $q = 5$ and $r = 2$, or with $q = 2$ and $r = 5$. In the first case, the theorem tells us that the *squares* of all the elements of $\mathbb{Z}_{11}^*$ should give a subgroup of order 5. This can be easily verified:

Element:	1	2	3	4	5	6	7	8	9	10
Square:	1	4	9	5	3	3	5	9	4	1

We have seen above that 3 is a generator of this subgroup. (In fact, since the subgroup is prime, every element of the subgroup besides 1 is a generator of the subgroup.) Taking $q = 2$ and $r = 5$, Theorem 8.64 tells us that taking 5th powers will give a subgroup of order 2. One can check that this gives the order-2 subgroup generated by 10 that we encountered earlier. ♢

Subgroups of finite fields. The discrete-logarithm problem is also believed to be hard in the multiplicative group of a finite field of large characteristic when the polynomial representation is used. (Appendix A.5 provides a brief background on finite fields.) Recall that for any prime p and integer $k \geq 1$ there is a (unique) field $\mathbb{F}_{p^k}$ of order p^k; the multiplicative group $\mathbb{F}_{p^k}^*$ of that field is a cyclic group of order $p^k - 1$ (cf. Theorem A.21). If q is a large prime factor of $p^k - 1$, then Theorem 8.64 shows that $\mathbb{F}_{p^k}^*$ has a cyclic subgroup of order q. (The only property of $\mathbb{Z}_p^*$ we used in the proof of that theorem was that $\mathbb{Z}_p^*$ is cyclic.) This offers another choice of prime-order groups in which the discrete-logarithm and Diffie–Hellman problems are believed to be hard. Our treatment of $\mathbb{Z}_p^*$ in this section corresponds to the special case $k = 1$.

8.3.4 Elliptic Curves

The groups we have concentrated on thus far have all been based directly on modular arithmetic. Another class of groups important for cryptography is given by groups consisting of *points on elliptic curves*. Such groups are especially interesting from a cryptographic perspective since, in contrast to $\mathbb{Z}_p^*$ or the multiplicative group of a finite field, there are currently no known sub-exponential time algorithms for solving the discrete-logarithm problem in elliptic-curve groups when chosen appropriately. (See Section 9.3 for further discussion.) For cryptosystems based on the discrete-logarithm or Diffie–Hellman assumptions, this means that implementations based on elliptic-curve groups will be more efficient than implementations based on prime-order subgroups of $\mathbb{Z}_p^*$ at any given level of security. In this section we provide only a brief introduction to this area. A deeper understanding of the issues discussed here requires more sophisticated mathematics than we are willing to assume on the part of the reader. Those interested in further exploring this topic are advised to consult the references at the end of this chapter.

Let $p \geq 5$ be a prime.[3] Consider an equation E in the variables x and y of the form:

$$y^2 = x^3 + Ax + B \bmod p, \tag{8.1}$$

where $A, B \in \mathbb{Z}_p$ are constants with $4A^3 + 27B^2 \neq 0 \bmod p$. (This ensures that the equation $x^3 + Ax + B = 0 \bmod p$ has no repeated roots.) Let $E(\mathbb{Z}_p)$ denote the set of pairs $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ satisfying the above equation along with a special value $\mathcal{O}$ whose purpose we will discuss shortly; that is,

$$E(\mathbb{Z}_p) \stackrel{\text{def}}{=} \big\{(x, y) \mid x, y \in \mathbb{Z}_p \text{ and } y^2 = x^3 + Ax + B \bmod p\big\} \cup \{\mathcal{O}\}.$$

The elements $E(\mathbb{Z}_p)$ are called the *points* on the *elliptic curve* E defined by Equation (8.1), and $\mathcal{O}$ is called the *point at infinity*.

Example 8.67

An element $y \in \mathbb{Z}_p^*$ is a *quadratic residue modulo* p if there is an $x \in \mathbb{Z}_p^*$ such that $x^2 = y \bmod p$; in that case, we say x is a *square root of* y. For $p > 2$ prime, half the elements in $\mathbb{Z}_p^*$ are quadratic residues, and every quadratic residue has exactly two square roots. (See Section 13.4.1.)

Let $f(x) \stackrel{\text{def}}{=} x^3 + 3x + 3$ and consider the curve $E : y^2 = f(x) \bmod 7$. Each value of x for which $f(x)$ is a quadratic residue modulo 7 yields two points on the curve; values x for which $f(x)$ is a non-quadratic residue are not on

[3] The theory can be adapted to deal with the case of $p = 2$ or 3 but this introduces additional complications. Elliptic curves can, in fact, be defined over arbitrary finite or infinite *fields* (cf. Section A.5), and our discussion largely carries over to fields of characteristic not equal to 2 or 3. *Binary curves* (i.e., curves over fields of characteristic 2) are particularly important in cryptography, but we will not discuss them here.

the curve; values of x for which $f(x) = 0 \bmod 7$ give one point on the curve. This allows us to determine the points on the curve:

- $f(0) = 3 \bmod 7$, a quadratic non-residue modulo 7.
- $f(1) = 0 \bmod 7$, so we obtain the point $(1, 0) \in E(\mathbb{Z}_7)$.
- $f(2) = 3 \bmod 7$, a quadratic non-residue modulo 7.
- $f(3) = 4 \bmod 7$, a quadratic residue modulo 7 with square roots 2 and 5. This yields the points $(3, 2), (3, 5) \in E(\mathbb{Z}_7)$.
- $f(4) = 2 \bmod 7$, a quadratic residue modulo 7 with square roots 3 and 4. This yields the points $(4, 3), (4, 4) \in E(\mathbb{Z}_7)$.
- $f(5) = 3 \bmod 7$, a quadratic non-residue modulo 7.
- $f(6) = 6 \bmod 7$, a quadratic non-residue modulo 7.

Including the point at infinity, there are 6 points in $E(\mathbb{Z}_7)$. $\diamondsuit$

A useful way to think about $E(\mathbb{Z}_p)$ is to look at the graph of Equation (8.1) over the reals (i.e., the equation $y^2 = x^3 + Ax + B$ without reduction modulo p) as in Figure 8.2. This figure does not correspond exactly to $E(\mathbb{Z}_p)$ because, for example, $E(\mathbb{Z}_p)$ has a finite number of points ($\mathbb{Z}_p$ is, after all, a finite set) while there are an infinite number of solutions to the same equation if we allow x and y to range over all real numbers. Nevertheless, the picture provides useful intuition. In such a figure, one can think of the "point at infinity" $\mathcal{O}$ as sitting at the top of the y-axis and lying on every vertical line.

It can be shown that every line intersecting $E(\mathbb{Z}_p)$ intersects it in exactly 3 points, where (1) a point P is counted twice if the line is tangent to the curve at P, and (2) the point at infinity is also counted when the line is vertical. This fact is used to define a binary operation, called "addition" and denoted by $+$, on points of $E(\mathbb{Z}_p)$ in the following way:

- The point $\mathcal{O}$ is defined to be an (additive) identity; that is, for all $P \in E(\mathbb{Z}_p)$ we define $P + \mathcal{O} = \mathcal{O} + P = P$.
- For two points $P_1, P_2 \neq \mathcal{O}$ on E, we evaluate their sum $P_1 + P_2$ by drawing the line through P_1, P_2 (if $P_1 = P_2$ then draw the line tangent to the curve at P_1) and finding the third point of intersection P_3 of this line with $E(\mathbb{Z}_p)$; the third point of intersection may be $P_3 = \mathcal{O}$ if the line is vertical. If $P_3 = (x, y) \neq \mathcal{O}$ then we define $P_1 + P_2 \stackrel{\text{def}}{=} (x, -y)$. (Graphically, this corresponds to reflecting P_3 in the x-axis.) If $P_3 = \mathcal{O}$ then $P_1 + P_2 \stackrel{\text{def}}{=} \mathcal{O}$.

If $P = (x, y) \neq \mathcal{O}$ is a point of $E(\mathbb{Z}_p)$, then $-P \stackrel{\text{def}}{=} (x, -y)$ (which is clearly also a point of $E(\mathbb{Z}_p)$) is the unique inverse of P. Indeed, the line

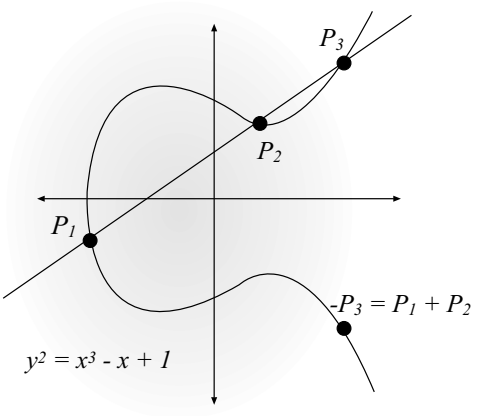

FIGURE 8.2: An elliptic curve over the reals.

through (x, y) and $(x, -y)$ is vertical, and so the addition rule implies that $P + (-P) = \mathcal{O}$. (If $y = 0$ then $P = (x, y) = (x, -y) = -P$ but then the tangent line at P will be vertical and so $P + (-P) = \mathcal{O}$ here as well.) Of course, $-\mathcal{O} = \mathcal{O}$.

It is straightforward, but tedious, to work out the addition law concretely. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points in $E(\mathbb{Z}_p)$, with $P_1, P_2 \neq \mathcal{O}$ and E as in Equation (8.1). To keep matters simple, suppose $x_1 \neq x_2$ (dealing with the case $x_1 = x_2$ is still straightforward but even more tedious). The slope of the line through these points is

$$m \stackrel{\text{def}}{=} \left[\frac{y_2 - y_1}{x_2 - x_1} \bmod p \right] ;$$

our assumption that $x_1 \neq x_2$ means that the inverse of $(x_2 - x_1)$ modulo p exists. The line passing through P_1 and P_2 has the equation

$$y = m \cdot (x - x_1) + y_1 \bmod p \,. \tag{8.2}$$

To find the third point of intersection of this line with E, substitute the above into the equation for E to obtain

$$\Big(m \cdot (x - x_1) + y_1 \Big)^2 = x^3 + Ax + B \bmod p.$$

The values of x that satisfy this equation are x_1, x_2, and

$$x_3 \stackrel{\text{def}}{=} [m^2 - x_1 - x_2 \bmod p\,].$$

The first two solutions correspond to the original points P_1 and P_2, while the third is the x-coordinate of the third point of intersection P_3. Plugging x_3 into Equation (8.2) we find that the y-coordinate corresponding to x_3 is

$y_3 = [m \cdot (x_3 - x_1) + y_1 \bmod p]$. To obtain the desired answer $P_1 + P_2$, we flip the sign of y to obtain:

$$(x_1, y_1) + (x_2, y_2) = \big([m^2 - x_1 - x_2 \bmod p],\ [m \cdot (x_1 - x_3) - y_1 \bmod p]\big).$$

We summarize and extend this in the following proposition.

PROPOSITION 8.68 *Let $p \geq 5$ be prime and let E be the elliptic curve given by $y^2 = x^3 + Ax + B \bmod p$ where $4A^3 + 27B^2 \neq 0 \bmod p$. Let $P_1, P_2 \neq \mathcal{O}$ be points on E, with $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.*

1. *If $x_1 \neq x_2$, then $P_1 + P_2 = (x_3, y_3)$ with*

   $$x_3 = [m^2 - x_1 - x_2 \bmod p] \quad \text{and} \quad y_3 = [m \cdot (x_1 - x_3) - y_1 \bmod p],$$

   where $m = \left[\frac{y_2 - y_1}{x_2 - x_1} \bmod p\right]$.

2. *If $x_1 = x_2$ but $y_1 \neq y_2$ then $P_1 = -P_2$ and so $P_1 + P_2 = \mathcal{O}$.*

3. *If $P_1 = P_2$ and $y_1 = 0$ then $P_1 + P_2 = 2P_1 = \mathcal{O}$.*

4. *If $P_1 = P_2$ and $y_1 \neq 0$ then $P_1 + P_2 = 2P_1 = (x_3, y_3)$ with*

   $$x_3 = [m^2 - 2x_1 \bmod p] \quad \text{and} \quad y_3 = [m \cdot (x_1 - x_3) - y_1 \bmod p],$$

   where $m = \left[\frac{3x_1^2 + A}{2y_1} \bmod p\right]$.

Somewhat amazingly, the set of points $E(\mathbb{Z}_p)$ along with the addition rule defined above form an abelian group, called the *elliptic-curve group of E*. Commutativity follows from the way addition is defined, $\mathcal{O}$ acts as the identity, and we have already seen that each point on $E(\mathbb{Z}_p)$ has an inverse in $E(\mathbb{Z}_p)$. The difficult property to verify is associativity, which the disbelieving reader can check through tedious calculation. A more illuminating proof that does not involve explicit calculation relies on algebraic geometry.

Example 8.69

Consider the curve from Example 8.67. We show associativity for three specific points. Let $P_1 = (1, 0)$, $P_2 = Q_2 = (4, 3)$. When computing $P_1 + P_2$ we get $m = [(3 - 0) \cdot (4 - 1)^{-1} \bmod 7] = 1$ and $[1^2 - 1 - 4 \bmod 7] = 3$. Thus,

$$P_3 \stackrel{\text{def}}{=} P_1 + P_2 = (3,\ [1 \cdot (1 - 3) - 0 \bmod 7]) = (3, 5);$$

note that this is indeed a point on $E(\mathbb{Z}_7)$. If we then compute $P_3 + Q_2$ we get $m = [(3 - 5) \cdot (4 - 3)^{-1} \bmod 7] = 5$ and $[5^2 - 3 - 4 \bmod 7] = 4$. Thus,

$$(P_1 + P_2) + Q_2 = P_3 + Q_2 = (4,\ [5 \cdot (3 - 4) - 5 \bmod 7]) = (4, 4).$$

If we compute $P_2 + Q_2 = 2P_2$ we obtain $m = [(3 \cdot 4^2 + 3) \cdot (2 \cdot 3)^{-1} \bmod 7] = 5$ and $[5^2 - 2 \cdot 4 \bmod 7] = 3$. Thus,

$$P_3' \stackrel{\text{def}}{=} P_2 + Q_2 = (3,\ \ [5 \cdot (4-3) - 3 \bmod 7]) = (3, 2).$$

If we then compute $P_1 + P_3'$ we find $m = [2 \cdot (3-1)^{-1} \bmod 7] = 1$ and $[1^2 - 1 - 3 \bmod 7] = 4$. So

$$P_1 + (P_2 + Q_2) = P_1 + P_3' = (4,\ \ [1 \cdot (1-4) - 0 \bmod 7]) = (4, 4),$$

and $P_1 + (P_2 + Q_2) = (P_1 + P_2) + Q_2$. ♢

Recall that when a group is written additively, "exponentiation" corresponds to repeated addition. Thus, if we fix some point P in an elliptic-curve group, the discrete-logarithm problem becomes (informally) the problem of computing the integer x from xP, while the decisional Diffie–Hellman problem becomes (informally) the problem of distinguishing tuples of the form (aP, bP, abP) from those of the form (aP, bP, cP). These problems are believed to be hard in elliptic-curve groups (or subgroups thereof) of large prime order, subject to a few technical conditions we will mention in passing below.

If we want an elliptic-curve group (or subgroup) of large prime order, the first question we must address is: how large are elliptic-curve groups? As noted in Example 8.67, the equation $y^2 = f(x) \bmod p$ has two solutions whenever $f(x)$ is a quadratic residue, and one solution when $f(x) = 0$. Since half the elements in $\mathbb{Z}_p^*$ are quadratic residues, we heuristically expect to find $2 \cdot (p-1)/2 + 1 = p$ points on the curve. Including the point at infinity, this means there should be about $p + 1$ points in an elliptic-curve group over $\mathbb{Z}_p$. The *Hasse bound* says this heuristic estimate is accurate, in the sense that every elliptic-curve group has "almost" this many points.

THEOREM 8.70 (Hasse bound) *Let p be prime, and let E be an elliptic curve over $\mathbb{Z}_p$. Then $p + 1 - 2\sqrt{p} \le |E(\mathbb{Z}_p)| \le p + 1 + 2\sqrt{p}$.*

This bound implies that it is always easy to *find* a point on a given elliptic curve $y^2 = f(x) \bmod p$: simply choose uniform $x \in \mathbb{Z}_p$, check whether $f(x)$ is 0 or a quadratic residue, and—if so—let y be a square root of $f(x)$. (Algorithms for deciding quadratic residuosity and computing square roots modulo a prime are discussed in Chapter 13.) Since points on the elliptic curve are plentiful, we will not have to try very many values of x before finding a point.

The Hasse bound only gives a range for the size of an elliptic-curve group. For a fixed prime p, however, the order of a random elliptic curve over $\mathbb{Z}_p$ (namely, a curve defined by Equation (8.1) in which A, B are chosen uniformly in $\mathbb{Z}_p$ subject to the constraint $4A^3 + 27B^2 \neq 0 \bmod p$) is heuristically found to be "close" to uniformly distributed in the Hasse interval. There also exist efficient algorithms—whose description and analysis are well beyond the scope of this book—for counting the number of points on an elliptic

curve. This suggests an approach to elliptic-curve parameter generation as in Algorithm 8.71.

ALGORITHM 8.71
Elliptic-curve group-generation algorithm $\mathcal{G}$

Input: Security parameter 1^n
Output: Cyclic group $\mathbb{G}$, its (prime) order q, and a generator g

generate a uniform n-bit prime p
until q is an n-bit prime **do**:
  choose $A, B \leftarrow \mathbb{Z}_p$ with $4A^3 + 27B^2 \neq 0 \bmod p$, defining elliptic curve E as in Equation (8.1)
  let q be the number of points on $E(\mathbb{Z}_p)$
choose $g \in E(\mathbb{Z}_p) \setminus \{\mathcal{O}\}$
return (A, B, p), q, g // $\mathbb{G}$ is the elliptic-curve group of E

Certain classes of curves are considered cryptographically weak and should be avoided. These include elliptic-curve groups over $\mathbb{Z}_p$ whose order is equal to p (*anomalous curves*) or $p+1$ (*supersingular curves*), or whose order divides $p^k - 1$ for "small" k. A full discussion is beyond the scope of this book. In practice, standardized curves (such as those recommended by NIST) are used, and generating a curve of one's own is not advised.

Efficiency Considerations

We conclude this section with a very brief discussion of some standard efficiency improvements when using elliptic curves.

Point compression. A useful observation is that the number of bits needed to represent a point on an elliptic curve can be reduced almost by half. To see this, note that for any point (x, y) on an elliptic curve $E : y^2 = f(x) \bmod p$ there are at most two points on the curve that have x-coordinate x: namely, (x, y) and $(x, -y)$. (It is possible that $y = 0$ in which case these are the same point.) Thus, we can specify any point $P = (x, y)$ by its x-coordinate and a bit b that distinguishes between the (at most) two possibilities for the value of its y-coordinate. One convenient way to do this is to set $b = 0$ if $y < p/2$ and $b = 1$ otherwise. Given x and b we can recover P by computing the two square roots y_1, y_2 of the equation $y^2 = f(x) \bmod p$; since $y_1 = -y_2 \bmod p$ and so $y_1 = p - y_2$, exactly one of y_1, y_2 will be less than $p/2$.

Projective coordinates. Representing elliptic-curve points as we have been doing until now—in which a point P on an elliptic curve is described by a pair of field elements (x, y)—is called using *affine coordinates*. There are alternate ways to represent points, using *projective coordinates*, that can offer efficiency

improvements. While these alternate representations can be motivated mathematically, we treat them simply as useful computational aids.

Points in projective coordinates are represented using *three* elements of $\mathbb{Z}_p$. An interesting feature is that a point has multiple representations. When using standard projective coordinates, a point $P \neq \mathcal{O}$ with representation (x, y) in affine coordinates is represented by any tuple $(X, Y, Z) \in \mathbb{Z}_p^3$ for which $X/Z = x \bmod p$ and $Y/Z = y \bmod p$. The point $\mathcal{O}$ is represented by any tuple $(0, Y, 0)$ with $Y \neq 0$, and these are the only points (X, Y, Z) with $Z = 0$. We can easily translate between coordinate systems: (x, y) in affine coordinates can be mapped to $(x, y, 1)$ in projective coordinates, and (X, Y, Z) (with $Z \neq 0$) in projective coordinates is mapped to the representation $([X/Z \bmod p\,], [Y/Z \bmod p\,])$ in affine coordinates.

The main advantage of using projective coordinates is that we can add points without having to compute inverses modulo p. (Adding points in affine coordinates requires computing inverses; see Proposition 8.68.) We accomplish this by exploiting the fact that points have multiple representations. To see this, let us work out the addition law for two points $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ with $P_1, P_2 \neq \mathcal{O}$ (so $Z_1, Z_2 \neq 0$) and $P_1 \neq \pm P_2$ (so $X_1/Z_1 \neq X_2/Z_2 \bmod p$). (If either P_1 or P_2 are equal to $\mathcal{O}$, addition is trivial. The case of $P_1 = \pm P_2$ can be handled as well, but we omit details here.) We can express P_1 and P_2 as $(X_1/Z_1, Y_1/Z_1)$ and $(X_2/Z_2, Y_2/Z_2)$ in affine coordinates, so

$$\begin{aligned} P_3 \stackrel{\text{def}}{=} P_1 + P_2 = \; & \big(m^2 - X_1/Z_1 - X_2/Z_2, \\ & m \cdot (X_1/Z_1 - m^2 + X_1/Z_1 + X_2/Z_2) - Y_1/Z_1,\ 1\big), \end{aligned}$$

where

$$m = (Y_2/Z_2 - Y_1/Z_1)(X_2/Z_2 - X_1/Z_1)^{-1} = (Y_2 Z_1 - Y_1 Z_2)(X_2 Z_1 - X_1 Z_2)^{-1}$$

and all computations are done modulo p. Note we are using projective coordinates to represent P_3, setting $Z_3 = 1$ above. But using projective coordinates means we are not limited to $Z_3 = 1$. Multiplying each coordinate by $Z_1 Z_2 (X_2 Z_1 - X_1 Z_2)^3 \neq 0 \bmod p$, we find that P_3 can also be represented as

$$P_3 = \Big(vw,\ \ u(v^2 X_1 Z_2 - w) - v^3 Y_1 Z_2,\ \ Z_1 Z_2 v^3\Big) \tag{8.3}$$

where

$$\begin{aligned} u = Y_2 Z_1 - Y_1 Z_2, \qquad v = X_2 Z_1 - X_1 Z_2, \\ w = u^2 Z_1 Z_2 - v^3 - 2v^2 X_1 Z_2. \end{aligned} \tag{8.4}$$

The point to notice is that the computations in Equations (8.3) and (8.4) can be carried out without having to perform any modular inversions.

Precisely because points have multiple representations in projective coordinates, some subtleties can arise when projective coordinates are used. (We

have explicitly assumed until now that group elements have unique representations as bit-strings.) Specifically, a point expressed in projective coordinates may reveal some information about how that point was obtained, which may depend on some secret information. To address this—as well as for reasons of efficiency—affine coordinates should be used for transmitting and storing points, with projective coordinates used only as an intermediate representation during the course of a computation (with points converted to/from projective coordinates at the beginning/end of the computation).

8.4 *Cryptographic Applications

We have spent a fair bit of time discussing number theory and group theory, and introducing computational hardness assumptions that are widely believed to hold. Applications of these assumptions will occupy us for the rest of the book, but we provide some brief examples here.

8.4.1 One-Way Functions and Permutations

One-way functions are the minimal cryptographic primitive, and they are both necessary and sufficient for private-key encryption and message authentication codes. A more complete discussion of the role of one-way functions in cryptography appears in Chapter 7; here we only provide a definition of one-way functions and demonstrate that their existence follows from the number-theoretic hardness assumptions we have seen in this chapter.

Informally, a function f is *one-way* if it is easy to compute but hard to invert. The following experiment and definition, a restatement of Definition 7.1, formalizes this.

The inverting experiment $\mathsf{Invert}_{\mathcal{A},f}(n)$**:**

1. *Choose uniform* $x \in \{0,1\}^n$ *and compute* $y := f(x)$.
2. $\mathcal{A}$ *is given* 1^n *and* y *as input, and outputs* x'.
3. *The output of the experiment is* 1 *if and only if* $f(x') = y$.

DEFINITION 8.72 *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is* one-way *if the following two conditions hold:*

1. **(Easy to compute:)** *There is a polynomial-time algorithm that on input* x *outputs* $f(x)$.
2. **(Hard to invert:)** *For all* PPT *algorithms* $\mathcal{A}$ *there is a negligible function* negl *such that* $\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n)$.

We now show formally that the factoring assumption implies the existence of a one-way function. Let Gen be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes except with probability negligible in n. (We use Gen rather than $\mathsf{GenModulus}$ here purely for notational convenience.) Since Gen runs in polynomial time, there is a polynomial upper bound on the number of random bits the algorithm uses. For simplicity, and in order to get the main ideas across, we assume Gen always uses at most n random bits on input 1^n. In Algorithm 8.73 we define a function f_{Gen} that uses its input as the random bits for running Gen. Thus, f_{Gen} is a *deterministic* function as required.

ALGORITHM 8.73
Algorithm computing f_{Gen}

Input: String x of length n
Output: Integer N

compute $(N, p, q) := \mathsf{Gen}(1^n; x)$
// i.e., run $\mathsf{Gen}(1^n)$ using x as the random tape
return N

If the factoring problem is hard relative to Gen then, intuitively, f_{Gen} is a one-way function. Certainly f_{Gen} is easy to compute. As for the hardness of inverting this function, note that the following distributions are identical:

1. The modulus N output by $f_{\mathsf{Gen}}(x)$, when $x \in \{0,1\}^n$ is chosen uniformly.
2. The modulus N output by (the randomized algorithm) $\mathsf{Gen}(1^n)$.

If moduli N generated according to the second distribution are hard to factor, then the same holds for moduli N generated according to the first distribution. Moreover, given any preimage x' of N with respect to f_{Gen} (i.e., an x' for which $f_{\mathsf{Gen}}(x') = N$; note that we do not require $x' = x$), it is easy to recover a factor of N by running $\mathsf{Gen}(1^n; x')$ to obtain (N, p, q) and outputting the factors p and q. Thus, finding a preimage of N with respect to f_{Gen} is as hard as factoring N. One can easily turn this into a formal proof of the following:

THEOREM 8.74 *If the factoring problem is hard relative to* Gen*, then* f_{Gen} *is a one-way function.*

One-Way Permutations

We can also use number-theoretic assumptions to construct a family of one-way *permutations*. We begin with a restatement of Definitions 7.2 and 7.3, specialized to the case of permutations:

DEFINITION 8.75 *A triple* $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ *of probabilistic polynomial-time algorithms is a* family of permutations *if the following hold:*

1. *The* parameter-generation algorithm Gen*, on input* 1^n*, outputs parameters* I *with* $|I| \geq n$*. Each value of* I *defines a set* $\mathcal{D}_I$ *that constitutes the domain and range of a permutation (i.e., bijection)* $f_I : \mathcal{D}_I \to \mathcal{D}_I$*.*

2. *The* sampling algorithm Samp*, on input* I*, outputs a uniformly distributed element of* $\mathcal{D}_I$*.*

3. *The deterministic* evaluation algorithm f*, on input* I *and* $x \in \mathcal{D}_I$*, outputs an element* $y \in \mathcal{D}_I$*. We write this as* $y := f_I(x)$*.*

Given a family of functions Π, consider the following experiment for any algorithm $\mathcal{A}$ and parameter n:

The inverting experiment $\mathsf{Invert}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain* I*, and then* $\mathsf{Samp}(I)$ *is run to choose a uniform* $x \in \mathcal{D}_I$*. Finally,* $y := f_I(x)$ *is computed.*
2. $\mathcal{A}$ *is given* I *and* y *as input, and outputs* x'*.*
3. *The output of the experiment is* 1 *if and only if* $f_I(x') = y$*.*

DEFINITION 8.76 *The family of permutations* $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ *is* one-way *if for all probabilistic polynomial-time algorithms* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Invert}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Given GenRSA as in Section 8.2.4, Construction 8.77 defines a family of permutations. It is immediate that if the RSA problem is hard relative to GenRSA then this family is one-way. It can similarly be shown that hardness of the discrete-logarithm problem in $\mathbb{Z}_p^*$, with p prime, implies the existence of a one-way family of permutations; see Section 7.1.2.

CONSTRUCTION 8.77

Let GenRSA be as before. Define a family of permutations as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d) and output $I = \langle N, e \rangle$. Set $\mathcal{D}_I = \mathbb{Z}_N^*$.
- Samp: on input $I = \langle N, e \rangle$, choose a uniform element of $\mathbb{Z}_N^*$.
- f: on input $I = \langle N, e \rangle$ and $x \in \mathbb{Z}_N^*$, output $[x^e \bmod N]$.

A family of permutations based on the RSA problem.

8.4.2 Constructing Collision-Resistant Hash Functions

Collision-resistant hash functions were introduced in Section 5.1. Although we have discussed constructions of collision-resistant hash functions used in practice in Section 6.3, we have not yet seen constructions that can be rigorously based on simpler assumptions. We show here a construction based on the discrete-logarithm assumption in prime-order groups. (A construction based on the RSA problem is described in Exercise 8.20.) Although these constructions are less efficient than the hash functions used in practice, they are important since they illustrate the *feasibility* of achieving collision resistance based on standard and well-studied number-theoretic assumptions.

Let $\mathcal{G}$ be a polynomial-time algorithm that, on input 1^n, outputs a (description of a) cyclic group $\mathbb{G}$, its order q (with $\|q\| = n$), and a generator g. Here we also require that q is *prime* except possibly with negligible probability. A fixed-length hash function based on $\mathcal{G}$ is given in Construction 8.78.

CONSTRUCTION 8.78

Let $\mathcal{G}$ be as described in the text. Define a fixed-length hash function (Gen, H) as follows:

- Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select a uniform $h \in \mathbb{G}$. Output $s := \langle \mathbb{G}, q, g, h \rangle$ as the key.
- H: given a key $s = \langle \mathbb{G}, q, g, h \rangle$ and input $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$, output $H^s(x_1, x_2) := g^{x_1} h^{x_2} \in \mathbb{G}$.

A fixed-length hash function.

Note that Gen and H can be computed in polynomial time. Before continuing with an analysis of the construction, we make some technical remarks:

- For a given $s = \langle \mathbb{G}, q, g, h \rangle$ with $n = \|q\|$, the function H^s is described as taking elements of $\mathbb{Z}_q \times \mathbb{Z}_q$ as input. However, H^s can be viewed as taking bit-strings of length $2 \cdot (n-1)$ as input if we parse an input $x \in \{0,1\}^{2(n-1)}$ as two strings x_1, x_2, each of length $n-1$, and then view x_1, x_2 as elements of $\mathbb{Z}_q$ in the natural way.
- The output of H^s is similarly specified as being an element of $\mathbb{G}$, but we can view this as a bit-string if we fix some representation of $\mathbb{G}$. To satisfy the requirements of Definition 5.2 (which requires the output length to be fixed as a function of n) we can pad the output as needed.
- Given the above, the construction only compresses its input for groups $\mathbb{G}$ in which elements of $\mathbb{G}$ can be represented using fewer than $2n-2$ bits. This holds both for the groups output by Algorithm 8.65 (assuming $n \ll \ell$, which is usually the case), as well as for elliptic-curve groups when point compression is used. A generalization of Construction 8.78

can be used to obtain compression from *any* $\mathcal{G}$ for which the discrete-logarithm problem is hard, regardless of the number of bits required to represent group elements; see Exercise 8.21.

THEOREM 8.79 *If the discrete-logarithm problem is hard relative to $\mathcal{G}$, then Construction 8.78 is a fixed-length collision-resistant hash function (subject to the discussion regarding compression, above).*

PROOF Let $\Pi = (\mathsf{Gen}, H)$ be as in Construction 8.78, and let $\mathcal{A}$ be a probabilistic polynomial-time algorithm with

$$\varepsilon(n) \stackrel{\text{def}}{=} \Pr[\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}(n) = 1]$$

(cf. Definition 5.2). We show how $\mathcal{A}$ can be used by an algorithm $\mathcal{A}'$ to solve the discrete-logarithm problem with success probability $\varepsilon(n)$:

Algorithm $\mathcal{A}'$:
The algorithm is given $\mathbb{G}, q, g, h$ as input.

1. Let $s := \langle \mathbb{G}, q, g, h \rangle$. Run $\mathcal{A}(s)$ and obtain output x and x'.
2. If $x \neq x'$ and $H^s(x) = H^s(x')$ then:
   (a) If $h = 1$ return 0.
   (b) Otherwise ($h \neq 1$), parse x as (x_1, x_2) and parse x' as (x'_1, x'_2), where $x_1, x_2, x'_1, x'_2 \in \mathbb{Z}_q$, and return the result $\left[(x_1 - x'_1) \cdot (x'_2 - x_2)^{-1} \bmod q\right]$.

Clearly, $\mathcal{A}'$ runs in polynomial time. Furthermore, the input s given to $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ is distributed exactly as in experiment $\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}$ for the same value of the security parameter n. (The input to $\mathcal{A}'$ is generated by running $\mathcal{G}(1^n)$ to obtain $\mathbb{G}, q, g$ and then choosing $h \in \mathbb{G}$ uniformly at random. This is exactly how s is generated by $\mathsf{Gen}(1^n)$.) So, with probability exactly $\varepsilon(n)$ there is a *collision*; i.e., $x \neq x'$ and $H^s(x) = H^s(x')$.

We claim that whenever there is a collision, $\mathcal{A}'$ returns the correct answer $\log_g h$. If $h = 1$ then this is clearly true (since $\log_g h = 0$ in this case). Assuming $h \neq 1$, the existence of a collision means that

$$\begin{aligned} H^s(x_1, x_2) = H^s(x'_1, x'_2) &\Rightarrow g^{x_1} h^{x_2} = g^{x'_1} h^{x'_2} \\ &\Rightarrow g^{x_1 - x'_1} = h^{x'_2 - x_2}. \end{aligned} \tag{8.5}$$

Note that $x'_2 - x_2 \neq 0 \bmod q$; otherwise, we would have $x_1 = x'_1 \bmod q$ but then $x = x'$ and we would not have a collision. Since q is prime, the inverse $\Delta \stackrel{\text{def}}{=} [(x'_2 - x_2)^{-1} \bmod q]$ exists. Raising each side of Equation (8.5) to this power gives:

$$g^{(x_1 - x'_1) \cdot \Delta} = \left(h^{x'_2 - x_2}\right)^{\Delta} = h^1 = h,$$

and so the output returned by $\mathcal{A}'$ is

$$\log_g h = [(x_1 - x_1') \cdot \Delta \bmod q] = \left[(x_1 - x_1') \cdot (x_2' - x_2)^{-1} \bmod q\right].$$

We see that $\mathcal{A}'$ correctly solves the discrete-logarithm problem with probability exactly $\varepsilon(n)$. Since, by assumption, the discrete-logarithm problem is hard relative to $\mathcal{G}$, we conclude that $\varepsilon(n)$ is negligible. ■

Using Exercise 8.21 in combination with the Merkle–Damgård transform (see Section 5.2) we obtain:

THEOREM 8.80 *If the discrete-logarithm problem is hard, then collision-resistant hash functions exist.*

References and Additional Reading

The book by Childs [44] has excellent coverage of the group theory discussed in this chapter (and more), in greater depth but at a similar level of exposition. Shoup [159] gives a more advanced, yet still accessible, treatment of much of this material also, with special focus on algorithmic aspects. (Our statement of Bertrand's postulate is taken from [159, Theorem 5.8].) Relatively gentle introductions to abstract algebra and group theory that go well beyond what we have space for here are available in the books by Fraleigh [67] and Herstein [89]; the interested reader will have no trouble finding more advanced algebra texts if they are so inclined.

The first efficient primality test was by Solovay and Strassen [164]. The Miller–Rabin test is due to Miller [127] and Rabin [146]. A deterministic primality test was discovered by Agrawal et al. [5]. See Dietzfelbinger [57] for a comprehensive survey of this area.

The RSA problem was publicly introduced by Rivest, Shamir, and Adleman [148], although it was revealed in 1997 that Ellis, Cocks, and Williamson, three members of GCHQ (a British intelligence agency), had explored similar ideas—without fully recognizing their importance—in a classified setting several years earlier. The discrete-logarithm and Diffie–Hellman problems were first considered, at least implicitly, by Diffie and Hellman [58].

Most treatments of elliptic curves require advanced mathematical background on the part of the reader. The book by Silverman and Tate [160] is perhaps an exception. As with many books on the subject written for mathematicians, however, that book has little coverage of elliptic curves over *finite* fields, which is the case most relevant to cryptography. The text by Washington [176], although a bit more advanced, deals heavily (but not exclusively)

with the finite-field case. Implementation issues related to elliptic-curve cryptography are covered by Hankerson et al. [83]. Recommended parameters for elliptic curves, as well as subgroups modulo a prime, are given by NIST [132].

The construction of a collision-resistant hash function based on the discrete-logarithm problem is due to [43], and an earlier construction based on the hardness of factoring is given in [81] (see also Exercise 8.20).

Exercises

8.1 Let $\mathbb{G}$ be an abelian group. Prove that there is a *unique* identity in $\mathbb{G}$, and that every element $g \in \mathbb{G}$ has a *unique* inverse.

8.2 Show that Proposition 8.36 does not necessarily hold when $\mathbb{G}$ is infinite.

Hint: Consider the set $\{1\} \cup \{2, 4, 6, 8, \ldots\} \subset \mathbb{R}$.

8.3 Let $\mathbb{G}$ be a finite group, and $g \in \mathbb{G}$. Show that $\langle g \rangle$ is a subgroup of $\mathbb{G}$. Is the set $\{g^0, g^1, \ldots\}$ necessarily a subgroup of $\mathbb{G}$ when $\mathbb{G}$ is infinite?

8.4 This question concerns the Euler phi function.

(a) Let p be prime and $e \geq 1$ an integer. Show that $\phi(p^e) = p^{e-1}(p-1)$.

(b) Let p, q be relatively prime. Show that $\phi(pq) = \phi(p) \cdot \phi(q)$. (You may use the Chinese remainder theorem.)

(c) Prove Theorem 8.19.

8.5 Compute the final two (decimal) digits of 3^{1000} (by hand).

Hint: The answer is $[3^{1000} \bmod 100]$.

8.6 Compute $[101^{4,800,000,002} \bmod 35]$ (by hand).

8.7 Compute $[46^{51} \bmod 55]$ (by hand) using the Chinese remainder theorem.

8.8 Prove that if $\mathbb{G}, \mathbb{H}$ are groups, then $\mathbb{G} \times \mathbb{H}$ is a group.

8.9 Let p, N be integers with $p \,|\, N$. Prove that for any integer X,

$$[[X \bmod N] \bmod p\,] = [X \bmod p\,].$$

Show that, in contrast, $[[X \bmod p\,] \bmod N]$ need not equal $[X \bmod N]$.

8.10 Corollary 8.21 shows that if $N = pq$ and $ed = 1 \bmod \phi(N)$ then for all $x \in \mathbb{Z}_N^*$ we have $(x^e)^d = x \bmod N$. Show that this holds for all $x \in \{0, \ldots, N-1\}$.

Hint: Use the Chinese remainder theorem.

8.11 Complete the details of the proof of the Chinese remainder theorem, showing that $\mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.

8.12 This exercise develops an efficient algorithm for testing whether an integer is a perfect power.

(a) Show that if $N = \hat{N}^e$ for some integers $\hat{N}, e > 1$ then $e \leq \|N\|$.

(b) Given N and e with $2 \leq e \leq \|N\| + 1$, show how to determine in $\mathsf{poly}(\|N\|)$ time whether there exists an integer $\hat{N}$ with $\hat{N}^e = N$.

Hint: Use binary search.

(c) Given N, show how to test in $\mathsf{poly}(\|N\|)$ time whether N is a perfect power.

8.13 Given N and $a \in \mathbb{Z}_N^*$, show how to test in polynomial time whether a is a strong witness that N is composite.

8.14 Fix N, e with $\gcd(e, \phi(N)) = 1$, and assume there is an adversary $\mathcal{A}$ running in time t for which

$$\Pr\left[\mathcal{A}\left([x^e \bmod N]\right) = x\right] = 0.01,$$

where the probability is taken over uniform choice of $x \in \mathbb{Z}_N^*$. Show that it is possible to construct an adversary $\mathcal{A}'$ for which

$$\Pr\left[\mathcal{A}'\left([x^e \bmod N]\right) = x\right] = 0.99$$

for *all* x. The running time t' of $\mathcal{A}'$ should be polynomial in t and $\|N\|$.

Hint: Use the fact that $y^{1/e} \cdot r = (y \cdot r^e)^{1/e} \bmod N$.

8.15 Formally define the CDH assumption. Prove that hardness of the CDH problem relative to $\mathcal{G}$ implies hardness of the discrete-logarithm problem relative to $\mathcal{G}$, and that hardness of the DDH problem relative to $\mathcal{G}$ implies hardness of the CDH problem relative to $\mathcal{G}$.

8.16 Determine the points on the elliptic curve $E : y^2 = x^3 + 2x + 1$ over $\mathbb{Z}_{11}$. How many points are on this curve?

8.17 Consider the elliptic-curve group from Example 8.67. (See also Example 8.69.) Compute $(1, 0) + (4, 3) + (4, 3)$ in this group by first converting to projective coordinates and then using Equations (8.3) and (8.4).

8.18 Prove the fourth statement in Proposition 8.68.

8.19 Can the following problem be solved in polynomial time? Given a prime p, a value $x \in \mathbb{Z}_{p-1}^*$, and $y := [g^x \bmod p]$ (where g is a uniform value in $\mathbb{Z}_p^*$), find g, i.e., compute $y^{1/x} \bmod p$. If your answer is "yes," give a polynomial-time algorithm. If your answer is "no," show a reduction to one of the assumptions introduced in this chapter.

8.20 Let GenRSA be as in Section 8.2.4. Prove that if the RSA problem is hard relative to GenRSA then Construction 8.81 is a fixed-length collision-resistant hash function.

CONSTRUCTION 8.81

Define (Gen, H) as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain N, e, d, and select $y \leftarrow \mathbb{Z}_N^*$. The key is $s := \langle N, e, y \rangle$.
- H: if $s = \langle N, e, y \rangle$, then H^s maps inputs in $\{0,1\}^{3n}$ to outputs in $\mathbb{Z}_N^*$. Let $f_0^s(x) \stackrel{\text{def}}{=} [x^e \bmod N]$ and $f_1^s(x) \stackrel{\text{def}}{=} [y \cdot x^e \bmod N]$. For a $3n$-bit long string $x = x_1 \cdots x_{3n}$, define

$$H^s(x) \stackrel{\text{def}}{=} f_{x_1}^s\left(f_{x_2}^s\left(\cdots\left(1\right)\cdots\right)\right).$$

8.21 Consider the following generalization of Construction 8.78:

CONSTRUCTION 8.82

Define a fixed-length hash function (Gen, H) as follows:

(a) Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, h_1)$ and then select $h_2, \ldots, h_t \leftarrow \mathbb{G}$. Output $s := \langle \mathbb{G}, q, (h_1, \ldots, h_t) \rangle$ as the key.

(b) H: given a key $s = \langle \mathbb{G}, q, (h_1, \ldots, h_t) \rangle$ and input $(x_1, \ldots, x_t)$ with $x_i \in \mathbb{Z}_q$, output $H^s(x_1, \ldots, x_t) := \prod_i h_i^{x_i}$.

(a) Prove that if the discrete-logarithm problem is hard relative to $\mathcal{G}$ and q is prime, then for any $t = \mathsf{poly}(n)$ this construction is a fixed-length collision-resistant hash function.

(b) Discuss how this construction can be used to obtain *compression* regardless of the number of bits needed to represent elements of $\mathbb{G}$ (as long as it is polynomial in n).

Chapter 9

*Algorithms for Factoring and Computing Discrete Logarithms

In the last chapter, we introduced several number-theoretic problems—most prominently, *factoring* the product of two large primes and *computing discrete logarithms* in certain groups—that are widely believed to be hard. As defined there, this means there are presumed to be no polynomial-time algorithms for these problems. This *asymptotic* notion of hardness, however, tells us little about how to set the security parameter—sometimes called the *key length*, although the terms are not interchangeable—to achieve some desired, *concrete* level of security in practice. A proper understanding of this issue is extremely important for the real-world deployment of cryptosystems based on these problems. Setting the security parameter too low means a cryptosystem may be vulnerable to attacks more efficient than anticipated; being overly conservative and setting the security parameter too high will give good security, but at the expense of efficiency for the honest users. The relative difficulty of different number-theoretic problems can also play a role in determining which problems to use as the basis for building cryptosystems in the first place.

The fundamental issue, of course, is that a brute-force search may not be the best algorithm for solving a given problem; thus, using key length n does not, in general, give security against attackers running for 2^n time. This is in contrast to the private-key setting where the best attacks on existing block ciphers have roughly the complexity of brute-force search (ignoring pre-computation). As a consequence, the key lengths used in the public-key setting tend to be significantly larger than those used in the private-key setting.

To gain a better appreciation of this point, we explore in this chapter several nonpolynomial-time algorithms for factoring and computing discrete logarithms that are far better than brute-force search. The goal is merely to give a taste of existing algorithms for these problems, as well as to provide some basic guidance for setting parameters in practice. Our focus is on the high-level ideas, and we consciously do not address many important implementation-level details that would be critical to deal with if these algorithms were to be used in practice. We also concentrate exclusively on *classical* algorithms, and refer the reader elsewhere for a discussion of known *quantum* polynomial-time(!) algorithms for factoring and computing discrete logarithms. (This was another conscious decision, both because we did not want to assume a background in quantum mechanics on the part of the reader, and because quantum computers seem unlikely in the near future.)

The reader may also notice that we only describe algorithms for factoring and computing discrete logarithms, and not algorithms for, say, solving the RSA or decisional Diffie–Hellman problems. Our choice is justified by the facts that the best known algorithms for solving RSA require factoring the modulus, and (in the groups discussed in Sections 8.3.3 and 8.3.4) the best known approaches for solving the decisional Diffie–Hellman problem involve computing discrete logarithms.

9.1 Algorithms for Factoring

Throughout, we assume that $N = pq$ is a product of two distinct primes with $p < q$. We will be most interested in the case when p and q each has the same (known) length n, and so $n = \Theta(\log N)$.

We will frequently use the Chinese remainder theorem along with the notation developed in Section 8.1.5. The Chinese remainder theorem states that

$$\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad \text{and} \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*,$$

with isomorphism given by $f(x) \stackrel{\text{def}}{=} ([x \bmod p], [x \bmod q])$. The fact that f is an isomorphism means, in particular, that it gives a bijection between elements $x \in \mathbb{Z}_N$ and pairs $(x_p, x_q) \in \mathbb{Z}_p \times \mathbb{Z}_q$. We write $x \leftrightarrow (x_p, x_q)$, with $x_p = [x \bmod p]$ and $x_q = [x \bmod q]$, to denote this bijection.

Recall from Section 8.2 that *trial division*—a trivial, brute-force factoring method—finds a factor of a given number N in time $\mathcal{O}(N^{1/2} \cdot \mathsf{polylog}(N))$. (This is an exponential-time algorithm, since the size of the input is the length of the binary representation of N, i.e., $\|N\| = \mathcal{O}(\log N)$.[1]) We cover three factoring algorithms with better performance:

- *Pollard's $p-1$ method* is effective if $p-1$ has only "small" prime factors.
- *Pollard's rho method* applies to arbitrary N. (As such, it is called a *general-purpose* factoring algorithm.) Its running time for N of the form discussed at the beginning of this section is $\mathcal{O}(N^{1/4} \cdot \mathsf{polylog}(N))$. Note this is still *exponential* in n, the length of N.
- The *quadratic sieve algorithm* is a general-purpose factoring algorithm that runs in time *sub-exponential* in the length of N. We give a high-level overview of how this algorithm works, but the details are somewhat complex and are beyond the scope of this book.

[1] Thus, a running time of $N^{\mathcal{O}(1)} = 2^{\mathcal{O}(\|N\|)}$ is exponential, a running time of $2^{o(\log N)} = 2^{o(\|N\|)}$ is sub-exponential, and a running time of $\log^{\mathcal{O}(1)} N = \|N\|^{\mathcal{O}(1)}$ is polynomial.

The fastest known general-purpose factoring algorithm is the *general number field sieve*. Heuristically, this algorithm factors its input N in expected time $2^{\mathcal{O}((\log N)^{1/3}\cdot(\log\log N)^{2/3})}$, which is sub-exponential in the length of N.

9.1.1 Pollard's $p-1$ Algorithm

If $N = pq$ and $p-1$ has *only* "small" prime factors, Pollard's $p-1$ algorithm can be used to efficiently factor N. The basic idea is simple. Let B be an integer for which $(p-1) \mid B$ and $(q-1) \nmid B$; we defer to below the details of how such a B is found. Say $B = \gamma \cdot (p-1)$ for some integer γ. Choose a uniform $x \in \mathbb{Z}_N^*$ and compute $y := [x^B - 1 \bmod N]$. (Note that y can be computed using the efficient exponentiation algorithm from Appendix B.2.3.) Since $1 \leftrightarrow (1,1)$, we have

$$\begin{aligned} y = [x^B - 1 \bmod N] &\leftrightarrow (x_p, x_q)^B - (1,1) \\ &= (x_p^B - 1 \bmod p,\ x_q^B - 1 \bmod q) \\ &= ((x_p^{p-1})^\gamma - 1 \bmod p,\ x_q^B - 1 \bmod q) \\ &= (0, [x_q^B - 1 \bmod q]) \end{aligned}$$

using Theorem 8.14 and the fact that the order of $\mathbb{Z}_p^*$ is $p-1$. We show below that, with high probability, $x_q^B \neq 1 \bmod q$. Assuming this is the case, we have obtained an integer y for which

$$y = 0 \bmod p \quad \text{but} \quad y \neq 0 \bmod q;$$

that is, $p \mid y$ but $q \nmid y$. This, in turn, implies that $\gcd(y, N) = p$. Thus, a simple gcd computation (which can be done efficiently as described in Appendix B.1.2) yields a prime factor of N.

ALGORITHM 9.1
Pollard's $p-1$ algorithm for factoring

Input: Integer N
Output: A non-trivial factor of N

$x \leftarrow \mathbb{Z}_N^*$
$y := [x^B - 1 \bmod N]$
  // B is as in the text
$p := \gcd(y, N)$
if $p \notin \{1, N\}$ **return** p

Let us first argue that the algorithm works (with high probability). Assume $(p-1) \mid B$ but $(q-1) \nmid B$. In that case, as long as $x_q \stackrel{\text{def}}{=} [x \bmod q]$ is a generator of $\mathbb{Z}_q^*$, we have $x_q^B \neq 1 \bmod q$. (This follows from Proposition 8.52.) It remains to analyze the probability that x_q is a generator. Here we rely on

some results proved in Appendix B.3.1. Since q is prime, $\mathbb{Z}_q^*$ is a cyclic group of order $q-1$ that has exactly $\phi(q-1)$ generators (cf. Theorem B.16). If x is chosen uniformly from $\mathbb{Z}_N^*$, then x_q is uniformly distributed in $\mathbb{Z}_q^*$. (This is a consequence of the fact that the Chinese remainder theorem gives a bijection between $\mathbb{Z}_N^*$ and $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.) Thus, the probability that x_q is a generator is $\frac{\phi(q-1)}{q-1} = \Omega(1/\log q) = \Omega(1/n)$ (cf. Theorem B.15). Multiple values of x can be chosen to boost the probability of success.

We are left with the problem of finding B such that $(p-1) \mid B$ but $(q-1) \nmid B$. One possibility is to choose $B = \prod_{i=1}^{k} p_i^{\lfloor n/\log p_i \rfloor}$ for some k, where p_i denotes the i^{th} prime (i.e., $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$) and n is the length of p. (Note that $p_i^{\lfloor n/\log p_i \rfloor}$ is the largest power of p_i that can possibly divide $p-1$.) If $p-1$ can be written as $\prod_{i=1}^{k} p_i^{e_i}$ with $e_i \geq 0$ (that is, if the largest prime factor of $p-1$ is less than p_k), we will have $(p-1) \mid B$. In contrast, if $q-1$ has *any* prime factor larger than p_k, then $(q-1) \nmid B$.

Choosing a larger value for k increases B and so increases the running time of the algorithm (which performs a modular exponentiation to the power B). A larger value of k also makes it more likely that $(p-1) \mid B$, but at the same time makes it less likely that $(q-1) \nmid B$. It is, of course, possible to run the algorithm repeatedly using multiple choices for k.

Pollard's $p-1$ algorithm is thwarted if both $p-1$ and $q-1$ have any large prime factors. (More precisely, the algorithm still works but only for B so large that the algorithm becomes impractical.) If p and q are uniform n-bit primes, then it is unlikely that either $p-1$ or $q-1$ will have only small prime factors. Nevertheless, when generating a modulus $N = pq$ for cryptographic applications, p and q are sometimes chosen to be *strong* primes. (Recall that p is a strong prime if $(p-1)/2$ is also prime.) Selecting p and q in this way is markedly less efficient than simply choosing p and q as *arbitrary* (random) primes. Because better factoring algorithms are available anyway (as we will see below), and due to the observation above, the current consensus is that the added computational cost of generating p and q as strong primes does not yield any appreciable security gains.

9.1.2 Pollard's Rho Algorithm

Pollard's rho algorithm can be used to factor an arbitrary integer $N = pq$; in that sense, it is a *general-purpose* factoring algorithm. Heuristically, the algorithm factors N with constant probability in $\mathcal{O}\left(N^{1/4} \cdot \mathsf{polylog}(N)\right)$ time; this is still exponential, but is a vast improvement over trial division.

The core idea of the approach is to find distinct values $x, x' \in \mathbb{Z}_N^*$ that are equivalent modulo p (i.e., for which $x = x' \bmod p$); call such a pair *good*. Note that for a good pair x, x' it holds that $\gcd(x - x', N) = p$ (since $x \neq x' \bmod N$), so computing the gcd gives a non-trivial factor of N.

How can we find a good pair? Say we choose values $x^{(1)}, \ldots, x^{(k)}$ uniformly from $\mathbb{Z}_N^*$, where $k = 2^{n/2} = \mathcal{O}(\sqrt{p})$. Viewing these in their Chinese-

ALGORITHM 9.2
Pollard's rho algorithm for factoring

Input: Integer N, a product of two n-bit primes
Output: A non-trivial factor of N

$x^{(0)} \leftarrow \mathbb{Z}_N^*$, $x' := x := x^{(0)}$
for $i = 1$ to $2^{n/2}$:
  $x := F(x)$
  $x' := F(F(x'))$
  $p := \gcd(x - x', N)$
  if $p \notin \{1, N\}$ **return** p and **stop**

remaindering representation as $(x_p^{(1)}, x_q^{(1)}), \ldots, (x_p^{(k)}, x_q^{(k)})$, we have that each $x_p^{(i)} \stackrel{\text{def}}{=} [x^{(i)} \bmod p]$ is uniform in $\mathbb{Z}_p^*$. (This follows from bijectivity between $\mathbb{Z}_N^*$ and $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.) Thus, using the birthday bound of Lemma A.16, we see that with high probability there exist distinct i, j with $x_p^{(i)} = x_p^{(j)}$ or, equivalently, $x^{(i)} = x^{(j)} \bmod p$. Moreover, Lemma A.15 shows that $x^{(i)} \neq x^{(j)}$ except with negligible probability. Thus, with high probability we obtain a good pair $x^{(i)}, x^{(j)}$ that can be used to find a non-trivial factor of N, as discussed earlier.

We can generate $k = \mathcal{O}(\sqrt{p}\,)$ uniform elements of $\mathbb{Z}_N^*$ in $\mathcal{O}(\sqrt{p}\,) = \mathcal{O}(N^{1/4})$ time. *Testing* all pairs of elements in order to identify a good pair, however, would require $\binom{k}{2} = \mathcal{O}(k^2) = \mathcal{O}(p) = \mathcal{O}\left(N^{1/2}\right)$ time! (Note that since p is unknown we cannot simply compute $x_p^{(1)}, \ldots, x_p^{(k)}$ explicitly and then sort the $x_p^{(i)}$ to find a good pair. Instead, for all distinct pairs i, j we must compute $\gcd(x^{(i)} - x^{(j)}, N)$ to see whether this gives a non-trivial factor of N.) Without further optimizations, this will be no better than trial division.

Pollard's idea was to use a technique we have seen in Section 5.4.2 in the context of small-space birthday attacks. Specifically, we compute the sequence $x^{(1)}, x^{(2)}, \ldots$ by letting each value be a function of the one before it; i.e., we fix some function $F : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, choose a uniform $x^{(0)} \in \mathbb{Z}_N^*$, and then set $x^{(i)} := F(x^{(i-1)})$ for $i = 1, \ldots, k$. We require F to have the property that if $x = x' \bmod p$, then $F(x) = F(x') \bmod p$; this ensures that once equivalence modulo p occurs, it persists. (A standard choice is $F(x) = [x^2 + 1 \bmod N]$, but any polynomial F will have this property.) If we model F as a random function (which works heuristically), then with high probability there is a good pair in the first k elements of this sequence. Proceeding roughly as in Algorithm 5.9 from Section 5.4.2, we can detect a good pair (if there is one) using only $\mathcal{O}(k)$ gcd computations; see Algorithm 9.2. In addition to improving the running time, Pollard's idea also drastically reduces the amount of memory needed.

9.1.3 The Quadratic Sieve Algorithm

Pollard's rho algorithm is better than trial division, but still runs in exponential time. The *quadratic sieve* algorithm runs in sub-exponential time. It

was the fastest known factoring algorithm until the early 1990s and remains the factoring algorithm of choice for numbers up to about 300 bits long. We describe the general principles of the algorithm but caution the reader that several important details are omitted.

An element $z \in \mathbb{Z}_N^*$ is a *quadratic residue modulo* N if there is an $x \in \mathbb{Z}_N^*$ such that $x^2 = z \bmod N$; in this case, we say that x is a *square root of* z. The following observations serve as our starting point:

- If N is a product of two distinct, odd primes, then every quadratic residue modulo N has exactly four square roots. (See Section 13.4.2.)

- Given x, y with $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$, it is possible to compute a nontrivial factor of N in polynomial time. This is by virtue of the fact that $x^2 = y^2 \bmod N$ implies

$$0 = x^2 - y^2 = (x - y)(x + y) \bmod N,$$

  and so $N \mid (x - y)(x + y)$. However, $N \nmid (x - y)$ and $N \nmid (x + y)$ because $x \neq \pm y \bmod N$. So it must be the case that $\gcd(x - y, N)$ is equal to one of the prime factors of N. (See also Lemma 13.35.)

The quadratic sieve algorithm tries to generate x, y with $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$. A naive way of doing this—which forms the basis of an older factoring algorithm due to Fermat—is to choose an $x \in \mathbb{Z}_N^*$, compute $q := [x^2 \bmod N]$, and then check whether q is a square *over the integers* (i.e., without reduction modulo N). If so, then $q = y^2$ for some integer y and so $x^2 = y^2 \bmod N$. Unfortunately, the probability that $[x^2 \bmod N]$ is a square is so low that this process must be repeated exponentially many times.

A significant improvement is obtained by generating a sequence of values $q_1 := [x_1^2 \bmod N], \ldots$ and identifying a subset of those values whose *product* is a square over the integers. In the quadratic sieve algorithm this is accomplished using the following two steps:

Step 1. Fix some bound B. Say an integer is *B-smooth* if all its prime factors are less than or equal to B. In the first phase of the algorithm, we search for integers of the form $q_i = [x_i^2 \bmod N]$ that are B-smooth and factor them. (Although factoring is hard, finding and factoring B-smooth numbers is feasible when B is small enough.) These $\{x_i\}$ are chosen by trying $x = \sqrt{N} + 1, \sqrt{N} + 2, \ldots$; this ensures a nontrivial reduction modulo N (since $x > \sqrt{N}$) and has the advantage that $q \stackrel{\text{def}}{=} [x^2 \bmod N] = x^2 - N$ is "small" so that q is more likely to be B-smooth.

Let $\{p_1, \ldots, p_k\}$ be the set of prime numbers less than or equal to B. Once we have found and factored the B-smooth $\{q_i\}$ as described above, we have a

set of equations of the form:

$$\begin{aligned} q_1 &= [x_1^2 \bmod N] = \prod_{i=1}^{k} p_i^{e_{1,i}} \\ &\vdots \\ q_\ell &= [x_\ell^2 \bmod N] = \prod_{i=1}^{k} p_i^{e_{\ell,i}}. \end{aligned} \tag{9.1}$$

(Note that the above equations are over the integers.)

Step 2. We next want to find some subset of the $\{q_i\}$ whose product is a square. If we multiply some subset S of the $\{q_i\}$, we see that the result

$$z = \prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}}$$

is a square if and only if the exponent of each prime p_i is even. This suggests that we care about the exponents $\{e_{j,i}\}$ in Equation (9.1) only modulo 2; moreover, we can use linear algebra to find a subset of the $\{q_i\}$ whose "exponent vectors" sum to the 0-vector modulo 2.

In more detail: if we reduce the exponents in Equation (9.1) modulo 2, we obtain the 0/1-matrix Γ given by

$$\begin{pmatrix} \gamma_{1,1} & \gamma_{1,2} & \cdots & \gamma_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{\ell,1} & \gamma_{\ell,2} & \cdots & \gamma_{\ell,k} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} [e_{1,1} \bmod 2] & [e_{1,2} \bmod 2] & \cdots & [e_{1,k} \bmod 2] \\ \vdots & \vdots & \ddots & \vdots \\ [e_{\ell,1} \bmod 2] & [e_{\ell,2} \bmod 2] & \cdots & [e_{\ell,k} \bmod 2] \end{pmatrix}.$$

If $\ell = k + 1$, then Γ has more rows than columns and there must be some nonempty subset S of the rows that sum to the 0-vector modulo 2. Such a subset can be found efficiently using linear algebra. Then:

$$z \stackrel{\text{def}}{=} \prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{(\sum_{j \in S} e_{j,i})/2} \right)^2,$$

using the fact that all the $\left\{ \sum_{j \in S} e_{j,i} \right\}$ are even. Since

$$z = \prod_{j \in S} q_j = \prod_{j \in S} x_j^2 = \left(\prod_{j \in S} x_j \right)^2 \bmod N,$$

we have obtained two square roots (modulo N) of z. Although there is no guarantee that these square roots will enable factorization of N (for reasons

discussed at the beginning of this section), heuristically they do with constant probability. By taking $\ell > k + 1$ we can obtain multiple subsets S with the desired property and try to factor N using each possibility.

Example 9.3
Take $N = 377753$. We have $6647 = [620^2 \bmod N]$, and we can factor 6647 (over the integers, without any modular reduction) as

$$\left[620^2 \bmod N\right] = 6647 = 17^2 \cdot 23.$$

Similarly,

$$\begin{aligned}\left[621^2 \bmod N\right] &= 2^4 \cdot 17 \cdot 29\\ \left[645^2 \bmod N\right] &= 2^7 \cdot 13 \cdot 23\\ \left[655^2 \bmod N\right] &= 2^3 \cdot 13 \cdot 17 \cdot 29.\end{aligned}$$

Letting our subset S include all four of the above equations, we see that

$$\begin{aligned}620^2 \cdot 621^2 \cdot 645^2 \cdot 655^2 &= 2^{14} \cdot 13^2 \cdot 17^4 \cdot 23^2 \cdot 29^2 \bmod N\\ \Rightarrow [620 \cdot 621 \cdot 645 \cdot 655 \bmod N]^2 &= \left[2^7 \cdot 13 \cdot 17^2 \cdot 23 \cdot 29 \bmod N\right]^2 \bmod N\\ \Rightarrow 127194^2 &= 45335^2 \bmod N,\end{aligned}$$

with $127194 \neq \pm 45335 \bmod N$. Computing $\gcd(127194 - 45335, 377753) = 751$ yields a non-trivial factor of N. ♢

Running time. Choosing a larger value of B makes it more likely that a uniform value $q = [x^2 \bmod N]$ is B-smooth; on the other hand, it means we have to work harder to identify and factor B-smooth numbers, and we will have to find more of them (since we require $\ell > k$, where k is the number of primes less than or equal to B). It also means that the matrix Γ will be larger, and so the linear-algebraic step will be slower. Choosing the optimal value of B gives an algorithm that (heuristically, at least) factors N in time $2^{\mathcal{O}(\sqrt{\log N \log\log N})}$. (In fact, the constant term in the exponent can be determined quite precisely.) The important point for our purposes is that this is sub-exponential in the length of N.

9.2 Algorithms for Computing Discrete Logarithms

Let $\mathbb{G}$ be a group of known order q. An instance of the discrete-logarithm problem in $\mathbb{G}$ specifies a *base* $g \in \mathbb{G}$ (which need not be a generator of $\mathbb{G}$) and an element $h \in \langle g \rangle$, the subgroup generated by h; the goal is to find x

such that $g^x = h$. (See Section 8.3.2.) The solution x is called the *discrete logarithm of h with respect to g*. A trivial brute-force search for x can be done in time $|\langle g \rangle| \leq q$ (by simply trying all possible values), and so we will only be interested in algorithms whose running time is better than this.

Algorithms for solving the discrete-logarithm problem fall into two categories: those that are *generic* and apply to arbitrary groups, and those that are tailored to work for some *specific* class of groups. We begin by discussing three generic algorithms:

- When the group order q is not prime and the factorization of q is known, or easy to determine, the *Pohlig–Hellman algorithm* reduces the problem of finding discrete logarithms in $\mathbb{G}$ to that of finding discrete logarithms in prime-order *subgroups* of $\mathbb{G}$. Roughly speaking, the effect is that the complexity of solving the discrete logarithm in a group of order q is no greater than the complexity of solving the discrete logarithm in a group of order q', where q' is the largest prime dividing q. This explains the preference for using prime-order groups (cf. Section 8.3.2).

- The *baby-step/giant-step* method, due to Shanks, computes the discrete logarithm in a group of order q in time $\mathcal{O}(\sqrt{q} \cdot \mathsf{polylog}(q))$ and storing $\mathcal{O}(\sqrt{q})$ group elements.

- *Pollard's rho algorithm* also enables computation of discrete logarithms in time $\mathcal{O}(\sqrt{q} \cdot \mathsf{polylog}(q))$, but using *constant* memory. It can be viewed as exploiting the connection between the discrete-logarithm problem and collision-resistant hashing that we have seen in Section 8.4.2. We describe a different algorithm that more clearly illustrates this connection.

It can be shown that the time complexity of the latter two algorithms is *optimal* as far as generic algorithms are concerned. Thus, to have any hope of doing better we must look at algorithms for specific groups that exploit the *representation* of group elements in those groups, where by "representation" we mean the way group elements are encoded as bit-strings.

This point bears some discussion. From a mathematical point of view, any two cyclic groups of the same order are isomorphic, meaning that the groups are identical up to a "renaming" of the group elements. From a computational/algorithmic point of view, however, this "renaming" can have a significant impact. For example, consider the cyclic group $\mathbb{Z}_q$ of integers $\{0, \ldots, q-1\}$ under *addition* modulo q. Computing discrete logarithms in this group is trivial. Say we are given $g, h \in \mathbb{Z}_q$ with g a generator (so $g \neq 0$), and we want to find x such that $x \cdot g = h \bmod q$. Since $g \neq 0$ we have $\gcd(g, q) = 1$ and so g has a multiplicative inverse g^{-1} modulo q. Moreover, g^{-1} can be computed efficiently, as described in Appendix B.2.2. But then $x = h \cdot g^{-1} \bmod q$ is the desired solution. Note that, formally, x here denotes an integer and not a group element—after all, the group operation is addition, not multiplication. Nevertheless, in solving the discrete-logarithm

problem in $\mathbb{Z}_q$ we can make use of the fact that another operation (namely, multiplication) can be defined on the elements of that group.

Turning to groups with cryptographic significance, we focus our attention on (subgroups of) $\mathbb{Z}_p^*$ for p prime. (See Section 8.3.3.) We give a high-level overview of the *index calculus algorithm* for solving the discrete-logarithm problem in such groups in sub-exponential time. Currently, the best known algorithm for this class of groups is the *number field sieve*,[2] which heuristically runs in time $2^{\mathcal{O}((\log p)^{1/3} \cdot (\log \log p)^{2/3})}$. Sub-exponential algorithms for computing discrete logarithms in multiplicative subgroups of finite fields of large characteristic are also known. In early 2013, significant advances were made in algorithms for computing discrete logarithms in multiplicative subgroups of finite fields of *small* characteristic; it seems prudent to avoid using such groups for cryptographic applications.

Importantly, no sub-exponential algorithms are known for computing discrete logarithms in certain elliptic-curve groups. This means that for a given security level, we can use groups of smaller order when working in elliptic-curve groups as compared to, say, working in $\mathbb{Z}_p^*$, resulting in cryptographic schemes with better asymptotic efficiency.

9.2.1 The Pohlig–Hellman Algorithm

The Pohlig–Hellman algorithm can be used to speed up the computation of discrete logarithms in a group $\mathbb{G}$ when any non-trivial factors of the group order q are known. Recall that the order of an element g, which we denote here by $\mathsf{ord}(g)$, is the smallest positive integer i for which $g^i = 1$. We will need the following lemma:

LEMMA 9.4 *Let* $\mathsf{ord}(g) = q$*, and say* $p \mid q$*. Then* $\mathsf{ord}(g^p) = q/p$*.*

PROOF Since $(g^p)^{q/p} = g^q = 1$, the order of g^p is at most q/p. Let $i > 0$ be such that $(g^p)^i = 1$. Then $g^{pi} = 1$ and, since q is the order of g, we must have $pi \geq q$ or equivalently $i \geq q/p$. The order of g^p is thus exactly q/p. ■

We will also use a generalization of the Chinese remainder theorem: if $q = \prod_{i=1}^{k} q_i$ and $\gcd(q_i, q_j) = 1$ for all $i \neq j$ then

$$\mathbb{Z}_q \simeq \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k} \quad \text{and} \quad \mathbb{Z}_q^* \simeq \mathbb{Z}_{q_1}^* \times \cdots \times \mathbb{Z}_{q_k}^*.$$

(This can be proved by induction on k, using the basic Chinese remainder theorem for $k = 2$.) Moreover, by an extension of the algorithm in Section 8.1.5

[2] It is no accident that the algorithm's name and its running time are similar to those of the general number field sieve for factoring, since the algorithms share many of the same underlying steps.

it is possible to convert efficiently between the representation of an element as an element of $\mathbb{Z}_q$ and its representation as an element of $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k}$.

We now describe the Pohlig–Hellman algorithm. We are given a generator g and an element h and wish to find an x such that $g^x = h$. Say a factorization $q = \prod_{i=1}^{k} q_i$ is known with the $\{q_i\}$ pairwise relatively prime. (This need not be the complete prime factorization of q.) We know that

$$\left(g^{q/q_i}\right)^x = (g^x)^{q/q_i} = h^{q/q_i} \text{ for } i = 1, \ldots, k. \tag{9.2}$$

Letting $g_i \stackrel{\text{def}}{=} g^{q/q_i}$ and $h_i \stackrel{\text{def}}{=} h^{q/q_i}$, we thus have k instances of a discrete-logarithm problem in k *smaller* groups. Specifically, each problem $g_i^x = h_i$ is in a subgroup of size $\mathsf{ord}(g_i) = q_i$ (by Lemma 9.4). (Note that each such problem only determines $[x \bmod q_i]$; this follows from Proposition 8.53.)

We can solve each of the k resulting instances using any algorithm for solving the discrete-logarithm problem. Solving these instances gives a set of answers $\{x_i\}_{i=1}^{k}$, with $x_i \in \mathbb{Z}_{q_i}$, for which $g_i^{x_i} = h_i = g_i^x$. Proposition 8.53 implies that $x = x_i \bmod q_i$ for all i. By the generalized Chinese remainder theorem discussed earlier, the constraints

$$\begin{aligned} x &= x_1 \bmod q_1 \\ &\vdots \\ x &= x_k \bmod q_k \end{aligned}$$

uniquely determine x modulo q, and the desired solution x can be efficiently reconstructed from the $\{x_i\}$.

Example 9.5

Consider the problem of computing discrete logarithms in $\mathbb{Z}_{31}^*$, a group of order $q = 30 = 5 \cdot 3 \cdot 2$. Say $g = 3$ and $h = 26 = g^x$ with x unknown. We have:

$$\begin{aligned} (g^{30/5})^x = h^{30/5} &\Rightarrow (3^6)^x = 26^6 \Rightarrow 16^x = 1 \\ (g^{30/3})^x = h^{30/3} &\Rightarrow (3^{10})^x = 26^{10} \Rightarrow 25^x = 5 \\ (g^{30/2})^x = h^{30/2} &\Rightarrow (3^{15})^x = 26^{15} \Rightarrow 30^x = 30. \end{aligned}$$

(All the above equations are modulo 31.) We have $\mathsf{ord}(16) = 5$, $\mathsf{ord}(25) = 3$, and $\mathsf{ord}(30) = 2$. Solving each equation, we obtain

$$x = 0 \bmod 5, \quad x = 2 \bmod 3, \quad \text{and} \quad x = 1 \bmod 2,$$

and so $x = 5 \bmod 30$. Indeed, $3^5 = 26 \bmod 31$. ♢

If q has (known) prime factorization $q = \prod_{i=1}^{k} p_i^{e_i}$ then, by using the Pohlig–Hellman algorithm, the time to compute discrete logarithms in a group of order q is dominated by the computation of a discrete logarithm in a subgroup of size $\max_i\{p_i^{e_i}\}$. This can be further reduced to computation of a discrete logarithm in a subgroup of size $\max_i\{p_i\}$; see Exercise 9.5.

9.2.2 The Baby-Step/Giant-Step Algorithm

The baby-step/giant-step algorithm computes discrete logarithms in a group of order q using $\mathcal{O}(\sqrt{q})$ group operations. The idea is simple. Given a generator $g \in \mathbb{G}$, we can imagine the powers of g as forming a cycle

$$1 = g^0,\ g^1,\ g^2,\ \ldots,\ g^{q-2},\ g^{q-1},\ g^q = 1.$$

We know that h must lie somewhere in this cycle. Computing all the points in this cycle would take $\Omega(q)$ time. Instead, we "mark off" the cycle at intervals of size $t \stackrel{\text{def}}{=} \lfloor \sqrt{q} \rfloor$; more precisely, we compute and store the $\lfloor q/t \rfloor + 1 = \mathcal{O}(\sqrt{q})$ elements

$$g^0, g^t, g^{2t}, \ldots, g^{\lfloor q/t \rfloor \cdot t}.$$

(These are the "giant steps.") Note that the gap between any consecutive "marks" is at most t. Furthermore, we know that $h = g^x$ lies in one of these gaps. Thus, if we next take "baby steps" and compute the t elements

$$h \cdot g^1,\ \ldots,\ h \cdot g^t,$$

each of which corresponds to a "shift" of h, we know that one of these values will be equal to one of the points we have marked off. Say we find $h \cdot g^i = g^{k \cdot t}$. We can then easily compute $\log_g h := [(kt - i) \bmod q]$. Pseudocode for this algorithm follows.

ALGORITHM 9.6
The baby-step/giant-step algorithm

Input: Elements $g, h \in \mathbb{G}$; the order q of $\mathbb{G}$
Output: $\log_g h$

$t := \lfloor \sqrt{q} \rfloor$
for $i = 0$ to $\lfloor q/t \rfloor$:
  compute $g_i := g^{i \cdot t}$
sort the pairs (i, g_i) by their second component
for $i = 1$ to t:
  compute $h_i := h \cdot g^i$
  if $h_i = g_k$ for some k, **return** $[kt - i \bmod q]$

The algorithm requires $\mathcal{O}(\sqrt{q})$ exponentiations/multiplications in $\mathbb{G}$. (In fact, other than the first value $g_1 = g^t$, each value g_i can be computed using a single multiplication as $g_i := g_{i-1} \cdot g_1$. Similarly, each h_i can be computed as $h_i := h_{i-1} \cdot g$.) Sorting the $\mathcal{O}(\sqrt{q})$ pairs $\{(i, g_i)\}$ takes time $\mathcal{O}(\sqrt{q} \cdot \log q)$, and we can then use binary search to check if each h_i is equal to some g_k in time $\mathcal{O}(\log q)$. The overall algorithm thus runs in time $\mathcal{O}(\sqrt{q} \cdot \mathsf{polylog}(q))$.

Example 9.7
We show an application of the algorithm in the cyclic group $\mathbb{Z}_{29}^*$ of order $q = 29 - 1 = 28$. Take $g = 2$ and $h = 17$. We set $t = 5$ and compute:

$$2^0 = 1, \;\; 2^5 = 3, \;\; 2^{10} = 9, \;\; 2^{15} = 27, \;\; 2^{20} = 23, \;\; 2^{25} = 11.$$

(It should be understood that all operations are in $\mathbb{Z}_{29}^*$.) Then compute:

$$17 \cdot 2^1 = 5, \quad 17 \cdot 2^2 = 10, \quad 17 \cdot 2^3 = 20, \quad 17 \cdot 2^4 = 11,$$

and notice that $17 \cdot 2^4 = 11 = 2^{25}$. We thus have $\log_2 17 = 25 - 4 = 21$. $\diamondsuit$

9.2.3 Discrete Logarithms from Collisions

A drawback of the baby-step/giant-step algorithm is that it uses a large amount of memory, as it requires storage of $\mathcal{O}(\sqrt{q})$ points. We can obtain an algorithm that uses constant memory—and has the same asymptotic running time—by exploiting the connection between the discrete-logarithm problem and collision-resistant hashing shown in Section 8.4.2, and recalling the small-space birthday attack for finding collisions from Section 5.4.2.

We describe the high-level idea. Fix a base $g \in \mathbb{G}$ and some element $h \in \langle g \rangle$. Recall from the results of Section 8.4.2 that if we define the hash function $H_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ by $H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}$, then finding a collision in $H_{g,h}$ implies the ability to compute $\log_g h$. (See the proof of Theorem 8.79.) We have thus reduced the problem of computing $\log_g h$ to that of finding a collision in a hash function, something we know how to do in time $\mathcal{O}(\sqrt{|\mathbb{G}|}) = \mathcal{O}(\sqrt{q})$ using a birthday attack! Moreover, a small-space birthday attack will give a collision in the same time and constant space.

It only remains to address a few technical details. One is that the small-space birthday attack described in Section 5.4.2 assumes that the range of the hash function is a subset of its domain; that is not the case here, and in fact (depending on the representation being used for elements of $\mathbb{G}$) it could even be that $H_{g,h}$ is not compressing. A second issue is that the analysis in Section 5.4.2 treated the hash function as a random function, whereas $H_{g,h}$ has a significant amount of algebraic structure.

Pollard's rho algorithm provides one way to deal with the above issues. We describe a different algorithm that can be viewed as a more direct implementation of the above ideas. (In practice, Pollard's algorithm would be more efficient, although both algorithms use only $\mathcal{O}(\sqrt{q})$ group operations.) Let $F : \mathbb{G} \to \mathbb{Z}_q \times \mathbb{Z}_q$ denote a cryptographic hash function obtained by, e.g., a suitable modification of SHA-1. Define $H : \mathbb{G} \to \mathbb{G}$ by $H(k) \stackrel{\text{def}}{=} H_{g,h}(F(k))$. We can use Algorithm 5.9, with natural modifications, to find a collision in H using $\mathcal{O}(\sqrt{|\mathbb{G}|}) = \mathcal{O}(\sqrt{q})$ evaluations of H in expectation (and constant memory). With overwhelming probability, this yields a collision in $H_{g,h}$. You are asked to flesh out the details in Exercise 9.6.

It is interesting to observe here that a security proof based on the hardness of the discrete-logarithm problem—namely, that it implies a collision-resistant hash function—leads to a better algorithm for solving that same problem! A little reflection should convince us that this is not surprising: a proof by reduction demonstrates that an attack on some construction (in this case, finding collisions in the hash function) directly yields an attack on the underlying assumption (here, the hardness of the discrete-logarithm problem), which is exactly the property the above algorithm exploits.

9.2.4 The Index Calculus Algorithm

We conclude with a brief look at the (non-generic) *index calculus algorithm* for computing discrete logarithms in the cyclic group $\mathbb{Z}_p^*$ (for p prime). In contrast to the preceding (generic) algorithms, this approach has sub-exponential running time. The algorithm bears some resemblance to the quadratic sieve algorithm introduced in Section 9.1.3, and we assume readers are familiar with the discussion there. As in that case, we discuss the main ideas of the index calculus method but leave a detailed analysis outside the scope of our treatment. Also, some simplifications are introduced to clarify the presentation.

The index calculus method uses a two-step process. Importantly, the first step requires knowledge only of the modulus p and the base g and so it can be run as a preprocessing step before h—the value whose discrete logarithm we wish to compute—is known. For the same reason, it suffices to run the first step only once in order to solve multiple instances of the discrete-logarithm problem (as long as all those instances share the same p and g).

Step 1. Fix some bound B, and let $\{p_1, \ldots, p_k\}$ be the set of prime numbers less than or equal to B. In this step, we find $\ell \geq k$ distinct values $x_1, \ldots, x_\ell \in \mathbb{Z}_{p-1}$ for which $g_i \stackrel{\text{def}}{=} [g^{x_i} \bmod p]$ is B-smooth. This is done by simply choosing uniform $\{x_i\}$ until suitable values are found.

Factoring the resulting B-smooth numbers, we have the ℓ equations:

$$\begin{aligned} g^{x_1} &= \prod_{i=1}^{k} p_i^{e_{1,i}} \bmod p \\ &\vdots \\ g^{x_\ell} &= \prod_{i=1}^{k} p_i^{e_{\ell,i}} \bmod p\,. \end{aligned}$$

Taking discrete logarithms, we can transform these into the linear equations

$$x_1 = \sum_{i=1}^{k} e_{1,i} \cdot \log_g p_i \mod (p-1) \tag{9.3}$$

$$\vdots$$

$$x_\ell = \sum_{i=1}^{k} e_{\ell,i} \cdot \log_g p_i \mod (p-1). \tag{9.4}$$

Note that the $\{x_i\}$ and the $\{e_{i,j}\}$ are known, while the $\{\log_g p_i\}$ are unknown.

Step 2. Now we are given an element h and want to compute $\log_g h$. Here, we find a value $x \in \mathbb{Z}_{p-1}$ for which $[g^x \cdot h \bmod p]$ is B-smooth. (Once again, this is done simply by choosing x uniformly.) Say

$$g^x \cdot h = \prod_{i=1}^{k} p_i^{e_i} \bmod p$$

$$\Rightarrow x + \log_g h = \sum_{i=1}^{k} e_i \cdot \log_g p_i \mod (p-1),$$

where x and the $\{e_i\}$ are known. Combined with Equations (9.3)–(9.4), we have $\ell + 1 \geq k + 1$ linear equations in the $k + 1$ unknowns $\{\log_g p_i\}_{i=1}^{k}$ and $\log_g h$. Using linear-algebraic[3] methods (and assuming the system of equations is not under-defined), we can solve for each of the unknowns and in particular obtain the desired solution $\log_g h$.

Example 9.8

Let $p = 101$, $g = 3$, and $h = 87$. We have $[3^{10} \bmod 101] = 65 = 5 \cdot 13$. Similarly, $[3^{12} \bmod 101] = 80 = 2^4 \cdot 5$ and $[3^{14} \bmod 101] = 13$. We thus have the linear equations

$$\begin{aligned} 10 &= \log_3 5 + \log_3 13 \mod 100 \\ 12 &= 4 \cdot \log_3 2 + \log_3 5 \mod 100 \\ 14 &= \log_3 13 \mod 100. \end{aligned}$$

We also have $3^5 \cdot 87 = 32 = 2^5 \bmod 101$, or

$$5 + \log_3 87 = 5 \cdot \log_3 2 \mod 100. \tag{9.5}$$

Adding the second and third equations and subtracting the first, we derive $4 \cdot \log_3 2 = 16 \bmod 100$. This doesn't determine $\log_3 2$ uniquely (since 4 is

[3]Technically, things are slightly more complicated since the linear equations are all modulo $p - 1$, which is not prime. Nevertheless, there exist techniques for dealing with this.

not invertible modulo 100), but it does tell us that $\log_3 2 = 4, 29, 54$, or 79 (cf. Exercise 9.3). Trying all possibilities gives $\log_3 2 = 29$. Plugging this into Equation (9.5) gives $\log_3 87 = 40$. ♢

Running time. Choosing a larger value of B makes it more likely that a uniform value in $\mathbb{Z}_p^*$ is B-smooth; however, it means we will have to work harder to identify and factor B-smooth numbers, and we will have to find more of them. Because the system of equations will be larger, solving the system will take longer. Choosing the optimal value of B gives an algorithm that (heuristically, at least) computes discrete logarithms in $\mathbb{Z}_p^*$ in time $2^{\mathcal{O}(\sqrt{\log p \log\log p})}$. (In fact, the constant term in the exponent can be determined quite precisely.) The important point for our purposes is that this is sub-exponential in the length of p.

9.3 Recommended Key Lengths

Knowledge of the best available algorithms for solving various cryptographic problems is essential for determining an appropriate key length to achieve some desired level of security. The following table summarizes the key lengths currently recommended by the US National Institute of Standards and Technology[4] (NIST) [13]:

	RSA	Discrete Logarithm	
Effective Key Length	**Modulus Length**	**Order-q Subgroup of $\mathbb{Z}_p^*$**	**Elliptic-Curve Group Order q**
112	2048	p: 2048, q: 224	224
128	3072	p: 3072, q: 256	256
192	7680	p: 7680, q: 384	384
256	15360	p: 15360, q: 512	512

All figures in the table are measured in bits. The "effective key length" is a value n such that the best known algorithm for solving a problem takes time roughly 2^n; i.e., the computational difficulty of solving a problem is approximately equivalent to that of performing a brute-force search against a symmetric-key scheme with an n-bit key, or the time to find collisions in a hash function with a $2n$-bit output length. NIST deems a 112-bit effective key length acceptable for security until the year 2030, but recommends 128-bit or higher key lengths for applications where security is required beyond then.

[4] Other groups have made their own recommendations; see `http://keylength.com`.

Given what we have learned in this chapter, it is instructive to look more closely at some of the numbers in the table. The first thing we may notice is that elliptic-curve groups can be used to realize any given level of security with smaller parameters than for RSA or subgroups of $\mathbb{Z}_p^*$. This is simply because no sub-exponential algorithms are known for solving the discrete-logarithm problem in such groups (when chosen appropriately). Achieving n-bit security, however, requires an elliptic-curve group whose order q is $2n$-bits long. This is a consequence of the generic algorithms we have seen in this chapter, which solve the discrete-logarithm problem (in any group) in time $\mathcal{O}(\sqrt{q}) \approx 2^n$.

Turning to the case of $\mathbb{Z}_p^*$ we see that here, too, a $2n$-bit value of q is needed for n-bit security. The length of p, however, must be significantly larger, because the number field sieve can be used to compute discrete logarithms in $\mathbb{Z}_p^*$ in time sub-exponential in the length of p. (That is, p and q are chosen such that the running time of the number field sieve, which depends on the length of p, and the running time of a generic algorithm, which depends on the length of q, will be approximately equal.) The practical ramifications of this are that, for any desired security level, elliptic-curve cryptosystems can use significantly smaller parameters, with asymptotically faster group operations for the honest parties, than cryptosystems based on subgroups of $\mathbb{Z}_p^*$.

References and Additional Reading

Pollard's $p-1$ algorithm was published in 1974 [139], and his rho method for factoring was described the following year [140]. The quadratic sieve algorithm is due to Pomerance [142], based on earlier ideas of Dixon [60].

The baby-step/giant-step algorithm is due to Shanks [153]. The Pohlig–Hellman algorithm was published in 1978 [138], as was Pollard's rho algorithm for computing discrete logarithms [141]. The index calculus algorithm as we have described it is by Adleman [4].

The texts by Wagstaff [173], Shoup [159], Crandall and Pomerance [51], Joux [96], and Galbraith [69] provide further information on algorithms for factoring and computing discrete logarithms, including descriptions of the (general) number field sieve. Very recently, an improved algorithm for the discrete-logarithm problem in finite fields of small characteristic was announced [11].

Lower bounds on the running time of generic algorithms for computing discrete logarithms, which asymptotically match the running times of the algorithms described here, were given by Nechaev [133] and Shoup [155].

Lenstra and Verheul [113] provide a comprehensive discussion of how known algorithms for factoring and computing discrete logarithms affect the choice of cryptographic parameters in practice.

Exercises

9.1 In order to speed up the key-generation algorithm for RSA, it has been suggested to generate a prime by generating many small random primes, multiplying them together, and adding one (of course, then checking that the result is prime). Ignoring the question of the probability that such a value really is prime, what do you think of this method?

9.2 In an execution of Algorithm 9.2, define $x^{(i)} \stackrel{\text{def}}{=} F^{(i)}(x^{(0)})$. Show that if, in a given execution of the algorithm, there exist $i, j \leq 2^{n/2}$ such that $x^{(i)} \neq x^{(j)}$ but $x^{(i)} = x^{(j)} \bmod p$, then that execution of the algorithm outputs p. (The analysis is a little different from the analysis of Algorithm 5.9, since the algorithms—and their goals—are slightly different.)

9.3 (a) Show that if $ab = c \bmod N$ and $\gcd(b, N) = d$, then:

i. $d \mid c$;

ii. $a \cdot (b/d) = (c/d) \bmod (N/d)$; and

iii. $\gcd(b/d, N/d) = 1$.

(b) Describe how to use the above to compute discrete logarithms in $\mathbb{Z}_N$ efficiently even when the base g is not a generator of $\mathbb{Z}_N$.

9.4 Here we show how to solve the discrete-logarithm problem in a cyclic group of order $q = p^e$ in time $\mathcal{O}(\mathsf{polylog}(q) \cdot \sqrt{p}\,)$. Given as input a generator g of order $q = p^e$ and a value h, we want to compute $x = \log_g h$.

(a) Show how to compute $[x \bmod p\,]$ in time $\mathcal{O}(\mathsf{polylog}(q) \cdot \sqrt{p}\,)$.
Hint: Solve the equation
$$\left(g^{p^{e-1}}\right)^{x_0} = h^{p^{e-1}}$$
and use the same ideas as in the Pohlig–Hellman algorithm.

(b) Say $x = x_0 + x_1 \cdot p + \cdots + x_{e-1} \cdot p^{e-1}$ with $0 \leq x_i < p$. In the previous step we determined x_0. Show how to compute in $\mathsf{polylog}(q)$ time a value h_1 such that $(g^p)^{x_1 + x_2 \cdot p + \cdots + x_{e-1} \cdot p^{e-2}} = h_1$.

(c) Use recursion to obtain the claimed running time for the original problem. (Note that $e = \mathcal{O}(\log q)$.)

9.5 Let q have prime factorization $q = \prod_{i=1}^{k} p_i^{e_i}$. Using the result from the previous problem, show a modification of the Pohlig–Hellman algorithm that solves the discrete-logarithm problem in a group of order q in time $\mathcal{O}\left(\mathsf{polylog}(q) \cdot \sum_{i=1}^{k} e_i \sqrt{p_i}\right) = \mathcal{O}\left(\mathsf{polylog}(q) \cdot \max_i\{\sqrt{p_i}\}\right)$.

9.6 Give pseudocode for the small-space algorithm for computing discrete logarithms described in Section 9.2.3, and give a heuristic analysis of the probability with which it succeeds.

Chapter 10

Key Management and the Public-Key Revolution

10.1 Key Distribution and Key Management

In Chapters 1–7 we have seen how private-key cryptography can be used to ensure secrecy and integrity for two parties communicating over an insecure channel, assuming the two parties are in possession of a shared, secret key. The question we have deferred since Chapter 1, however, is:

How can the parties share a secret key in the first place?

Clearly, the key cannot simply be sent over the public communication channel because an eavesdropping adversary would then be able to observe it *en route*. Some other mechanism must be used instead.

In some situations, the parties may have access to a secure channel that they can use to reliably share a secret key. One common example is when the two parties are co-located at some time, at which point they can share a key. Alternatively, the parties might be able to use a trusted courier service as a secure channel. We stress that the fact that the parties can share a key—and thus must have access to a secure channel at some point—does *not* make private-key cryptography useless: in the first example, the parties have a secure channel at one point in time but not indefinitely; in the second example, utilizing the secure channel might be slower and more costly than communicating over an insecure channel.

The above approaches have been used to share keys in government, diplomatic, and military contexts. For example, the "red phone" connecting Moscow and Washington in the 1960s was encrypted using a one-time pad, with keys shared by couriers who flew from one country to the other carrying briefcases full of print-outs. Such approaches can also be used in corporations, e.g., to set up a shared key between a central database and a new employee on his/her first day of work. (We return to this example in the next section.)

Relying on a secure channel to distribute keys, however, does not work well in many other situations. For example, consider a large, multinational corporation in which *every pair* of employees might need the ability to communicate securely, with their communication protected from other employees as well.

It will be inconvenient, to say the least, for each pair of employees to meet so they can securely share a key; for employees working in different cities, this may even be impossible. Even if the current set of employees could somehow share keys with each other, it would be impractical for them to share keys with new employees who join after this initial sharing is done.

Assuming these N employees are somehow able to securely share keys with each other, another significant drawback is that each employee will have to manage and store $N - 1$ secret keys (one for each other employee). In fact, this may significantly under-count the number of keys stored by each user, because employees may also need keys to communicate securely with remote resources such as databases, servers, printers, and so on. The proliferation of so many secret keys is a significant logistical problem. Moreover, all these keys must be stored *securely*. The more keys there are, the harder it is to protect them, and the higher the chance of some keys being stolen by an attacker. Computer systems are often infected by viruses, worms, and other forms of malicious software which can steal secret keys and send them quietly over the network to an attacker. Thus, storing keys on employees' personal computers is not always a safe solution.

To be clear, potential compromise of secret keys is always a concern, irrespective of the number of keys each party holds. When only a few keys need to be stored, however, there are good solutions available for dealing with this threat. A typical solution today is to store keys on *secure hardware* such as a smartcard. A smartcard can carry out cryptographic computations using the stored secret keys, ensuring that these keys never make their way onto users' personal computers. If designed properly, the smartcard can be much more resilient to attack than a personal computer—for example, it typically cannot be infected by malware—and so offers a good means of protecting users' secret keys. Unfortunately, smartcards are typically quite limited in memory and so cannot store hundreds (or thousands) of keys.

The concerns outlined above can all be addressed—in principle, even if not in practice—in "closed" organizations consisting of a well-defined population of users, all of whom are willing to follow the same policies for distributing and storing keys. They break down, however, in "open systems" where users have transient interactions, cannot arrange a physical meeting, and may not even be aware of each other's existence until the time they want to communicate. This is, in fact, a more common situation than one might at first realize: consider using encryption to send credit-card information to an Internet merchant from whom you have not previously purchased anything, or sending email to someone whom you have never met in person. In such cases, private-key cryptography alone simply does not provide a solution, and we must look further for adequate solutions.

To summarize, there are at least three distinct problems related to the use of private-key cryptography. The first is that of *key distribution*; the second is that of *storing and managing large numbers of secret keys*; the third is the *inapplicability of private-key cryptography to open systems.*

10.2 A Partial Solution: Key-Distribution Centers

One way to address some of the concerns from the previous section is to use a *key-distribution center (KDC)* to establish shared keys. Consider again the case of a large corporation where all pairs of employees must be able to communicate securely. In such a setting, we can leverage the fact that all employees may *trust* some entity—say, the system administrator—at least with respect to the security of work-related information. This trusted entity can then act as a KDC and help all the employees share pairwise keys.

When a new employee joins, the KDC can share a key with that employee (in person, in a secure location) as part of that employee's first day of work. At the same time, the KDC could also distribute shared keys between that employee and all existing employees. That is, when the ith employee joins, the KDC could (in addition to sharing a key between itself and this new employee) generate $i-1$ keys $k_1, \ldots, k_{i-1}$, give these keys to the new employee, and then send key k_j to the jth existing employee by encrypting it using the key that employee already shares with the KDC. Following this, the new employee shares a key with every other employee (as well as with the KDC).

A better approach, which avoids requiring employees to store and manage multiple keys, is to utilize the KDC in an *online* fashion to generate keys "on demand" whenever two employees wish to communicate securely. As before, the KDC will share a (different) key with each employee, something that can be done securely on an employee's first day of work. Say the KDC shares key k_A with employee Alice, and k_B with employee Bob. At some later time, when Alice wishes to communicate securely with Bob, she can simply send the message ``I, Alice, want to talk to Bob'' to the KDC. (If desired, this message can be authenticated using the key shared by Alice and the KDC.) The KDC then chooses a new, random key—called a *session key*—and sends this key k to Alice encrypted using k_A, and to Bob encrypted using k_B. (This protocol is too simplistic to be used in practice; see further discussion below.) Once Alice and Bob both recover this session key, they can use it to communicate securely. When they are done with their conversation, they can (and should) erase the session key because they can always contact the KDC again if they wish to communicate at some later time.

Consider the advantages of this approach:

1. Each employee needs to store only *one* long-term secret key (namely, the one they share with the KDC). Employees still need to manage and store session keys, but these are short-term keys that are erased once a communication session concludes.

   The KDC needs to store many long-term keys. However, the KDC can be kept in a secure location and be given the highest possible protection against network attacks.

2. When an employee joins the organization, all that must be done is to set up a key between this employee and the KDC. No other employees need to update the set of keys they hold.

Thus, KDCs can alleviate two of the problems we have seen with regard to private-key cryptography: they can simplify key distribution (since only one new key must be shared when an employee joins, and it is reasonable to assume a secure channel between the KDC and that employee on their first day of work), and can reduce the complexity of key storage (since each employee only needs to store a single key). KDCs go a long way toward making private-key cryptography practical in large organizations where there is a single entity who is trusted by everyone.

There are, however, some drawbacks to relying on KDCs:

1. A successful attack on the KDC will result in a complete break of the system: an attacker can compromise all keys and subsequently eavesdrop on all network traffic. This makes the KDC a high-value target. Note that even if the KDC is well-protected against external attacks, there is always the possibility of insider attacks by employees who have access to the KDC (for example, the IT manager).

2. The KDC is a single point of failure: if the KDC is down, secure communication is temporarily impossible. If employees are constantly contacting the KDC and asking for session keys to be established, the load on the KDC can be very high, thereby increasing the chances that it may fail or be slow to respond.

A simple solution to the second problem is to replicate the KDC. This works (and is done in practice), but also means that there are now more points of attack on the system. Adding more KDCs also makes it more difficult to add new employees, since updates must be securely propagated to every KDC.

Protocols for key distribution using a KDC. There are a number of protocols in the literature for secure key distribution using a KDC. We mention in particular the *Needham–Schroeder protocol*, which forms the core of *Kerberos*, an important and widely used service for performing authentication and supporting secure communication. (Kerberos is used in many universities and corporations, and is the default mechanism for supporting secure networked authentication and communication in Windows and many UNIX systems.) We only highlight one feature of this protocol. When Alice contacts the KDC and asks to communicate with Bob, the KDC does not send the encrypted session key to both Alice and Bob as we have described earlier. Instead, the KDC sends to Alice the session key encrypted under Alice's key *in addition to* the session key encrypted under Bob's key. Alice then forwards the second ciphertext to Bob as in Figure 10.1. The second ciphertext is sometimes called a *ticket*, and can be viewed as a credential that allows Alice to talk to Bob (and allows Bob to be assured that he is talking to Alice). Indeed,

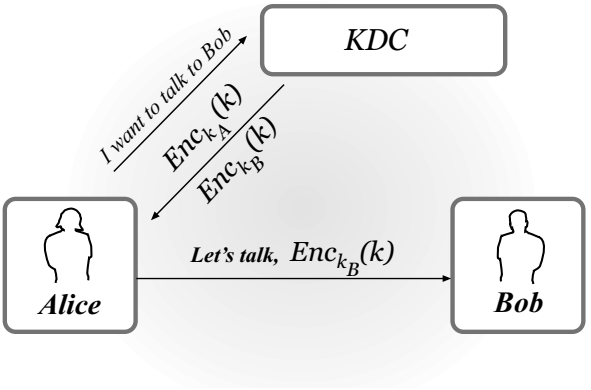

FIGURE 10.1: A general template for key-distribution protocols.

although we have not stressed this point in our discussion, a KDC-based approach can provide a useful means of performing authentication as well. Note also that Alice and Bob need not both be users; Alice might be a user and Bob a resource such as a server, a remote disk, or a printer.

The protocol was designed in this way to reduce the load on the KDC. In the protocol as described, the KDC does not need to initiate a second connection to Bob, and need not worry whether Bob is on-line when Alice initiates the protocol. Moreover, if Alice retains the ticket (and her copy of the session key), then she can re-initiate secure communication with Bob by simply re-sending the ticket to Bob, without the involvement of the KDC at all. (In practice, tickets expire and eventually need to be renewed. But a session could be re-established within some acceptable time period.)

We conclude by noting that in practice the key that Alice shares with the KDC might be a short, easy-to-memorize password. In this case, many additional security problems arise that must be dealt with. We have also been implicitly assuming an attacker who only passively eavesdrops, rather than one who might actively try to interfere with the protocol. We refer the interested reader to the references at the end of this chapter for more information about how such issues can be addressed.

10.3 Key Exchange and the Diffie–Hellman Protocol

KDCs and protocols like Kerberos are commonly used in practice. But these approaches to the key-distribution problem still require, at some point, a *private* and *authenticated* channel that can be used to share keys. (In particular, we assumed the existence of such a channel between the KDC and the employees on their first day.) Thus, they still cannot solve the problem

of key distribution in open systems like the Internet, where there may be no private channel available between two users who wish to communicate.

To achieve private communication without ever communicating over a private channel, a radically different approach is needed. In 1976, Whitfield Diffie and Martin Hellman published a paper with the innocent-looking title "New Directions in Cryptography." In that work they observed that there is often *asymmetry* in the world; in particular, there are certain actions that can be easily performed but not easily reversed. For example, padlocks can be locked without a key (i.e., easily), but then cannot be reopened. More strikingly, it is easy to shatter a glass vase but extremely difficult to put it back together again. Algorithmically (and more germane for our purposes), it is easy to multiply two large primes but difficult to recover those primes from their product. (This is exactly the factoring problem discussed in previous chapters.) Diffie and Hellman realized that such phenomena could be used to derive interactive protocols for *secure key exchange* that allow two parties to share a secret key, via communication over a public channel, by having the parties perform operations that they can reverse but that an eavesdropper cannot.

The existence of secure key-exchange protocols is quite amazing. It means that you and a friend could agree on a secret by simply shouting across a room (and performing some local computation); the secret would be unknown to anyone else, even if they had listened to everything that was said. Indeed, until 1976 it was generally believed that secure communication could not be achieved without first sharing some secret information using a private channel.

The influence of Diffie and Hellman's paper was enormous. In addition to introducing a fundamentally new way of looking at cryptography, it was one of the first steps toward moving cryptography out of the private domain and into the public one. We quote the first two paragraphs of their paper:

> *We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals.*
>
> *In turn, such applications create a need for new types of cryptographic systems which minimize the necessity of secure key distribution channels. ... At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science.*

Diffie and Hellman were not exaggerating, and the revolution they spoke of was due in great part to their work.

In this section we present the Diffie–Hellman key-exchange protocol. We prove its security against eavesdropping adversaries or, equivalently, under

the assumption that the parties communicate over a public but *authenticated* channel (so an attacker cannot interfere with their communication). Security against an eavesdropping adversary is a relatively weak guarantee, and in practice key-exchange protocols must satisfy stronger notions of security that are beyond our present scope. (Moreover, we are interested here in the setting where the communicating parties have *no* prior shared information, in which case there is nothing that can be done to prevent an adversary from impersonating one of the parties. We return to this point later.)

The setting and definition of security. We consider a setting with two parties—traditionally called Alice and Bob—who run a probabilistic protocol Π in order to generate a shared, secret key; Π can be viewed as the set of instructions for Alice and Bob in the protocol. Alice and Bob begin by holding the security parameter 1^n; they then run Π using (independent) random bits. At the end of the protocol, Alice and Bob output keys $k_A, k_B \in \{0,1\}^n$, respectively. The basic correctness requirement is that $k_A = k_B$. Since we will only deal with protocols that satisfy this requirement, we will speak simply of *the* key $k = k_A = k_B$ generated by an honest execution of Π.

We now turn to defining security. Intuitively, a key-exchange protocol is secure if the key output by Alice and Bob is completely unguessable by an eavesdropping adversary. This is formally defined by requiring that an adversary who has eavesdropped on an execution of the protocol should be unable to distinguish the key k generated by that execution (and now shared by Alice and Bob) from a *uniform* key of length n. This is much stronger than simply requiring that the adversary be unable to *compute* k exactly, and this stronger notion is necessary if the parties will subsequently use k for some cryptographic application (e.g., as a key for a private-key encryption scheme).

Formalizing the above, let Π be a key-exchange protocol, $\mathcal{A}$ an adversary, and n the security parameter. We have the following experiment:

The key-exchange experiment $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

1. *Two parties holding* 1^n *execute protocol* Π. *This results in a transcript* trans *containing all the messages sent by the parties, and a key* k *output by each of the parties.*
2. *A uniform bit* $b \in \{0,1\}$ *is chosen. If* $b = 0$ *set* $\hat{k} := k$, *and if* $b = 1$ *then choose* $\hat{k} \in \{0,1\}^n$ *uniformly at random.*
3. $\mathcal{A}$ *is given* trans *and* $\hat{k}$, *and outputs a bit* b'.
4. *The output of the experiment is defined to be* 1 *if* $b' = b$, *and* 0 *otherwise. (In case* $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1$, *we say that* $\mathcal{A}$ succeeds.)

$\mathcal{A}$ is given trans to capture the fact that $\mathcal{A}$ eavesdrops on the entire execution of the protocol and thus sees all messages exchanged by the parties. In the real world, $\mathcal{A}$ would not be given any key; in the experiment the adversary is given $\hat{k}$ only as a means of defining what it means for $\mathcal{A}$ to "break" the security of Π.

That is, the adversary succeeds in "breaking" Π if it can correctly determine whether the key $\hat{k}$ is the real key corresponding to the given execution of the protocol, or whether $\hat{k}$ is a uniform key that is independent of the transcript. As expected, we say Π is secure if the adversary succeeds with probability that is at most negligibly greater than 1/2. That is:

DEFINITION 10.1 *A key-exchange protocol* Π *is* secure in the presence of an eavesdropper *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

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

The aim of a key-exchange protocol is almost always to generate a shared key k that will be used by the parties for some further cryptographic purpose, e.g., to encrypt and authenticate their subsequent communication using, say, an authenticated encryption scheme. Intuitively, using a shared key generated by a secure key-exchange protocol should be "as good as" using a key shared over a private channel. It is possible to prove this formally; see Exercise 10.1.

The Diffie–Hellman key-exchange protocol. We now describe the key-exchange protocol that appeared in the original paper by Diffie and Hellman (although they were less formal than we will be here). Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n, outputs a description of a cyclic group $\mathbb{G}$, its order q (with $\|q\| = n$), and a generator $g \in \mathbb{G}$. (See Section 8.3.2.) The Diffie–Hellman key-exchange protocol is described formally as Construction 10.2 and illustrated in Figure 10.2.

CONSTRUCTION 10.2

- **Common input:** The security parameter 1^n
- **The protocol:**
  1. Alice runs $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$.
  2. Alice chooses a uniform $x \in \mathbb{Z}_q$, and computes $h_A := g^x$.
  3. Alice sends $(\mathbb{G}, q, g, h_A)$ to Bob.
  4. Bob receives $(\mathbb{G}, q, g, h_A)$. He chooses a uniform $y \in \mathbb{Z}_q$, and computes $h_B := g^y$. Bob sends h_B to Alice and outputs the key $k_B := h_A^y$.
  5. Alice receives h_B and outputs the key $k_A := h_B^x$.

The Diffie–Hellman key-exchange protocol.

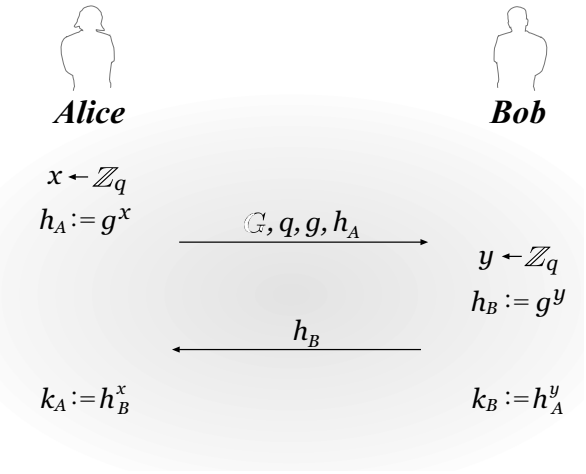

FIGURE 10.2: The Diffie–Hellman key-exchange protocol.

In our description, we have assumed that Alice generates $(\mathbb{G}, q, g)$ and sends these parameters to Bob as part of her first message. In practice, these parameters are *standardized* and are fixed and known to both parties before the protocol begins. In that case Alice need only send h_A, and Bob need not wait to receive Alice's message before computing and sending h_B.

It is not hard to see that the protocol is correct: Bob computes the key

$$k_B = h_A^y = (g^x)^y = g^{xy}$$

and Alice computes the key

$$k_A = h_B^x = (g^y)^x = g^{xy},$$

and so $k_A = k_B$. (The observant reader will note that the shared key is a group element, not a bit-string. We will return to this point later.)

Diffie and Hellman did not prove security of their protocol; indeed, the appropriate notions (both the definitional framework as well as the idea of formulating precise assumptions) were not yet in place. Let us see what sort of assumption will be needed in order for the protocol to be secure. A first observation, made by Diffie and Hellman, is that a *minimal* requirement for security here is that the discrete-logarithm problem be hard relative to $\mathcal{G}$. If not, then an adversary given the transcript (which, in particular, includes h_A) can compute the secret value of one of the parties (i.e., x) and then easily compute the shared key using that value. So, hardness of the discrete-logarithm problem is necessary for the protocol to be secure. It is not, however, sufficient, as it is possible that there are other ways of computing the key $k_A = k_B$ without explicitly computing x or y. The *computational* Diffie–Hellman assumption—which would only guarantee that the key g^{xy} is

hard to compute in its *entirety* from the transcript—does not suffice either. What is required by Definition 10.1 is that the shared key g^{xy} should be *indistinguishable from uniform* for any adversary given g, g^x, and g^y. This is exactly the *decisional* Diffie–Hellman assumption introduced in Section 8.3.2.

As we will see, a proof of security for the protocol follows almost immediately from the decisional Diffie–Hellman assumption. This should not be surprising, as the Diffie–Hellman assumptions were introduced—well after Diffie and Hellman published their paper—as a way of abstracting the properties underlying the (conjectured) security of the Diffie–Hellman protocol. Given this, it is fair to ask whether anything is gained by defining and proving security here. By this point in the book, hopefully you are convinced the answer is *yes*. Precisely defining secure key exchange forces us to think about exactly what security properties we require; specifying a precise assumption (namely, the decisional Diffie–Hellman assumption) means we can study this assumption independently of any particular application and—once we are convinced of its plausibility—construct other protocols based on it; finally, proving security shows that the assumption does, indeed, suffice for the protocol to meet our desired notion of security.

In our proof of security, we use a modified version of Definition 10.1 in which it is required that the shared key be indistinguishable from a *uniform element of* $\mathbb{G}$ rather than from a *uniform n-bit string*. This discrepancy will need to be addressed before the protocol can be used in practice—after all, group elements are not typically useful as cryptographic keys, and the representation of a uniform group element will not, in general, be a uniform bit-string—and we briefly discuss one standard way to do so following the proof. For now, we let $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ denote a modified experiment where if $b = 1$ the adversary is given $\hat{k}$ chosen uniformly from $\mathbb{G}$ instead of a uniform n-bit string.

THEOREM 10.3 *If the decisional Diffie–Hellman problem is hard relative to $\mathcal{G}$, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper (with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}$).*

PROOF Let $\mathcal{A}$ be a PPT adversary. Since $\Pr[b = 0] = \Pr[b = 1] = 1/2$, we have

$$\begin{aligned}&\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right]\\ &\quad = \frac{1}{2}\cdot \Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0\right] + \frac{1}{2}\cdot \Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1\right].\end{aligned}$$

In experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ the adversary $\mathcal{A}$ receives $(\mathbb{G}, q, g, h_A, h_B, \hat{k})$, where $(\mathbb{G}, q, g, h_A, h_B)$ represents the transcript of the protocol execution, and $\hat{k}$ is either the actual key computed by the parties (if $b = 0$) or a uniform group element (if $b = 1$). Distinguishing between these two cases is exactly

equivalent to solving the decisional Diffie–Hellman problem. That is

$$\begin{aligned}
&\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0\right] + \frac{1}{2} \cdot \Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1\right] \\
&= \frac{1}{2} \cdot \Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] \\
&= \frac{1}{2} \cdot \Big(1 - \Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1]\Big) + \frac{1}{2} \cdot \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] \\
&= \frac{1}{2} + \frac{1}{2} \cdot \Big(\Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\Big) \\
&\leq \frac{1}{2} + \frac{1}{2} \cdot \Big|\Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\Big|,
\end{aligned}$$

where the probabilities are all taken over $(\mathbb{G}, q, g)$ output by $\mathcal{G}(1^n)$, and uniform choice of $x, y, z \in \mathbb{Z}_q$. (Note that since g is a generator, g^z is a uniform element of $\mathbb{G}$ when z is uniformly distributed in $\mathbb{Z}_q$.) If the decisional Diffie–Hellman assumption is hard relative to $\mathcal{G}$, that exactly means that there is a negligible function negl for which

$$\big|\Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\big| \leq \mathsf{negl}(n).$$

We conclude that

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

completing the proof. ∎

Uniform group elements vs. uniform bit-strings. The previous theorem shows that the key output by Alice and Bob in the Diffie–Hellman protocol is indistinguishable (for a polynomial-time eavesdropper) from a uniform group element. In order to use the key for subsequent cryptographic applications—as well as to meet Definition 10.1—the key output by the parties should instead be indistinguishable from a uniform bit-string of the appropriate length. The Diffie–Hellman protocol can be modified to achieve this by having the parties apply an appropriate *key-derivation function* (cf. Section 5.6.4) to the shared group element g^{xy} they each compute.

Active adversaries. So far we have considered only an eavesdropping adversary. Although eavesdropping attacks are by far the most common (as they are the easiest to carry out), they are by no means the only possible attack. *Active* attacks, in which the adversary sends messages of its own to one or both of the parties, are also a concern, and any protocol used in practice must be resilient to such attacks as well. When considering active attacks, it is useful to distinguish, informally, between *impersonation* attacks where

the adversary impersonates one party while interacting with the other party, and *man-in-the-middle* attacks where both honest parties are executing the protocol and the adversary is intercepting and modifying messages being sent from one party to the other. We will not formally define security against either class of attacks, as such definitions are rather involved and cannot be achieved without the parties sharing *some* information in advance. Nevertheless, it is worth remarking that the Diffie–Hellman protocol is *completely insecure* against man-in-the-middle attacks. In fact, a man-in-the-middle adversary can act in such a way that Alice and Bob terminate the protocol with different keys k_A and k_B that are both known to the adversary, yet neither Alice nor Bob can detect that any attack was carried out. We leave the details of this attack as an exercise.

Diffie–Hellman key exchange in practice. The Diffie–Hellman protocol in its basic form is typically not used in practice due to its insecurity against man-in-the-middle attacks, as discussed above. This does not detract in any way from its importance. The Diffie–Hellman protocol served as the first demonstration that asymmetric techniques (and number-theoretic problems) could be used to alleviate the problems of key distribution in cryptography. Furthermore, the Diffie–Hellman protocol is at the core of standardized key-exchange protocols that are resilient to man-in-the-middle attacks and are in wide use today. One notable example is TLS; see Section 12.8.

10.4 The Public-Key Revolution

In addition to key exchange, Diffie and Hellman also introduced in their ground-breaking work the notion of *public-key* (or *asymmetric*) cryptography. In the public-key setting (in contrast to the private-key setting we have studied in Chapters 1–7), a party who wishes to communicate securely generates a *pair* of keys: a *public key* that is widely disseminated, and a *private key* that it keeps secret. (The fact that there are now two different keys is what makes the scheme asymmetric.) Having generated these keys, a party can use them to ensure secrecy for messages it receives using a *public-key encryption scheme*, or integrity for messages it sends using a *digital signature scheme*. (See Figure 10.3.) We provide a brief taste of these primitives here, and discuss them in extensive detail in Chapters 11 and 12, respectively.

In a public-key encryption scheme, the public key generated by some party serves as an *encryption key*; anyone who knows that public key can use it to encrypt messages and generate corresponding ciphertexts. The private key serves as a *decryption key* and is used by the party who knows it to recover the original message from any ciphertext generated using the matching public key. Furthermore—and it is amazing that something like this exists!—

	Private-Key Setting	**Public-Key Setting**
Secrecy	Private-key encryption	Public-key encryption
Integrity	Message authentication codes	Digital signature schemes

FIGURE 10.3: Cryptographic primitives in the private-key and the public-key settings.

the secrecy of encrypted messages is preserved *even against an adversary who knows the encryption key* (but not the decryption key). In other words, the (public) encryption key is of no use for an attacker trying to decrypt ciphertexts encrypted using that key. To enable secret communication, then, a receiver can simply send her public key to a potential sender (without having to worry about an eavesdropping adversary who observes her public key), or publicize her public key on her webpage or in some central database. A public-key encryption scheme thus enables private communication without relying on a private channel for key distribution.[1]

A digital signature scheme is a public-key analogue of message authentication codes (MACs). Here, the private key serves as an "authentication key" (more typically called a *signing key*) that enables the party who knows this key to generate "authentication tags" (i.e., *signatures*) for messages it sends. The public key acts as a *verification key*, allowing anyone who knows it to verify signatures issued by the sender. As with MACs, a digital signature scheme can be used to prevent undetected tampering of a message. The fact that verification can be done by anyone who knows the public key of the sender, however, turns out to have far-reaching ramifications. Specifically, it makes it possible to take a document that was signed by Alice and present it to a third party (say, a judge) as proof that Alice indeed signed the document. This property is called *non-repudiation* and has extensive applications in electronic commerce. For example, it is possible to digitally sign contracts, send signed electronic purchase orders or promises of payments, and so on. Digital signatures are also used for the secure distribution of public keys as part of a *public-key infrastructure*, as discussed in more detail in Section 12.7.

In their paper, Diffie and Hellman set forth the notion of public-key cryptography but did not give any candidate constructions, A year later, Ron Rivest, Adi Shamir, and Len Adleman proposed the *RSA problem* and presented the first public-key encryption and digital signature schemes based on the hardness of this problem. Variants of their schemes are now among the most widely used cryptographic primitives today. In 1985, Taher El Gamal presented an encryption scheme that is essentially a slight twist on the Diffie–Hellman key-exchange protocol, and is now also widely used. Thus, although

[1] For now, however, we do assume an *authenticated* channel that allows the sender to obtain a legitimate copy of the receiver's public key. In Section 12.7 we show how public-key cryptography can be used to reduce this assumption as well.

Diffie and Hellman did not succeed in constructing a (non-interactive) public-key encryption scheme, they came very close.

We close by summarizing how public-key cryptography addresses the limitations of the private-key setting discussed in Section 10.1:

1. Public-key encryption allows key distribution to be done over public (but authenticated) channels. This can simplify the distribution and updating of key material.

2. Public-key cryptography reduces the need for users to store many secret keys. Consider again the setting of a large corporation where each pair of employees needs the ability to communicate securely. Using public-key cryptography, it suffices for each employee to store just a *single* private key (their own) and the public keys of all other employees. Importantly, these latter keys do *not* need to be stored in secret; they could even be stored in some central (public) repository.

3. Finally, public-key cryptography is (more) suitable for open environments where parties who have never previously interacted want the ability to communicate securely. As one commonplace example, an Internet merchant can post their public key on-line; a user making a purchase can then obtain the merchant's public key, as needed, when they need to encrypt their credit card information.

The invention of public-key encryption was a revolution in cryptography. It is no coincidence that until the late 1970s and early 1980s, encryption and cryptography in general belonged to the domain of intelligence and military organizations, and only with the advent of public-key techniques did the use of cryptography spread to the masses.

References and Additional Reading

We have only briefly discussed the problems of key distribution and key management. For more information, we recommend looking at textbooks on network security. Kaufman et al. [102] provide a good treatment of protocols for secure key distribution, what they aim to achieve, and how they work.

We have not made any attempt to capture the full history of the development of public-key cryptography. Others besides Diffie and Hellman were working on similar ideas in the 1970s. One researcher in particular doing similar and independent work was Ralph Merkle, considered by many to be a co-inventor of public-key cryptography (although he published after Diffie and Hellman). We also mention Michael Rabin, who developed constructions of signature schemes and public-key encryption schemes based on the hardness of factoring about one year after the work of Rivest, Shamir, and Adleman [148].

We highly recommend reading the original paper by Diffie and Hellman [58], and refer the reader to the book by Levy [114] for more on the political and historical aspects of the public-key revolution.

Interestingly, aspects of public-key cryptography were discovered in the intelligence community before being published in the open scientific literature. In the early 1970s, James Ellis, Clifford Cocks, and Malcolm Williamson of the British intelligence agency GCHQ invented the notion of public-key cryptography, a variant of RSA encryption, and a variant of the Diffie–Hellman key-exchange protocol. Their work was not declassified until 1997. Although the underlying mathematics of public-key cryptography may have been discovered before 1976, it is fair to say that the widespread ramifications of this new technology were not appreciated until Diffie and Hellman came along.

Exercises

10.1 Let Π be a key-exchange protocol, and $(\mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme. Consider the following *interactive* protocol Π' for encrypting a message: first, the sender and receiver run Π to generate a shared key k. Next, the sender computes $c \leftarrow \mathsf{Enc}_k(m)$ and sends c to the other party, who decrypts and recovers m using k.

(a) Formulate a definition of indistinguishable encryptions in the presence of an eavesdropper (cf. Definition 3.8) appropriate for this interactive setting.

(b) Prove that if Π is secure in the presence of an eavesdropper and $(\mathsf{Enc}, \mathsf{Dec})$ has indistinguishable encryptions in the presence of an eavesdropper, then Π' satisfies your definition.

10.2 Show that, for either of the groups considered in Sections 8.3.3 or 8.3.4, a uniform group element (expressed using the natural representation) is easily distinguishable from a uniform bit-string of the same length.

10.3 Describe a man-in-the-middle attack on the Diffie–Hellman protocol where the adversary shares a key k_A with Alice and a (different) key k_B with Bob, and Alice and Bob cannot detect that anything is wrong.

10.4 Consider the following key-exchange protocol:

(a) Alice chooses uniform $k, r \in \{0,1\}^n$, and sends $s := k \oplus r$ to Bob.

(b) Bob chooses uniform $t \in \{0,1\}^n$, and sends $u := s \oplus t$ to Alice.

(c) Alice computes $w := u \oplus r$ and sends w to Bob.

(d) Alice outputs k and Bob outputs $w \oplus t$.

Show that Alice and Bob output the same key. Analyze the security of the scheme (i.e., either prove its security or show a concrete attack).

Chapter 11

Public-Key Encryption

11.1 Public-Key Encryption – An Overview

The introduction of public-key encryption marked a revolution in cryptography. Until that time, cryptographers had relied exclusively on shared, *secret* keys to achieve private communication. Public-key techniques, in contrast, enable parties to communicate privately without having agreed on *any* secret information in advance. As we have already noted, it is quite amazing and counterintuitive that this is possible: it means that two people on opposite sides of a room who can only communicate by shouting to each other, and have no initial secret, can talk in such a way that no one else in the room learns anything about what they are saying!

In the setting of private-key encryption, two parties agree on a secret key that can be used, by either party, for both encryption and decryption. Public-key encryption is *asymmetric* in both these respects. One party (the *receiver*) generates a *pair* of keys (pk, sk), called the *public key* and the *private key*, respectively. The public key is used by a *sender* to encrypt a message; the receiver uses the private key to decrypt the resulting ciphertext.

Since the goal is to avoid the need for two parties to meet in advance to agree on any information, how does the sender learn pk? At an abstract level, there are two ways this can occur. Call the receiver Alice and the sender Bob. In the first approach, when Alice learns that Bob wants to communicate with her, she can at that point generate (pk, sk) (assuming she hasn't done so already) and then send pk to Bob *in the clear*; Bob can then use pk to encrypt his message. We emphasize that the channel between Alice and Bob may be public, but is assumed to be authenticated, meaning that the adversary cannot modify the public key sent by Alice to Bob (and, in particular, cannot replace it with its own key). See Section 12.7 for a discussion of how public keys can be distributed over unauthenticated channels.

An alternative approach is for Alice to generate her keys (pk, sk) in advance, *independently* of any particular sender. (In fact, at the time of key generation Alice need not even be aware that Bob wants to talk to her, or even that Bob exists.) Alice can widely disseminate her public key pk by, say, publishing it on her webpage, putting it on her business cards, or placing it in a public directory. Now, *anyone* who wishes to communicate privately with Alice can

look up her public key and proceed as above. Note that multiple senders can communicate multiple times with Alice using the same public key pk for encrypting all their communication.

Note that pk is inherently public—and can thus be learned easily by an attacker—in either of the above scenarios. In the first case, an adversary eavesdropping on the communication between Alice and Bob obtains pk directly; in the second case, an adversary could just as well look up Alice's public key on its own. We see that the security of public-key encryption cannot rely on secrecy of pk, but must instead rely on secrecy of sk. It is therefore crucial that Alice not reveal her private key to anyone, including the sender Bob.

Comparison to Private-Key Encryption

Perhaps the most obvious difference between private- and public-key encryption is that the former assumes *complete* secrecy of all cryptographic keys, whereas the latter requires secrecy for only the private key sk. Although this may seem like a minor distinction, the ramifications are huge: in the private-key setting the communicating parties must somehow be able to share the secret key without allowing any third party to learn it, whereas in the public-key setting the public key can be sent from one party to the other over a public channel without compromising security. For parties shouting across a room or, more realistically, communicating over a public network like a phone line or the Internet, public-key encryption is the only option.

Another important distinction is that private-key encryption schemes use the same key for both encryption and decryption, whereas public-key encryption schemes use different keys for each operation. That is, public-key encryption is inherently *asymmetric*. This asymmetry in the public-key setting means that the roles of sender and receiver are *not* interchangeable as they are in the private-key setting: a single key-pair allows communication in one direction only. (Bidirectional communication can be achieved in a number of ways; the point is that a single invocation of a public-key encryption scheme forces a distinction between one user who acts as a receiver and other users who act as senders.) In addition, a single instance of a public-key encryption scheme enables multiple senders to communicate privately with a single receiver, in contrast to the private-key case where a secret key shared between two parties enables private communication only between those two parties.

Summarizing and elaborating the preceding discussion, we see that public-key encryption has the following advantages relative to private-key encryption:

- Public-key encryption addresses (to some extent) the *key-distribution problem*, since communicating parties do not need to secretly share a key in advance of their communication. Two parties can communicate secretly even if *all* communication between them is monitored.
- When a single receiver is communicating with N senders (e.g., an on-line merchant processing credit card orders from multiple purchasers), it is

much more convenient for the receiver to store a *single* private key sk rather than to share, store, and manage N different secret keys (i.e., one for each sender). In fact, when using public-key encryption the number and identities of potential senders need not be known at the time of key generation. This allows enormous flexibility in "open systems."

The fact that public-key encryption schemes allow anyone to act as a sender can be a drawback when a receiver only wants to receive messages from one specific individual. In that case, an authenticated (private-key) encryption scheme would be a better choice than public-key encryption.

The main disadvantage of public-key encryption is that it is roughly *2 to 3 orders of magnitude* slower than private-key encryption.[1] It can be a challenge to implement public-key encryption in severely resource-constrained devices like smartcards or radio-frequency identification (RFID) tags. Even when a desktop computer is performing cryptographic operations, carrying out thousands of such operations per second (as in the case of an on-line merchant processing credit card transactions) may be prohibitive. Thus, when private-key encryption is an option (i.e., if two parties *can* securely share a key in advance), then it typically should be used.

In fact, as we will see in Section 11.3, private-key encryption is used *in the public-key setting* to improve efficiency for the (public-key) encryption of long messages. A thorough understanding of private-key encryption is therefore crucial to appreciate how public-key encryption is implemented in practice.

Secure Distribution of Public Keys

In our entire discussion thus far, we have implicitly assumed that the adversary is *passive*; that is, the adversary only eavesdrops on communication between the sender and receiver but does not *actively* interfere with the communication. If the adversary has the ability to tamper with *all* communication between the honest parties, and these honest parties share no keys in advance, then privacy simply cannot be achieved. For example, if a receiver Alice sends her public key pk to Bob but the adversary replaces it with a key pk' of his own (for which it knows the matching private key sk'), then even though Bob encrypts his message using pk' the adversary will easily be able to recover the message (using sk'). A similar attack works if an adversary is able to change the value of Alice's public key that is stored in some public directory, or if the adversary can tamper with the public key as it is transmitted from the public directory to Bob. If Alice and Bob do not share any information in advance, and are not willing to rely on some mutually trusted third party, there is nothing Alice or Bob can do to prevent active attacks of this sort, or

[1] It is difficult to give an exact comparison since the relative efficiency depends on the exact schemes under consideration as well as various implementation details.

even to tell that such an attack is taking place.[2]

Importantly, our treatment of public-key encryption in this chapter *assumes that senders are able to obtain a legitimate copy of the receiver's public key.* (This will be implicit in the security definitions we provide.) That is, we assume *secure key distribution.* This assumption is made not because active attacks of the type discussed above are of no concern—in fact, they represent a serious threat that must be dealt with in any real-world system that uses public-key encryption. Rather, this assumption is made because there exist other mechanisms for preventing active attacks (see, for example, Section 12.7), and it is therefore convenient (and useful) to decouple the study of secure public-key encryption from the study of secure public-key distribution.

11.2 Definitions

We begin by defining the syntax of public-key encryption. The definition is very similar to Definition 3.7, with the exception that instead of working with just one key, we now have distinct encryption and decryption keys.

DEFINITION 11.1 *A* public-key encryption scheme *is a triple of probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a pair of keys* (pk, sk)*. We refer to the first of these as the* public key *and the second as the* private key*. We assume for convenience that* pk *and* sk *each has length at least* n*, and that* n *can be determined from* pk, sk*.*

2. *The* encryption algorithm Enc *takes as input a public key* pk *and a message* m *from some message space (that may depend on* pk*). It outputs a ciphertext* c*, and we write this as* $c \leftarrow \mathsf{Enc}_{pk}(m)$*. (Looking ahead,* Enc *will need to be probabilistic to achieve meaningful security.)*

3. *The deterministic* decryption algorithm Dec *takes as input a private key* sk *and a ciphertext* c*, and outputs a message* m *or a special symbol* $\perp$ *denoting failure. We write this as* $m := \mathsf{Dec}_{sk}(c)$*.*

It is required that, except possibly with negligible probability over (pk, sk) *output by* $\mathsf{Gen}(1^n)$*, we have* $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$ *for any (legal) message* m*.*

The important difference from the private-key setting is that the key-generation algorithm Gen now outputs *two* keys instead of one. The public

[2] In our "shouting-across-a-room" scenario, Alice and Bob can detect when an adversary interferes with the communication. But this is only because: (1) the adversary cannot prevent Alice's messages from reaching Bob, and (2) Alice and Bob "share" in advance information (e.g., the sound of their voices) that allows them to "authenticate" their communication.

key pk is used for encryption, while the private key sk is used for decryption. Reiterating our earlier discussion, pk is assumed to be widely distributed so that anyone can encrypt messages for the party who generated this key, but sk must be kept private by the receiver in order for security to possibly hold.

We allow for a negligible probability of decryption error and, indeed, some of the schemes we present will have a negligible error probability (e.g., if a prime needs to be chosen but with negligible probability a composite is obtained instead). Despite this, we will generally ignore the issue from here on.

For practical usage of public-key encryption, we will want the message space to be $\{0,1\}^n$ or $\{0,1\}^*$ (and, in particular, to be independent of the public key). Although we will sometimes describe encryption schemes using some message space $\mathcal{M}$ that does not contain all bit-strings of some fixed length (and that may also depend on the public key), we will in such cases also specify how to encode bit-strings as elements of $\mathcal{M}$. This encoding must be both efficiently computable and efficiently reversible, so the receiver can recover the bit-string that was encrypted.

11.2.1 Security against Chosen-Plaintext Attacks

We initiate our treatment of security by introducing the "natural" counterpart of Definition 3.8 in the public-key setting. Since extensive motivation for this definition (as well as the others we will see) has already been given in Chapter 3, the discussion here will be relatively brief and will focus primarily on the differences between the private-key and the public-key settings.

Given a public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$, consider the following experiment:

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

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk*, and outputs a pair of equal-length messages* m_0, m_1 *in the message space.*
3. *A uniform bit* $b \in \{0,1\}$ *is chosen, and then a ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$*. We call* c *the* challenge ciphertext.
4. $\mathcal{A}$ *outputs a bit* b'*. The output of the experiment is* 1 *if* $b' = b$*, and* 0 *otherwise. If* $b' = b$ *we say that* $\mathcal{A}$ succeeds.

DEFINITION 11.2 *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions in the presence of an eavesdropper *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

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

The main difference between the above definition and Definition 3.8 is that here $\mathcal{A}$ *is given the public key* pk. Furthermore, we allow $\mathcal{A}$ to choose its messages m_0 and m_1 based on this public key. This is essential when defining security of public-key encryption since, as discussed previously, we assume that the adversary knows the public key of the recipient.

The seemingly "minor" modification of giving the adversary $\mathcal{A}$ the public key pk being used to encrypt the message has a tremendous impact: it effectively gives $\mathcal{A}$ access to an encryption oracle *for free*. (The concept of an encryption oracle is explained in Section 3.4.2.) This is true because the adversary, given pk, can encrypt any message m on its own by simply computing $\mathsf{Enc}_{pk}(m)$. (As always, $\mathcal{A}$ is assumed to know the algorithm Enc.) The upshot is that Definition 11.2 is *equivalent* to CPA-security (i.e., security against chosen-plaintext attacks), defined in a manner analogous to Definition 3.22, with the only difference being that the attacker is given the public key in the corresponding experiment. We thus have:

PROPOSITION 11.3 *If a public-key encryption scheme has indistinguishable encryptions in the presence of an eavesdropper, it is CPA-secure.*

This is in contrast to the private-key setting, where there exist schemes that have indistinguishable encryptions in the presence of an eavesdropper but are insecure under a chosen-plaintext attack (see Proposition 3.20). Further differences from the private-key setting that follow almost immediately as consequences of the above are discussed next.

Impossibility of perfectly secret public-key encryption. *Perfectly secret* public-key encryption could be defined analogously to Definition 2.3 by conditioning on the entire view of an eavesdropper (i.e., including the public key). Equivalently, it could be defined by extending Definition 11.2 to require that for *all* adversaries $\mathcal{A}$ (not only efficient ones), we have:

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}.$$

In contrast to the private-key setting, however, perfectly secret public-key encryption is *impossible*, regardless of how long the keys are or how small the message space is. In fact, an unbounded adversary given pk and a ciphertext c computed via $c \leftarrow \mathsf{Enc}_{pk}(m)$ can determine m with probability 1. A proof of this is left as Exercise 11.1.

Insecurity of deterministic public-key encryption. As noted in the context of private-key encryption, no deterministic encryption scheme can be CPA-secure. The same is true here:

THEOREM 11.4 *No deterministic public-key encryption scheme is CPA-secure.*

Because Theorem 11.4 is so important, it merits a bit more discussion. The theorem is *not* an "artifact" of our security definition, or an indication that our definition is too strong. Deterministic public-key encryption schemes are vulnerable to *practical* attacks in *realistic* scenarios and should never be used. The reason is that a deterministic scheme not only allows the adversary to determine when the same message is sent twice (as in the private-key setting), but also allows the adversary to recover the message, with probability 1, if the set of possible messages being encrypted is small. For example, consider a professor encrypting students' grades. Here, an eavesdropper knows that each student's grade must be one of $\{A, B, C, D, F\}$. If the professor uses a deterministic public-key encryption scheme, an eavesdropper can quickly determine any student's actual grade by encrypting all possible grades and comparing the result to the given ciphertext.

Although the above theorem seems deceptively simple, for a long time *many real-world systems were designed using deterministic public-key encryption.* When public-key encryption was introduced, it is fair to say that the importance of probabilistic encryption was not yet fully realized. The seminal work of Goldwasser and Micali, in which (something equivalent to) Definition 11.2 was proposed and Theorem 11.4 was stated, marked a turning point in the field of cryptography. The importance of pinning down one's intuition in a formal definition and looking at things the right way for the first time—even if seemingly simple in retrospect—should not be underestimated.

11.2.2 Multiple Encryptions

As in Chapter 3, it is important to understand the effect of using the same key (in this case, the same public key) for encrypting multiple messages. We could formulate security in such a setting by having an adversary output two *lists* of plaintexts, as in Definition 3.19. For the reasons discussed in Section 3.4.2, however, we choose instead to use a definition in which the attacker is given access to a "left-or-right" oracle $\mathsf{LR}_{pk,b}$ that, on input a pair of equal-length messages m_0, m_1, computes the ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ and returns c. The attacker is allowed to query this oracle as many times as it likes, and the definition therefore models security when multiple (unknown) messages are encrypted using the same public key.

Formally, consider the following experiment defined for a public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and adversary $\mathcal{A}$:

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

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *A uniform bit* $b \in \{0,1\}$ *is chosen.*
3. *The adversary* $\mathcal{A}$ *is given input* pk *and oracle access to* $\mathsf{LR}_{pk,b}(\cdot,\cdot)$.
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. If* $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1$, *we say that* $\mathcal{A}$ succeeds.

DEFINITION 11.5 *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable multiple encryptions *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that:*

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

We will show that any CPA-secure scheme *automatically* has indistinguishable multiple encryptions; that is, in the public-key setting, security for encryption of a single message implies security for encryption of multiple messages. This means we can prove security of some scheme with respect to Definition 11.2, which is simpler and easier to work with, and conclude that the scheme satisfies Definition 11.5, a seemingly stronger definition that more accurately models real-world usage of public-key encryption. A proof of the following theorem is given below.

THEOREM 11.6 *If public-key encryption scheme* Π *is CPA-secure, then it also has indistinguishable* multiple *encryptions.*

An analogous result in the private-key setting was stated, but not proved, as Theorem 3.24.

Encrypting arbitrary-length messages. An immediate consequence of Theorem 11.6 is that a CPA-secure public-key encryption scheme for *fixed-length* messages implies a public-key encryption scheme for *arbitrary-length* messages satisfying the same notion of security. We illustrate this in the extreme case when the original scheme encrypts only 1-bit messages. Say $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an encryption scheme for single-bit messages. We can construct a new scheme $\Pi' = (\mathsf{Gen}, \mathsf{Enc}', \mathsf{Dec}')$ that has message space $\{0,1\}^*$ by defining Enc' as follows:

$$\mathsf{Enc}'_{pk}(m) = \mathsf{Enc}_{pk}(m_1), \ldots, \mathsf{Enc}_{pk}(m_\ell), \tag{11.1}$$

where $m = m_1 \cdots m_\ell$. (The decryption algorithm Dec' is constructed in the obvious way.) We have:

CLAIM 11.7 *Let* Π *and* Π' *be as above. If* Π *is CPA-secure, then so is* Π'.

The claim follows since we can view encryption of the message m using Π' as encryption of t messages $(m_1, \ldots, m_t)$ using scheme Π.

A note on terminology. We have introduced three definitions of security for public-key encryption schemes—indistinguishable encryptions in the presence of an eavesdropper, CPA-security, and indistinguishable *multiple* encryptions—that are all equivalent. Following the usual convention in the cryptographic literature, we will simply use the term "CPA-security" to refer to schemes meeting these notions of security.

*Proof of Theorem 11.6

The proof of Theorem 11.6 is rather involved. We therefore provide some intuition before turning to the details. For this intuitive discussion we assume for simplicity that $\mathcal{A}$ makes only *two* calls to the LR oracle in experiment $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n)$. (In the full proof, the number of calls is arbitrary.)

Fix an arbitrary PPT adversary $\mathcal{A}$ and a CPA-secure public-key encryption scheme Π, and consider an experiment $\mathsf{PubK}^{\mathsf{LR\text{-}cpa2}}_{\mathcal{A},\Pi}(n)$ where $\mathcal{A}$ can make only *two* queries to the LR oracle. Denote the queries made by $\mathcal{A}$ to the oracle by $(m_{1,0}, m_{1,1})$ and $(m_{2,0}, m_{2,1})$; note that the second pair of messages may depend on the first ciphertext obtained by $\mathcal{A}$ from the oracle. In the experiment, $\mathcal{A}$ receives either a pair of ciphertexts $(\mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0}))$ (if $b = 0$), or a pair of ciphertexts $(\mathsf{Enc}_{pk}(m_{1,1}), \mathsf{Enc}_{pk}(m_{2,1}))$ (if $b = 1$). We write $\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0}))$ to denote the output of $\mathcal{A}$ in the first case, and analogously for the second.

Let $\vec{C}_0$ denote the distribution of ciphertext pairs in the first case, and $\vec{C}_1$ the distribution of ciphertext pairs in the second case. To show that Definition 11.5 holds (for $\mathsf{PubK}^{\mathsf{LR\text{-}cpa2}}_{\mathcal{A},\Pi}$), we need to prove that $\mathcal{A}$ cannot distinguish between being given a pair of ciphertexts distributed according to $\vec{C}_0$, or a pair of ciphertexts distributed according to $\vec{C}_1$. That is, we need to prove that there is a negligible function negl such that

$$\begin{aligned}\big|\Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0})) = 1] \\ - \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,1}), \mathsf{Enc}_{pk}(m_{2,1})) = 1]\big| \leq \mathsf{negl}(n). \qquad (11.2)\end{aligned}$$

(This is equivalent to Definition 11.5 for the same reason that Definition 3.9 is equivalent to Definition 3.8.) To prove this, we will show that

1. CPA-security of Π implies that $\mathcal{A}$ cannot distinguish between the case when it is given a pair of ciphertexts distributed according to $\vec{C}_0$, or a pair of ciphertexts $(\mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,1}))$, which corresponds to encrypting the *first* message in $\mathcal{A}$'s first oracle query and the *second* message in $\mathcal{A}$'s second oracle query. (Although this cannot occur in $\mathsf{PubK}^{\mathsf{LR\text{-}cpa2}}_{\mathcal{A},\Pi}(n)$, we can still ask what $\mathcal{A}$'s behavior would be if given such a ciphertext pair.) Let $\vec{C}_{01}$ denote the distribution of ciphertext pairs in this latter case.

2. Similarly, CPA-security of Π implies that $\mathcal{A}$ cannot distinguish between the case when it is given a pair of ciphertexts distributed according to $\vec{C}_{01}$, or a pair of ciphertexts distributed according to $\vec{C}_1$.

The above says that $\mathcal{A}$ cannot distinguish between distributions $\vec{C}_0$ and $\vec{C}_{01}$, nor between distributions $\vec{C}_{01}$ and $\vec{C}_1$. We conclude (using simple algebra) that $\mathcal{A}$ cannot distinguish between distributions $\vec{C}_0$ and $\vec{C}_1$.

The crux of the proof, then, is showing that $\mathcal{A}$ cannot distinguish between being given a pair of ciphertexts distributed according to $\vec{C}_0$, or a pair of

ciphertexts distributed according to $\vec{C}_{01}$. (The other case follows similarly.) That is, we want to show that there is a negligible function negl for which

$$\big|\Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0})) = 1] - \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,1})) = 1]\big| \leq \mathsf{negl}(n). \quad (11.3)$$

Note that the only difference between the input of the adversary $\mathcal{A}$ in each case is in the second element. Intuitively, indistinguishability follows from the single-message case since $\mathcal{A}$ can generate $\mathsf{Enc}_{pk}(m_{1,0})$ by itself. Formally, consider the following PPT adversary $\mathcal{A}'$ running in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)$:

Adversary $\mathcal{A}'$:

1. On input pk, adversary $\mathcal{A}'$ runs $\mathcal{A}(pk)$ as a subroutine.
2. When $\mathcal{A}$ makes its first query $(m_{1,0}, m_{1,1})$ to the LR oracle, $\mathcal{A}'$ computes $c_1 \leftarrow \mathsf{Enc}_{pk}(m_{1,0})$ and returns c_1 to $\mathcal{A}$ as the response from the oracle.
3. When $\mathcal{A}$ makes its second query $(m_{2,0}, m_{2,1})$ to the LR oracle, $\mathcal{A}'$ outputs $(m_{2,0}, m_{2,1})$ and receives back a challenge ciphertext c_2. This is returned to $\mathcal{A}$ as the response from the LR oracle.
4. $\mathcal{A}'$ outputs the bit b' output by $\mathcal{A}$.

Looking at experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)$, we see that when $b = 0$ then the challenge ciphertext c_2 is computed as $\mathsf{Enc}_{pk}(m_{2,0})$. Thus,

$$\Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,0})) = 1] = \Pr[\mathcal{A}(\mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0})) = 1]. \quad (11.4)$$

(We suppress explicit mention of pk to save space.) In contrast, when $b = 1$ in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)$, then c_2 is computed as $\mathsf{Enc}_{pk}(m_{2,1})$ and so

$$\Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,1})) = 1] = \Pr[\mathcal{A}(\mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,1})) = 1]. \quad (11.5)$$

CPA-security of Π implies that there is a negligible function negl such that

$$|\Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,0})) = 1] - \Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,1})) = 1]| \leq \mathsf{negl}(n).$$

This, together with Equations (11.4) and (11.5), yields Equation (11.3).

In almost exactly the same way, we can prove that:

$$\big|\Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,1})) = 1] - \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,1}), \mathsf{Enc}_{pk}(m_{2,1})) = 1]\big| \leq \mathsf{negl}(n). \quad (11.6)$$

Equation (11.2) follows by combining Equations (11.3) and (11.6).

The main complication that arises in the general case is that the number of queries to the LR oracle is no longer fixed but may instead be an arbitrary

polynomial of n. In the formal proof this is handled using a *hybrid argument*. (Hybrid arguments were used also in Chapter 7.)

PROOF (of Theorem 11.6) Let Π be a CPA-secure public-key encryption scheme and $\mathcal{A}$ an arbitrary PPT adversary in experiment $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n)$. Let $t = t(n)$ be a polynomial upper bound on the number of queries made by $\mathcal{A}$ to the LR oracle, and assume without loss of generality that $\mathcal{A}$ always queries the oracle *exactly* this many times. For a given public key pk and $0 \leq i \leq t$, let LR^i_{pk} denote the oracle that on input (m_0, m_1) returns $\mathsf{Enc}_{pk}(m_0)$ for the first i queries it receives, and returns $\mathsf{Enc}_{pk}(m_1)$ for the next $t - i$ queries it receives. (That is, for the first i queries the first message in the input pair is encrypted, and in response to the remaining queries the second message in the input pair is encrypted.) We stress that each encryption is computed using uniform, independent randomness. Using this notation, we have

$$\Pr\left[\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1\right] = \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{LR}^t_{pk}}(pk) = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{LR}^0_{pk}}(pk) = 1]$$

because, from the point of view of $\mathcal{A}$ (who makes exactly t queries), oracle LR^t_{pk} is equivalent to $\mathsf{LR}_{pk,0}$, and oracle LR^0_{pk} is equivalent to $\mathsf{LR}_{pk,1}$. To prove that Π satisfies Definition 11.5, we will show that for any PPT $\mathcal{A}$ there is a negligible function negl' such that

$$\left|\Pr[\mathcal{A}^{\mathsf{LR}^t_{pk}}(pk) = 1] - \Pr[\mathcal{A}^{\mathsf{LR}^0_{pk}}(pk) = 1]\right| \leq \mathsf{negl}'(n). \tag{11.7}$$

(As before, this is equivalent to Definition 11.5 for the same reason that Definition 3.9 is equivalent to Definition 3.8.)

Consider the following PPT adversary $\mathcal{A}'$ that eavesdrops on the encryption of a *single* message:

Adversary $\mathcal{A}'$:

1. $\mathcal{A}'$, given pk, chooses a uniform index $i \leftarrow \{1, \ldots, t\}$.
2. $\mathcal{A}'$ runs $\mathcal{A}(pk)$, answering its jth oracle query $(m_{j,0}, m_{j,1})$ as follows:
   (a) For $j < i$, adversary $\mathcal{A}'$ computes $c_j \leftarrow \mathsf{Enc}_{pk}(m_{j,0})$ and returns c_j to $\mathcal{A}$ as the response from its oracle.
   (b) For $j = i$, adversary $\mathcal{A}'$ outputs $(m_{j,0}, m_{j,1})$ and receives back a challenge ciphertext c_j. This is returned to $\mathcal{A}$ as the response from its oracle.
   (c) For $j > i$, adversary $\mathcal{A}'$ computes $c_j \leftarrow \mathsf{Enc}_{pk}(m_{j,1})$ and returns c_j to $\mathcal{A}$ as the response from its oracle.
3. $\mathcal{A}'$ outputs the bit b' that is output by $\mathcal{A}$.

Consider experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)$. Fixing some choice of $i = i^*$, note that if c_{i^*} is an encryption of $m_{i^*,0}$ then the interaction of $\mathcal{A}$ with its oracle is identical to an interaction with oracle $\mathsf{LR}^{i^*}_{pk}$. Thus,

$$\begin{aligned}\Pr[\mathcal{A}' \text{ outputs } 1 \mid b=0] &= \sum_{i^*=1}^{t} \Pr[i=i^*] \cdot \Pr[\mathcal{A}' \text{ outputs } 1 \mid b=0 \wedge i=i^*] \\ &= \sum_{i^*=1}^{t} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right].\end{aligned}$$

On the other hand, if c_{i^*} is an encryption of $m_{i^*,1}$ then the interaction of $\mathcal{A}$ with its oracle is identical to an interaction with oracle $\mathsf{LR}^{i^*-1}_{pk}$, and so

$$\begin{aligned}\Pr[\mathcal{A}' \text{ outputs } 1 \mid b=1] &= \sum_{i^*=1}^{t} \Pr[i=i^*] \cdot \Pr[\mathcal{A}' \text{ outputs } 1 \mid b=1 \wedge i=i^*] \\ &= \sum_{i^*=1}^{t} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*-1}_{pk}}(pk)=1\right] \\ &= \sum_{i^*=0}^{t-1} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right],\end{aligned}$$

where the third equality is obtained just by shifting the indices of summation.

Since $\mathcal{A}'$ runs in polynomial time, the assumption that Π is CPA-secure means that there exists a negligible function negl such that

$$\left|\Pr[\mathcal{A}' \text{ outputs } 1 \mid b=0] - \Pr[\mathcal{A}' \text{ outputs } 1 \mid b=1]\right| \leq \mathsf{negl}(n).$$

But this means that

$$\begin{aligned}\mathsf{negl}(n) &\geq \left|\sum_{i^*=1}^{t} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right] - \sum_{i^*=0}^{t-1} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right]\right| \\ &= \frac{1}{t} \cdot \left|\Pr\left[\mathcal{A}^{\mathsf{LR}^{t}_{pk}}(pk)=1\right] - \Pr\left[\mathcal{A}^{\mathsf{LR}^{0}_{pk}}(pk)=1\right]\right|,\end{aligned}$$

since all but one of the terms in each summation cancel. We conclude that

$$\left|\Pr\left[\mathcal{A}^{\mathsf{LR}^{t}_{pk}}(pk)=1\right] - \Pr\left[\mathcal{A}^{\mathsf{LR}^{0}_{pk}}(pk)=1\right]\right| \leq t(n) \cdot \mathsf{negl}(n).$$

Because t is polynomial, the function $t \cdot \mathsf{negl}(n)$ is negligible. Since $\mathcal{A}$ was an arbitrary PPT adversary, this shows that Equation (11.7) holds and so completes the proof that Π has indistinguishable multiple encryptions. ■

11.2.3 Security against Chosen-Ciphertext Attacks

Chosen-ciphertext attacks, in which an adversary is able to obtain the decryption of arbitrary ciphertexts of its choice (with one technical restriction described below), are a concern in the public-key setting just as they are in the private-key setting. In fact, they are arguably *more* of a concern in the public-key setting since there a receiver expects to receive ciphertexts from multiple senders who are possibly unknown in advance, whereas a receiver in the private-key setting intends to communicate only with a single, known sender using any particular secret key.

Assume an eavesdropper $\mathcal{A}$ observes a ciphertext c sent by a sender $\mathcal{S}$ to a receiver $\mathcal{R}$. Broadly speaking, in the public-key setting there are two classes of chosen-ciphertext attacks:

- $\mathcal{A}$ might send a modified ciphertext c' to $\mathcal{R}$ *on behalf of* $\mathcal{S}$. (For example, in the context of encrypted e-mail, $\mathcal{A}$ might construct an encrypted e-mail c' and forge the "`From`" field so that it appears the e-mail originated from $\mathcal{S}$.) In this case, although it is unlikely that $\mathcal{A}$ would be able to obtain the entire decryption m' of c', it might be possible for $\mathcal{A}$ to infer some information about m' based on the subsequent behavior of $\mathcal{R}$. Based on this information, $\mathcal{A}$ might be able to learn something about the original message m.

- $\mathcal{A}$ might send a modified ciphertext c' to $\mathcal{R}$ *in its own name*. In this case, $\mathcal{A}$ might obtain the entire decryption m' of c' if $\mathcal{R}$ responds directly to $\mathcal{A}$. Even if $\mathcal{A}$ learns nothing about m', this modified message may have a known *relation* to the original message m that can be exploited by $\mathcal{A}$; see the third scenario below for an example.

The second class of attacks is specific to the context of public-key encryption, and has no analogue in the private-key setting.

It is not hard to identify a number of realistic scenarios illustrating the above types of attacks:

Scenario 1. Say a user $\mathcal{S}$ logs in to her bank account by sending to her bank an encryption of her password pw concatenated with a timestamp. Assume further that there are two types of error messages the bank sends: it returns "password incorrect" if the encrypted password does not match the stored password of $\mathcal{S}$, and "timestamp incorrect" if the password is correct but the timestamp is not.

If an adversary obtains a ciphertext c sent by $\mathcal{S}$ to the bank, the adversary can now mount a chosen-ciphertext attack by sending ciphertexts c' to the bank on behalf of $\mathcal{S}$ and observing the error messages that result. (This is similar to the padding-oracle attack that we saw in Section 3.7.2.) In some cases, this information may be enough to allow the adversary to determine the user's entire password.

Scenario 2. Say $\mathcal{S}$ sends an encrypted e-mail c to $\mathcal{R}$, and this e-mail is observed by $\mathcal{A}$. If $\mathcal{A}$ sends, in its own name, an encrypted e-mail c' to $\mathcal{R}$, then $\mathcal{R}$ might reply to this e-mail *and quote the decrypted text m' corresponding to c'*. In this case, $\mathcal{R}$ is essentially acting as a decryption oracle for $\mathcal{A}$ and might potentially decrypt any ciphertext that $\mathcal{A}$ sends it.

Scenario 3. An issue that is closely related to that of chosen-ciphertext security is potential *malleability* of ciphertexts. Since a formal definition is quite involved, we do not provide one here but instead only give the intuitive idea. A scheme is *malleable* if it has the following property: given an encryption c of some unknown message m, it is possible to come up with a ciphertext c' that is an encryption of a message m' *that is related in some known way to m*. For example, perhaps given an encryption of m, it is possible to construct an encryption of $2m$. (We will see natural examples of CPA-secure schemes with this and similar properties later; see Section 13.2.3.)

Now imagine that $\mathcal{R}$ is running an auction, where two parties $\mathcal{S}$ and $\mathcal{A}$ submit their bids by encrypting them using the public key of $\mathcal{R}$. If a malleable encryption scheme is used, it may be possible for an adversary $\mathcal{A}$ to always place the highest bid (without bidding the maximum) by carrying out the following attack: wait until $\mathcal{S}$ sends a ciphertext c corresponding to its bid m (that is unknown to $\mathcal{A}$); then send a ciphertext c' corresponding to the bid $m' = 2m$. Note that m (and m', for that matter) remain unknown to $\mathcal{A}$ until $\mathcal{R}$ announces the results, and so the possibility of such an attack does not contradict the fact that the encryption scheme is CPA-secure. CCA-secure schemes, on the other hand, can be shown to be non-malleable, meaning they are not vulnerable to such attacks.

The definition. Security against chosen-ciphertext attacks is defined by suitable modification of the analogous definition from the private-key setting (Definition 3.33). Given a public-key encryption scheme Π and an adversary $\mathcal{A}$, consider the following experiment:

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

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *The adversary $\mathcal{A}$ is given pk and access to a decryption oracle $\mathsf{Dec}_{sk}(\cdot)$. It outputs a pair of messages m_0, m_1 of the same length. (These messages must be in the message space associated with pk.)*
3. *A uniform bit $b \in \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ is computed and given to $\mathcal{A}$.*
4. *$\mathcal{A}$ continues to interact with the decryption oracle, but may not request a decryption of c itself. Finally, $\mathcal{A}$ outputs a bit b'.*
5. *The output of the experiment is defined to be* 1 *if $b' = b$, and* 0 *otherwise.*

DEFINITION 11.8 *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under a chosen-ciphertext attack (*or is* CCA-secure) *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

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

The natural analogue of Theorem 11.6 holds for CCA-security as well. That is, if a scheme has indistinguishable encryptions under a chosen-ciphertext attack then it has indistinguishable *multiple* encryptions under a chosen-ciphertext attack, where this is defined appropriately. Interestingly, however, the analogue of Claim 11.7 does *not* hold for CCA-security.

As in Definition 3.33, we must prevent the attacker from submitting the challenge ciphertext c to the decryption oracle for the definition to be achievable. But this restriction does not make the definition meaningless and, in particular, for each of the three motivating scenarios given earlier one can argue that setting $c' = c$ is of no benefit to the attacker:

- In the first scenario involving password-based login, the attacker learns nothing about $\mathcal{S}$'s password by replaying c to the bank since in this case it already knows that no error message will be generated.
- In the second scenario involving encrypted email, sending $c' = c$ to the receiver would likely make the receiver suspicious and so it would refuse to respond at all.
- In the final scenario involving an auction, $\mathcal{R}$ could easily detect cheating if the adversary's encrypted bid is identical to the other party's encrypted bid. Anyway, in that case all the attacker achieves by replaying c is that it submits the *same* bid as the honest party.

11.3 Hybrid Encryption and the KEM/DEM Paradigm

Claim 11.7 shows that any CPA-secure public-key encryption scheme for ℓ'-bit messages can be used to obtain a CPA-secure public-key encryption scheme for messages of arbitrary length. Encrypting an ℓ-bit message using this approach requires $\gamma \stackrel{\text{def}}{=} \lceil \ell/\ell' \rceil$ invocations of the original encryption scheme, meaning that both the computation and the ciphertext length are increased by a multiplicative factor of γ relative to the underlying scheme.

It is possible to do better by using private-key encryption *in tandem with* public-key encryption. This improves efficiency because private-key encryption is significantly faster than public-key encryption, and improves bandwidth because private-key schemes have lower ciphertext expansion. The resulting combination is called *hybrid encryption* and is used extensively in

practice. The basic idea is to use public-key encryption to obtain a shared key k, and then encrypt the message m using a private-key encryption scheme and key k. The receiver uses its long-term (asymmetric) private key to derive k, and then uses private-key decryption (with key k) to recover the original message. We stress that although private-key encryption is used as a component, this is a full-fledged public-key encryption scheme by virtue of the fact that the sender and receiver do not share any secret key *in advance*.

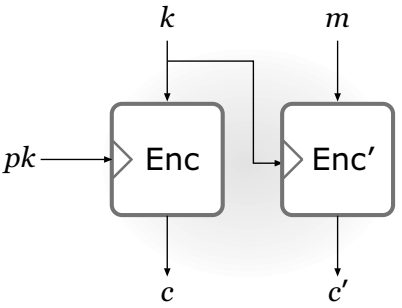

FIGURE 11.1: Hybrid encryption. Enc denotes a public-key encryption scheme, while Enc' is a private-key encryption scheme.

In a direct implementation of this idea (see Figure 11.1), the sender would share k by (1) choosing a uniform value k and then (2) encrypting k using a public-key encryption scheme. A more direct approach is to use a public-key primitive called a *key-encapsulation mechanism* (KEM) to accomplish both of these "in one shot." This is advantageous both from a conceptual point of view and in terms of efficiency, as we will see later.

A KEM has three algorithms similar in spirit to those of a public-key encryption scheme. As before, the key-generation algorithm Gen is used to generate a pair of public and private keys. In place of encryption, we now have an *encapsulation* algorithm Encaps that takes only a public key as input (and no message), and outputs a ciphertext c along with a key k. A corresponding *decapsulation* algorithm Decaps is run by the receiver to recover k from the ciphertext c using the private key. Formally:

DEFINITION 11.9 *A* key-encapsulation mechanism (KEM) *is a tuple of probabilistic polynomial-time algorithms* (Gen, Encaps, Decaps) *such that:*

1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a public-/private-key pair* (pk, sk)*. We assume* pk *and* sk *each has length at least* n*, and that* n *can be determined from* pk*.*

2. *The* encapsulation algorithm Encaps *takes as input a public key* pk *and the security parameter* 1^n*. It outputs a ciphertext* c *and a key* $k \in \{0,1\}^{\ell(n)}$ *where* ℓ *is the* key length*. We write this as* $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$*.*

3. *The deterministic* decapsulation algorithm Decaps *takes as input a private key sk and a ciphertext c, and outputs a key k or a special symbol $\perp$ denoting failure. We write this as $k := \mathsf{Decaps}_{sk}(c)$.*

It is required that with all but negligible probability over (sk, pk) output by $\mathsf{Gen}(1^n)$, if $\mathsf{Encaps}_{pk}(1^n)$ outputs (c, k) then $\mathsf{Decaps}_{sk}(c)$ outputs k.

In the definition we assume for simplicity that Encaps always outputs (a ciphertext c and) a key of some fixed length $\ell(n)$. One could also consider a more general definition in which Encaps takes 1^ℓ as an additional input and outputs a key of length ℓ.

Any public-key encryption scheme trivially gives a KEM by choosing a random key k and encrypting it. As we will see, however, dedicated constructions of KEMs can be more efficient.

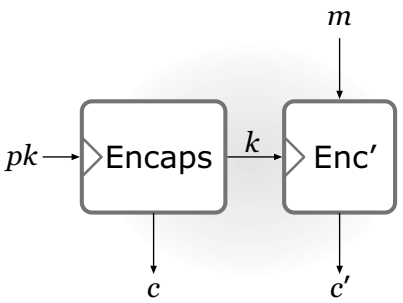

FIGURE 11.2: Hybrid encryption using the KEM/DEM approach.

Using a KEM (with key length n), we can implement hybrid encryption as in Figure 11.2. The sender runs $\mathsf{Encaps}_{pk}(1^n)$ to obtain c along with a key k; it then uses a private-key encryption scheme to encrypt its message m, using k as the key. In this context, the private-key encryption scheme is called a *data-encapsulation mechanism* (DEM) for obvious reasons. The ciphertext sent to the receiver includes both c and the ciphertext c' from the private-key scheme. Construction 11.10 gives a formal specification.

What is the efficiency of the resulting hybrid encryption scheme Π^{hy}? For some fixed value of n, let α denote the cost of encapsulating an n-bit key using Encaps, and let β denote the cost (per bit of plaintext) of encryption using Enc'. Assume $|m| > n$, which is the interesting case. Then the cost, per bit of plaintext, of encrypting a message m using Π^{hy} is

$$\frac{\alpha + \beta \cdot |m|}{|m|} = \frac{\alpha}{|m|} + \beta, \tag{11.8}$$

which approaches β for sufficiently long m. In the limit of very long messages, then, the cost per bit incurred by the *public-key* encryption scheme Π^{hy} is the

CONSTRUCTION 11.10

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM with key length n, and let $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a private-key encryption scheme. Construct a public-key encryption scheme $\Pi^{\mathsf{hy}} = (\mathsf{Gen}^{\mathsf{hy}}, \mathsf{Enc}^{\mathsf{hy}}, \mathsf{Dec}^{\mathsf{hy}})$ as follows:

- $\mathsf{Gen}^{\mathsf{hy}}$: on input 1^n run $\mathsf{Gen}(1^n)$ and use the public and private keys (pk, sk) that are output.
- $\mathsf{Enc}^{\mathsf{hy}}$: on input a public key pk and a message $m \in \{0,1\}^*$ do:
  1. Compute $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.
  2. Compute $c' \leftarrow \mathsf{Enc}'_k(m)$.
  3. Output the ciphertext $\langle c, c' \rangle$.
- $\mathsf{Dec}^{\mathsf{hy}}$: on input a private key sk and a ciphertext $\langle c, c' \rangle$ do:
  1. Compute $k := \mathsf{Decaps}_{sk}(c)$.
  2. Output the message $m := \mathsf{Dec}'_k(c')$.

Hybrid encryption using the KEM/DEM paradigm.

same as the cost per bit of the *private-key* scheme Π'. Hybrid encryption thus allows us to achieve the *functionality* of public-key encryption at the *efficiency* of private-key encryption, at least for sufficiently long messages.

A similar calculation can be used to measure the effect of hybrid encryption on the ciphertext length. For some fixed value of n, let L denote the length of the ciphertext output by Encaps, and say the private-key encryption of a message m using Enc' results in a ciphertext of length $n + |m|$ (this can be achieved using one of the modes of encryption discussed in Section 3.6; actually, even ciphertext length $|m|$ is possible since, as we will see, Π' need not be CPA-secure). Then the total length of a ciphertext in scheme Π^{hy} is

$$L + n + |m|. \tag{11.9}$$

In contrast, when using block-by-block encryption as in Equation (11.1), and assuming that public-key encryption of an n-bit message using Enc results in a ciphertext of length L, encryption of a message m would result in a ciphertext of length $L \cdot \lceil |m|/n \rceil$. The ciphertext length reported in Equation (11.9) is a significant improvement for sufficiently long m.

We can use some rough estimates to get a sense for what the above results mean in practice. (We stress that these numbers are only meant to give the reader a feel for the improvement; realistic values would depend on a variety of factors.) A typical value for the length of the key k might be $n = 128$. Furthermore, a "native" public-key encryption scheme might yield 256-bit ciphertexts when encrypting 128-bit messages; assume a KEM has ciphertexts of the same length when encapsulating a 128-bit key. Letting α, as before, denote the computational cost of public-key encryption/encapsulation of a 128-bit key, we see that block-by-block encryption as in Equation (11.1) would

encrypt a 1 MB ($= 10^6$-bit) message with computational cost $\alpha \cdot \lceil 10^6/128 \rceil \approx 7800 \cdot \alpha$ and the ciphertext would be 2 MB long. Compare this to the efficiency of hybrid encryption. Letting β, as before, denote the per-bit computational cost of private-key encryption, a reasonable approximation is $\beta \approx \alpha/10^5$. Using Equation (11.8), we see that the overall computational cost for hybrid encryption for a 1 Mb message is

$$\alpha + 10^6 \cdot \frac{\alpha}{10^5} = 11 \cdot \alpha \,,$$

and the ciphertext would be only slightly longer than 1 MB. Thus, hybrid encryption improves the computational efficiency in this case by a factor of 700, and the ciphertext length by a factor of 2.

It remains to analyze the security of Π^{hy}. This, of course, depends on the security of its underlying components Π and Π'. In the following sections we define notions of CPA-security and CCA-security for KEMs, and show:

- If Π is a CPA-secure KEM and the private-key scheme Π' has indistinguishable encryptions in the presence of an eavesdropper, then Π^{hy} is a CPA-secure public-key encryption scheme. Notice that it suffices for Π' to satisfy a weaker definition of security—which, recall, does *not* imply CPA-security in the private-key setting—in order for the hybrid scheme Π^{hy} to be CPA-secure. Intuitively, the reason is that a *fresh*, uniform key k is chosen each time a new message is encrypted. Since each key k is used only once, indistinguishability of a single encryption of Π' suffices for security of the hybrid scheme Π^{hy}. This means that basic private-key encryption using a pseudorandom generator (or stream cipher), as in Construction 3.17, suffices.

- If Π is a CCA-secure KEM and Π' is a CCA-secure private-key encryption scheme, then Π^{hy} is a CCA-secure public-key encryption scheme.

11.3.1 CPA-Security

For simplicity, we assume in this and the next section a KEM with key length n. We define a notion of CPA-security for KEMs by analogy with Definition 11.2. As there, the adversary here eavesdrops on a single ciphertext c. Definition 11.2 requires that the attacker is unable to distinguish whether c is an encryption of some message m_0 or some other message m_1. With a KEM there is no message, and we require instead that the encapsulated key k is indistinguishable from a uniform key that is independent of the ciphertext c.

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and $\mathcal{A}$ an arbitrary adversary.

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

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk). *Then* $\mathsf{Encaps}_{pk}(1^n)$ *is run to generate* (c, k) *with* $k \in \{0,1\}^n$.
2. *A uniform bit* $b \in \{0,1\}$ *is chosen. If* $b = 0$ *set* $\hat{k} := k$. *If* $b = 1$ *then choose a uniform* $\hat{k} \in \{0,1\}^n$.
3. *Give* $(pk, c, \hat{k})$ *to* $\mathcal{A}$, *who outputs a bit* b'. *The output of the experiment is defined to be* 1 *if* $b' = b$, *and* 0 *otherwise.*

In the experiment, $\mathcal{A}$ is given the ciphertext c and either the actual key k corresponding to c, or an independent, uniform key. The KEM is CPA-secure if no efficient adversary can distinguish between these possibilities.

DEFINITION 11.11 *A key-encapsulation mechanism* Π *is* CPA-secure *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

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

In the remainder of this section we prove the following theorem:

THEOREM 11.12 *If* Π *is a CPA-secure KEM and* Π' *is a private-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper, then* Π^{hy} *as in Construction 11.10 is a CPA-secure public-key encryption scheme.*

Before proving the theorem formally, we give some intuition. Let the notation "$X \stackrel{\mathrm{c}}{\equiv} Y$" denote the event that no polynomial-time adversary can distinguish between two distributions X and Y. (This concept is treated more formally in Section 7.8, although we do not rely on that section here.) For example, let $\mathsf{Encaps}^{(1)}_{pk}(1^n)$ (resp., $\mathsf{Encaps}^{(2)}_{pk}(1^n)$) denote the ciphertext (resp., key) output by Encaps. The fact that Π is CPA-secure means that

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Encaps}^{(2)}_{pk}(1^n)\right) \stackrel{\mathrm{c}}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), k'\right),$$

where pk is generated by $\mathsf{Gen}(1^n)$ and k' is chosen independently and uniformly from $\{0,1\}^n$. Similarly, the fact that Π' has indistinguishable encryptions in the presence of an eavesdropper means that for any m_0, m_1 output by $\mathcal{A}$ we have $\mathsf{Enc}'_k(m_0) \stackrel{\mathrm{c}}{\equiv} \mathsf{Enc}'_k(m_1)$ if k is chosen uniformly at random.

In order to prove CPA-security of Π^{hy} we need to show that

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0)\right) \stackrel{\mathrm{c}}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_1)\right) \quad (11.10)$$

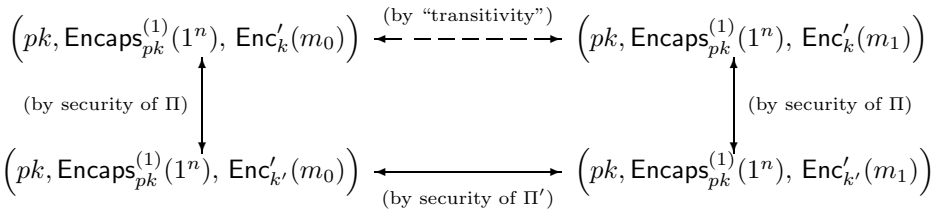

FIGURE 11.3: High-level structure of the proof of Theorem 11.12 (the arrows represent indistinguishability).

for m_0, m_1 output by a PPT adversary $\mathcal{A}$. (Equation (11.10) suffices to show that Π^{hy} has indistinguishable encryptions in the presence of an eavesdropper, and by Proposition 11.3 this implies that Π^{hy} is CPA-secure.)

The proof proceeds in three steps. (See Figure 11.3.) First we prove that

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0)\right) \stackrel{c}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)\right), \quad (11.11)$$

where on the left k is output by $\mathsf{Encaps}^{(2)}_{pk}(1^n)$, and on the right k' is an independent, uniform key. This follows via a fairly straightforward reduction, since CPA-security of Π means exactly that $\mathsf{Encaps}^{(2)}_{pk}(1^n)$ cannot be distinguished from a uniform key k' even given pk and $\mathsf{Encaps}^{(1)}_{pk}(1^n)$.

Next, we prove that

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)\right) \stackrel{c}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)\right). \quad (11.12)$$

Here the difference is between encrypting m_0 or m_1 using Π' *and a uniform, independent key* k'. Equation (11.12) thus follows using the fact that Π' has indistinguishable encryptions in the presence of an eavesdropper.

Exactly as in the case of Equation (11.11), we can also show that

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_1)\right) \stackrel{c}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)\right), \quad (11.13)$$

by relying again on the CPA-security of Π. Equations (11.11)–(11.13) imply, by transitivity, the desired result of Equation (11.10). (Transitivity will be implicit in the proof we give below.)

We now present the full proof.

PROOF (of Theorem 11.12) We prove that Π^{hy} has indistinguishable encryptions in the presence of an eavesdropper; by Proposition 11.3, this implies it is CPA-secure.

Fix an arbitrary PPT adversary $\mathcal{A}^{\mathsf{hy}}$, and consider experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}^{\mathsf{hy}},\Pi^{\mathsf{hy}}}(n)$. Our goal is to prove that there is a negligible function negl such that

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

By definition of the experiment, we have

$$\begin{aligned}
&\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}^{\mathsf{hy}},\Pi^{\mathsf{hy}}}(n)=1] \qquad (11.14)\\
&= \frac{1}{2}\cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0))=0]\\
&\quad + \frac{1}{2}\cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_1))=1],
\end{aligned}$$

where in each case k equals $\mathsf{Encaps}^{(2)}_{pk}(1^n)$. Consider the following PPT adversary $\mathcal{A}_1$ attacking Π.

Adversary $\mathcal{A}_1$:

1. $\mathcal{A}_1$ is given $(pk, c, \hat{k})$.
2. $\mathcal{A}_1$ runs $\mathcal{A}^{\mathsf{hy}}(pk)$ to obtain two messages m_0, m_1. Then $\mathcal{A}_1$ computes $c' \leftarrow \mathsf{Enc}'_{\hat{k}}(m_0)$, gives ciphertext $\langle c, c'\rangle$ to $\mathcal{A}^{\mathsf{hy}}$, and outputs the bit b' that $\mathcal{A}^{\mathsf{hy}}$ outputs.

Consider the behavior of $\mathcal{A}_1$ when attacking Π in experiment $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A}_1,\Pi}(n)$. When $b=0$ in that experiment, then $\mathcal{A}_1$ is given $(pk, c, \hat{k})$ where c and $\hat{k}$ were both output by $\mathsf{Encaps}_{pk}(1^n)$. This means that $\mathcal{A}^{\mathsf{hy}}$ is given a ciphertext of the form $\langle c, c'\rangle = \langle c, \mathsf{Enc}'_k(m_0)\rangle$, where k is the key encapsulated by c. So,

$$\Pr[\mathcal{A}_1 \text{ outputs } 0 \mid b=0] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0))=0].$$

On the other hand, when $b=1$ in experiment $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A}_1,\Pi}(n)$ then $\mathcal{A}_1$ is given $(pk, c, \hat{k})$ with $\hat{k}$ uniform and independent of c. If we denote such a key by k', this means $\mathcal{A}^{\mathsf{hy}}$ is given a ciphertext of the form $\langle c, \mathsf{Enc}'_{k'}(m_0)\rangle$, and

$$\Pr[\mathcal{A}_1 \text{ outputs } 1 \mid b=1] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0))=1].$$

Since Π is a CPA-secure KEM, there is a negligible function negl_1 such that

$$\begin{aligned}
\frac{1}{2}+\mathsf{negl}_1(n) &\geq \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A}_1,\Pi}(n)=1] \qquad (11.15)\\
&= \frac{1}{2}\cdot\Pr[\mathcal{A}_1 \text{ outputs } 0 \mid b=0] + \frac{1}{2}\cdot\Pr[\mathcal{A}_1 \text{ outputs } 1 \mid b=1]\\
&= \frac{1}{2}\cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0))=0]\\
&\quad + \frac{1}{2}\cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0))=1]
\end{aligned}$$

where k is equal to $\mathsf{Encaps}^{(2)}_{pk}(1^n)$ and k' is a uniform and independent key.

Next, consider the following PPT adversary $\mathcal{A}'$ that eavesdrops on a message encrypted using the private-key scheme Π'.

Adversary $\mathcal{A}'$:

1. $\mathcal{A}'(1^n)$ runs $\mathsf{Gen}(1^n)$ on its own to generate keys (pk, sk). It also computes $c \leftarrow \mathsf{Encaps}^{(1)}_{pk}(1^n)$.
2. $\mathcal{A}'$ runs $\mathcal{A}^{\mathsf{hy}}(pk)$ to obtain two messages m_0, m_1. These are output by $\mathcal{A}'$, and it is given in return a ciphertext c'.
3. $\mathcal{A}'$ gives the ciphertext $\langle c, c' \rangle$ to $\mathcal{A}^{\mathsf{hy}}$, and outputs the bit b' that $\mathcal{A}^{\mathsf{hy}}$ outputs.

When $b = 0$ in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi'}(n)$, adversary $\mathcal{A}'$ is given a ciphertext c' which is an encryption of m_0 using a key k' that is uniform and independent of anything else. So $\mathcal{A}^{\mathsf{hy}}$ is given a ciphertext of the form $\langle c, \mathsf{Enc}'_{k'}(m_0) \rangle$ where k' is uniform and independent of c, and

$$\Pr[\mathcal{A}' \text{ outputs } 0 \mid b = 0] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)) = 0].$$

On the other hand, when $b = 1$ in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi'}(n)$, then $\mathcal{A}'$ is given an encryption of m_1 using a uniform, independent key k'. This means $\mathcal{A}^{\mathsf{hy}}$ is given a ciphertext of the form $\langle c, \mathsf{Enc}'_{k'}(m_1) \rangle$ and so

$$\Pr[\mathcal{A}' \text{ outputs } 1 \mid b = 1] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)) = 1].$$

Since Π' has indistinguishable encryptions in the presence of an eavesdropper, there is a negligible function negl' such that

$$\begin{aligned}
\frac{1}{2} + \mathsf{negl}'(n) &\geq \Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi'}(n) = 1] \qquad (11.16)\\
&= \frac{1}{2} \cdot \Pr[\mathcal{A}' \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}' \text{ outputs } 1 \mid b = 1]\\
&= \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)) = 0]\\
&\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)) = 1].
\end{aligned}$$

Proceeding exactly as we did to prove Equation (11.15), we can show that there is a negligible function negl_2 such that

$$\begin{aligned}
\frac{1}{2} + \mathsf{negl}_2(n) &\geq \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A}_2,\Pi}(n) = 1] \qquad (11.17)\\
&= \frac{1}{2} \cdot \Pr[\mathcal{A}_2 \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}_2 \text{ outputs } 1 \mid b = 1]\\
&= \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k}(m_1)) = 1]\\
&\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)) = 0].
\end{aligned}$$

Summing Equations (11.15)–(11.17) and using the fact that the sum of three negligible functions is negligible, we see there exists a negligible function negl such that

$$\begin{aligned}
&\frac{3}{2} + \mathsf{negl}(n) \geq \\
&\frac{1}{2} \cdot \Big(\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_k(m_0)) = 0] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 1] \\
&\quad + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 0] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 1] \\
&\quad + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_k(m_1)) = 1] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 0] \Big),
\end{aligned}$$

where $c = \mathsf{Encaps}^{(1)}_{pk}(1^n)$ in all the above. Note that

$$\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 1] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 0] = 1,$$

since the probabilities of complementary events always sum to 1. Similarly,

$$\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 1] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 0] = 1.$$

Therefore,

$$\begin{aligned}
&\frac{1}{2} + \mathsf{negl}(n) \\
&\geq \frac{1}{2} \cdot \Big(\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_k(m_0)) = 0] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_k(m_1)) = 1] \Big) \\
&= \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}^{\mathsf{hy}}, \Pi^{\mathsf{hy}}}(n) = 1]
\end{aligned}$$

(using Equation (11.14) for the last equality), proving the theorem. ■

11.3.2 CCA-Security

If the private-key encryption scheme Π' is not itself secure against chosen-ciphertext attacks, then (regardless of the KEM used) neither is the resulting hybrid encryption scheme Π^{hy}. As a simple, illustrative example, say we take Construction 3.17 as our private-key encryption scheme. Then, leaving the KEM unspecified, encryption of a message m by Π^{hy} is done by computing $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$ and then outputting the ciphertext

$$\langle c, G(k) \oplus m \rangle,$$

where G is a pseudorandom generator. Given a ciphertext $\langle c, c' \rangle$, an attacker can simply flip the last bit of c' to obtain a modified ciphertext that is a valid encryption of m with its last bit flipped.

The natural way to fix this is to use a CCA-secure private-key encryption scheme. But this is clearly not enough if the KEM is susceptible to chosen-ciphertext attacks. Since we have not yet defined this notion, we do so now.

As in Definition 11.11, we require that an adversary given a ciphertext c cannot distinguish the key k encapsulated by that ciphertext from a uniform and independent key k'. Now, however, we additionally allow the attacker to request *decapsulation* of ciphertexts of its choice (as long as they are different from the challenge ciphertext).

Formally, let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM with key length n and $\mathcal{A}$ an adversary, and consider the following experiment:

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

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk). *Then* $\mathsf{Encaps}_{pk}(1^n)$ *is run to generate* (c, k) *with* $k \in \{0,1\}^n$.
2. $\mathcal{A}$ *uniform bit* $b \in \{0,1\}$ *is chosen. If* $b = 0$ *set* $\hat{k} := k$. *If* $b = 1$ *then choose a uniform* $\hat{k} \in \{0,1\}^n$.
3. $\mathcal{A}$ *is given* $(pk, c, \hat{k})$ *and access to an oracle* $\mathsf{Decaps}_{sk}(\cdot)$, *but may not request decapsulation of* c *itself.*
4. $\mathcal{A}$ *outputs a bit* b'. *The output of the experiment is defined to be* 1 *if* $b' = b$, *and* 0 *otherwise.*

DEFINITION 11.13 *A key-encapsulation mechanism* Π *is* CCA-secure *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

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

Fortunately, we can show that using a CCA-secure KEM in combination with a CCA-secure private-key encryption scheme results in a public-key encryption scheme secure against chosen-ciphertext attacks.

THEOREM 11.14 *If* Π *is a CCA-secure KEM and* Π' *is a CCA-secure private-key encryption scheme, then* Π^{hy} *as in Construction 11.10 is a CCA-secure public-key encryption scheme.*

A proof is obtained by suitable modification of the proof of Theorem 11.12.

11.4 CDH/DDH-Based Encryption

So far we have discussed public-key encryption abstractly, but have not yet seen any concrete examples of public-key encryption schemes (or KEMs). Here we explore some constructions based on the *Diffie–Hellman problems*. (The Diffie–Hellman problems are introduced in Section 8.3.2.)

11.4.1 El Gamal Encryption

In 1985, Taher El Gamal observed that the Diffie–Hellman key-exchange protocol (cf. Section 10.3) could be adapted to give a public-key encryption scheme. Recall that in the Diffie–Hellman protocol, Alice sends a message to Bob and then Bob responds with a message to Alice; based on these messages, Alice and Bob can derive a shared value k which is indistinguishable (to an eavesdropper) from a uniform element of some group $\mathbb{G}$. We could imagine Bob using that shared value to encrypt a message $m \in \mathbb{G}$ by simply sending $k \cdot m$ to Alice; Alice can clearly recover m using her knowledge of k, and we will argue below that an eavesdropper learns nothing about m.

In the *El Gamal encryption scheme* we simply change our perspective on the above interaction. We view Alice's initial message as her public key, and Bob's reply (both his initial response and $k \cdot m$) as a ciphertext. CPA-security based on the decisional Diffie–Hellman (DDH) assumption follows fairly easily from security of the Diffie–Hellman key-exchange protocol (Theorem 10.3).

In our formal treatment, we begin by stating and proving a simple lemma that underlies the El Gamal encryption scheme. Let $\mathbb{G}$ be a finite group, and let $m \in \mathbb{G}$ be an arbitrary element. The lemma states that multiplying m by a uniform group element k yields a uniformly distributed group element k'. Importantly, the distribution of k' is independent of m; this means that k' *contains no information about* m.

LEMMA 11.15 *Let $\mathbb{G}$ be a finite group, and let $m \in \mathbb{G}$ be arbitrary. Then choosing uniform $k \in \mathbb{G}$ and setting $k' := k \cdot m$ gives the same distribution for k' as choosing uniform $k' \in \mathbb{G}$. Put differently, for any $\hat{g} \in \mathbb{G}$ we have*

$$\Pr[k \cdot m = \hat{g}] = 1/|\mathbb{G}|,$$

where the probability is taken over uniform choice of $k \in \mathbb{G}$.

PROOF Let $\hat{g} \in \mathbb{G}$ be arbitrary. Then

$$\Pr[k \cdot m = \hat{g}] = \Pr[k = \hat{g} \cdot m^{-1}].$$

Since k is uniform, the probability that k is equal to the fixed element $\hat{g} \cdot m^{-1}$ is exactly $1/|\mathbb{G}|$. ■

The above lemma suggests a way to construct a perfectly secret *private-key* encryption scheme with message space $\mathbb{G}$. The sender and receiver share as their secret key a uniform element $k \in \mathbb{G}$. To encrypt the message $m \in \mathbb{G}$, the sender computes the ciphertext $k' := k \cdot m$. The receiver can recover the message from the ciphertext k' by computing $m := k'/k$. Perfect secrecy follows immediately from the lemma above. In fact, we have already seen this scheme in a different guise—the one-time pad encryption scheme is an

instantiation of this approach, with the underlying group being the set of strings of some fixed length under the operation of bit-wise XOR.

We can adapt the above ideas to the public-key setting by providing the parties with a way to generate a shared, "random-looking" value k by interacting over a public channel. This should sound familiar since it is exactly what the Diffie–Hellman protocol provides. We proceed with the details.

As in Section 8.3.2, let $\mathcal{G}$ be a polynomial-time algorithm that takes as input 1^n and (except possibly with negligible probability) outputs a description of a cyclic group $\mathbb{G}$, its order q (with $\|q\| = n$), and a generator g. The El Gamal encryption scheme is described in Construction 11.16.

CONSTRUCTION 11.16

Let $\mathcal{G}$ be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. Then choose a uniform $x \in \mathbb{Z}_q$ and compute $h := g^x$. The public key is $\langle \mathbb{G}, q, g, h \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$. The message space is $\mathbb{G}$.
- Enc: on input a public key $pk = \langle \mathbb{G}, q, g, h \rangle$ and a message $m \in \mathbb{G}$, choose a uniform $y \in \mathbb{Z}_q$ and output the ciphertext
$$\langle g^y,\ h^y \cdot m \rangle.$$
- Dec: on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $\langle c_1, c_2 \rangle$, output
$$\hat{m} := c_2 / c_1^x.$$

The El Gamal encryption scheme.

To see that decryption succeeds, let $\langle c_1, c_2 \rangle = \langle g^y,\ h^y \cdot m \rangle$ with $h = g^x$. Then

$$\hat{m} = \frac{c_2}{c_1^x} = \frac{h^y \cdot m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{g^{xy}} = \frac{g^{xy} \cdot m}{g^{xy}} = m.$$

Example 11.17

Let $q = 83$ and $p = 2q + 1 = 167$, and let $\mathbb{G}$ denote the group of *quadratic residues* (i.e., squares) modulo p. (Since p and q are prime, $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ with order q. See Section 8.3.3.) Since the order of $\mathbb{G}$ is prime, any element of $\mathbb{G}$ except 1 is a generator; take $g = 2^2 = 4 \bmod 167$. Say the receiver chooses secret key $37 \in \mathbb{Z}_{83}$ and so the public key is

$$pk = \langle p, q, g, h \rangle = \langle 167, 83, 4, [4^{37} \bmod 167] \rangle = \langle 167, 83, 4, 76 \rangle,$$

where we use p to represent $\mathbb{G}$ (it is assumed that the receiver knows that the group is the set of quadratic residues modulo p).

Say a sender encrypts the message $m = 65 \in \mathbb{G}$ (note $65 = 30^2 \bmod 167$ and so 65 is an element in the subgroup). If $y = 71$, the ciphertext is

$$\langle [4^{71} \bmod 167],\ [76^{71} \cdot 65 \bmod 167] \rangle = \langle 132,\ 44 \rangle.$$

To decrypt, the receiver first computes $124 = [132^{37} \bmod 167]$; then, since $66 = [124^{-1} \bmod 167]$, the receiver recovers $m = 65 = [44 \cdot 66 \bmod 167]$. ♢

We now prove security of the scheme. (The reader may want to compare the proof of the following to the proofs of Theorems 3.18 and 10.3.)

THEOREM 11.18 *If the DDH problem is hard relative to $\mathcal{G}$, then the El Gamal encryption scheme is CPA-secure.*

PROOF Let Π denote the El Gamal encryption scheme. We prove that Π has indistinguishable encryptions in the presence of an eavesdropper; by Proposition 11.3, this implies it is CPA-secure.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We want to show that there is a negligible function negl such that

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

Consider the modified "encryption scheme" $\widetilde{\Pi}$ where Gen is the same as in Π, but encryption of a message m with respect to the public key $\langle \mathbb{G}, q, g, h \rangle$ is done by choosing uniform $y, z \in \mathbb{Z}_q$ and outputting the ciphertext

$$\langle g^y,\ g^z \cdot m \rangle.$$

Although $\widetilde{\Pi}$ is not actually an encryption scheme (as there is no way for the receiver to decrypt), the experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)$ is still well-defined since that experiment depends only on the key-generation and encryption algorithms.

Lemma 11.15 and the discussion that immediately follows it imply that the second component of the ciphertext in scheme $\widetilde{\Pi}$ is a uniformly distributed group element and, in particular, is independent of the message m being encrypted. (Remember that g^z is a uniform element of $\mathbb{G}$ when z is chosen uniformly from $\mathbb{Z}_q$.) The first component of the ciphertext is trivially independent of m. Taken together, this means that the entire ciphertext contains no information about m. It follows that

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

Now consider the following PPT algorithm D that attempts to solve the DDH problem relative to $\mathcal{G}$. Recall that D receives $(\mathbb{G}, q, g, h_1, h_2, h_3)$ where $h_1 = g^x$, $h_2 = g^y$, and h_3 is either g^{xy} or g^z (for uniform x, y, z); the goal of D is to determine which is the case.

Algorithm D:
The algorithm is given $(\mathbb{G}, q, g, h_1, h_2, h_3)$ as input.

- Set $pk = \langle \mathbb{G}, q, g, h_1 \rangle$ and run $\mathcal{A}(pk)$ to obtain two messages $m_0, m_1 \in \mathbb{G}$.
- Choose a uniform bit b, and set $c_1 := h_2$ and $c_2 := h_3 \cdot m_b$.
- Give the ciphertext $\langle c_1, c_2 \rangle$ to $\mathcal{A}$ and obtain an output bit b'. If $b' = b$, output 1; otherwise, output 0.

Let us analyze the behavior of D. There are two cases to consider:

Case 1: Say the input to D is generated by running $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$, then choosing uniform $x, y, z \in \mathbb{Z}_q$, and finally setting $h_1 := g^x$, $h_2 := g^y$, and $h_3 := g^z$. Then D runs $\mathcal{A}$ on a public key constructed as

$$pk = \langle \mathbb{G}, q, g, g^x \rangle$$

and a ciphertext constructed as

$$\langle c_1, c_2 \rangle = \langle g^y,\ g^z \cdot m_b \rangle.$$

We see that in this case the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to $\mathcal{A}$'s view in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)$. Since D outputs 1 exactly when the output b' of $\mathcal{A}$ is equal to b, we have that

$$\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] = \frac{1}{2}.$$

Case 2: Say the input to D is generated by running $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$, then choosing uniform $x, y \in \mathbb{Z}_q$, and finally setting $h_1 := g^x$, $h_2 := g^y$, and $h_3 := g^{xy}$. Then D runs $\mathcal{A}$ on a public key constructed as

$$pk = \langle \mathbb{G}, q, g, g^x \rangle$$

and a ciphertext constructed as

$$\langle c_1, c_2 \rangle = \langle g^y,\ g^{xy} \cdot m_b \rangle = \langle g^y,\ (g^x)^y \cdot m_b \rangle.$$

We see that in this case the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to $\mathcal{A}$'s view in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$. Since D outputs 1 exactly when the output b' of $\mathcal{A}$ is equal to b, we have that

$$\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1].$$

Under the assumption that the DDH problem is hard relative to $\mathcal{G}$, there is a negligible function negl such that

$$\begin{aligned} \mathsf{negl}(n) &\geq \left| \Pr[D(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[D(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] \right| \\ &= \left| \frac{1}{2} - \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \right|. \end{aligned}$$

This implies $\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$, completing the proof. ∎

El Gamal Implementation Issues

We briefly discuss some practical issues related to El Gamal encryption.

Sharing public parameters. Our description of the El Gamal encryption scheme in Construction 11.16 requires the receiver to run $\mathcal{G}$ to generate $\mathbb{G}, q, g$. In practice, it is common for these parameters to be generated and fixed "once-and-for-all," and then shared by multiple receivers. (Of course, each receiver must choose their own secret value x and publish their own public key $h = g^x$.) For example, NIST has published a set of recommended parameters suitable for use in the El Gamal encryption scheme. Sharing parameters in this way does not impact security (assuming the parameters were generated correctly and honestly in the first place). Looking ahead, we remark that this is in contrast to the case of RSA, where parameters cannot safely be shared (see Section 11.5.1).

Choice of group. As discussed in Section 8.3.2, the group order q is generally chosen to be prime. As far as specific groups are concerned, elliptic curves are one increasingly popular choice; an alternative is to let $\mathbb{G}$ be a prime-order subgroup of $\mathbb{Z}_p^*$, for p prime. We refer to Section 9.3 for a tabulation of recommended key lengths for achieving different levels of security.

The message space. An inconvenient aspect of the El Gamal encryption scheme is that the message space is a group $\mathbb{G}$ rather than bit-strings of some specified length. For some choices of the group, it is possible to address this by defining a reversible encoding of bit-strings as group elements. In such cases, the sender can first encode their message $m \in \{0,1\}^\ell$ as a group element $\hat{m} \in \mathbb{G}$ and then apply El Gamal encryption to $\hat{m}$. The receiver can decrypt as in Construction 11.16 to obtain the encoded message $\hat{m}$, and then reverse the encoding to recover the original message m.

A simpler approach is to use (a variant of) El Gamal encryption as part of a hybrid encryption scheme. For example, the sender could choose a uniform group element $m \in \mathbb{G}$, encrypt this using the El Gamal encryption scheme, and then encrypt their actual message using a private-key encryption scheme and key $H(m)$, where $H : \mathbb{G} \to \{0,1\}^n$ is an appropriate key-derivation function (see the following section). In this case, it would be more efficient to use the DDH-based KEM that we describe next.

11.4.2 DDH-Based Key Encapsulation

At the end of the previous section we noted that El Gamal encryption can be used as part of a hybrid encryption scheme by simply encrypting a uniform group element m and using a hash of that element as a key. But this is wasteful! The proof of security for El Gamal encryption shows that c_1^x (where c_1 is the first component of the ciphertext, and x is the private key of the receiver) is already indistinguishable from a uniform group element, so the sender/receiver may as well use that. Construction 11.19 illustrates the KEM

that follows this approach. Note that the resulting encapsulation consists of just a single group element. In contrast, if we were to use El Gamal encryption of a uniform group element, the ciphertext would contain two group elements.

CONSTRUCTION 11.19

Let $\mathcal{G}$ be as in the previous section. Define a KEM as follows:

- Gen: on input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. choose a uniform $x \in \mathbb{Z}_q$ and set $h := g^x$. Also specify a function $H : \mathbb{G} \to \{0,1\}^{\ell(n)}$ for some function ℓ (see text). The public key is $\langle \mathbb{G}, q, g, h, H \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$.
- Encaps: on input a public key $pk = \langle \mathbb{G}, q, g, h, H \rangle$ choose a uniform $y \in \mathbb{Z}_q$ and output the ciphertext g^y and the key $H(h^y)$.
- Decaps: on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $c \in \mathbb{G}$, output the key $H(c^x)$.

An "El Gamal-like" KEM.

As described, the construction leaves the key-derivation function H unspecified, and there are several options for it. (See Section 5.6.4 for more on key derivation in general.) One possibility is to choose a function $H : \mathbb{G} \to \{0,1\}^\ell$ that is (close to) *regular*, meaning that for each possible key $k \in \{0,1\}^\ell$ the number of group elements that map to k is approximately the same. (Formally, we need a negligible function negl such that for each $k \in \{0,1\}^\ell$

$$2^\ell \cdot \left| \Pr[H(g) = k] - 2^{-\ell} \right| \leq \mathsf{negl}(n),$$

where the probability is taken over uniform choice of $g \in \mathbb{G}$. This ensures that the distribution on the key k is statistically close to uniform.) Both the complexity of H, as well as the achievable key length ℓ, will depend on the specific group $\mathbb{G}$ being used.

A second possibility is to let H be a *keyed* function, where the (uniform) key for H is included as part of the receiver's public key. This works if H is a strong extractor, as mentioned briefly in Section 5.6.4. Appropriate choice of ℓ here (to ensure that the resulting key is statistically close to uniform) will depend on the size of $\mathbb{G}$.

In either of the above cases, a proof of CPA-security based on the decisional Diffie–Hellman (DDH) assumption follows easily by adapting the proof of security for the Diffie–Hellman key-exchange protocol (Theorem 10.3).

THEOREM 11.20 *If the DDH problem is hard relative to $\mathcal{G}$, and H is chosen as described, then Construction 11.19 is a CPA-secure KEM.*

If one is willing to model H as a *random oracle*, then Construction 11.19 can be proven CPA-secure based on the (weaker) *computational* Diffie–Hellman (CDH) assumption. We discuss this in the following section.

11.4.3 *A CDH-Based KEM in the Random-Oracle Model

In this section, we show that if one is willing to model H as a *random oracle*, then Construction 11.19 can be proven CPA-secure based on the CDH assumption. (Readers may want to review Section 5.5 to remind themselves of the random-oracle model.) Intuitively, the CDH assumption implies that an attacker observing $h = g^x$ (from the public key) and the ciphertext $c = g^y$ cannot compute $\mathsf{DH}_g(h, c) = h^y$. In particular, then, an attacker cannot query h^y to the random oracle. But this means that the encapsulated key $H(h^y)$ is completely random from the attacker's point of view. This intuition is turned into a formal proof below.

As indicated by the intuition above, the proof inherently relies on modeling H as a random oracle.[3] Specifically, the proof relies on the facts that (1) the only way to learn $H(h^y)$ is to explicitly query h^y to H, which would mean that the attacker has solved a CDH instance (this is called "extractability" in Section 5.5.1), and (2) if an attacker does *not* query h^y to H, then the value $H(h^y)$ is uniform from the attacker's point of view. These properties only hold—indeed, they only make sense—if H is modeled as a random oracle.

THEOREM 11.21 *If the CDH problem is hard relative to $\mathcal{G}$, and H is modeled as a random oracle, then Construction 11.19 is CPA-secure.*

PROOF Let Π denote Construction 11.19, and let $\mathcal{A}$ be a PPT adversary. We want to show that there is a negligible function negl such that

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

The above probability is also taken over uniform choice of the function H, to which $\mathcal{A}$ is given oracle access.

Consider an execution of experiment $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$ in which the public key is $\langle \mathbb{G}, q, g, h \rangle$ and the ciphertext is $c = g^y$, and let Query be the event that $\mathcal{A}$ queries $\mathsf{DH}_g(h, c) = h^y$ to H. We have

$$\begin{aligned}\Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] &= \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathsf{Query}}] \\ &\quad + \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \mathsf{Query}] \\ &\leq \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathsf{Query}}] + \Pr[\mathsf{Query}]. \quad (11.18)\end{aligned}$$

If $\Pr[\overline{\mathsf{Query}}] = 0$ then $\Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathsf{Query}}] = 0$. Otherwise,

$$\begin{aligned}\Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathsf{Query}}] &= \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \mid \overline{\mathsf{Query}}] \cdot \Pr[\overline{\mathsf{Query}}] \\ &\leq \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \mid \overline{\mathsf{Query}}].\end{aligned}$$

[3] This is true as long as we wish to rely only on the CDH assumption. As noted earlier, a proof without random oracles is possible if we rely on the stronger DDH assumption.

In experiment $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$, the adversary $\mathcal{A}$ is given the public key and the ciphertext, plus either the encapsulated key $k \stackrel{\text{def}}{=} H(h^y)$ or a uniform key. If Query does not occur, then k is uniformly distributed from the perspective of the adversary, and so there is no way $\mathcal{A}$ can distinguish between these two possibilities. This means that

$$\Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \mid \overline{\mathsf{Query}}] = \frac{1}{2}.$$

Returning to Equation (11.18), we thus have

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

We next show that $\Pr[\mathsf{Query}]$ is negligible, completing the proof.

Let $t = t(n)$ be a (polynomial) upper bound on the number of queries that $\mathcal{A}$ makes to the random oracle H. Define the following PPT algorithm $\mathcal{A}'$ for the CDH problem relative to $\mathcal{G}$:

> **Algorithm** A':
> The algorithm is given $\mathbb{G}, q, g, h, c$ as input.
>
> - Set $pk = \langle \mathbb{G}, q, g, h \rangle$ and choose a uniform $k \in \{0,1\}^\ell$.
> - Run $\mathcal{A}(pk, c, k)$. When $\mathcal{A}$ makes a query to H, answer it by choosing a fresh, uniform ℓ-bit string.
> - At the end of $\mathcal{A}$'s execution, let $y_1, \ldots, y_t$ be the list of queries that $\mathcal{A}$ has made to H. Choose a uniform index $i \in \{1, \ldots, t\}$ and output y_i.

We are interested in the probability with which $\mathcal{A}'$ solves the CDH problem, i.e., $\Pr[\mathcal{A}'(\mathbb{G}, q, g, h, c) = \mathsf{DH}_g(h, c)]$, where the probability is taken over $\mathbb{G}, q, g$ output by $\mathcal{G}(1^n)$, uniform $h, c \in \mathbb{G}$, and the randomness of $\mathcal{A}'$. To analyze this probability, note first that event Query is still well-defined in the execution of $\mathcal{A}'$, even though $\mathcal{A}'$ cannot detect whether it occurs. Moreover, the probability of event Query when $\mathcal{A}$ is run as a sub-routine by $\mathcal{A}'$ is identical to the probability of event Query in experiment $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$. This follows because the view of $\mathcal{A}$ is identical in both cases until event Query occurs: in each case, $\mathbb{G}, q, g$ are output by $\mathcal{G}(1^n)$; in each case, h and c are uniform elements of $\mathbb{G}$ and k is a uniform, ℓ-bit string; and in each case, queries to H *other than* $H(\mathsf{DH}_g(h, c))$ are answered with a uniform ℓ-bit string. (In $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$, the query $H(\mathsf{DH}_g(h, c))$ is answered with the actual encapsulated key, which is equal to k with probability $1/2$, whereas when $\mathcal{A}$ is run as a subroutine by $\mathcal{A}'$ the query $H(\mathsf{DH}_g(h, c))$ is answered with a uniform ℓ-bit string that is independent of k. But when this query is made, event Query occurs.)

Finally, observe that when Query occurs then $\mathsf{DH}_g(h, c) \in \{y_1, \ldots, y_t\}$ by definition, and so $\mathcal{A}'$ outputs the correct result $\mathsf{DH}_g(h, c)$ with probability at

least $1/t$. We therefore conclude that

$$\Pr[\mathcal{A}'(\mathbb{G}, q, g, h, c) = \mathsf{DH}_g(h, c)] \geq \Pr[\mathsf{Query}]/t,$$

or $\Pr[\mathsf{Query}] \leq t \cdot \Pr[\mathcal{A}'(\mathbb{G}, q, g, h, c) = \mathsf{DH}_g(h, c)]$. Since the CDH problem is hard for $\mathcal{G}$, this latter probability is negligible; since t is polynomial, this implies that $\Pr[\mathsf{Query}]$ is negligible as well. This completes the proof. ■

In the next section we will see that Construction 11.19 can be shown to be CCA-secure under a stronger variant of the CDH assumption (if we continue to model H as a random oracle).

11.4.4 Chosen-Ciphertext Security and DHIES/ECIES

The El Gamal encryption scheme is vulnerable to chosen-ciphertext attacks. This follows from the fact that it is *malleable*. Recall that an encryption scheme is malleable, informally, if given a ciphertext c that is an encryption of some unknown message m, it is possible to generate a modified ciphertext c' that is an encryption of a message m' having some known relation to m. In the case of El Gamal encryption, consider an adversary $\mathcal{A}$ who intercepts a ciphertext $c = \langle c_1, c_2 \rangle$ encrypted using the public key $pk = \langle \mathbb{G}, q, g, h \rangle$, and who then constructs the modified ciphertext $c' = \langle c_1, c_2' \rangle$ where $c_s' = c_2 \cdot \alpha$ for some $\alpha \in \mathbb{G}$. If c is an encryption of a message $m \in \mathbb{G}$ (which may be unknown to $\mathcal{A}$), we have $c_1 = g^y$ and $c_2 = h^y \cdot m$ for some $y \in \mathbb{Z}_q$. But then

$$c_1 = g^y \quad \text{and} \quad c_2' = h^y \cdot (\alpha \cdot m),$$

and so c' is a valid encryption of the message $\alpha \cdot m$. In other words, $\mathcal{A}$ *can transform an encryption of the (unknown) message m into an encryption of the (unknown) message $\alpha \cdot m$.* As discussed in Scenario 3 in Section 11.2.3, this sort of attack can have serious consequences.

The KEM discussed in the previous section might also be malleable depending on the specific key-derivation function H being used. If H is modeled as a random oracle, however, then such attacks no longer seem possible. In fact, one can prove in this case that Construction 11.19 is CCA-secure based on the so-called *gap-CDH assumption*. Recall that the CDH assumption says that given group elements g^x and g^y (for some generator g), it is infeasible to compute g^{xy}. The gap-CDH assumption says that this remains infeasible even given access to an oracle $\mathcal{O}$ such that $\mathcal{O}(U, V)$ returns 1 exactly when $V = U^y$. Stated differently, the CDH problem remains hard even given an oracle that solves the DDH problem. (We do not give a formal definition since we will not use this assumption in the rest of the book.) This assumption is believed to hold for the classes of groups we have discussed in this book. A proof of the following is very similar to the proof of Theorem 11.38.

THEOREM 11.22 *If the gap-CDH problem is hard relative to $\mathcal{G}$, and H is modeled as a random oracle, then Construction 11.19 is a CCA-secure KEM.*

It is interesting to observe that the same construction (namely, Construction 11.19) can be analyzed under different assumptions and in different models, yielding different results. Assuming only that the DDH problem is hard (and for H chosen appropriately), the scheme is CPA-secure. If we model H as a random oracle (which imposes more stringent requirements on H), then under the weaker CDH assumption we obtain CPA-security, and under the stronger gap-CDH assumption we obtain CCA-security.

CONSTRUCTION 11.23

Let $\mathcal{G}$ be as in the text. Let $\Pi_E = (\mathsf{Enc}', \mathsf{Dec}')$ be a private-key encryption scheme, and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code. Define a public-key encryption scheme as follows:

- Gen: On input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. choose a uniform $x \in \mathbb{Z}_q$, set $h := g^x$, and specify a function $H : \mathbb{G} \to \{0,1\}^{2n}$. The public key is $\langle \mathbb{G}, q, g, h, H \rangle$ and the private key is $\langle \mathbb{G}, q, g, x, H \rangle$.
- Enc: On input a public key $pk = \langle \mathbb{G}, q, g, h, H \rangle$, choose a uniform $y \in \mathbb{Z}_q$ and set $k_E \| k_M := H(h^y)$. Compute $c' \leftarrow \mathsf{Enc}'_{k_E}(m)$, and output the ciphertext $\langle g^y, c', \mathsf{Mac}_{k_M}(c') \rangle$.
- Dec: On input a private key $sk = \langle \mathbb{G}, q, g, x, H \rangle$ and a ciphertext $\langle c, c', t \rangle$, output $\perp$ if $c \notin \mathbb{G}$. Else, compute $k_E \| k_M := H(c^x)$. If $\mathsf{Vrfy}_{k_M}(c', t) \neq 1$ then output $\perp$; otherwise, output $\mathsf{Dec}'_{k_E}(c')$.

DHIES/ECIES.

CCA-secure encryption with Construction 11.19. Combining the KEM in Construction 11.19 with any CCA-secure private-key encryption scheme yields a CCA-secure public-key encryption scheme. (See Theorem 11.14.) Instantiating this approach using Construction 4.18 for the private-key component matches what is done in DHIES/ECIES, variants of which are included in the ISO/IEC 18033-2 standard for public-key encryption. (See Construction 11.23.) Encryption of a message m in these schemes takes the form

$$\langle g^y,\ \mathsf{Enc}'_{k_E}(m), \mathsf{Mac}_{k_M}(c') \rangle,$$

where Enc' denotes a CPA-secure private-key encryption scheme and c' denotes $\mathsf{Enc}'_{k_E}(m)$. DHIES, the *Diffie–Hellman Integrated Encryption Scheme*, can be used generically to refer to any scheme of this form, or to refer specifically to the case when the group $\mathbb{G}$ is a cyclic subgroup of a finite field. ECIES, the *Elliptic Curve Integrated Encryption Scheme*, refers to the case when $\mathbb{G}$ is an elliptic-curve group. We remark that, in Construction 11.23, it is critical to check during decryption that c, the first component of the ciphertext, is in $\mathbb{G}$. Otherwise, an attacker might request decryption of a malformed

ciphertext $\langle c, c', t\rangle$ in which $c \notin \mathbb{G}$; decrypting such a ciphertext (i.e., without returning $\perp$) might leak information about the private key.

By Theorem 4.19, encrypting a message and then applying a (strong) message authentication code yields a CCA-secure private-key encryption scheme. Combining this with Theorem 11.14, we conclude:

COROLLARY 11.24 *Let Π_E be a CPA-secure private-key encryption scheme, and let Π_M be a strongly secure message authentication code. If the gap-CDH problem is hard relative to $\mathcal{G}$, and H is modeled as a random oracle, then Construction 11.23 is a CCA-secure public-key encryption scheme.*

11.5 RSA Encryption

In this section we turn our attention to encryption schemes based on the *RSA assumption* defined in Section 8.2.4. We remark that although RSA-based encryption is in widespread use today, there is also currently a gradual shift away from using RSA—and toward using CDH/DDH-based cryptosystems relying on elliptic-curve groups—because of the longer key lengths required for RSA-based schemes. We refer to Section 9.3 for further discussion.

11.5.1 Plain RSA

We begin by describing a simple encryption scheme based on the RSA problem. Although the scheme is insecure, it provides a useful starting point for the secure schemes that follow.

Let GenRSA be a PPT algorithm that, on input 1^n, outputs a modulus N that is the product of two n-bit primes, along with integers e, d satisfying $ed = 1 \bmod \phi(N)$. (As usual, the algorithm may fail with negligible probability but we ignore that here.) Recall from Section 8.2.4 that such an algorithm can be easily constructed from any algorithm $\mathsf{GenModulus}$ that outputs a composite modulus N along with its factorization; see Algorithm 11.25.

ALGORITHM 11.25
RSA key generation GenRSA

Input: Security parameter 1^n
Output: N, e, d as described in the text

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return N, e, d

Let N, e, d be as above, and let $c = m^e \bmod N$. RSA encryption relies on the fact that someone who knows d can recover m from c by computing $[c^d \bmod N]$; this works because

$$c^d = (m^e)^d = m^{ed} = m \bmod N,$$

as discussed in Section 8.2.4. On the other hand, *without* knowledge of d (even if N and e are known) the RSA assumption (cf. Definition 8.46) implies that it is difficult to recover m from c, at least if m is chosen uniformly from $\mathbb{Z}_N^*$. This naturally suggests the public-key encryption scheme shown as Construction 11.26: The receiver runs GenRSA to obtain N, e, d; it publishes N and e as its public key, and keeps d in its private key. To encrypt a message[4] $m \in \mathbb{Z}_N^*$, a sender computes the ciphertext $c := [m^e \bmod N]$. As we have just noted, the receiver—who knows d—can decrypt c and recover m.

CONSTRUCTION 11.26

Let GenRSA be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n run GenRSA(1^n) to obtain N, e, and d. The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
- Enc: on input a public key $pk = \langle N, e\rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the ciphertext
$$c := [m^e \bmod N].$$
- Dec: on input a private key $sk = \langle N, d\rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message
$$m := [c^d \bmod N].$$

The plain RSA encryption scheme.

The following gives a worked example of the above (see also Example 8.49).

Example 11.27

Say GenRSA outputs $(N, e, d) = (391, 3, 235)$. (Note that $391 = 17 \cdot 23$ and so $\phi(391) = 16 \cdot 22 = 352$. Moreover, $3 \cdot 235 = 1 \bmod 352$.) So the public key is $(391, 3)$ and the private key is $(391, 235)$.

To encrypt the message $m = 158 \in \mathbb{Z}_{391}^*$ using the public key $(391, 3)$, we simply compute $c := [158^3 \bmod 391] = 295$; this is the ciphertext. To decrypt, the receiver computes $[295^{235} \bmod 391] = 158$. ♢

Is the plain RSA encryption scheme secure? The factoring assumption implies that it is computationally infeasible for an attacker who is given the

[4] We assume that $m \in \mathbb{Z}_N^*$. If factoring N is hard, it is computationally difficult to find an $m \in \{1, \ldots, N-1\}$ with $m \notin \mathbb{Z}_N^*$ (since then $\gcd(m, N)$ is a nontrivial factor of N).

public key to derive the corresponding private key; see Section 8.2.5. This is necessary—but not sufficient—for a public-key encryption scheme to be secure. The RSA assumption implies that if the message m is *chosen uniformly from* $\mathbb{Z}_N^*$ then an eavesdropper given N, e, and c (namely, the public key and the ciphertext) *cannot recover* m. But these are weak guarantees, and fall far short of the level of security we want! In particular, they leave open the possibility that an attacker can recover the message when it is *not* chosen uniformly from $\mathbb{Z}_N^*$ (indeed, when m is chosen from a small range it is easy to see that an attacker can compute m from the public key and ciphertext). In addition, it does not rule out the possibility that an attacker can learn *partial information* about the message, even when it is uniform (in fact, this is known to be possible). Moreover, plain RSA encryption is *deterministic* and so must be insecure, as we have already discussed in Section 11.2.1.

More Attacks on Plain RSA

We have already noted that plain RSA encryption is not CPA-secure. Nevertheless, there may be a temptation to use plain RSA for encrypting "random messages" and/or in situations where leaking a few bits of information about the message is acceptable. We warn against this in general, and provide here just a few examples of what can go wrong.

(Some of the attacks that follow assume $e = 3$. In some cases the attacks can be extended, at least partially, to larger e; in any case, as noted in Section 8.2.4, setting $e = 3$ is often done in practice. The attacks should be taken as demonstrating that Construction 11.26 is inadequate, not as indicating that setting $e = 3$ is necessarily a bad choice.)

A quadratic improvement in recovering m. Since plain RSA encryption is deterministic, we know that if $m < B$ then an attacker can determine m from the ciphertext $c = [m^e \bmod N]$ in time $\mathcal{O}(B)$ using the brute-force attack discussed in Section 11.2.1. One might hope, however, that plain RSA encryption can be used if B is large, i.e., if the message is chosen from a reasonably large set of values. One possible scenario where this might occur is in the context of hybrid encryption (see Section 11.3), where the "message" is a random n-bit key and so $B = 2^n$. Unfortunately, there is a clever attack that recovers m, with high probability, in time roughly $\mathcal{O}(\sqrt{B})$. This can make a significant difference in practice: a 2^{80}-time attack (say) is infeasible, but an attack running in time 2^{40} is relatively easy to carry out.

A description of the attack is given as Algorithm 11.28. In our description, we assume $B = 2^n$ and let $\alpha \in (\frac{1}{2}, 1)$ denote some fixed constant (see below).

The time complexity of the algorithm is dominated by the time required to sort the $2^{\alpha n}$ pairs (r, x_r); this can be done in time $\mathcal{O}(n \cdot 2^{\alpha n})$. Binary search is used in the second-to-last line to check whether there exists an r with $x_r = [s^e \bmod N]$.

We now sketch why the attack recovers m with high probability. Let $c =$

ALGORITHM 11.28
An attack on plain RSA encryption

Input: Public key $\langle N, e\rangle$; ciphertext c
Output: $m < 2^n$ such that $m^e = c \bmod N$

set $T := 2^{\alpha n}$
for $r = 1$ to T:
  $x_r := [c/r^e \bmod N]$
sort the pairs $\{(r, x_r)\}_{r=1}^{T}$ by their second component
for $s = 1$ to T:
  if $x_r \stackrel{?}{=} [s^e \bmod N]$ for some r
    return $[r \cdot s \bmod N]$

$m^e \bmod N$. For appropriate choice of $\alpha > \frac{1}{2}$, it can be shown that if m is a uniform n-bit integer then with high probability there exist r, s with $1 < r \leq s \leq 2^{\alpha n}$ for which $m = r \cdot s$. (For example, if $n = 64$ and so m is a random 64-bit string, then with probability 0.35 there exist r, s of length at most 34 bits such that $m = r \cdot s$. See the references at the end of the chapter for details.) Assuming this to be the case, the above algorithm finds m since

$$c = m^e = (r \cdot s)^e = r^e \cdot s^e \bmod N,$$

and so $x_r = c/r^e = s^e \bmod N$ with $r, s < T$.

Encrypting short messages using small *e*. The previous attack shows how to recover a message m known to be smaller than some bound B in time roughly $\mathcal{O}(\sqrt{B})$. Here we show how to do the same thing in time $\mathsf{poly}(\|N\|)$ if $B \leq N^{1/e}$ (where this means the eth root of N as a real number).

The attack relies on the observation that when $m < N^{1/e}$, raising m to the eth power modulo N involves no modular reduction; i.e., $[m^e \bmod N]$ is equal to the integer m^e. This means that given the ciphertext $c = [m^e \bmod N]$, an attacker can determine m by computing $m := c^{1/e}$ *over the integers* (i.e., not modulo N); this can be done easily in time $\mathsf{poly}(\|c\|) = \mathsf{poly}(\|N\|)$ since finding eth roots is easy over the integers and hard only when working mod N.

For small e this represents a serious weakness of plain RSA encryption. For example, if we take $e = 3$ and assume $\|N\| \approx 1024$ bits, then the attack works even when m is a uniform 300-bit integer; this once again rules out security of plain RSA even when used as part of a hybrid encryption scheme.

Encrypting a partially known message. This attack can be viewed as a generalization of the previous one. It assumes a sender who encrypts a message, part of which is known (something that should not result in an attack when using a secure encryption scheme). Here we rely on a powerful result of Coppersmith that we state without proof:

THEOREM 11.29 *Let $p(x)$ be a polynomial of degree e. Then in time $\mathsf{poly}(\|N\|, e)$ one can find all m such that $p(m) = 0 \bmod N$ and $|m| \leq N^{1/e}$.*

Due to the dependence of the running time on e, the attack is only practical for small e. In what follows we assume $e = 3$ for concreteness.

Assume a sender encrypts a message $m = m_1 \| m_2$ to a receiver with public key $\langle N, 3 \rangle$, where the first portion m_1 of the message is known but the second portion m_2 is not. For concreteness, say m_2 is k bits long, so $m = B \cdot m_1 + m_2$ where we let $B = 2^k$. Given the resulting ciphertext $c = [(m_1 \| m_2)^3 \bmod N]$, an eavesdropper can define $p(x) \stackrel{\text{def}}{=} (2^k \cdot m_1 + x)^3 - c$, a cubic polynomial. This polynomial has m_2 as a root (modulo N), and $|m_2| < B$. Theorem 11.29 thus implies that the attacker can compute m_2 efficiently as long as $B \leq N^{1/3}$. A similar attack works when m_2 is known but m_1 is not.

Encrypting related messages.[5] This attack assumes a sender who encrypts two related messages to the same receiver (something that should not result in an attack when using a secure encryption scheme). Assume the sender encrypts both m and $m + \delta$ to a receiver with public key $\langle N, e \rangle$, where the offset δ is known but m is not. Given the two ciphertexts $c_1 = [m^e \bmod N]$ and $c_2 = [(m + \delta)^e \bmod N]$, an eavesdropper can define the two polynomials $f_1(x) \stackrel{\text{def}}{=} x^e - c_1$ and $f_2(x) \stackrel{\text{def}}{=} (x + \delta)^e - c_2$ (modulo N), each of degree e. Note that $x = m$ is a root (modulo N) of both polynomials, and so the linear term $(x - m)$ is a factor of both. Thus, if the greatest common divisor of $f_1(x)$ and $f_2(x)$ (as polynomials over $\mathbb{Z}_N^*$) is linear, it will reveal m. The greatest common divisor can be computed in time $\mathsf{poly}(\|N\|, e)$ using an algorithm similar to the one in Appendix B.1.2; thus, this attack is feasible for small e.

Sending the same message to multiple receivers.[6] Our final attack assumes a sender who encrypts the same message to multiple receivers (something that, once again, should not result in an attack when using a secure encryption scheme). Let $e = 3$, and say the same message m is encrypted to three different parties holding public keys $pk_1 = \langle N_1, 3 \rangle$, $pk_2 = \langle N_2, 3 \rangle$, and $pk_3 = \langle N_3, 3 \rangle$, respectively. Assume $\gcd(N_i, N_j) = 1$ for distinct i, j; if not, then at least one of the moduli can be factored immediately and the message m can be easily recovered. An eavesdropper sees

$$c_1 = [m^3 \bmod N_1], \quad c_2 = [m^3 \bmod N_2], \quad \text{and} \quad c_3 = [m^3 \bmod N_3].$$

Let $N^* = N_1 N_2 N_3$. An extended version of the Chinese remainder theorem says that there exists a unique non-negative integer $\hat{c} < N^*$ such that

$$\begin{aligned} \hat{c} &= c_1 \bmod N_1 \\ \hat{c} &= c_2 \bmod N_2 \\ \hat{c} &= c_3 \bmod N_3. \end{aligned}$$

[5] This attack relies on some algebra slightly beyond what we have covered in this book.

[6] This attack relies on the Chinese remainder theorem presented in Section 8.1.5.

Moreover, using techniques similar to those shown in Section 8.1.5 it is possible to compute $\hat{c}$ efficiently given the public keys and the above ciphertexts. Note finally that m^3 satisfies the above equations, and $m^3 < N^*$ since $m < \min\{N_1, N_2, N_3\}$. As in the previous attack, this means that $\hat{c} = m^3$ *over the integers* (i.e., with no modular reduction taking place), and so the message m can be recovered by computing the integer cube root of $\hat{c}$.

11.5.2 Padded RSA and PKCS #1 v1.5

Although plain RSA is insecure, it does suggest one general approach to public-key encryption based on the RSA problem: to encrypt a message m using public key $\langle N, e\rangle$, first map m to an element $\hat{m} \in \mathbb{Z}_N^*$; then compute the ciphertext $c = [\hat{m}^e \bmod N]$. To decrypt a ciphertext c, the receiver computes $\hat{m} = [c^d \bmod N]$ and then recovers the original message m. For the receiver to be able to recover the message, the mapping from messages to elements of $\mathbb{Z}_N^*$ must be (efficiently) *reversible*. For a scheme following this approach to have a hope of being CPA-secure, the mapping must be *randomized* so encryption is not deterministic. This is, of course, a necessary condition but not a sufficient one, and security of the encryption scheme depends critically on the specific mapping that is used.

One simple implementation of the above idea is to *randomly pad* the message before encrypting. That is, to map a message m (viewed as a bit-string) to an element of $\mathbb{Z}_N^*$, the sender chooses a uniform bit-string $r \in \{0,1\}^\ell$ (for some appropriate ℓ) and sets $\hat{m} := r \| m$; the resulting value can naturally be interpreted as an integer in $\mathbb{Z}_N^*$, and this mapping is clearly reversible. See Construction 11.30. (The bounds on $\ell(n)$ and the length of m ensure that the integer $\hat{m}$ is smaller than N.)

The construction is parameterized by a value ℓ that determines the length of the random padding used. Security of the scheme depends on ℓ. There is an obvious brute-force attack on the scheme that runs in time 2^ℓ, so if ℓ is too short (in particular, if $\ell(n) = \mathcal{O}(\log n)$), the scheme is insecure. At the other extreme, we show in the following section (essentially) that when the padding is as large as possible, and m is just a single bit, then it is possible to prove security based on the RSA assumption. In intermediate cases, the situation is less clear: for certain ranges of ℓ we cannot prove security based on the RSA assumption but no polynomial-time attacks are known either. We defer further discussion until after our treatment of PKCS #1 v1.5 next.

RSA PKCS #1 v1.5. The *RSA Laboratories Public-Key Cryptography Standard (PKCS) #1 version 1.5*, issued in 1993, utilizes a variant of padded RSA encryption. For a public key $pk = \langle N, e\rangle$ of the usual form, let k denote the length of N in bytes; i.e., k is the integer satisfying $2^{8(k-1)} \leq N < 2^{8k}$. Messages m to be encrypted are assumed to be a multiple of 8 bits long and can have length anywhere from one to $k - 11$ bytes. Encryption of a message

CONSTRUCTION 11.30

Let GenRSA be as before, and let ℓ be a function with $\ell(n) \leq 2n-4$ for all n. Define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d). Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0,1\}^{\|N\|-\ell(n)-2}$, choose a uniform string $r \in \{0,1\}^{\ell(n)}$ and interpret $\hat{m} := r \| m$ as an element of $\mathbb{Z}_N^*$. Output the ciphertext
$$c := [\hat{m}^e \bmod N].$$
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute
$$\hat{m} := [c^d \bmod N],$$
and output the $\|N\| - \ell(n) - 2$ least-significant bits of $\hat{m}$.

The padded RSA encryption scheme.

m that is D-bytes long is computed as

$$[(\mathtt{0x00} \| \mathtt{0x02} \| r \| \mathtt{0x00} \| m)^e \bmod N],$$

where r is a randomly generated, $(k-D-3)$-byte string with none of its bytes equal to 0x00. (This latter condition enables the message to be unambiguously recovered upon decryption.) Note that the maximum allowed length of m ensures that the length of r is at least 8 bytes.

Unfortunately, PKCS #1 v1.5 as specified is not CPA-secure because it allows using random padding that is too short. This is best illustrated by showing that an attacker can determine the initial portion of a message known to have many trailing 0s. For simplicity, say $m = b \| \underbrace{0 \cdots 0}_{L}$ where $b \in \{0,1\}$ is unknown and m is as long as possible (so $L = 8 \cdot (k-11) - 1$). Encryption of m gives a ciphertext c with

$$c = (\mathtt{0x00} \| \mathtt{0x02} \| r \| \mathtt{0x00} \| b \| 0 \cdots 0)^e \bmod N.$$

An attacker can compute $c' = c/(2^L)^e \bmod N$; note that

$$c' = \left(\frac{\mathtt{0x00} \| \mathtt{0x02} \| r \| \mathtt{0x00} \| b \| 0 \cdots 0}{10 \cdots 0} \right)^e = (\mathtt{0x00} \| \mathtt{0x02} \| r \| \mathtt{0x00} \| b)^e \bmod N.$$

The integer $\mathtt{0x02} \| r \| \mathtt{0x00} \| b$ is 75 bits long (note that $\mathtt{0x02} = 0000\,0010$, and all the high-order 0-bits don't count), and so an attacker can now apply the "short-message attack," or the attack based on encrypting a partially known message, from the previous section. To avoid these attacks we need to take r of length at least $\|N\|/e$. Even if e is large, however, the "quadratic-improvement attack" from the previous section shows that r can be recovered, with high probability, in time roughly $2^{\|r\|/2}$.

If we force r to be roughly half the length of N, and correspondingly reduce the maximum message length, then it is reasonable to conjecture that the encryption scheme in PKCS #1 v1.5 is CPA-secure. (We stress, however, that no proof of security based on the RSA assumption is known.) Nevertheless, because of a serious chosen-*ciphertext* attack on the scheme, described briefly in Section 11.5.5, newer versions of the PKCS #1 standard have been introduced and should be used instead.

11.5.3 *CPA-Secure Encryption without Random Oracles

In this section we show an encryption scheme that can be proven to be CPA-secure based on the RSA assumption. We begin by describing a specific hard-core predicate (see Section 7.1.3) for the RSA problem and then show how to use that hard-core predicate to encrypt a single bit. We then extend this scheme to give a KEM.

The schemes described in this section are mainly of theoretical interest and are not used in practice. This is because they are less efficient than alternative RSA-based constructions that can be proven secure in the random-oracle model (cf. Section 5.5). We will see examples of such encryption schemes in the sections that follow.

A hard-core predicate for the RSA problem. Loosely speaking, the RSA assumption says that given N, e, and $[x^e \bmod N]$ (for x chosen uniformly from $\mathbb{Z}_N^*$), it is infeasible to recover x. By itself, this says nothing about the computational difficulty of computing some specific information about x. Can we isolate some particular bit of information about x that is hard to compute from N, e and $[x^e \bmod N]$? The notion of a *hard-core predicate* captures exactly this requirement. (Hard-core predicates were introduced in Section 7.1.3. The fact that the RSA assumption gives a family of one-way permutations is discussed in Section 8.4.1. Our treatment here is self-contained, however.) It turns out that the least-significant bit of x, denoted $\mathsf{lsb}(x)$, is a hard-core predicate for the RSA problem.

Define the following experiment for a given algorithm GenRSA (with the usual behavior) and algorithm $\mathcal{A}$:

The RSA hard-core predicate experiment $\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(1^n)$:

1. *Run* $\mathsf{GenRSA}(1^n)$ *to obtain* (N, e, d).
2. *choose a uniform* $x \in \mathbb{Z}_N^*$ *and compute* $y := [x^e \bmod N]$.
3. $\mathcal{A}$ *is given* N, e, y, *and outputs a bit* b.
4. *The output of the experiment is* 1 *if and only if* $\mathsf{lsb}(x) = b$.

Observe that $\mathsf{lsb}(x)$ is a uniform bit when $x \in \mathbb{Z}_N^*$ is uniform. $\mathcal{A}$ can guess $\mathsf{lsb}(x)$ with probability $1/2$ by simply outputting a uniform bit b. The following theorem states that if the RSA problem is hard, then no efficient

algorithm $\mathcal{A}$ can do significantly better than this; i.e., the least-significant bit is a hard-core predicate of the RSA permutation.

THEOREM 11.31 *If the RSA problem is hard relative to* GenRSA *then for all probabilistic polynomial-time algorithms $\mathcal{A}$ there is a negligible function* negl *such that* $\Pr[\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$.

A full proof of this theorem is beyond the scope of this book. However, we provide some intuition for the theorem by sketching a proof of a weaker result: that the RSA assumption implies $\Pr[\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] < 1$ for all probabilistic polynomial-time $\mathcal{A}$. To prove this we show that an efficient algorithm that *always* correctly computes $\mathsf{lsb}(r)$ from N, e, and $[r^e \bmod N]$ can be used to efficiently recover x (in its entirety) from N, e, and $[x^e \bmod N]$.

Fix N and e, and let $\mathcal{A}$ be an algorithm such that $\mathcal{A}([r^e \bmod N]) = \mathsf{lsb}(r)$. Given N, e, and $y = [x^e \bmod N]$, we will recover the bits of x one-by-one, from least to most significant. To determine $\mathsf{lsb}(x)$ we simply run $\mathcal{A}(y)$. There are now two cases:

Case 1: $\mathsf{lsb}(x) = 0$**.** Note that $y/2^e = (x/2)^e \bmod N$, and because x is even (i.e., $\mathsf{lsb}(x) = 0$), 2 divides the integer x. So $x/2$ is just the right-wise bit-shift of x, and $\mathsf{lsb}(x/2)$ is equal to $\mathsf{2sb}(x)$, the 2nd-least-significant bit of x. So we can obtain $\mathsf{2sb}(x)$ by computing $y' := [y/2^e \bmod N]$ and then running $\mathcal{A}(y')$.

Case 2: $\mathsf{lsb}(x) = 1$**.** Here $[x/2 \bmod N] = (x + N)/2$. So $\mathsf{lsb}([x/2 \bmod N])$ is equal to $\mathsf{2sb}(x + N)$; the latter is equal to $1 \oplus \mathsf{2sb}(N) \oplus \mathsf{2sb}(x)$ (we have a carry bit in the second position because both x and N are odd). So if we compute $y' := [y/2^e \bmod N]$, then $\mathsf{2sb}(x) = \mathcal{A}(y') \oplus 1 \oplus \mathsf{2sb}(N)$.

Continuing in this way, we can recover all the bits of x.

Encrypting one bit. We can use the hard-core predicate identified above to encrypt a single bit. The idea is straightforward: to encrypt the message $m \in \{0, 1\}$, the sender chooses uniform $r \in \mathbb{Z}_N^*$ subject to the constraint that $\mathsf{lsb}(r) = m$; the ciphertext is $c := [r^e \bmod N]$. See Construction 11.32.

THEOREM 11.33 *If the RSA problem is hard relative to* GenRSA *then Construction 11.32 is CPA-secure.*

PROOF Let Π denote Construction 11.32. We prove that Π has indistinguishable encryptions in the presence of an eavesdropper; by Proposition 11.3, this implies it is CPA-secure.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary. Without loss of gener-

CONSTRUCTION 11.32

Let GenRSA be as usual, and define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d). Output the public key $pk = \langle N, e\rangle$, and the private key $sk = \langle N, d\rangle$.
- Enc: on input a public key $pk = \langle N, e\rangle$ and a message $m \in \{0, 1\}$, choose a uniform $r \in \mathbb{Z}_N^*$ subject to the constraint that $\mathsf{lsb}(r) = m$. Output the ciphertext $c := [r^e \bmod N]$.
- Dec: on input a private key $sk = \langle N, d\rangle$ and a ciphertext c, compute $r := [c^d \bmod N]$ and output $\mathsf{lsb}(r)$.

Single-bit encryption using a hard-core predicate for RSA.

ality, we may assume $m_0 = 0$ and $m_1 = 1$ in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$. So

$$\begin{aligned}\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathcal{A}(N, e, c) = 0 \mid c \text{ is an encryption of } 0] \\ + \frac{1}{2} \cdot \Pr[\mathcal{A}(N, e, c) = 1 \mid c \text{ is an encryption of } 1].\end{aligned}$$

Consider running $\mathcal{A}$ in experiment $\mathsf{RSA\text{-}lsb}$. By definition,

$$\Pr[\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] \;=\; \Pr[\mathcal{A}\left(N, e, [r^e \bmod N]\right) = \mathsf{lsb}(r)],$$

where r is uniform in $\mathbb{Z}_N^*$. Since $\Pr[\mathsf{lsb}(r) = 1] = 1/2$, we have

$$\begin{aligned}\Pr[\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathcal{A}\left(N, e, [r^e \bmod N]\right) = 0 \mid \mathsf{lsb}(r) = 0] \\ + \frac{1}{2} \cdot \Pr[\mathcal{A}\left(N, e, [r^e \bmod N]\right) = 1 \mid \mathsf{lsb}(r) = 1].\end{aligned}$$

Noting that encrypting $m \in \{0, 1\}$ corresponds exactly to choosing uniform r subject to the constraint that $\mathsf{lsb}(r) = m$, we see that

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \Pr[\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1].$$

Theorem 11.31 thus implies that there is a negligible function negl such that

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

as desired. ■

Constructing a KEM. We now show how to extend Construction 11.32 so as to obtain a KEM with key length n. A naive way of doing this would be to simply choose a uniform, n-bit key k and then encrypt the bits of k one-by-one

using n invocations of Construction 11.32. This would result in a rather long ciphertext consisting of n elements of $\mathbb{Z}_N$.

A better approach is for the sender to apply the RSA permutation (namely, raising to the eth power modulo N) repeatedly, starting from an initial, uniform value c_1. That is, the sender will successively compute c_1^e, followed by $(c_1^e)^e = c_1^{e^2}$, and so on, up to $c_1^{e^n}$ (all modulo N). The final value $[c_1^{e^n} \bmod N]$ will be the ciphertext, and the sequence of bits $\mathsf{lsb}(c_1), \mathsf{lsb}(c_1^e), \ldots, \mathsf{lsb}(c_1^{e^{n-1}})$ is the key. To decrypt a ciphertext c, the receiver simply reverses this process, successively computing $c^d, (c^d)^d = c^{d^2}$ up to c^{d^n} (again, all modulo N) to recover the initial value $c_1 = c^{d^n}$ used by the sender. Having recovered c_1, the receiver can then recompute $c_1^e, \ldots, c_1^{e^n}$ and obtain the key.

It is possible to implement decryption more efficiently using the fact that the receiver knows the order of the group $\mathbb{Z}_N^*$. At key-generation time, the receiver can pre-compute $d' := [d^n \bmod \phi(N)]$ and store d' as part of its private key. Then, rather than computing the n successive modular exponentiations $c^d, c^{d^2}, \ldots$ to obtain c_1, the receiver can instead directly compute $c_1 := [c^{d'} \bmod N]$. This works, of course, since

$$c^{d^n} \bmod N = c^{[d^n \bmod \phi(N)]} = c^{d'} \bmod N.$$

The above is formally described as Construction 11.34.

CONSTRUCTION 11.34

Let GenRSA be as usual, and define a KEM as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d). Then compute $d' := [d^n \bmod \phi(N)]$ (note that $\phi(N)$ can be computed from (N, e, d) or obtained during the course of running GenRSA). Output $pk = \langle N, e \rangle$ and $sk = \langle N, d' \rangle$.
- Encaps: on input $pk = \langle N, e \rangle$ and 1^n, choose a uniform $c_1 \in \mathbb{Z}_N^*$. Then for $i = 1, \ldots, n$ do:
  1. Compute $k_i := \mathsf{lsb}(c_i)$.
  2. Compute $c_{i+1} := [c_i^e \bmod N]$.

  Output the ciphertext c_{n+1} and the key $k = k_1 \cdots k_n$.
- Decaps: on input $sk = \langle N, d' \rangle$ and a ciphertext c, compute $c_1 := [c^{d'} \bmod N]$. Then for $i = 1, \ldots, n$ do:
  1. Compute $k_i := \mathsf{lsb}(c_i)$.
  2. Compute $c_{i+1} := [c_i^e \bmod N]$.

  Output the key $k = k_1 \cdots k_n$.

A KEM using a hard-core predicate for RSA.

The construction is reminiscent of the approach used to construct a pseudorandom generator from a one-way permutation toward the end of Section 7.4.2. If we let f denote the RSA permutation relative to some public key $\langle N, e\rangle$ (i.e., $f(x) \stackrel{\text{def}}{=} [x^e \bmod N]$), then CPA-security of Construction 11.34 is equivalent to pseudorandomness of $\mathsf{lsb}(f^{n-1}(c_1)), \ldots, \mathsf{lsb}(c_1)$ even conditioned on the value $c = f^n(c_1)$. This, in turn, can be proven using Theorem 11.31 and the techniques from Section 7.4.2. (The only difference is that in Section 7.4.2 the value $f^n(c_1)$ was itself a uniform n-bit string, whereas here it is a uniform element of $\mathbb{Z}_N^*$. Pseudorandomness of the successive hard-core predicates is independent of the domain of f.) Summarizing:

THEOREM 11.35 *If the RSA problem is hard relative to* GenRSA *then Construction 11.34 is a CPA-secure KEM.*

Efficiency. Construction 11.34 is reasonably efficient. To be concrete, assume that $n = 128$, the RSA modulus N is 2048 bits long, and the public exponent e is 3 so that exponentiation to the power e modulo N can be computed using two modular multiplications. (See Appendix B.2.3.) Encryption then requires $2n = 256$ modular multiplications. Decryption can be done with one full modular exponentiation (at the cost of approximately $1.5 \cdot 2048 = 3072$ modular multiplications) plus an additional 256 modular multiplications. The cost of decryption is thus only about 8% less efficient than for the plain RSA encryption scheme. In contrast, encryption is significantly more expensive than in plain RSA, but in many applications decryption time is much more crucial (since it may be implemented by a server that is performing thousands of decryptions simultaneously).

11.5.4 OAEP and RSA PKCS #1 v2.0

We have so far not considered RSA-based encryption schemes that are CCA-secure. We first show that all the RSA-based encryption schemes we have seen so far are vulnerable to chosen-ciphertext attacks.

Plain RSA encryption. Plain RSA is not even CPA-secure. But it does ensure that if $m \in \mathbb{Z}_N^*$ is uniform then an attacker who eavesdrops on the encryption $c = [m^e \bmod N]$ of m with respect to the public key $\langle N, e\rangle$ cannot recover m. Even this weak guarantee no longer holds in a setting where chosen-ciphertext attacks are possible. As in the case of El Gamal encryption, this is a consequence of the fact that plain RSA is *malleable*: given the encryption $c = [m^e \bmod N]$ of an unknown message m, it is easy to generate a ciphertext c' that is an encryption of $[2m \bmod N]$ by setting

$$\begin{aligned} c' &:= [2^e \cdot c \bmod N] \\ &= 2^e \cdot m^e = (2m)^e \bmod N. \end{aligned}$$

RSA PKCS #1 v1.5. Padded RSA encryption, which is conjectured to be CPA-secure for the right setting of the parameters, is vulnerable to essentially the same attack as plain RSA encryption is. But there is also a more interesting chosen-ciphertext attack on PKCS #1 v1.5 encryption that, in contrast to the attack presented above, does *not* require full access to a decryption oracle; it only requires access to a "partial" decryption oracle that indicates whether or not decryption of some ciphertext returns an error. This makes the attack much more practical, as the attack can be carried out whenever an attacker can distinguish the behavior of the receiver after a decryption success from its behavior after a decryption failure, as in the case of the padding-oracle attack shown in Section 3.7.2.

Recall that the public-key encryption scheme defined in the PKCS #1 v1.5 standard uses a variant of padded RSA encryption where the padding is done in a specific way. In particular, the two high-order bytes of the padded message are always $\texttt{0x00}\|\texttt{0x02}$. When decrypting, the receiver is supposed to check that the two high-order bytes match these values, and return an error if this is not the case. In 1998, Bleichenbacher developed a chosen-ciphertext attack that exploits the fact that this check is done. Roughly, given a ciphertext c that corresponds to an honest encryption of some unknown message m with respect to a public key $\langle N, e\rangle$, the attack repeatedly chooses uniform $s \in \mathbb{Z}_N^*$ and submits the ciphertext $c' := [s^e \cdot c \bmod N]$ to the receiver. Say $c = [\hat{m}^e \bmod N]$ where

$$\hat{m} = \texttt{0x00}\|\texttt{0x02}\|r\|\texttt{0x00}\|m,$$

as specified by PKCS #1 v1.5. Then decryption of c' will give the intermediate result $\hat{m}' = [s \cdot \hat{m} \bmod N]$, and the receiver will return an error unless the top two bytes of $\hat{m}'$ are exactly $\texttt{0x00}\|\texttt{0x02}$. (Other checks are done as well, but we ignore those for simplicity.) Any time decryption succeeds, the attacker learns that the top two bytes of $s \cdot \hat{m} \bmod N$ are $\texttt{0x00}\|\texttt{0x02}$, where s is known. A number of equations of this type suffice for the attacker to learn $\hat{m}$ and recover all of the original message m.

The CPA-secure KEM. In Section 11.5.3 we showed a construction of a KEM that can be proven CPA-secure based on the RSA assumption. That construction is also insecure against a chosen-ciphertext attack; we leave the details as an exercise.

RSA-OAEP

In this section we explore a construction of RSA-based CCA-secure encryption using *optimal asymmetric encryption padding* (OAEP). The resulting RSA-OAEP scheme follows the idea (used also in Section 11.5.2) of taking a message m, transforming it to an element $\hat{m} \in \mathbb{Z}_N^*$, and then letting $c = [\hat{m}^e \bmod N]$ be the ciphertext. The transformation here, however, is

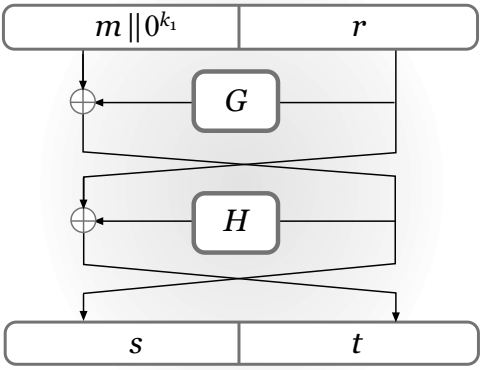

FIGURE 11.4: The OAEP mechanism.

more complex than before. A version of RSA-OAEP has been standardized as part of RSA PKCS #1 since version 2.0.

Let $\ell(n), k_0(n), k_1(n)$ be integer-valued functions with $k_0(n), k_1(n) = \Theta(n)$ and such that $\ell(n) + k_0(n) + k_1(n)$ is less than the minimum bit-length of moduli output by $\mathsf{GenRSA}(1^n)$. Fix n, and let $\ell = \ell(n), k_0 = k_0(n)$, and $k_1 = k_1(n)$. Let $G : \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$ and $H : \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$ be two hash functions that will be modeled as independent random oracles. (Although using more than one random oracle was not discussed in Section 5.5.1, this is interpreted in the natural way.) The transformation defined by OAEP is a two-round Feistel network with G and H as round functions; see Figure 11.4. In detail, padding of a message $m \in \{0,1\}^{\ell}$ is done as follows: first set $m' := m \| 0^{k_1}$ and choose a uniform $r \in \{0,1\}^{k_0}$. Then compute

$$s := m' \oplus G(r) \in \{0,1\}^{\ell+k_1}, \quad t := r \oplus H(s) \in \{0,1\}^{k_0},$$

and set $\hat{m} := s \| t$.

To encrypt a message m with respect to the public key $\langle N, e \rangle$, the sender generates $\hat{m}$ as above and outputs the ciphertext $c := \hat{m}^e \bmod N$. (Note that $\hat{m}$, interpreted as an integer, is less than N because of the constraints on ℓ, k_0, k_1.) To decrypt, the receiver computes $\hat{m} := [c^d \bmod N]$ and lets $s \| t := \hat{m}$ with s and t of the appropriate lengths. It then inverts the Feistel network by computing $r := H(s) \oplus t$ and $m' := G(r) \oplus s$. Importantly, the receiver then verifies that the trailing k_1 bits of m' are all 0; if not, the ciphertext is rejected and an error message is returned. Otherwise, the k_1 least-significant 0s of m' are discarded, and the remaining ℓ bits of m' are output as the message. This process is described in Construction 11.36.

The proof of CCA-security for RSA-OAEP is rather complicated, and we do not give it here. Instead, we merely provide some intuition. First consider

CONSTRUCTION 11.36

Let GenRSA be as in the previous sections, and ℓ, k_0, k_1 be as described in the text. Let $G : \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$ and $H : \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$ be functions. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d). The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
- Enc: on input a public key $\langle N, e\rangle$ and a message $m \in \{0,1\}^\ell$, set $m' := m\|0^{k_1}$ and choose a uniform $r \in \{0,1\}^{k_0}$. Then compute

$$s := m' \oplus G(r), \quad t := r \oplus H(s)$$

  and set $\hat{m} := s\|t$. Output the ciphertext $c := [\hat{m}^e \bmod N]$.
- Dec: on input a private key $\langle N, d\rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $\hat{m} := [c^d \bmod N]$. If $\|\hat{m}\| > \ell + k_0 + k_1$, output $\perp$. Otherwise, parse $\hat{m}$ as $s\|t$ with $s \in \{0,1\}^{\ell+k_1}$ and $t \in \{0,1\}^{k_0}$. Compute $r := H(s) \oplus t$ and $m' := G(r) \oplus s$. If the least-significant k_1 bits of m' are not all 0, output $\perp$. Otherwise, output the ℓ most-significant bits of $\hat{m}$.

The RSA-OAEP encryption scheme.

CPA-security. During encryption the sender computes

$$m' := m\|0^{k_1}, \quad s := m' \oplus G(r), \quad t := r \oplus H(s)$$

for uniform r; the ciphertext is $[(s\|t)^e \bmod N]$. If the attacker never queries r to G then, since we model G as a random function, the value $G(r)$ is uniform from the attacker's point of view and so m is masked with a uniform string just as in the one-time pad encryption scheme. Thus, if the attacker never queries r to G then no information about the message is leaked.

Can the attacker query r to G? Note that the value of r is itself masked by $H(s)$. So the attacker has no information about r unless it first queries s to H. If the attacker does not query s to H then the attacker may get lucky and guess r anyway, but if we set the length of r (i.e., k_0) sufficiently long then the probability of this is negligible.

So, the only way for the attacker to learn anything about m is to first query s to H. This would require the attacker to compute s from the (uniform) ciphertext $[(s\|t)^e \bmod N]$. Note that computing s from $[(s\|t)^e \bmod N]$ is not the RSA problem, which instead involves computing *both* s and t. Nevertheless, for the right settings of the parameters we can use Theorem 11.29 to show that recovering s enables recovery of t in polynomial time, and so recovering s is computationally infeasible if the RSA problem is hard.

Arguing CCA-security involves additional complications, but the basic idea is to show that every decryption-oracle query c made by the attacker falls into one of two categories: either the attacker obtained c by legally encrypting some message m (in which case the attacker learns nothing from the decryption query), or else decryption of c returns an error. This is a consequence of

the fact that the receiver checks that the k_1 low-order bits of $\hat{m}$ are 0 during decryption; if the attacker did not construct some ciphertext c by legally encrypting some message, the probability that this condition holds is negligible. The formal proof is complicated by the fact that the attacker's decryption-oracle queries must be answered correctly without knowledge of the private key, which means there must be an efficient way to determine whether to return an error or not and, if not, what message to return. This is accomplished by looking at the adversary's queries to the random oracles G, H.

Manger's chosen-ciphertext attack on PKCS #1 v2.0. In 2001, James Manger showed a chosen-ciphertext attack against certain implementations of the RSA encryption scheme specified in PKCS #1 v2.0—even though what was specified was a variant of RSA-OAEP! Since Construction 11.36 is CCA-secure (assuming the RSA problem is hard), how is this possible?

Examining the decryption algorithm in Construction 11.36, note that there are two ways an error can occur: either $\hat{m} \in \mathbb{Z}_N^*$ is too large, or $m' \in \{0,1\}^{\ell+k_1}$ does not have enough trailing 0s. In Construction 11.36, the receiver is supposed to return the *same* error (denoted $\perp$) in either case. In some implementations, however, the receiver would output *different* errors depending on which step failed. This single bit of additional information enables an attacker to mount a chosen-ciphertext attack that recovers a message m in its entirety from an encryption of that message, using only about $\|N\|$ queries to an oracle that leaks the error message upon decryption. This shows the importance of implementing cryptographic schemes exactly as specified, since the resulting proof and analysis may no longer apply if aspects of the scheme are changed.

Even if the same error is returned in both cases, an attacker can determine where the error occurs if the *time* to return the error is different. (This is a great example of how an attacker is not limited to examining the inputs/outputs of an algorithm, but can use *side-channel information* to attack a scheme.) Implementations must be careful to ensure that the time to return an error is identical regardless of where the error occurs.

11.5.5 *A CCA-Secure KEM in the Random-Oracle Model

We show here a construction of an RSA-based KEM that is CCA-secure in the random-oracle model. (Recall from Theorem 11.14 that any such construction can be used in conjunction with any CCA-secure *private*-key encryption scheme to give a CCA-secure *public*-key encryption scheme.) As compared to the RSA-OAEP scheme from the previous section, the main advantage is the simplicity of both the construction and its proof of security. Its main disadvantage is that it results in longer ciphertexts when encrypting short messages since it requires the KEM/DEM paradigm whereas RSA-OAEP does not. For encrypting long messages, however, RSA-OAEP would also be used as part of a hybrid encryption scheme, and would result in an encryption scheme having similar efficiency to what could be obtained using the KEM shown here.

The KEM we describe is included as part of the ISO/IEC 18033-2 standard for public-key encryption. In the scheme, the public key includes $\langle N, e\rangle$ as usual, and a function $H : \mathbb{Z}_N^* \to \{0,1\}^n$ is specified that will be modeled as a random oracle in the analysis. (This function can be based on some underlying cryptographic hash function, as discussed in Section 5.5. We omit the details.) To encapsulate a key, the sender chooses uniform $r \in \mathbb{Z}_N^*$ and then computes the ciphertext $c := [r^e \bmod N]$ and the key $k := H(r)$. To decrypt a ciphertext c, the receiver simply recovers r in the usual way and then re-derives the same key $k := H(r)$. See Construction 11.37.

CONSTRUCTION 11.37

Let GenRSA be as usual, and construct a KEM as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to compute (N, e, d). The public key is $\langle N, e\rangle$, and the private key is $\langle N, d\rangle$.

  As part of key generation, a function $H : \mathbb{Z}_N^* \to \{0,1\}^n$ is specified, but we leave this implicit.
- Encaps: on input public key $\langle N, e\rangle$ and 1^n, choose a uniform $r \in \mathbb{Z}_N^*$. Output the ciphertext $c := [r^e \bmod N]$ and the key $k := H(r)$.
- Decaps: on input private key $\langle N, d\rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $r := [c^d \bmod N]$ and output the key $k := H(r)$.

A CCA-secure KEM (in the random-oracle model).

CPA-security of the scheme is immediate. Indeed, the ciphertext c is equal to $[r^e \bmod N]$ for uniform $r \in \mathbb{Z}_N^*$, and so the RSA assumption implies that an eavesdropper who observes c will be unable to compute r. This means, in turn, that the eavesdropper will not query r to H, and thus the value of the key $k \stackrel{\text{def}}{=} H(r)$ remains uniform from the attacker's point of view.

In fact, the above extends to show CCA-security as well. This is because answering a decapsulation-oracle query for any ciphertext $\tilde{c} \neq c$ only involves evaluating H at some input $[\tilde{c}^d \bmod N] = \tilde{r} \neq r$. Thus, the attacker's decapsulation-oracle queries do not reveal any additional information about the key $H(r)$ encapsulated by the challenge ciphertext. (A formal proof is slightly more involved since we must show how it is possible to *simulate* the answers to decapsulation-oracle queries without knowledge of the private key. Nevertheless, this turns out not to be very difficult.)

THEOREM 11.38 *If the RSA problem is hard relative to* GenRSA *and H is modeled as a random oracle, then Construction 11.37 is CCA-secure.*

PROOF Let Π denote Construction 11.37, and let $\mathcal{A}$ be a probabilistic polynomial-time adversary. For convenience, and because this is the first proof where we use the full power of the random-oracle model, we explicitly describe the steps of experiment $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{GenRSA}(1^n)$ is run to obtain (N, e, d). In addition, a random function $H : \mathbb{Z}_N^* \to \{0,1\}^n$ is chosen.

2. Uniform $r \in \mathbb{Z}_N^*$ is chosen, and the ciphertext $c := [r^e \bmod N]$ and key $k := H(r)$ are computed.

3. A uniform bit $b \in \{0,1\}$ is chosen. If $b = 0$ set $\hat{k} := k$. If $b = 1$ then choose a uniform $\hat{k} \in \{0,1\}^n$.

4. $\mathcal{A}$ is given $pk = \langle N, e\rangle$, c, and $\hat{k}$, and may query $H(\cdot)$ (on any input) and the decapsulation oracle $\mathsf{Decaps}_{\langle N,d\rangle}(\cdot)$ on any ciphertext $\hat{c} \neq c$.

5. $\mathcal{A}$ outputs a bit b'. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

In an execution of experiment $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$, let Query be the event that, at any point during its execution, $\mathcal{A}$ queries r to the random oracle H. We let $\mathsf{Success}$ denote the event that $b' = b$ (i.e., the experiment outputs 1). Then

$$\begin{aligned}\Pr[\mathsf{Success}] &= \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] + \Pr[\mathsf{Success} \wedge \mathsf{Query}] \\ &\leq \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] + \Pr[\mathsf{Query}],\end{aligned}$$

where all probabilities are taken over the randomness used in experiment $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$. We show that $\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \frac{1}{2}$ and that $\Pr[\mathsf{Query}]$ is negligible. The theorem follows.

We first argue that $\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \frac{1}{2}$. If $\Pr[\overline{\mathsf{Query}}] = 0$ this is immediate. Otherwise, $\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \Pr\left[\mathsf{Success} | \overline{\mathsf{Query}}\right]$. Now, conditioned on $\overline{\mathsf{Query}}$, the value of the correct key $k = H(r)$ is uniform because H is a random function. Consider $\mathcal{A}$'s information about k in experiment $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$. The public key pk and ciphertext c, by themselves, do not contain any information about k. (They do uniquely determine r, but since H is chosen independently of anything else, this gives no information about $H(r)$.) Queries that $\mathcal{A}$ makes to H also do not reveal any information about r, unless $\mathcal{A}$ queries r to H (in which case Query occurs); this, again, relies on the fact that H is a random function. Finally, queries that $\mathcal{A}$ makes to its decapsulation oracle only reveal $H(\tilde{r})$ for $\tilde{r} \neq r$. This follows from the fact that $\mathsf{Decaps}_{\langle N,d\rangle}(\tilde{c}) = H(\tilde{r})$ where $\tilde{r} = [\tilde{c}^d \bmod N]$, but $\tilde{c} \neq c$ implies $\tilde{r} \neq r$. Once again, this and the fact that H is a random function mean that no information about $H(r)$ is revealed unless Query occurs.

The above shows that, as long as Query does not occur, the value of the correct key k is uniform even given $\mathcal{A}$'s view of the public key, ciphertext, and

the answers to all its oracle queries. In that case, then, there is no way $\mathcal{A}$ can distinguish (any better than random guessing) whether $\hat{k}$ is the correct key or a uniform, independent key. Therefore, $\Pr\left[\mathsf{Success}|\overline{\mathsf{Query}}\right] = \frac{1}{2}$.

We highlight that nowhere in the above argument did we rely on the fact that $\mathcal{A}$ is computationally bounded, and in fact $\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \frac{1}{2}$ even if no computational restrictions are placed on $\mathcal{A}$. This indicates part of the power of the random-oracle model.

To complete the proof of the theorem, we show

CLAIM 11.39 *If the RSA problem is hard relative to* GenRSA *and H is modeled as a random oracle, then* $\Pr[\mathsf{Query}]$ *is negligible.*

To prove this, we construct an algorithm $\mathcal{A}'$ that uses $\mathcal{A}$ as a subroutine. $\mathcal{A}'$ is given an instance N, e, c of the RSA problem, and its goal is to compute r for which $r^e = c \bmod N$. To do so, it will run $\mathcal{A}$, answering its queries to H and Decaps. Handling queries to H is simple, since $\mathcal{A}'$ can just return a random value. Queries to Decaps are trickier, however, since $\mathcal{A}'$ does not know the private key associated with the effective public key $\langle N, e\rangle$.

On further thought, however, decapsulation queries are also easy to answer since $\mathcal{A}'$ can just return a random value here as well. That is, although the query $\mathsf{Decaps}(\tilde{c})$ is supposed to be computed by first computing $\tilde{r}$ such that $\tilde{r}^e = \tilde{c} \bmod N$ and then evaluating $H(\tilde{r})$, the result is just a uniform value. Thus, $\mathcal{A}'$ can simply return a random value without performing the intermediate computation. The only "catch" is that $\mathcal{A}'$ must ensure consistency between its answers to H-queries and Decaps-queries; namely, it must ensure that for any $\tilde{r}, \tilde{c}$ with $\tilde{r}^e = \tilde{c} \bmod N$ it holds that $H(\tilde{r}) = \mathsf{Decaps}(\tilde{c})$. This is handled using simple bookkeeping and lists L_H and L_{Decaps} that keep track of the answers $\mathcal{A}'$ has given in response to the respective oracle queries. We now give the details.

Algorithm $\mathcal{A}'$:
The algorithm is given (N, e, c) as input.

1. Initialize empty lists L_H, L_{Decaps}. choose a uniform $k \in \{0,1\}^n$ and store (c, k) in L_{Decaps}.
2. Choose a uniform bit $b \in \{0,1\}$. If $b = 0$ set $\hat{k} := k$. If $b = 1$ then choose a uniform $\hat{k} \in \{0,1\}^n$. Run $\mathcal{A}$ on $\langle N, e\rangle$, c, and $\hat{k}$.

   When $\mathcal{A}$ makes a query $H(\tilde{r})$, answer it as follows:

   - If there is an entry in L_H of the form $(\tilde{r}, k)$ for some k, return k.
   - Otherwise, let $\tilde{c} := [\tilde{r}^e \bmod N]$. If there is an entry in L_{Decaps} of the form $(\tilde{c}, k)$ for some k, return k and store $(\tilde{r}, k)$ in L_H.

- Otherwise, choose a uniform $k \in \{0,1\}^n$, return k, and store $(\tilde{r}, k)$ in L_H.

When $\mathcal{A}$ makes a query $\mathsf{Decaps}(\tilde{c})$, answer it as follows:

- If there is an entry in L_{Decaps} of the form $(\tilde{c}, k)$ for some k, return k.
- Otherwise, for each entry $(\tilde{r}, k) \in L_H$, check if $\tilde{r}^e = \tilde{c} \bmod N$ and, if so, output k.
- Otherwise, choose a uniform $k \in \{0,1\}^n$, return k, and store $(\tilde{c}, k)$ in L_{Decaps}.

3. At the end of $\mathcal{A}$'s execution, if there is an entry (r, k) in L_H for which $r^e = c \bmod N$ then return r.

Clearly $\mathcal{A}'$ runs in polynomial time, and the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ in experiment $\mathsf{RSA\text{-}inv}_{\mathcal{A}',\mathsf{GenRSA}}(n)$ is *identical* to the view of $\mathcal{A}$ in experiment $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$: the inputs given to $\mathcal{A}$ clearly have the right distribution, the answers to $\mathcal{A}$'s oracle queries are consistent, and the responses to all H-queries are uniform and independent. Finally, $\mathcal{A}'$ outputs the correct solution exactly when Query occurs. Hardness of the RSA problem relative to GenRSA thus implies that $\Pr[\mathsf{Query}]$ is negligible, as required. ■

It is worth remarking on the various properties of the random-oracle model (see Section 5.5.1) that are used in the above proof. First, we rely on the fact that the value $H(r)$ is uniform unless r is queried to H—even if H is queried on multiple other values $\tilde{r} \neq r$. We also, implicitly, use extractability to argue that the attacker cannot query r to H; otherwise, we could use this attacker to solve the RSA problem. Finally, the proof relies on programmability in order to simulate the adversary's decapsulation-oracle queries.

11.5.6 RSA Implementation Issues and Pitfalls

We close this section with a brief discussion of some issues related to the implementation of RSA-based schemes, and some pitfalls to be aware of.

Using Chinese remaindering. In implementations of RSA-based encryption, the receiver can use the Chinese remainder theorem (Section 8.1.5) to speed up computation of eth roots modulo N during decryption. Specifically, let $N = pq$ and say the receiver wishes to compute the eth root of some value y using $d = [e^{-1} \bmod \phi(N)]$. The receiver can use the correspondence $[y^d \bmod N] \leftrightarrow ([y^d \bmod p\,], [y^d \bmod q])$ to compute the partial results

$$x_p := [y^d \bmod p] = \left[y^{[d \bmod (p-1)]} \bmod p\right] \tag{11.19}$$

and

$$x_q := [y^d \bmod q] = \left[y^{[d \bmod (q-1)]} \bmod q\right], \tag{11.20}$$

and then combine these to obtain $x \leftrightarrow (x_p, x_q)$, as discussed in Section 8.1.5. Note that $[d \bmod (p-1)]$ and $[d \bmod (q-1)]$ could be pre-computed since they are independent of y.

Why is this better? Assume exponentiation modulo an ℓ-bit integer takes $\gamma \cdot \ell^3$ operations for some constant γ. If p, q are each n bits long, then naively computing $[y^d \bmod N]$ takes $\gamma \cdot (2n)^3 = 8\gamma \cdot n^3$ steps (because $\|N\| = 2n$). Using Chinese remaindering reduces this to roughly $2 \cdot (\gamma \cdot n^3)$ steps (because $\|p\| = \|q\| = n$), or roughly 1/4 of the time.

Example 11.40

We revisit Example 8.49. Recall that $N = 143 = 11 \cdot 13$ and $d = 103$, and $y = 64$ there. To calculate $[64^{103} \bmod 143]$ we compute

$$\begin{aligned}\big([64 \bmod 11],\ [64 \bmod 13]\big)^{103} &= \big([(-2)^{103} \bmod 11],\ [(-1)^{103} \bmod 13]\big) \\ &= \Big([(-2)^{[103 \bmod 10]} \bmod 11],\ -1\Big) \\ &= \big([-8 \bmod 11],\ -1\big) = (3,\ -1).\end{aligned}$$

We can compute $1_p = 78 \leftrightarrow (1, 0)$ and $1_q = 66 \leftrightarrow (0, 1)$, as discussed in Section 8.1.5. (Note these values can be pre-computed, as they are independent of y.) Then $(3, -1) \leftrightarrow 3 \cdot 1_p - 1_q = 3 \cdot 78 - 66 = 168 = 25 \bmod 143$, in agreement with the answer previously obtained. ♢

A fault attack when using Chinese remaindering. When using Chinese remaindering as just described, one should be aware of a potential attack that can be carried out if *faults* occur (or can be induced to occur by an attacker, e.g., by hardware tampering) during the course of the computation.

Consider what happens if $[y^d \bmod N]$ is computed twice: the first time with no error (giving the correct result x), but the second time with an error during computation of Equation (11.20) but not Equation (11.19) (the same attack applies in the opposite case). The second computation yields an incorrect result x' for which $x' = x \bmod p$ but $x' \neq x \bmod q$. This means that $p \,|\, (x' - x)$ but $q \nmid (x' - x)$. But then $\gcd(x' - x, N) = p$, yielding the factorization of N.

One possible countermeasure is to verify correctness of the result before using it, by checking that $x^e = y \bmod N$. (Since $\|e\| \ll \|d\|$, using Chinese remaindering still gives better efficiency.) This is recommended in hardware implementations.

Dependent public keys I. When multiple receivers wish to utilize the same encryption scheme, they should use *independent* public keys. This and the following attack demonstrate what can go wrong when this is not done.

Imagine a company wants to use the same modulus N for each of its employees. Since it is not desirable for messages encrypted to one employee to be read by any other employee, the company issues different (e_i, d_i) pairs to

each employee. That is, the public key of the ith employee is $pk_i = \langle N, e_i \rangle$ and their private key is $sk = \langle N, d_i \rangle$, where $e_i \cdot d_i = 1 \bmod \phi(N)$ for all i.

This approach is insecure and allows any employee to read messages encrypted to all other employees. The reason is that, as noted in Section 8.2.4, given N and e_i, d_i with $e_i \cdot d_i = 1 \bmod \phi(N)$, the factorization of N can be efficiently computed. Given the factorization of N, of course, it is possible to compute $d_j := e_j^{-1} \bmod \phi(N)$ for any j.

Dependent public keys II. The attack just shown allows any employee to decrypt messages sent to any other employee. This still leaves the possibility that sharing the modulus N is fine as long as all employees trust each other (or, alternatively, as long as confidentiality need only be preserved against outsiders but not against other members of the company). Here we show a scenario indicating that sharing a modulus is still a bad idea, at least when plain RSA encryption is used.

Say the same message m is encrypted and sent to two different (known) employees with public keys (N, e_1) and (N, e_2) where $e_1 \neq e_2$. Assume further that $\gcd(e_1, e_2) = 1$. Then an eavesdropper sees the two ciphertexts

$$c_1 = m^{e_1} \bmod N \quad \text{and} \quad c_2 = m^{e_2} \bmod N.$$

Since $\gcd(e_1, e_2) = 1$, there exist integers X, Y such that $Xe_1 + Ye_2 = 1$ by Proposition 8.2. Moreover, given the public exponents e_1 and e_2 it is possible to efficiently compute X and Y using the extended Euclidean algorithm (see Appendix B.1.2). We claim that $m = [c_1^X \cdot c_2^Y \bmod N]$, which can easily be calculated. This is true because

$$c_1^X \cdot c_2^Y = m^{Xe_1} m^{Ye_2} = m^{Xe_1 + Ye_2} = m^1 = m \bmod N.$$

A similar attack applies when using padded RSA or RSA-OAEP if the sender uses the same transformed message $\hat{m}$ when encrypting to two users.

Randomness quality in RSA key generation. Throughout this book, we always assume that honest parties have access to sufficient, high-quality randomness. When this assumption is violated then security may fail to hold. In particular, if an ℓ-bit string is chosen from some set $S \subset \{0,1\}^\ell$ rather than uniformly from $\{0,1\}^\ell$, then an attacker can perform a brute-force search (in time $\mathcal{O}(|S|)$) to attack the system.

In some cases the situation may be even worse. Consider in particular the case of RSA key generation, where one sample of random bits r_p is used to choose the first prime p, and a second sample r_q is used to generate the second prime q. Assume further that many public/private keys are generated using the same source of poor-quality randomness, in which r_p, r_q are chosen uniformly from some set S of size 2^s. After generating roughly $2^{s/2}$ public keys (see Appendix A.4), we expect to obtain two different moduli N, N' that were generated using identical randomness $r_p = r'_p$. These two moduli share a prime factor which can be easily found by computing $\gcd(N, N')$. An attacker

can thus scrape the Internet for a large set of RSA public keys, compute their pairwise gcd's, and hope to factor some subset of them. Although computing pairwise gcd's of $2^{s/2}$ moduli would naively take time $\mathcal{O}(2^s)$, it turns out that this can be significantly improved using a "divide-and-conquer" approach that is beyond the scope of this book. The upshot is that an attacker can perform a brute-force search in time much less than 2^s. Moreover, the attack works even if the set S is unknown to the attacker!

The above scenario was verified experimentally by two research teams working independently, who carried out exactly the above attack on public keys scraped over the Internet and were able to successfully factor a significant fraction of the keys they found.

References and Additional Reading

The idea of public-key encryption was first proposed in the open literature by Diffie and Hellman [58]. Rivest, Shamir, and Adleman [148] introduced the RSA assumption and proposed a public-key encryption scheme based on this assumption. As pointed out in the previous chapter, other pioneers of public-key cryptography include Merkle and Rabin (in academic publications) and Ellis, Cocks, and Williamson (in classified publications).

Definition 11.2 is rooted in the seminal work of Goldwasser and Micali [80], who were also the first to recognize the necessity of *probabilistic* encryption for satisfying this definition. As noted in Chapter 4, chosen-ciphertext attacks were first formally defined by Naor and Yung [129] and Rackoff and Simon [147]. The expository article by Shoup [156] discusses the importance of security against chosen-ciphertext attacks. Bellare et al. give a unified, modern treatment of various security notions for public-key encryption [16].

A proof of CPA-security for hybrid encryption was first given by Blum and Goldwasser [36]. The case of CCA-security was treated in [56].

Somewhat amazingly, the El Gamal encryption scheme [70] was not suggested until 1984, even though it can be viewed as a direct transformation of the Diffie–Hellman key-exchange protocol (see Exercise 11.4). DHIES was introduced in [2]. The ISO/IEC 18033-2 standard for public-key encryption can be found at `http://www.shoup.net/iso`.

Plain RSA encryption corresponds to the original scheme introduced by Rivest, Shamir, and Adleman [148]. The attacks on plain RSA encryption described in Section 11.5.1 are due to [161, 55, 84, 47, 40]; see [120, Chapter 8] and [38] for additional attacks and further information. Proofs of Coppersmith's theorem can be found in the original work [46] or several subsequent expositions (e.g., [119]).

The *PKCS #1 RSA Cryptography Standards* (both previous and current

versions) are available at `http://www.emc.com/emc-plus/rsa-labs`. The chosen-plaintext attack on PKCS #1 v1.5 described here is due to [49]. A description of Bleichenbacher's chosen-ciphertext attack on PKCS #1 v1.5 can be found in the original paper [34]. See [12] for subsequent improvements.

Proofs of Theorem 11.31, and generalizations, can be found in [8, 86, 66, 7]. See Section 13.1.2 for a general treatment of schemes of this form. Construction 11.37 appears to have been introduced and first analyzed by Shoup [157]. OAEP was introduced by Bellare and Rogaway [22]. The original proof of OAEP was later found to be flawed; the interested reader is referred to [39, 158, 68]. For details of Manger's chosen-ciphertext attack on implementations of PKCS #1 v2.0, see [117].

The pairwise-gcd attack described in Section 11.5.6 was carried out by Lenstra et al. [112] and Heninger et al. [88].

When using any encryption scheme in practice, the question arises as to what key length to use. This issue should not be taken lightly, and we refer the reader to Section 9.3 and references therein for an in-depth treatment.

The first efficient CCA-secure public-key encryption scheme not relying on the random-oracle model was shown by Cramer and Shoup [50] based on the DDH assumption. Subsequently, Hoffheinz and Kiltz have shown an efficient CCA-secure scheme without random oracles based on the RSA assumption [92].

Exercises

11.1 Assume a public-key encryption scheme for single-bit messages with no decryption error. Show that, given pk and a ciphertext c computed via $c \leftarrow \mathsf{Enc}_{pk}(m)$, it is possible for an unbounded adversary to determine m with probability 1.

11.2 Show that for any CPA-secure public-key encryption scheme for single-bit messages, the length of the ciphertext must be superlogarithmic in the security parameter.

Hint: If not, the range of possible ciphertexts has polynomial size.

11.3 Say a public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for n-bit messages is *one-way* if any PPT adversary $\mathcal{A}$ has negligible probability of success in the following experiment:

- $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
- *A message* $m \in \{0,1\}^n$ *is chosen uniformly at random, and a ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m)$ *is computed.*
- $\mathcal{A}$ *is given* pk *and* c*, and outputs a message* m'*. We say* $\mathcal{A}$ *succeeds if* $m' = m$.

(a) Give a construction of a CPA-secure KEM in the random-oracle model based on any one-way public-key encryption scheme.

(b) Can a *deterministic* public-key encryption scheme be one-way? If not, prove impossibility; if so, give a construction based on any of the assumptions introduced in this book.

11.4 Show that any two-round key-exchange protocol (that is, where each party sends a single message) satisfying Definition 10.1 can be converted into a CPA-secure public-key encryption scheme.

11.5 Show that Claim 11.7 does not hold in the setting of CCA-security.

11.6 Consider the following public-key encryption scheme. The public key is $(\mathbb{G}, q, g, h)$ and the private key is x, generated exactly as in the El Gamal encryption scheme. In order to encrypt a bit b, the sender does the following:

(a) If $b = 0$ then choose a uniform $y \in \mathbb{Z}_q$ and compute $c_1 := g^y$ and $c_2 := h^y$. The ciphertext is $\langle c_1, c_2 \rangle$.

(b) If $b = 1$ then choose independent uniform $y, z \in \mathbb{Z}_q$, compute $c_1 := g^y$ and $c_2 := g^z$, and set the ciphertext equal to $\langle c_1, c_2 \rangle$.

Show that it is possible to decrypt efficiently given knowledge of x. Prove that this encryption scheme is CPA-secure if the decisional Diffie–Hellman problem is hard relative to $\mathcal{G}$.

11.7 Consider the following variant of El Gamal encryption. Let $p = 2q + 1$, let $\mathbb{G}$ be the group of squares modulo p (so $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ of order q), and let g be a generator of $\mathbb{G}$. The private key is $(\mathbb{G}, g, q, x)$ and the public key is $(\mathbb{G}, g, q, h)$, where $h = g^x$ and $x \in \mathbb{Z}_q$ is chosen uniformly. To encrypt a message $m \in \mathbb{Z}_q$, choose a uniform $r \in \mathbb{Z}_q$, compute $c_1 := g^r \bmod p$ and $c_2 := h^r + m \bmod p$, and let the ciphertext be $\langle c_1, c_2 \rangle$. Is this scheme CPA-secure? Prove your answer.

11.8 Consider the following protocol for two parties A and B to flip a fair coin (more complicated versions of this might be used for Internet gambling): (1) a trusted party T publishes her public key pk; (2) then A chooses a uniform bit b_A, encrypts it using pk, and announces the ciphertext c_A to B and T; (3) next, B acts symmetrically and announces a ciphertext $c_B \neq c_A$; (4) T decrypts both c_A and c_B, and the parties XOR the results to obtain the value of the coin.

(a) Argue that even if A is dishonest (but B is honest), the final value of the coin is uniformly distributed.

(b) Assume the parties use El Gamal encryption (where the bit b is encoded as the group element g^b before being encrypted—note that efficient decrypt is still possible). Show how a dishonest B can bias the coin to any value he likes.

(c) Suggest what type of encryption scheme would be appropriate to use here. Can you define an appropriate notion of security and prove that your suggestion achieves this definition?

11.9 Prove formally that the El Gamal encryption scheme is not CCA-secure.

11.10 In Section 11.4.4 we showed that El Gamal encryption is malleable, and specifically that given a ciphertext $\langle c_1, c_2 \rangle$ that is the encryption of some unknown message m, it is possible to produce a ciphertext $\langle c_1, c_2' \rangle$ that is the encryption of $\alpha \cdot m$ (for known α). A receiver who receives both these ciphertexts might be suspicious since both ciphertexts share the first component. Show that it is possible to generate $\langle c_1', c_2' \rangle$ that is the encryption of $\alpha \cdot m$, with $c_1' \neq c_1$ and $c_2' \neq c_2$.

11.11 Prove Theorem 11.22.

11.12 One of the attacks on plain RSA discussed in Section 11.5.1 involves a sender who encrypts two related messages using the same public key. Formulate an appropriate definition of security ruling out such attacks, and show that any CPA-secure public-key encryption scheme satisfies your definition.

11.13 One of the attacks on plain RSA discussed in Section 11.5.1 involves a sender who encrypts the same message to three different receivers. Formulate an appropriate definition of security ruling out such attacks, and show that any CPA-secure public-key encryption scheme satisfies your definition.

11.14 Consider the following modified version of padded RSA encryption: Assume messages to be encrypted have length exactly $\|N\|/2$. To encrypt, first compute $\hat{m} := \texttt{0x00} \| r \| \texttt{0x00} \| m$ where r is a uniform string of length $\|N\|/2 - 16$. Then compute the ciphertext $c := [\hat{m}^e \bmod N]$. When decrypting a ciphertext c, the receiver computes $\hat{m} := [c^d \bmod N]$ and returns an error if $\hat{m}$ does not consist of `0x00` followed by $\|N\|/2 - 16$ arbitrary bits followed by `0x00`. Show that this scheme is not CCA-secure. Why is it easier to construct a chosen-ciphertext attack on this scheme than on PKCS #1 v1.5?

11.15 Consider the RSA-based encryption scheme in which a user encrypts a message $m \in \{0,1\}^\ell$ with respect to the public key $\langle N, e \rangle$ by computing $\hat{m} := H(m) \| m$ and outputting the ciphertext $[\hat{m}^e \bmod N]$. (Here, let $H : \{0,1\}^\ell \to \{0,1\}^n$ and assume $\ell + n < \|N\|$.) The receiver recovers $\hat{m}$ in the usual way and verifies that it has the correct form before outputting the ℓ least-significant bits as m. Prove or disprove that this scheme is CCA-secure if H is modeled as a random oracle.

11.16 Show a chosen-ciphertext attack on Construction 11.34.

11.17 Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure public-key encryption scheme, and let $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a CCA-secure private-key encryption scheme. Consider the following construction:

CONSTRUCTION 11.41

Let $H : \{0,1\}^n \to \{0,1\}^n$ be a function. Construct a public-key encryption scheme as follows:

- Gen^*: on input 1^n, run $\mathsf{Gen}(1^n)$ to obtain (pk, sk). Output these as the public and private keys, respectively.
- Enc^*: on input a public key pk and a message $m \in \{0,1\}^n$, choose a uniform $r \in \{0,1\}^n$ and output the ciphertext

$$\left\langle \mathsf{Enc}_{pk}(r),\ \mathsf{Enc}'_{H(r)}(m) \right\rangle .$$

- Dec^*: on input a private key sk and a ciphertext $\langle c_1, c_2 \rangle$, compute $r := \mathsf{Dec}_{sk}(c_1)$ and set $k := H(r)$. Then output $\mathsf{Dec}'_k(c_2)$.

Does the above construction have indistinguishable encryptions under a chosen-*ciphertext* attack, if H is modeled as a random oracle? If yes, provide a proof. If not, where does the approach used to prove Theorem 11.38 break down?

11.18 Consider the following variant of Construction 11.32:

CONSTRUCTION 11.42

Let GenRSA be as usual, and define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d). Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0,1\}$, choose a uniform $r \in \mathbb{Z}_N^*$. Output the ciphertext $\langle [r^e \bmod N],\ \mathsf{lsb}(r) \oplus m \rangle$.
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $\langle c, b \rangle$, compute $r := [c^d \bmod N]$ and output $\mathsf{lsb}(r) \oplus b$.

Prove that this scheme is CPA-secure. Discuss its advantages and disadvantages relative to Construction 11.32.

11.19 Say three users have RSA public keys $\langle N_1, 3 \rangle$, $\langle N_2, 3 \rangle$, and $\langle N_3, 3 \rangle$ (i.e., they all use $e = 3$), with $N_1 < N_2 < N_3$. Consider the following method for sending the same message $m \in \{0,1\}^\ell$ to each of these parties: choose a uniform $r \leftarrow \mathbb{Z}_{N_1}^*$, and send to everyone the same ciphertext

$$\left\langle [r^3 \bmod N_1], [r^3 \bmod N_2], [r^3 \bmod N_3], H(r) \oplus m \right\rangle ,$$

where $H : \mathbb{Z}_{N_1}^* \to \{0,1\}^\ell$. Assume $\ell \gg n$.

(a) Show that this is not CPA-secure, and an adversary can recover m from the ciphertext even when H is modeled as a random oracle.

Hint: See Section 11.5.1.

(b) Show a simple way to fix this and get a CPA-secure method that transmits a ciphertext of length $3\ell + \mathcal{O}(n)$.

(c) Show a better approach that is still CPA-secure but with a ciphertext of length $\ell + \mathcal{O}(n)$.

11.20 Fix an RSA public key $\langle N, e\rangle$ and assume we have an algorithm $\mathcal{A}$ that always correctly computes $\mathsf{lsb}(x)$ given $[x^e \bmod N]$. Write full pseudocode for an algorithm $\mathcal{A}'$ that computes x from $[x^e \bmod N]$.

11.21 Fix an RSA public key $\langle N, e\rangle$ and define

$$\mathsf{half}(x) = \begin{cases} 0 & \text{if } 0 < x < N/2 \\ 1 & \text{if } N/2 < x < N \end{cases}$$

Prove that half is a hard-core predicate for the RSA problem.

Hint: Reduce to hardness of computing lsb.

Chapter 12

Digital Signature Schemes

12.1 Digital Signatures – An Overview

In the previous chapter we explored how public-key encryption can be used to achieve *secrecy* in the public-key setting. *Integrity* (or *authenticity*) in the public-key setting is provided using *digital signature schemes*. These can be viewed as the public-key analogue of message authentication codes, although, as we will see, there are several important differences between these primitives.

Signature schemes allow a *signer* S who has established a public key pk to "sign" a message using the associated private key sk in such a way that anyone who knows pk (and knows that this public key was established by S) can *verify* that the message originated from S and was not modified in transit. (Note that, in contrast to public-key encryption, in the context of digital signatures the owner of the public key acts as the *sender*.) As a prototypical application, consider a software company that wants to disseminate software updates in an authenticated manner; that is, when the company releases an update it should be possible for any of its clients to verify that the update is authentic, and a malicious third party should never be able to fool a client into accepting an update that was not actually released by the company. To do this, the company can generate a public key pk along with a private key sk, and then distribute pk in some reliable manner to its clients while keeping sk secret. (As in the case of public-key encryption, we assume that this initial distribution of the public key is carried out correctly so that all clients have a correct copy of pk. In the current example, pk could be bundled with the original software purchased by a client.) When releasing a software update m, the company computes a digital signature σ on m using its private key sk, and sends (m, σ) to every client. Each client can verify the authenticity of m by checking that σ is a correct signature on m with respect to the public key pk.

A malicious party might try to issue a fraudulent update by sending (m', σ') to a client, where m' represents an update that was never released by the company. This m' might be a modified version of some previous update, or it might be completely new and unrelated to any prior updates. If the signature scheme is "secure" (in a sense we will define more carefully soon), however, then when the client attempts to verify σ' it will find that this is an *invalid* signature on m' with respect to pk, and will therefore *reject* the signature. The

client will reject even if m' is modified only slightly from a genuine update m.

The above is not just a theoretical application of digital signatures, but one that is used extensively today for distributing software updates.

Comparison to Message Authentication Codes

Both message authentication codes and digital signature schemes are used to ensure the integrity of transmitted messages. Although the discussion in Chapter 10 comparing the public-key and private-key settings focused mainly on encryption, that discussion applies also to message integrity. Using digital signatures rather than message authentication codes simplifies key distribution and management, especially when a sender needs to communicate with multiple receivers as in the software-update example above. By using a digital signature scheme the sender avoids having to establish a distinct secret key with each potential receiver, and avoids having to compute a separate MAC tag with respect to each such key. Instead, the sender need only compute a single signature that can be verified by all recipients.

A *qualitative* advantage that digital signatures have as compared to message authentication codes is that signatures are *publicly verifiable*. This means that if a receiver verifies that a signature on a given message is legitimate, then all other parties who receive this signed message will also verify it as legitimate. This feature is not achieved by message authentication codes if the signer shares a separate key with each receiver: in such a setting a malicious sender might compute a correct MAC tag with respect to the key it shares with receiver A but an incorrect MAC tag with respect to the key it shares with a different user B. In this case, A knows that he received an authentic message from the sender but has no guarantee that B will agree.

Public verifiability implies that signatures are *transferable*: a signature σ on a message m by a signer S can be shown to a third party, who can then verify herself that σ is a legitimate signature on m with respect to S's public key (here, we assume this third party also knows S's public key). By making a copy of the signature, this third party can then show the signature to another party and convince *them* that S authenticated m, and so on. Public verifiability and transferability are essential for the application of digital signatures to certificates and public-key infrastructures, as we will discuss in Section 12.7.

Digital signature schemes also provide the very important property of *non-repudiation*. This means that once S signs a message he cannot later deny having done so (assuming the public key of S is widely publicized and distributed). This aspect of digital signatures is crucial for legal applications where a recipient may need to prove to a third party (say, a judge) that a signer did indeed "certify" a particular message (e.g., a contract): assuming S's public key is known to the judge, or is otherwise publicly available, a valid signature on a message serves as convincing evidence that S indeed signed this message. Message authentication codes simply cannot provide non-repudiation. To see this, say users S and R share a key k_{SR}, and S

sends a message m to R along with a (valid) MAC tag t computed using this key. Since the judge does *not* know k_{SR} (indeed, this key is kept secret by S and R), there is no way for the judge to determine whether t is valid or not. If R were to reveal the key k_{SR} to the judge, there would be no way for the judge to know whether this is the "actual" key that S and R shared, or whether it is some "fake" key manufactured by R. Finally, even if we assume the judge can somehow obtain the actual key k_{SR} shared by the parties, there is no way for the judge to distinguish whether S generated t or whether R did—this follows from the fact that message authentication codes are a *symmetric*-key primitive; anything S can do, R can do also.

As in the case of private-key vs. public-key encryption, message authentication codes have the advantage of being shorter and roughly 2–3 orders of magnitude more efficient to generate/verify than digital signatures. Thus, in situations where public verifiability, transferability, and/or non-repudiation are not needed, and the sender communicates primarily with a single recipient (with whom it is able to share a secret key), message authentication codes should be used.

Relation to Public-Key Encryption

Digital signatures are often mistakenly viewed as the "inverse" of public-key encryption, with the roles of the sender and receiver interchanged. Historically,[1] in fact, it has been suggested that digital signatures can be obtained by "reversing" public-key encryption, i.e., signing a message m by decrypting it (using the private key) to obtain σ, and verifying a signature σ by encrypting it (using the corresponding public key) and checking whether the result is m. The suggestion to construct signature schemes in this way is *completely unfounded*: in most cases, it is simply inapplicable, and in cases where it is applicable it results in signature schemes that are not secure.

12.2 Definitions

Digital signatures are the public-key counterpart of message authentication codes, and their syntax and security guarantees are analogous. The algorithm that the sender applies to a message is here denoted Sign (rather than Mac), and the output of this algorithm is now called a *signature* (rather than a tag).

[1] This view no doubt arose because, as we will see in Section 12.4.1, plain RSA signatures are the reverse of plain RSA encryption. However, neither plain RSA signatures nor plain RSA encryption meet even minimal notions of security.

The algorithm that the receiver applies to a message and a signature in order to verify validity is still denoted Vrfy.

DEFINITION 12.1 *A* (digital) signature scheme *consists of three probabilistic polynomial-time algorithms* (Gen, Sign, Vrfy) *such that:*

1. *The* key-generation algorithm Gen *takes as input a security parameter* 1^n *and outputs a pair of keys* (pk, sk)*. These are called the* public key *and the* private key*, respectively. We assume that* pk *and* sk *each has length at least* n*, and that* n *can be determined from* pk *or* sk*.*

2. *The* signing algorithm Sign *takes as input a private key* sk *and a message* m *from some message space (that may depend on* pk*). It outputs a signature* σ*, and we write this as* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$*.*

3. *The deterministic* verification algorithm Vrfy *takes as input a public key* pk*, a message* m*, and a signature* σ*. It outputs a bit* b*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid*. We write this as* $b := \mathsf{Vrfy}_{pk}(m, \sigma)$*.*

It is required that except with negligible probability over (pk, sk) *output by* $\mathsf{Gen}(1^n)$*, it holds that* $\mathsf{Vrfy}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$ *for every (legal) message* m*.*

If there is a function ℓ *such that for every* (pk, sk) *output by* $\mathsf{Gen}(1^n)$ *the message space is* $\{0,1\}^{\ell(n)}$*, then we say that* (Gen, Sign, Vrfy) *is a* signature scheme for messages of length $\ell(n)$*.*

We call σ a *valid signature* on a message m (with respect to some public key pk understood from the context) if $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

A signature scheme is used in the following way. One party S, who acts as the *sender*, runs $\mathsf{Gen}(1^n)$ to obtain keys (pk, sk). The public key pk is then publicized as belonging to S; e.g., S can put the public key on its webpage or place it in some public directory. As in the case of public-key encryption, we assume that any other party is able to obtain a legitimate copy of S's public key (see discussion below). When S wants to authenticate a message m, it computes the signature $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$ and sends (m, σ). Upon receipt of (m, σ), a receiver who knows pk can verify the authenticity of m by checking whether $\mathsf{Vrfy}_{pk}(m, \sigma) \stackrel{?}{=} 1$. This establishes both that S sent m, and also that m was not modified in transit. As in the case of message authentication codes, however, it does not say anything about *when* m was sent, and replay attacks are still possible (see Section 4.2).

The assumption that parties are able to obtain a legitimate copy of S's public key implies that S is able to transmit at least one message (namely, pk itself) in a reliable and authenticated manner. If we are able to transmit messages reliably, however, then why do we need a signature scheme at all? The answer is that reliable distribution of pk *is* a difficult and expensive task, but using a signature scheme means that this need only be carried out

once, after which an unlimited number of messages can subsequently be sent reliably. Furthermore, as we will discuss in Section 12.7, signature schemes themselves are used to ensure the reliable distribution of *other* public keys. They thus serve as a central tool for setting up a "public-key infrastructure" to address the key-distribution problem.

Security of signature schemes. For a fixed public key pk generated by a signer S, a *forgery* is a message m along with a valid signature σ, where m was not previously signed by S. Security of a signature scheme means that an adversary should be unable to output a forgery even if it obtains signatures on many other messages of its choice. This is the direct analogue of the definition of security for message authentication codes, and we refer the reader to Section 4.2 for motivation and further discussion.

We now present the formal definition of security, which is essentially the same as Definition 4.2. Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and consider the following experiment for an adversary $\mathcal{A}$ and parameter n:

The signature experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *and access to an oracle* $\mathsf{Sign}_{sk}(\cdot)$. *The adversary then outputs* (m, σ). *Let* $\mathcal{Q}$ *denote the set of all queries that* $\mathcal{A}$ *asked its oracle.*
3. $\mathcal{A}$ succeeds *if and only if (1)* $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ *and (2)* $m \notin \mathcal{Q}$. *In this case the output of the experiment is defined to be* 1.

DEFINITION 12.2 *A signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* existentially unforgeable under an adaptive chosen-message attack, *or just* secure, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$, *there is a negligible function* negl *such that:*

$$\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

12.3 The Hash-and-Sign Paradigm

As in the case of public-key vs. private-key encryption, "native" signature schemes are orders of magnitude less efficient than message authentication codes. Fortunately, as with hybrid encryption (see Section 11.3), it is possible to obtain the functionality of digital signatures at the asymptotic cost of a private-key operation, at least for sufficiently long messages. This can be done using the *hash-and-sign* approach, discussed next.

The intuition behind the hash-and-sign approach is straightforward. Say we have a signature scheme for messages of length ℓ, and wish to sign a (longer) message $m \in \{0,1\}^*$. Rather than sign m itself, we can instead use a hash function H to *hash* the message down to a fixed-length *digest* of length ℓ, and then sign the resulting digest. This approach is exactly analogous to the hash-and-MAC approach discussed in Section 5.3.1.

CONSTRUCTION 12.3

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme for messages of length $\ell(n)$, and let $\Pi_H = (\mathsf{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Construct signature scheme $\Pi' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ as follows:

- Gen': on input 1^n, run $\mathsf{Gen}(1^n)$ to obtain (pk, sk) and run $\mathsf{Gen}_H(1^n)$ to obtain s; the public key is $\langle pk, s\rangle$ and the private key is $\langle sk, s\rangle$.
- Sign': on input a private key $\langle sk, s\rangle$ and a message $m \in \{0,1\}^*$, output $\sigma \leftarrow \mathsf{Sign}_{sk}(H^s(m))$.
- Vrfy': on input a public key $\langle pk, s\rangle$, a message $m \in \{0,1\}^*$, and a signature σ, output 1 if and only if $\mathsf{Vrfy}_{pk}(H^s(m), \sigma) \stackrel{?}{=} 1$.

The hash-and-sign paradigm.

THEOREM 12.4 *If Π is a secure signature scheme for messages of length ℓ and Π_H is collision resistant, then Construction 12.3 is a secure signature scheme (for arbitrary-length messages).*

The proof of this theorem is almost identical to that of Theorem 5.6.

12.4 RSA Signatures

We begin our consideration of concrete signature schemes with a discussion of schemes based on the RSA assumption.

12.4.1 Plain RSA

We first describe a simple, RSA-based signature scheme. Although the scheme is insecure, it serves as a useful starting point.

As usual, let GenRSA be a PPT algorithm that, on input 1^n, outputs a

modulus N that is the product of two n-bit primes (except with negligible probability), along with integers e, d satisfying $ed = 1 \bmod \phi(N)$. Key generation in plain RSA involves simply running GenRSA, and outputting $\langle N, e\rangle$ as the public key and $\langle N, d\rangle$ as the private key. To sign a message $m \in \mathbb{Z}_N^*$, the signer computes $\sigma := [m^d \bmod N]$. Verification of a signature σ on a message m with respect to the public key $\langle N, e\rangle$ is carried out by checking whether $m \stackrel{?}{=} \sigma^e \bmod N$. See Construction 12.5.

CONSTRUCTION 12.5

Let GenRSA be as in the text. Define a signature scheme as follows:

- Gen: on input 1^n run $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d). The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
- Sign: on input a private key $sk = \langle N, d\rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the signature
$$\sigma := [m^d \bmod N].$$
- Vrfy: on input a public key $pk = \langle N, e\rangle$, a message $m \in \mathbb{Z}_N^*$, and a signature $\sigma \in \mathbb{Z}_N^*$, output 1 if and only if
$$m \stackrel{?}{=} [\sigma^e \bmod N].$$

The plain RSA signature scheme.

It is easy to see that verification of a legitimately generated signature is always successful since

$$\sigma^e = (m^d)^e = m^{[ed \bmod \phi(N)]} = m^1 = m \bmod N.$$

One might expect this scheme to be secure since, for an adversary knowing only the public key $\langle N, e\rangle$, computing a valid signature on a message m seems to require solving the RSA problem (since the signature is exactly the eth root of m). Unfortunately, this reasoning is incorrect. For one thing, the RSA assumption only implies hardness of computing a signature (that is, computing an eth root) of a *uniform* message m; it says nothing about hardness of computing a signature on a nonuniform m or on some message m of the attacker's choice. Moreover, the RSA assumption says nothing about what an attacker might be able to do once it learns signatures on *other* messages. The following examples demonstrate that both of these observations lead to attacks on the plain RSA signature scheme.

A no-message attack. The first attack we describe generates a forgery *using the public key alone*, without obtaining any signatures from the legitimate signer. The attack works as follows: given a public key $pk = \langle N, e\rangle$, choose a uniform $\sigma \in \mathbb{Z}_N^*$ and compute $m := [\sigma^e \bmod N]$. Then output the forgery

(m, σ). It is immediate that σ is a valid signature on m, and this is a forgery since no signatures at all were issued by the owner of the public key. We conclude that the plain RSA signature scheme does not satisfy Definition 12.2.

One might argue that this does not constitute a "realistic" attack since the adversary has "no control" over the message m for which it forges a valid signature. This is irrelevant as far as Definition 12.2 is concerned, and we have already discussed (in Chapter 4) why it is dangerous to assume any semantics for messages that are going to be authenticated using any cryptographic scheme. Moreover, the adversary does have *some* control over m: for example, by choosing multiple, uniform values of σ it can (with high probability) obtain an m with a few bits set in some desired way. By choosing σ in some specific manner, it may also be possible to influence the resulting message for which a forgery is output.

Forging a signature on an arbitrary message. A more damaging attack on the plain RSA signature scheme requires the adversary to obtain *two* signatures from the signer, but allows the adversary to output a forged signature on any message of its choice. Say the adversary wants to forge a signature on the message $m \in \mathbb{Z}_N^*$ with respect to the public key $pk = \langle N, e \rangle$. The adversary chooses arbitrary $m_1, m_2 \in \mathbb{Z}_N^*$ distinct from m such that $m = m_1 \cdot m_2 \bmod N$. It then obtains signatures σ_1, σ_2 on m_1, m_2, respectively. Finally, it outputs $\sigma := [\sigma_1 \cdot \sigma_2 \bmod N]$ as a valid signature on m. This works because

$$\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot m_2^d)^e = m_1^{ed} \cdot m_2^{ed} = m_1 \cdot m_2 = m \bmod N,$$

using the fact that σ_1, σ_2 are valid signatures on m_1, m_2.

Being able to forge a signature on an arbitrary message is devastating. Nevertheless, one might argue that this attack is unrealistic since an adversary will not be able to convince a signer to sign the exact messages m_1 and m_2. Once again, this is irrelevant as far as Definition 12.2 is concerned. Furthermore, it is dangerous to make assumptions about what messages the signer may or may not be willing to sign. For example, a client may use a signature scheme to authenticate to a server by signing a random challenge sent by the server. Here, a malicious server would be able to obtain a signature on any message(s) of its choice. More generally, it may be possible for the adversary to choose m_1 and m_2 as "legitimate" messages that the signer will agree to sign. Finally, note that the attack can be generalized: if an adversary obtains valid signatures on q arbitrary messages $M = \{m_1, \ldots, m_q\}$, then the adversary can output a valid signature on any of $2^q - q$ other messages obtained by taking products of subsets of M (of size different from 1).

12.4.2 RSA-FDH and PKCS #1 v2.1

One can attempt to prevent the attacks from the previous section by applying some transformation to messages before signing them. That is, the signer will now specify as part of its public key a (deterministic) function H

with certain cryptographic properties (described below) mapping messages to $\mathbb{Z}_N^*$; the signature on a message m will be $\sigma := [H(m)^d \bmod N]$, and verification of the signature σ on the message m will be done by checking whether $\sigma^e \stackrel{?}{=} H(m) \bmod N$. See Construction 12.6.

CONSTRUCTION 12.6

Let GenRSA be as in the previous sections, and construct a signature scheme as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ to compute (N, e, d). The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
  As part of key generation, a function $H : \{0,1\}^* \to \mathbb{Z}_N^*$ is specified, but we leave this implicit.
- Sign: on input a private key $\langle N, d\rangle$ and a message $m \in \{0,1\}^*$, compute
  $$\sigma := [H(m)^d \bmod N].$$
- Vrfy: on input a public key $\langle N, e\rangle$, a message m, and a signature σ, output 1 if and only if $\sigma^e \stackrel{?}{=} H(m) \bmod N$.

The RSA-FDH signature scheme.

What properties does H need in order for this construction to be secure? At a minimum, to prevent the no-message attack it should be infeasible for an attacker to start with σ, compute $\hat{m} := [\sigma^e \bmod N]$, and then find a message m such that $H(m) = \hat{m}$. This, in particular, means that H should be hard to invert in some sense. To prevent the second attack, we need an H that does not admit "multiplicative relations," that is, for which it is hard to find three messages m, m_1, m_2 with $H(m) = H(m_1) \cdot H(m_2) \bmod N$. Finally, it must be hard to find collisions in H: if $H(m_1) = H(m_2)$, then m_1 and m_2 have the same signature and forgery becomes trivial.

There is no known way to choose H so that Construction 12.6 can be proven secure. However, it *is* possible to prove security if H is modeled as a random oracle that maps its inputs uniformly onto $\mathbb{Z}_N^*$; the scheme in this case is called the *RSA full-domain hash* (RSA-FDH) signature scheme. One can check that a random function of this sort satisfies the requirements discussed in the previous paragraph: a random function (with large range) is hard to invert, does not have any easy-to-find multiplicative relations, and is collision resistant. Of course, this informal reasoning does not rule out all possible attacks, but the proof of security below does.

Before turning to the formal proof, we provide some intuition. Our goal is to prove that if the RSA problem is hard relative to GenRSA, then RSA-FDH is secure when H is modeled as a random oracle. We consider first security against a no-message attack, i.e., when the adversary $\mathcal{A}$ cannot request any

signatures. Here the adversary is limited to making queries to the random oracle, and we assume without loss of generality that $\mathcal{A}$ always makes exactly q (distinct) queries to H and that if the adversary outputs a forgery (m, σ) then it had previously queried m to H.

Say there is an efficient adversary $\mathcal{A}$ that carries out a no-message attack and makes exactly q queries to H. We construct an efficient algorithm $\mathcal{A}'$ solving the RSA problem relative to GenRSA. Given input (N, e, y), algorithm $\mathcal{A}'$ runs $\mathcal{A}$ on the public key $pk = \langle N, e \rangle$. Let $m_1, \ldots, m_q$ denote the q (distinct) queries that $\mathcal{A}$ makes to H. Our algorithm $\mathcal{A}'$ answers these random-oracle queries of $\mathcal{A}$ with uniform elements of $\mathbb{Z}_N^*$ except for one query—say, the ith query, chosen uniformly from the oracle queries of $\mathcal{A}$—that is answered with y itself. Note that, from the point of view of $\mathcal{A}$, all its random-oracle queries are answered with uniform elements of $\mathbb{Z}_N^*$ (recall that y is uniform as well, although it is not chosen by $\mathcal{A}'$), and so $\mathcal{A}$ has no information about i. Moreover, the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ is identically distributed to the view of $\mathcal{A}$ when attacking the original signature scheme.

If $\mathcal{A}$ outputs a forgery (m, σ) then, because $m \in \{m_1, \ldots, m_q\}$, with probability $1/q$ we will have $m = m_i$. In that case,

$$\sigma^e = H(m) = H(m_i) = y \bmod N$$

and $\mathcal{A}'$ can output σ as the solution to its given RSA instance (N, e, y). We conclude that if $\mathcal{A}$ outputs a forgery with probability ε, then $\mathcal{A}'$ solves the RSA problem with probability ε/q. Since q is polynomial, we conclude that ε must be negligible if the RSA problem is hard relative to GenRSA.

Handling the case when the adversary is allowed to request signatures on messages of its choice is more difficult. The complication arises since our algorithm $\mathcal{A}'$ above does not know the decryption exponent d, yet now has to compute valid signatures on messages queried by $\mathcal{A}$ to its signing oracle. This seems impossible (and possibly even contradictory!) until we realize that $\mathcal{A}'$ *can* correctly compute a signature on a message m as long as it sets $H(m)$ to equal to $[\sigma^e \bmod N]$ for a *known* value σ. (Here we are using the fact that the random oracle is "programmable.") If σ is uniform then $[\sigma^e \bmod N]$ is uniform as well, and so the random oracle is still emulated "properly" by $\mathcal{A}'$.

The above intuition is formalized in the proof of the following:

THEOREM 12.7 *If the RSA problem is hard relative to* GenRSA *and H is modeled as a random oracle, then Construction 12.6 is secure.*

PROOF Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ denote Construction 12.6, and let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We assume without loss of generality that if $\mathcal{A}$ requests a signature on a message m, or outputs a forgery (m, σ), then it previously queried m to H. Let $q(n)$ be a polynomial upper bound on the number of queries $\mathcal{A}$ makes to H on security parameter n; we assume without loss of generality that $\mathcal{A}$ makes exactly $q(n)$ distinct queries to H.

For convenience, we list the steps of experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{GenRSA}(1^n)$ *is run to obtain* (N, e, d). *A random function* $H : \{0,1\}^* \to \mathbb{Z}_N^*$ *is chosen.*
2. *The adversary* $\mathcal{A}$ *is given* $pk = \langle N, e\rangle$, *and may query* H *as well as a signing oracle* $\mathsf{Sign}_{\langle N,d\rangle}(\cdot)$ *that, on input a message* m, *returns* $\sigma := [H(m)^d \bmod N]$.
3. $\mathcal{A}$ *outputs* (m, σ), *where it had not previously requested a signature on* m. *The output of the experiment is* 1 *if and only if* $\sigma^e = H(m) \bmod N$.

We define a modified experiment $\mathsf{Sig\text{-}forge}'_{\mathcal{A},\Pi}(n)$ in which a guess is made at the outset as to which message (from among the q messages that $\mathcal{A}$ queries to H) will correspond to the eventual forgery (if any) output by $\mathcal{A}$:

1. *Choose uniform* $j \in \{1, \ldots, q\}$.
2. $\mathsf{GenRSA}(1^n)$ *is run to obtain* (N, e, d). *A random function* $H : \{0,1\}^* \to \mathbb{Z}_N^*$ *is chosen.*
3. *The adversary* $\mathcal{A}$ *is given* $pk = \langle N, e\rangle$, *and may query* H *as well as a signing oracle* $\mathsf{Sign}_{\langle N,d\rangle}(\cdot)$ *that, on input a message* m, *returns* $\sigma := [H(m)^d \bmod N]$.
4. $\mathcal{A}$ *outputs* (m, σ), *where it had not previously requested a signature on* m. *Let* i *be such that* $m = m_i$.[2] *The output of the experiment is* 1 *if and only if* $\sigma^e = H(m) \bmod N$ *and* $j = i$.

Since j is uniform and independent of everything else, the probability that $j = i$ (even conditioned on the event that $\mathcal{A}$ outputs a forgery) is exactly $1/q$. Therefore $\Pr[\mathsf{Sig\text{-}forge}'_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{q(n)} \cdot \Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1]$.

Now consider the modified experiment $\mathsf{Sig\text{-}forge}''_{\mathcal{A},\Pi}(n)$ in which the experiment is aborted if $\mathcal{A}$ ever requests a signature on the message m_j (where m_j denotes the jth message queried to H, and j is the uniform value chosen at the outset). This does not change the probability that the output of the experiment is 1, since if $\mathcal{A}$ ever requests a signature on m_j then it cannot possible output a forgery on m_j. In words,

$$\begin{aligned}\Pr[\mathsf{Sig\text{-}forge}''_{\mathcal{A},\Pi}(n) = 1] &= \Pr[\mathsf{Sig\text{-}forge}'_{\mathcal{A},\Pi}(n) = 1] \\ &= \frac{\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1]}{q(n)}. \end{aligned} \tag{12.1}$$

[2]Here m_i denotes the ith query made to H. Recall, by assumption, that if $\mathcal{A}$ requests a signature on a message m, then it must have previously queried m to H.

Finally, consider the following algorithm $\mathcal{A}'$ solving the RSA problem:

Algorithm $\mathcal{A}'$:
The algorithm is given (N, e, y) as input.

1. Choose uniform $j \in \{1, \ldots, q\}$.
2. Run $\mathcal{A}$ on input the public key $pk = \langle N, e \rangle$. Store triples $(\cdot, \cdot, \cdot)$ in a table, initially empty. An entry (m_i, σ_i, y_i) indicates that $\mathcal{A}'$ has set $H(m_i) = y_i$, and $\sigma_i^e = y_i \bmod N$.
3. When $\mathcal{A}$ makes its ith random-oracle query $H(m_i)$, answer it as follows:
   - If $i = j$, return y as the answer to the query.
   - Else choose uniform $\sigma_i \in \mathbb{Z}_N^*$, compute $y_i := [\sigma_i^e \bmod N]$, return y_i as the answer to the query, and store (m_i, σ_i, y_i) in the table.

   When $\mathcal{A}$ requests a signature on message m, let i be such that $m = m_i$ and answer the query as follows[3]
   - If $i = j$ then $\mathcal{A}'$ aborts.
   - If $i \neq j$ then there is an entry (m_i, σ_i, y_i) in the table. Return σ_i as the answer to the query.
4. At the end of $\mathcal{A}$'s execution, it outputs (m, σ). If $m = m_j$ and $\sigma^e = y \bmod N$, then output σ.

Clearly, $\mathcal{A}'$ runs in probabilistic polynomial time. Say the input (N, e, y) to $\mathcal{A}'$ is generated by running $\mathsf{GenRSA}(1^n)$ to obtain (N, e, d), and then choosing uniform $y \in \mathbb{Z}_N^*$. The crucial observation is that the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ is identical to the view of $\mathcal{A}$ in experiment $\mathsf{Sig\text{-}forge}''_{\mathcal{A},\Pi}(n)$. In particular, all Sign-oracle queries are answered correctly, and each of the random-oracle queries of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ is answered with a uniform element of $\mathbb{Z}_N^*$:

- The query $H(m_j)$ is answered with y, a uniform element of $\mathbb{Z}_N^*$.
- Queries $H(m_i)$ with $i \neq j$ are answered with $y_i = [\sigma_i^e \bmod N]$, where σ_i is uniform in $\mathbb{Z}_N^*$. Since exponentiation to the eth power is a one-to-one function, y_i is uniformly distributed as well.

Finally, observe that whenever experiment $\mathsf{Sig\text{-}forge}''_{\mathcal{A},\Pi}(n)$ would output 1, then $\mathcal{A}'$ outputs a correct solution to its given RSA instance. This follows since $\mathsf{Sig\text{-}forge}''_{\mathcal{A},\Pi}(n) = 1$ implies that $j = i$ and $\sigma^e = H(m_i) \bmod N$. Now, when $j = i$, algorithm $\mathcal{A}'$ does not abort and in addition $H(m_i) = y$. Thus, $\sigma^e =$

[3]Here m_i denotes the ith query made to H. Recall, by assumption, that if $\mathcal{A}$ requests a signature on a message m, then it must have previously queried m to H.

$H(m_i) = y \bmod N$, and so σ is the desired inverse. Using Equation (12.1), this means that

$$\begin{aligned}\Pr[\mathsf{RSA\text{-}inv}_{\mathcal{A}',\mathsf{GenRSA}}(n) = 1] &= \Pr[\mathsf{Sig\text{-}forge}''_{\mathcal{A},\Pi}(n) = 1] \\ &= \frac{\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1]}{q(n)}. \end{aligned} \tag{12.2}$$

If the RSA problem is hard relative to GenRSA, there is a negligible function negl such that $\Pr[\mathsf{RSA\text{-}inv}_{\mathcal{A}',\mathsf{GenRSA}}(n) = 1] \leq \mathsf{negl}(n)$. Since $q(n)$ is polynomial, we conclude from Equation (12.2) that $\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1]$ is negligible as well. This completes the proof. ■

RSA PKCS #1 v2.1. The RSA PKCS #1 v2.1 standard includes a signature scheme that can be viewed as a variant of RSA-FDH. A bit more precisely, the standard defines a scheme in which the signature on a message depends on a *salt* (i.e., a random value) chosen by the signer at the time of signature generation. If this salt is fixed to NULL by the signer—something that is allowed by the standard—the resulting scheme is very similar to RSA-FDH.

It is critical for the range of H to be (close to) *all* of $\mathbb{Z}_N^*$; in particular it does not suffice to simply let H be an "off-the-shelf" cryptographic hash function such as SHA-1. (The output length of SHA-1 is much smaller than the length of RSA moduli used in practice.) Indeed, *practical attacks* on Construction 12.6 are known if the output length of H is too small (e.g., if the output length is 160 bits as would be the case if SHA-1 were used directly as H). In the PKCS #1 v2.1 signature scheme, H is constructed via repeated application of an underlying cryptographic hash function.

12.5 Signatures from the Discrete-Logarithm Problem

Signature schemes can be based on the discrete-logarithm assumption as well, although the assumption does not lend itself as readily to signatures as the RSA assumption does. In Section 12.5.1 we describe a signature scheme introduced by Claus Schnorr that can be proven secure in the random-oracle model. In Section 12.5.2 we describe the DSA and ECDSA signature schemes which are used widely.

12.5.1 The Schnorr Signature Scheme

The underlying intuition for the Schnorr signature scheme is best explained by taking a slight detour to discuss (public-key) *identification schemes*. We then describe the *Fiat–Shamir transform* that can be used to convert identification schemes to signature schemes in the random-oracle model. Finally,

we present the Schnorr identification scheme—and corresponding signature scheme—based on the discrete-logarithm problem.

Identification Schemes

An identification scheme is an interactive protocol that allows one party to prove its identity (i.e., to *authenticate* itself) to another. This is a very natural notion, and it is common nowadays to authenticate oneself when logging in to a website. We call the party identifying herself (e.g., the user) the "prover," and the party verifying the identity (e.g., the web server) the "verifier." Here, we are interested in the public-key setting where the prover and verifier do not share any secret information (such as a password) in advance; instead, the verifier only knows the public key of the prover. Successful execution of the identification protocol convinces the verifier that it is communicating with the intended prover rather than an imposter.

We will only consider three-round identification protocols of a specific form, where the prover is specified by two algorithms $\mathcal{P}_1, \mathcal{P}_2$ and the verifier's side of the protocol is specified by an algorithm $\mathcal{V}$. The prover runs $\mathcal{P}_1(sk)$ using its private key sk to obtain an initial message I along with some state st, and initiates the protocol by sending I to the verifier. In response, the verifier sends a challenge r chosen uniformly from some set Ω_{pk} defined by the prover's public key pk. Next, the prover runs $\mathcal{P}_2(sk, \mathsf{st}, r)$ to compute a response s that it sends back to the verifier. Finally, the verifier computes $\mathcal{V}(pk, r, s)$ and accepts if and only if this results in the initial message I; see Figure 12.1. Of course, for correctness we require that if the legitimate prover executes the protocol correctly then the verifier should always accept.

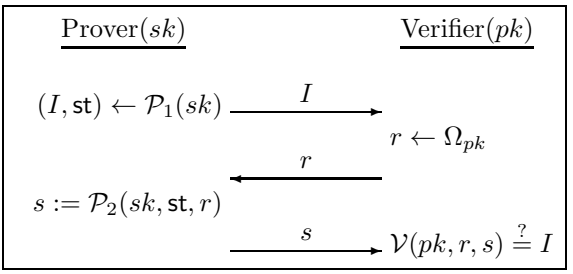

FIGURE 12.1: A three-round identification scheme.

For technical reasons, we assume identification schemes that are "non-degenerate," which intuitively means that there are many possible initial messages I, and none has a high probability of being sent. Formally, a scheme is *non-degenerate* if for every private key sk and any fixed initial message I, the

probability that $\mathcal{P}_1(sk)$ outputs I is negligible. (Any identification scheme can be trivially modified to be non-degenerate by sending a uniform n-bit string along with the initial message.)

The basic security requirement of an identification scheme is that an adversary who does not know the prover's secret key should be unable to fool the verifier into accepting. This should hold even if the attacker is able to passively eavesdrop on multiple (honest) executions of the protocol between the prover and verifier. We formalize such eavesdropping via an oracle Trans_{sk} that, when called without any input, runs an honest execution of the protocol and returns to the adversary the entire transcript (I, r, s) of the interaction.

Let $\Pi = (\mathsf{Gen}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ be an identification scheme, and consider the following experiment for an adversary $\mathcal{A}$ and parameter n:

The identification experiment $\mathsf{Ident}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *and access to an oracle* Trans_{sk} *that it can query as often as it likes.*
3. *At any point during the experiment,* $\mathcal{A}$ *outputs a message* I. *A uniform challenge* $r \in \Omega_{pk}$ *is chosen and given to* $\mathcal{A}$*, who responds with some* s. *(*$\mathcal{A}$ *may continue to query* Trans_{sk} *even after receiving* r*.)*
4. *The experiment outputs* 1 *if and only if* $\mathcal{V}(pk, r, s) \stackrel{?}{=} I$.

DEFINITION 12.8 *An identification scheme* $\Pi = (\mathsf{Gen}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ *is* secure against a passive attack*, or just* secure*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

It is also possible to consider stronger notions of security, for example, where the adversary can also carry out *active* attacks on the protocol by impersonating a verifier and possibly sending maliciously chosen values r. We will not need this for our application to signature schemes.

From Identification Schemes to Signatures

The Fiat–Shamir transform (Construction 12.9) provides a way to convert any (interactive) identification scheme into a (non-interactive) signature scheme. The basic idea is for the signer to act as a prover, running the identification protocol *by itself*. That is, to sign a message m, the signer first computes I, and next generates the challenge r by applying some function H to I and m. It then derives the correct response s. The signature on m is (r, s), which can be verified by (1) recomputing $I := \mathcal{V}(pk, r, s)$ and then (2) checking that $H(I, m) \stackrel{?}{=} r$.

CONSTRUCTION 12.9

Let $(\mathsf{Gen}_{\mathsf{id}}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ be an identification scheme, and construct a signature scheme as follows:

- Gen: on input 1^n, simply run $\mathsf{Gen}_{\mathsf{id}}(1^n)$ to obtain keys pk, sk. The public key pk specifies a set of challenges Ω_{pk}. As part of key generation, a function $H : \{0,1\}^* \to \Omega_{pk}$ is specified, but we leave this implicit.
- Sign: on input a private key sk and a message $m \in \{0,1\}^*$, do:
  1. Compute $(I, \mathsf{st}) \leftarrow \mathcal{P}_1(sk)$.
  2. Compute $r := H(I, m)$.
  3. Compute $s := \mathcal{P}_2(sk, \mathsf{st}, r)$.

  Output the signature (r, s).
- Vrfy: on input a public key pk, a message m, and a signature (r, s), compute $I := \mathcal{V}(pk, r, s)$ and output 1 if and only if $H(I, m) \stackrel{?}{=} r$.

The Fiat–Shamir transform.

A signature (r, s) is "bound" to a specific message m because r is a function of both I and m; changing m thus results in a completely different r. If H is modeled as a random oracle mapping inputs uniformly onto Ω_{pk}, then the challenge r is uniform; intuitively, it will be just as difficult for an adversary (who does not know sk) to find a valid signature (r, s) on a message m as it would be to impersonate the prover in an honest execution of the protocol. This intuition is formalized in the proof of the following theorem.

THEOREM 12.10 *Let Π be an identification scheme, and let Π' be the signature scheme that results by applying the Fiat–Shamir transform to it. If Π is secure and H is modeled as a random oracle, then Π' is secure.*

PROOF Let $\mathcal{A}'$ be a probabilistic polynomial-time adversary attacking the signature scheme Π', with $q = q(n)$ an upper bound on the number of queries that $\mathcal{A}'$ makes to H. We make a number of simplifying assumptions without loss of generality. First, we assume that $\mathcal{A}'$ makes any given query to H only once. We also assume that after being given a signature (r, s) on a message m with $\mathcal{V}(pk, r, s) = I$, the adversary $\mathcal{A}'$ never queries $H(I, m)$ (since it knows the answer will be r). Finally, we assume that if $\mathcal{A}'$ outputs a forged signature (r, s) on a message m with $\mathcal{V}(pk, r, s) = I$, then $\mathcal{A}'$ had previously queried $H(I, m)$.

We construct an efficient adversary $\mathcal{A}$ that uses $\mathcal{A}'$ as a subroutine and attacks the identification scheme Π:

Algorithm $\mathcal{A}$:
The algorithm is given pk and access to an oracle Trans_{sk}.

1. Choose uniform $j \in \{1, \ldots, q\}$.
2. Run $\mathcal{A}'(pk)$. Answer its queries as follows:
   When $\mathcal{A}'$ makes its ith random-oracle query $H(I_i, m_i)$, answer it as follows:
   - If $i = j$, output I_j and receive in return a challenge r. Return r to $\mathcal{A}'$ as the answer to its query.
   - If $i \neq j$, choose a uniform $r \in \Omega_{pk}$ and return r as the answer to the query.

   When $\mathcal{A}'$ requests a signature on m, answer it as follows:
   (a) Query Trans_{sk} to obtain a transcript (I, r, s) of an honest execution of the protocol.
   (b) Return the signature (r, s).
3. If $\mathcal{A}'$ outputs a forged signature (r, s) on a message m, compute $I := \mathcal{V}(pk, r, s)$ and check whether $(I, m) \stackrel{?}{=} (I_j, m_j)$. If so, then output s. Otherwise, abort.

The view of $\mathcal{A}'$ when run as a subroutine by $\mathcal{A}$ in experiment $\mathsf{Ident}_{\mathcal{A},\Pi}(n)$ is *almost* identical to the view of $\mathcal{A}'$ in experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n)$. Indeed, all the H-queries that $\mathcal{A}'$ makes are answered with a uniform value from Ω_{pk}, and all the signing queries that $\mathcal{A}'$ makes are answered with valid signatures having the correct distribution. The only difference between the views is that when $\mathcal{A}'$ is run as a subroutine by $\mathcal{A}$ it is possible for there to be an inconsistency in the answers $\mathcal{A}'$ receives from its queries to H: specifically, this happens if $\mathcal{A}$ ever answers a signing query for a message m using a transcript (I, r, s) for which $H(I, m)$ is already defined (that is, $\mathcal{A}'$ had previously queried (I, m) to H) and $H(I, m) \neq r$. However, if Π is non-degenerate then this only ever happens with negligible probability. Thus, the probability that $\mathcal{A}'$ outputs a forgery when run as a subroutine by $\mathcal{A}$ is $\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n) - \mathsf{negl}(n)$ for some negligible function negl.

Consider an execution of experiment $\mathsf{Ident}_{\mathcal{A},\Pi}(n)$ in which $\mathcal{A}'$ outputs a forged signature (r, s) on a message m, and let $I := \mathcal{V}(pk, r, s)$. Since j is uniform and independent of everything else, the probability that $(I, m) = (I_j, m_j)$ (even conditioned on the event that $\mathcal{A}'$ outputs a forgery) is exactly $1/q$. (Recall we assume that if $\mathcal{A}'$ outputs a forged signature (r, s) on a message m with $\mathcal{V}(pk, r, s) = I$, then $\mathcal{A}'$ had previously queried $H(I, m)$.) When both events happen, $\mathcal{A}$ successfully impersonates the prover. Indeed, $\mathcal{A}$ sends I_j as its initial message, receives in response a challenge r, and responds with s. But $H(I_j, m_j) = r$ and (since the forged signature is valid) $\mathcal{V}(pk, r, s) = I$. Putting everything together, we see that

$$\Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] \geq \frac{1}{q(n)} \cdot \big(\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1] - \mathsf{negl}(n)\big)$$

or

$$\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1] \leq q(n) \cdot \Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] + \mathsf{negl}(n).$$

If Π is secure then $\Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1]$ is negligible; since $q(n)$ is polynomial this implies that $\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1]$ is also negligible. Because $\mathcal{A}'$ was arbitrary, this means Π' is secure. ■

The Schnorr Identification Scheme

The Schnorr identification scheme is based on hardness of the discrete-logarithm problem. Let $\mathcal{G}$ be a polynomial-time algorithm that takes as input 1^n and (except possibly with negligible probability) outputs a description of a cyclic group $\mathbb{G}$, its order q (with $\|q\| = n$), and a generator g. To generate its keys, the prover runs $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$, chooses a uniform $x \in \mathbb{Z}_q$, and sets $y := g^x$; the public key is $\langle \mathbb{G}, q, g, y \rangle$ and the private key is x. To execute the protocol (see Figure 12.2), the prover begins by choosing a uniform $k \in \mathbb{Z}_q^*$ and setting $I := g^k$; it sends I as the initial message. The verifier chooses and sends a uniform challenge $r \in \mathbb{Z}_q$; in response, the prover computes $s := [rx + k \bmod q]$. The verifier accepts if and only if $g^s \cdot y^{-r} \stackrel{?}{=} I$. Correctness holds because

$$g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^k = I.$$

Note that I is uniform in $\mathbb{G}$, and so the scheme is non-degenerate.

Before giving the proof, we provide some high-level intuition. A first important observation is that passive eavesdropping is of no help to the attacker. The reason is that the attacker can *simulate* transcripts of honest executions on its own, based only on the public key and *without* knowledge of the private key. To do this, the attacker just reverses the order of the steps: it first chooses uniform and independent $r, s \in \mathbb{Z}_q$ and then sets $I := g^s \cdot y^{-r}$. In an honest transcript (I, r, s), the initial message I is a uniform element of $\mathbb{G}$, the challenge is an independent, uniform element of $\mathbb{Z}_q$, and s is then uniquely determined as $s = \log_g(I \cdot y^r)$. Simulated transcripts constructed by an attacker

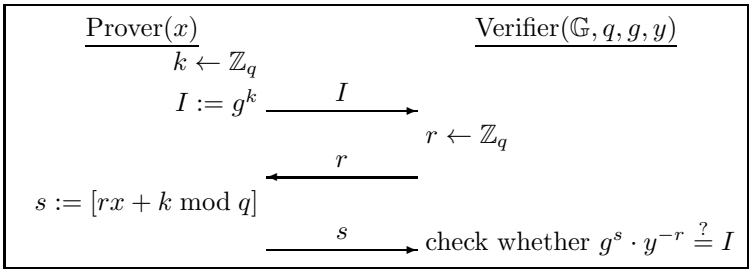

FIGURE 12.2: An execution of the Schnorr identification scheme.

have the same distribution: $r \in \mathbb{Z}_q$ is uniform and, because s is uniform in $\mathbb{Z}_q$ and independent of r, we see that I is uniform in $\mathbb{G}$ and independent of r. Finally, s is uniquely determined as satisfying the same constraint as before. Due to this, we may effectively assume that when attacking the identification scheme, an attacker does not eavesdrop on honest executions at all.

So, we have reduced to an attacker who gets a public key y, sends an initial message I, is given in response a uniform challenge r, and then must send a response s for which $g^s \cdot y^{-r} = I$. Informally, if an attacker is able to do this with high probability then it must, in particular, be able to compute correct responses s_1, s_2 to at least two different challenges $r_1, r_2 \in \mathbb{Z}_q$. Note

$$g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2},$$

and so $g^{s_1 - s_2} = y^{r_1 - r_2}$. But this implies that the attacker (who, recall, is able to generate s_1 in response to r_1, and s_2 in response to r_2) can implicitly compute the discrete logarithm

$$\log_g y = [(s_1 - s_2) \cdot (r_1 - r_2)^{-1} \bmod q],$$

contradicting the assumed hardness of the discrete-logarithm problem.

THEOREM 12.11 *If the discrete-logarithm problem is hard relative to $\mathcal{G}$, then the Schnorr identification scheme is secure.*

PROOF Let Π denote the Schnorr identification scheme, and let $\mathcal{A}$ be a PPT adversary attacking the scheme. We construct the following PPT algorithm $\mathcal{A}'$ solving the discrete-logarithm problem relative to $\mathcal{G}$:

> **Algorithm $\mathcal{A}'$:**
> The algorithm is given $\mathbb{G}, q, g, y$ as input.
>
> 1. Run $\mathcal{A}(pk)$, answering all its queries to Trans_{sk} as described in the intuition given previously.
> 2. When $\mathcal{A}$ outputs I, choose a uniform $r_1 \in \mathbb{Z}_q$ as the challenge. Give r_1 to $\mathcal{A}$, who responds with s_1.
> 3. Run $\mathcal{A}(pk)$ a second time (from the beginning), using the same randomness as before except for uniform and independent $r_2 \in \mathbb{Z}_q$. Eventually, $\mathcal{A}$ responds with s_2.
> 4. If $g^{s_1} \cdot h^{-r_1} = I$ and $g^{s_2} \cdot h^{-r_2} = I$ and $r_1 \neq r_2$ then output $[(s_1 - s_2) \cdot (r_1 - r_2)^{-1} \bmod q]$. Else, output nothing.

Considering a single run of $\mathcal{A}$ as a subroutine of $\mathcal{A}'$, let ω denote the randomness used in that execution except for the challenge itself. So, ω comprises any randomness used by $\mathcal{G}$, the choice of (unknown) private key x, any randomness used by $\mathcal{A}$ itself, and the randomness used by $\mathcal{A}'$ when answering queries

to Trans_{sk}. Define $V(\omega, r)$ to be equal to 1 if and only if $\mathcal{A}$ correctly responds to challenge r when randomness ω is used in the rest of the execution. For any fixed ω, define $\delta_\omega \stackrel{\text{def}}{=} \Pr_r[V(\omega, r) = 1]$; having fixed ω, this is the probability over choice of the challenge r that $\mathcal{A}$ responds correctly.

Define $\delta(n) \stackrel{\text{def}}{=} \Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1]$. Since the simulation of the Trans_{sk} oracle is perfect, we have

$$\delta(n) = \Pr_{\omega,r}[V(\omega, r) = 1] = \textstyle\sum_\omega \Pr[\omega] \cdot \delta_\omega.$$

Moreover, the intuition preceding the proof shows that $\mathcal{A}'$ correctly computes the discrete logarithm of y whenever $\mathcal{A}$ succeeds twice and $r_1 \neq r_2$. Thus:

$$\begin{aligned}
\Pr[\mathsf{DLog}_{\mathcal{A}',\mathcal{G}}(n) = 1] &= \Pr_{\omega,r_1,r_2}[V(\omega, r_1) \ \wedge \ V(\omega, r_2) \ \wedge \ r_1 \neq r_2] \\
&\geq \Pr_{\omega,r_1,r_2}[V(\omega, r_1) \ \wedge \ V(\omega, r_2)] - \Pr_{\omega,r_1,r_2}[r_1 = r_2] \\
&= \textstyle\sum_\omega \Pr[\omega] \cdot (\delta_\omega)^2 - 1/q \\
&\geq \left(\textstyle\sum_\omega \Pr[\omega] \cdot \delta_\omega\right)^2 - 1/q \\
&= \delta(n)^2 - 1/q,
\end{aligned}$$

using Jensen's inequality[4] in the second-to-last step. If the discrete-logarithm problem is hard relative to $\mathcal{G}$ then $\Pr[\mathsf{DLog}_{\mathcal{A}',\mathcal{G}}(n) = 1]$ is negligible. Since $1/q$ is negligible (because $\|q\| = n$), this implies that $\delta(n)$ is also negligible, and so Π is a secure identification scheme. ■

The Schnorr signature scheme is obtained by applying the Fiat–Shamir transform to the Schnorr identification scheme. See Construction 12.12.

CONSTRUCTION 12.12

Let $\mathcal{G}$ be as described in the text.

- Gen: run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. Choose a uniform $x \in \mathbb{Z}_q$ and set $y := g^x$. The private key is x and the public key is $(\mathbb{G}, q, g, y)$. As part of key generation, a function $H : \{0,1\}^* \to \mathbb{Z}_q$ is specified, but we leave this implicit.
- Sign: on input a private key x and a message $m \in \{0,1\}^*$, choose uniform $k \in \mathbb{Z}_q$ and set $I := g^k$. Then compute $r := H(I, m)$, followed by $s := [rx + k \bmod q]$. Output the signature (r, s).
- Vrfy: on input a public key $(\mathbb{G}, q, g, y)$, a message m, and a signature (r, s), compute $I := g^s \cdot y^{-r}$ and output 1 if $H(I, m) \stackrel{?}{=} r$.

The Schnorr signature scheme.

[4] Jensen's inequality says that $\sum_i a_i \cdot b_i^2 \geq \left(\sum_i a_i\right)^{-1} \cdot \left(\sum_i a_i \cdot b_i\right)^2$ for positive $\{a_i\}$.

12.5.2 DSA and ECDSA

The *Digital Signature Algorithm* (DSA) and *Elliptic Curve Digital Signature Algorithm* (ECDSA) are based on the discrete-logarithm problem in different classes of groups. They have been around in some form since 1991, and are both included in the current *Digital Signature Standard* (DSS) issued by NIST.

Both schemes follow a common template and can be viewed as being constructed from an underlying identification scheme (see the previous section). Let $\mathbb{G}$ be a cyclic group of prime order q with generator g. Consider the following identification scheme in which the prover's private key is x and public key is $(\mathbb{G}, q, g, y)$ with $y = g^x$:

1. The prover chooses uniform $k \in \mathbb{Z}_q^*$ and sends $I := g^k$.
2. The verifier chooses and sends uniform $\alpha, r \in \mathbb{Z}_q$ as the challenge.
3. The prover sends $s := [k^{-1} \cdot (\alpha + xr) \bmod q]$ as the response.
4. The verifier accepts if $s \neq 0$ and $g^{\alpha s^{-1}} \cdot y^{rs^{-1}} \stackrel{?}{=} I$.

Note $s \neq 0$ unless $\alpha = -xr \bmod q$, which occurs with negligible probability. Assuming $s \neq 0$, the inverse $s^{-1} \bmod q$ exists and

$$g^{\alpha s^{-1}} \cdot y^{rs^{-1}} = g^{\alpha s^{-1}} \cdot g^{xrs^{-1}} = g^{(\alpha + xr) \cdot s^{-1}} = g^{(\alpha+xr) \cdot k \cdot (\alpha+xr)^{-1}} = I.$$

We thus see that correctness holds with all but negligible probability.

One can show that this identification scheme is secure if the discrete-logarithm problem is hard relative to $\mathcal{G}$. We merely sketch the argument, assuming familiarity with the results of the previous section. First of all, transcripts of honest executions can be simulated: to do so, simply choose uniform $\alpha, r \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q^*$, and then set $I := g^{\alpha s^{-1}} \cdot y^{rs^{-1}}$. (This no longer gives a *perfect* simulation, but it is close enough.) Moreover, if an attacker outputs an initial message I for which it can give correct responses $s_1, s_2 \in \mathbb{Z}_q^*$ to distinct challenges $(\alpha, r_1), (\alpha, r_2)$ then

$$g^{\alpha s_1^{-1}} \cdot y^{r_1 s_1^{-1}} = I = g^{\alpha s_2^{-1}} \cdot y^{r_2 s_2^{-1}},$$

and so $g^{\alpha(s_1^{-1} - s_2^{-1})} = h^{r_1 s_1^{-1} - r_2 s_2^{-1}}$ and $\log_g h$ can be computed as in the previous section. The same holds if the attacker gives correct responses to distinct challenges $(\alpha_1, r), (\alpha_2, r)$.

The DSA/ECDSA signature schemes are constructed by "collapsing" the above identification scheme into a non-interactive algorithm run by the signer. In contrast to the Fiat–Shamir transform, however, the transformation here is carried out as follows (see Construction 12.13):

- Set $\alpha := H(m)$, where m is the message being signed and H is a cryptographic hash function.

- Set $r := F(I)$ for a (specified) function $F : \mathbb{G} \to \mathbb{Z}_q$. Here, F is a "simple" function that is *not* intended to act like a random oracle.

The function F depends on the group $\mathbb{G}$m, which in turn depends on the scheme. In DSA, $\mathbb{G}$ is taken to be an order-q subgroup of $\mathbb{Z}_p^*$, for p prime (cf. Section 8.3.3), and $F(I) \stackrel{\text{def}}{=} [I \bmod q]$. In ECDSA, $\mathbb{G}$ is an order-q subgroup of an elliptic-curve group $E(\mathbb{Z}_p)$, for p prime.[5] Recall from Section 8.3.4 that any element of such a group can be represented as a pair $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$. The function F in this case is defined as $F((x, y)) \stackrel{\text{def}}{=} [x \bmod q]$.

CONSTRUCTION 12.13

Let $\mathcal{G}$ be as in the text.

- Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x$. The public key is $\langle \mathbb{G}, q, g, y \rangle$ and the private key is x.

  As part of key generation, two functions $H : \{0,1\}^* \to \mathbb{Z}_q$ and $F : \mathbb{G} \to \mathbb{Z}_q$ are specified, but we leave this implicit.

- Sign: on input the private key x and a message $m \in \{0,1\}^*$, choose uniform $k \in \mathbb{Z}_q^*$ and set $r := F(g^k)$. Then compute $s := [k^{-1} \cdot (H(m) + xr) \bmod q]$. (If $r = 0$ or $s = 0$ then start again with a fresh choice of k.) Output the signature (r, s).

- Vrfy: on input a public key $\langle \mathbb{G}, q, g, y \rangle$, a message $m \in \{0,1\}^*$, and a signature (r, s) with $r, s \neq 0 \bmod q$, output 1 if and only if

$$r \stackrel{?}{=} F\left(g^{H(m) \cdot s^{-1}} y^{r \cdot s^{-1}}\right).$$

DSA and ECDSA—abstractly.

Assuming hardness of the discrete-logarithm problem, DSA and ECDSA can be proven secure if H and F are modeled as random oracles. As we have discussed above, however, while the random-oracle model may be reasonable for H, it is *not* an appropriate model for F. No proofs of security are known for the specific choices of F in the standard. Nevertheless, DSA and ECDSA have been used and studied for decades without any attacks being found.

Proper generation of k. The DSA/ECDSA schemes specify that the signer should choose a uniform $k \in \mathbb{Z}_q^*$ when computing a signature. Failure to choose k properly (e.g., due to poor random-number generation) can lead to catastrophic results. For starters, if an attacker can predict the value of k used to compute a signature (r, s) on a message m, then they can compute

[5] ECDSA also allows elliptic curves over other fields, but we have only covered the case of prime fields in Section 8.3.4.

the signer's private key. This is true because $s = k^{-1} \cdot (H(m) + xr) \bmod q$, and if k is known then the only unknown is the private key x.

Even if k is unpredictable, the attacker can compute the signer's private key if the *same* k is ever used to generate two different signatures. The attacker can easily tell when this happens because then r repeats as well. Say (r, s_1) and (r, s_2) are signatures on messages m_1 and m_2, respectively. Then

$$s_1 = k^{-1} \cdot (H(m_1) + xr) \bmod q$$
$$s_2 = k^{-1} \cdot (H(m_2) + xr) \bmod q.$$

Subtracting gives $s_1 - s_2 = k^{-1}(H(m_1) - H(m_2)) \bmod q$, from which k can be computed; given k, the attacker can determine the private key x as in the previous paragraph. This very attack was used by hackers to extract the master private key from the Sony PlayStation (PS3) in 2010.

12.6 *Signatures from Hash Functions

Interestingly—and perhaps somewhat surprisingly—signature schemes can be constructed based on cryptographic hash functions, *without* relying on number-theoretic assumptions. (This is in contrast to public-key encryption, which seems to require hard problems with some algebraic structure.) In this section we explore such constructions. The schemes we will see are also interesting in that they do not rely on random oracles, as opposed to all the constructions we have described previously in this chapter.

12.6.1 Lamport's Signature Scheme

We initiate our study of signature schemes based on hash functions by considering the relatively weak notion of *one-time-secure signature schemes*. Informally, such schemes are "secure" as long as a given private key is used to sign only a *single* message. Schemes satisfying this notion of security may be appropriate for some applications, and also serve as useful "building blocks" for achieving stronger notions of security, as we will see in the following section.

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and consider the following experiment for an adversary $\mathcal{A}$ and parameter n:

The one-time signature experiment $\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *and asks a* single *query* m' *to its oracle* $\mathsf{Sign}_{sk}(\cdot)$. $\mathcal{A}$ *then outputs* (m, σ) *with* $m \neq m'$.
3. *The output of the experiment is defined to be* 1 *if and only if* $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

DEFINITION 12.14 *Signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* existentially unforgeable under a single-message attack*, or is a* one-time-secure signature scheme*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\Pr\left[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \mathsf{negl}(n).$$

Leslie Lamport showed a construction of a one-time-secure signature scheme in 1979. We illustrate the idea for the case of signing 3-bit messages. Let H be a cryptographic hash function. A private key consists of six uniform values $x_{1,0}, x_{1,1}, x_{2,0}, x_{2,1}, x_{3,0}, x_{3,1} \in \{0,1\}^n$, and the corresponding public key contains the results obtained by applying H to each of these elements. These keys can be visualized as two-dimensional arrays:

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}.$$

To sign a message $m = m_1 m_2 m_3$ (where $m_i \in \{0,1\}$), the signer releases the appropriate preimage x_{i,m_i} for each bit of the message; the signature σ consists of the three values $(x_{1,m_1}, x_{2,m_2}, x_{3,m_3})$. Verification is carried out in the natural way: presented with the candidate signature (x_1, x_2, x_3) on the message $m = m_1 m_2 m_3$, accept if and only if $H(x_i) \stackrel{?}{=} y_{i,m_i}$ for $1 \leq i \leq 3$. This is shown graphically in Figure 12.3, and the general case—for messages of any length ℓ—is described formally in Construction 12.15.

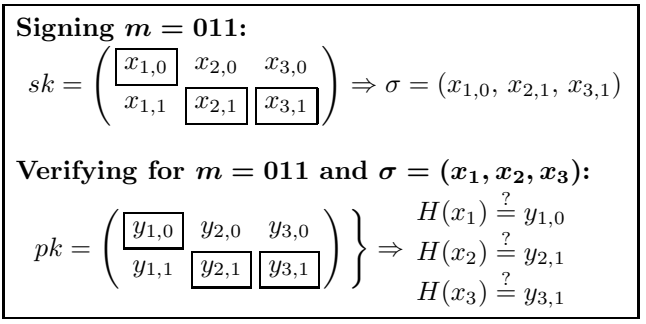

FIGURE 12.3: The Lamport scheme used to sign the message $m = 011$.

After observing a signature on a message, an attacker who wishes to forge a signature on any *other* message must find a preimage of one of the three "unused" elements in the public key. If H is *one-way* (see Definition 8.72), then finding any such preimage is computationally difficult.

THEOREM 12.16 *Let* ℓ *be any polynomial. If* H *is a one-way function, then Construction 12.15 is a one-time-secure signature scheme.*

CONSTRUCTION 12.15

Let $H : \{0,1\}^* \to \{0,1\}^*$ be a function. Construct a signature scheme for messages of length $\ell = \ell(n)$ as follows:

- Gen: on input 1^n, proceed as follows for $i \in \{1, \ldots, \ell\}$:
  1. Choose uniform $x_{i,0}, x_{i,1} \in \{0,1\}^n$.
  2. Compute $y_{i,0} := H(x_{i,0})$ and $y_{i,1} := H(x_{i,1})$.

  The public key pk and the private key sk are

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}.$$

- Sign: on input a private key sk as above and a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, output the signature $(x_{1,m_1}, \ldots, x_{\ell,m_\ell})$.
- Vrfy: on input a public key pk as above, a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, and a signature $\sigma = (x_1, \ldots, x_\ell)$, output 1 if and only if $H(x_i) = y_{i,m_i}$ for all $1 \le i \le \ell$.

The Lamport signature scheme.

PROOF Let $\ell = \ell(n)$ throughout. As noted a moment ago, the key observation is this: say an attacker $\mathcal{A}$ requests a signature on a message m', and consider any other message $m \neq m'$. There must be at least one position $i^* \in \{1, \ldots, \ell\}$ on which m and m' differ. Say $m_{i^*} = b \neq m'_{i^*}$. Then forging a signature on m requires, at least, finding a preimage (under H) of element y_{i^*,b^*} of the public key. Since H is one-way, this should be infeasible. We now formalize this intuition.

Let Π denote the Lamport scheme, and let $\mathcal{A}$ be a probabilistic polynomial-time adversary. In a particular execution of $\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n)$, let m' denote the message whose signature is requested by $\mathcal{A}$ (we assume without loss of generality that $\mathcal{A}$ always requests a signature on a message), and let (m, σ) be the final output of $\mathcal{A}$. We say that $\mathcal{A}$ *outputs a forgery at* (i, b) if $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ and furthermore $m_i \neq m'_i$ (i.e., messages m and m' differ on their ith position) and $m_i = b \neq m'_i$. Note that whenever $\mathcal{A}$ outputs a forgery, it outputs a forgery at *some* (i, b).

Consider the following PPT algorithm $\mathcal{I}$ attempting to invert H:

Algorithm $\mathcal{I}$:
The algorithm is given 1^n and y as input.

1. Choose uniform $i^* \in \{1, \ldots, \ell\}$ and $b^* \in \{0,1\}$. Set $y_{i^*,b^*} := y$.
2. For all $i \in \{1, \ldots, \ell\}$ and $b \in \{0,1\}$ with $(i, b) \neq (i^*, b^*)$:
   - Choose uniform $x_{i,b} \in \{0,1\}^n$ and set $y_{i,b} := H(x_{i,b})$.
3. Run $\mathcal{A}$ on input $pk := \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$.

4. When $\mathcal{A}$ requests a signature on the message m':
   - If $m'_{i^*} = b^*$, then $\mathcal{I}$ aborts the execution.
   - Otherwise, $\mathcal{I}$ returns the signature $\sigma = (x_{1,m'_1}, \ldots, x_{\ell,m'_\ell})$.
5. When $\mathcal{A}$ outputs (m, σ) with $\sigma = (x_1, \ldots, x_\ell)$:
   - If $\mathcal{A}$ outputs a forgery at (i^*, b^*), then output x_{i^*}.

Whenever $\mathcal{A}$ outputs a forgery at (i^*, b^*), algorithm $\mathcal{I}$ succeeds in inverting its given input y. We are interested in the probability that this occurs when the input to $\mathcal{I}$ is generated by choosing uniform $x \in \{0,1\}^n$ and setting $y := H(x)$ (cf. Definition 8.72). Imagine a "mental experiment" in which $\mathcal{I}$ is given x at the outset, sets $x_{i^*,b^*} := x$, and then always returns a signature to $\mathcal{A}$ in step 4 (i.e., even if $m'_{i^*} = b^*$). The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{I}$ in this mental experiment is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n)$. Because (i^*, b^*) was chosen uniformly at the beginning of the experiment, and the view of $\mathcal{A}$ is independent of this choice, the probability that $\mathcal{A}$ outputs a forgery at (i^*, b^*), conditioned on the fact that $\mathcal{A}$ outputs a forgery at all, is at least $1/2\ell$. (This is because a signature forgery implies a forgery for at least one point (i, b). Since there are $2\ell(n)$ points, the probability of the forgery being at (i^*, b^*) is at least $1/2\ell$.) We conclude that, in this mental experiment, the probability that $\mathcal{A}$ outputs a forgery at (i^*, b^*) is at least $\frac{1}{2\ell} \cdot \Pr[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1]$.

Returning to the real experiment involving $\mathcal{I}$ as initially described, the key observation is that *the probability that $\mathcal{A}$ outputs a forgery at (i^*, b^*) is unchanged*. This is because the mental experiment and the real experiment coincide if $\mathcal{A}$ outputs a forgery at (i^*, b^*). That is, the experiments only differ if $m'_{i^*} = b^*$, but if this happens then it is impossible (by definition) for $\mathcal{A}$ to subsequently output a forgery at (i^*, b^*). So the probability that $\mathcal{A}$ outputs a forgery at (i^*, b^*) is still at least $\frac{1}{2\ell} \cdot \Pr[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1]$. In other words,

$$\Pr[\mathsf{Invert}_{\mathcal{I},H}(n) = 1] \geq \frac{1}{2\ell} \cdot \Pr[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1].$$

Because H is a one-way function, there is a negligible function negl such that

$$\mathsf{negl}(n) \geq \Pr[\mathsf{Invert}_{\mathcal{I},H}(n) = 1].$$

Since ℓ is polynomial this implies that $\Pr[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1]$ is negligible, completing the proof. ■

COROLLARY 12.17 *If one-way functions exist, then for any polynomial ℓ there is a one-time-secure signature scheme for messages of length ℓ.*

12.6.2 Chain-Based Signatures

Being able to sign only a single message with a given private key is obviously a significant drawback. We show here an approach based on collision-resistant hash functions that allows a signer to sign *arbitrarily many* messages, at the expense of maintaining *state* that must be updated after each signature is generated. In Section 12.6.3 we discuss a more efficient variant of this approach (that still requires state), and then describe how this modified construction can be made *stateless*. The result shows that signature schemes satisfying Definition 12.2 can be constructed based on collision-resistant hash functions.

We first define signature schemes that allow the signer to maintain *state* that is updated after every signature is produced.

DEFINITION 12.18 *A* stateful signature scheme *is a tuple of probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *satisfying the following:*

1. *The* key-generation algorithm Gen *takes as input a security parameter* 1^n *and outputs* (pk, sk, s_0)*. These are called the* public key*,* private key*, and* initial state*, respectively. We assume* pk *and* sk *each has length at least* n*, and that* n *can be determined from* pk, sk*.*

2. *The* signing algorithm Sign *takes as input a private key* sk*, a value* s_{i-1}*, and a message* $m \in \{0,1\}^*$*. It outputs a signature* σ *and a value* s_i*.*

3. *The deterministic* verification algorithm Vrfy *takes as input a public key* pk*, a message* m*, and a signature* σ*. It outputs a bit* b*.*

We require that for every n*, every* (pk, sk, s_0) *output by* $\mathsf{Gen}(1^n)$*, and any messages* $m_1, \ldots, m_t \in \{0,1\}^*$*, if we iteratively compute* $(\sigma_i, s_i) \leftarrow \mathsf{Sign}_{sk, s_{i-1}}(m_i)$ *for* $i = 1, \ldots, t$*, then for every* $i \in \{1, \ldots, t\}$*, it holds that* $\mathsf{Vrfy}_{pk}(m_i, \sigma_i) = 1$*.*

We emphasize that the verifier does not need to know the signer's state in order to verify a signature; in fact, in some schemes the state must be kept secret by the signer in order for security to hold. Signature schemes that do not maintain state (as in Definition 12.1) are called *stateless* to distinguish them from stateful schemes. Clearly, stateless schemes are preferable (although stateful schemes can still potentially be useful). We introduce stateful signatures as a stepping stone to an eventual stateless construction.

Security for stateful signatures schemes is exactly analogous to Definition 12.2, with the only subtleties being that the signing oracle returns only the signature (and *not* the state), and that the signing oracle updates the state each time it is invoked.

For any polynomial $t = t(n)$, we can easily construct a stateful "t-time-secure" signature scheme. (The definition of security here would be the obvious generalization of Definition 12.14.) We can do this by simply letting the public key (resp., private key) consist of t independently generated public keys (resp., private keys) for any one-time-secure signature scheme; i.e.,

set $pk := \langle pk_1, \ldots, pk_t \rangle$ and $sk := \langle sk_1, \ldots, sk_t \rangle$ where each (pk_i, sk_i) is an independently generated key-pair for some one-time-secure signature scheme. The state is a counter i initially set to 1. To sign a message m using the private key sk and current state $i \leq t$, compute $\sigma \leftarrow \mathsf{Sign}_{sk_i}(m)$ (that is, generate a signature on m using the private key sk_i) and output (σ, i); the state is updated to $i := i + 1$. Since the state starts at 1, this means the ith message is signed using sk_i. Verification of a signature (σ, i) on a message m is done by checking whether σ is a valid signature on m with respect to pk_i. This scheme is secure if used to sign t messages since each private key of the underlying one-time-secure scheme is used to sign only a *single* message.

As described, signatures have constant length (i.e., independent of t), but the public key has length *linear* in t. It is possible to trade off the length of the public key and signature by having the signer compute a Merkle tree $h := \mathcal{MT}_t(pk_1, \ldots, pk_t)$ (see Section 5.6.2) over the t underlying public keys from the one-time-secure scheme. That is, the public key will now be $\langle t, h \rangle$, and the signature on the ith message will include (σ, i), as before, along with the ith value pk_i and a proof π_i that this is the correct value corresponding to h. (Verification is done in the natural way.) The public key now has constant size, and the signature length grows only logarithmically with t.

Since t can be an arbitrary polynomial, why don't the previous schemes give us the solution we are looking for? The main drawback is that they require the upper bound t on the number of messages that can be signed *to be fixed in advance*, at the time of key generation. This is a potentially severe limitation since once the upper bound is reached a new public key would have to be generated and distributed. We would prefer instead to have a single, fixed public key that can be used to sign an *unbounded* number of messages.

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a one-time-secure signature scheme. In the scheme we have just described (ignoring the Merkle-tree optimization), the signer runs t invocations of Gen to obtain public keys $pk_1, \ldots, pk_t$, and includes each of these in its actual public key pk. The signer is then restricted to signing at most t messages. We can do better by using a "chain-based" scheme in which the signer generates additional public keys *on-the-fly*, as needed.

In the chain-based scheme, the public key consists of just a single public key pk_1 generated using Gen, and the private key is just the associated private key sk_1. To sign the first message m_1, the signer first generates a new key-pair (pk_2, sk_2) using Gen, and then signs both m_1 and pk_2 using sk_1 to obtain $\sigma_1 \leftarrow \mathsf{Sign}_{sk_1}(m_1 \| pk_2)$. The signature that is output includes both pk_2 and σ_1, and the signer adds $(m_1, pk_2, sk_2, \sigma_1)$ to its current state. In general, when it comes time to sign the ith message the signer will have stored $\{(m_j, pk_{j+1}, sk_{j+1}, \sigma_j)\}_{j=1}^{i-1}$ as part of its state. To sign the ith message m_i, the signer first generates a new key-pair (pk_{i+1}, sk_{i+1}) using Gen, and then signs m_i and pk_{i+1} using sk_i to obtain a signature $\sigma_i \leftarrow \mathsf{Sign}_{sk_i}(m_i \| pk_{i+1})$. The actual signature that is output includes pk_{i+1}, σ_i, and also the values $\{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1}$. The signer then adds $(m_i, pk_{i+1}, sk_{i+1}, \sigma_i)$ to its state. See Figure 12.4 for a graphical depiction of this process.

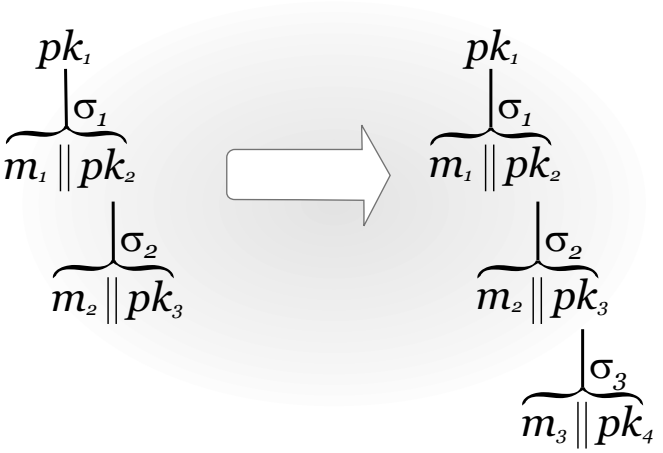

FIGURE 12.4: Chain-based signatures: the situation before and after signing the third message m_3.

To verify a signature $(pk_{i+1}, \sigma_i, \{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1})$ on a message $m = m_i$ with respect to public key pk_1, the receiver verifies each link between a public key pk_j and the next public key pk_{j+1} in the chain, as well as the link between the last public key pk_i and m. That is, verification outputs 1 if and only if $\mathsf{Vrfy}_{pk_j}(m_j \| pk_{j+1}, \sigma_j) \stackrel{?}{=} 1$ for all $j \in \{1, \ldots, i\}$. (Refer to Figure 12.4.)

It is not hard to be convinced—at least on an intuitive level—that this signature scheme is existentially unforgeable under an adaptive chosen-message attack (regardless of how many messages are signed). Informally, this is once again due to the fact that each key-pair (pk_i, sk_i) is used to sign only a single "message," where in this case the "message" is actually a message/public-key pair $m_i \| pk_{i+1}$. Since we will prove security of a more efficient scheme in the next section, we do not prove security for the chain-based scheme here.

In the chain-based scheme, each public key pk_i is used to sign both a message and another public key. Thus, it is essential that the underlying one-time-secure signature scheme Π *is capable of signing messages longer than the public key.* The Lamport scheme presented in Section 12.6.1 does *not* have this property. However, if we apply the hash-and-sign paradigm from Section 12.3 to the Lamport scheme, we *do* obtain a one-time-secure signature scheme that can sign messages of arbitrary length. (Although Theorem 12.4 was stated only with regard to signature schemes satisfying Definition 12.2, it is not hard to see that an identical proof works for one-time-secure signature schemes.) Because this result is crucial for the next section, we state it formally. (Note that the existence of collision-resistant hash functions implies the existence of one-way functions; see Exercise 7.4.)

LEMMA 12.19 *If collision-resistant hash functions exist, then there exists a one-time-secure signature scheme (for messages of arbitrary length).*

The chain-based signature scheme is a stateful signature scheme that is existentially unforgeable under an adaptive chosen-message attack. It has a number of disadvantages, though. For one, there is no immediate way to eliminate the state (recall that our ultimate goal is a stateless scheme satisfying Definition 12.2). It is also not very efficient, in that the signature length, size of the state, and verification time are all linear in the number of messages that have been signed. Finally, each signature reveals all previously signed messages, and this may be undesirable in some contexts.

12.6.3 Tree-Based Signatures

The signer in the chain-based scheme of the previous section can be viewed as maintaining a *tree* of degree 1, rooted at the public key pk_1, and with depth equal to the number of messages signed so far (cf. Figure 12.4). A natural way to improve efficiency is to use a *binary* tree in which each node has degree 2. As before, a signature will correspond to a "signed" path in the tree from a leaf to the root; as long as the tree has polynomial depth (even if it has exponential size!), verification can be done in polynomial time.

Concretely, to sign messages of length n we will work with a binary tree of depth n having 2^n leaves. As before, the signer will add nodes to the tree "on-the-fly," as needed. In contrast to the chain-based scheme, however, only leaves (and not internal nodes) will be used for signing messages. Each leaf of the tree will correspond to one of the possible messages of length n.

In more detail, we imagine a binary tree of depth n where the root is labeled by ε (i.e., the empty string), and a node that is labeled with the binary string w (of length less than n) has left-child labeled $w0$ and right-child labeled $w1$. This tree is never constructed in its entirety (note that it has exponential size), but is instead built up by the signer as needed.

For every node w, we associate a pair of keys pk_w, sk_w for a one-time-secure signature scheme Π. The public key of the root, pk_ε, is the actual public key of the signer. To sign a message $m \in \{0,1\}^n$, the signer does the following:

1. It first generates keys (as needed) for all nodes on the path from the root to the leaf labeled m. (Some of these public keys may have been generated in the process of signing previous messages, and in that case are not generated again.)

2. Next, it "certifies" the path from the root to the leaf labeled m by computing a signature on $pk_{w0} \| pk_{w1}$, using private key sk_w, for each string w that is a proper prefix of m.

3. Finally, it "certifies" m itself by computing a signature on m using the private key sk_m.

The final signature on m consists of the signature on m with respect to pk_m, as well as all the information needed to verify the path from the leaf labeled

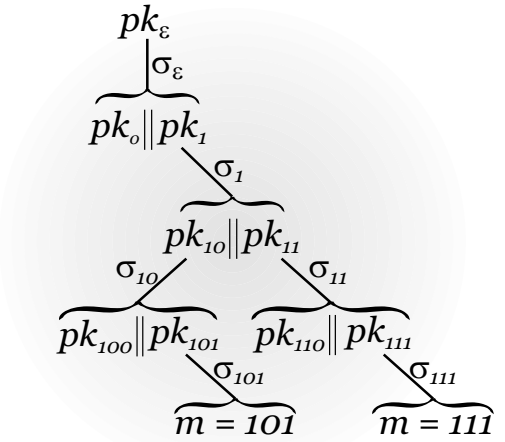

FIGURE 12.5: Tree-based signatures (conceptually).

m to the root; see Figure 12.5. Additionally, the signer updates its state by storing all the keys generated as part of the above signing process. A formal description of this scheme is given as Construction 12.20.

Notice that each of the underlying keys in this scheme is used to sign only a *single* "message." Each key associated with an internal node signs a pair of public keys, and a key at a leaf is used to sign only a single message. Since each key is used to sign a *pair* of other keys, we again need the one-time-secure signature scheme Π to be capable of signing messages longer than the public key. Lemma 12.19 shows that such schemes can be constructed based on collision-resistant hash functions.

Before proving security of this tree-based approach, note that it improves on the chain-based scheme in a number of respects. It still allows for signing an unbounded number of messages. (Although there are only 2^n leaves, the message space contains only 2^n messages. In any case, 2^n is eventually larger than any polynomial function of n.) In terms of efficiency, the signature length and verification time are now proportional to the message length n but are *independent* of the number of messages signed. The scheme is still stateful, but we will see how this can be avoided after we prove the following result.

THEOREM 12.21 *Let Π be a one-time-secure signature scheme. Then Construction 12.20 is a secure signature scheme.*

PROOF Let Π^* denote Construction 12.20. Let $\mathcal{A}^*$ be a probabilistic polynomial time adversary, let $\ell^* = \ell^*(n)$ be a (polynomial) upper bound on the number of signing queries made by $\mathcal{A}^*$, and set $\ell = \ell(n) \stackrel{\text{def}}{=} 2n\ell^*(n) + 1$.

CONSTRUCTION 12.20

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. For a binary string m, let $m|_i \stackrel{\text{def}}{=} m_1 \cdots m_i$ denote the i-bit prefix of m (with $m|_0 \stackrel{\text{def}}{=} \varepsilon$, the empty string). Construct the scheme $\Pi^* = (\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*)$ as follows:

- Gen^*: on input 1^n, compute $(pk_\varepsilon, sk_\varepsilon) \leftarrow \mathsf{Gen}(1^n)$ and output the public key pk_ε. The private key and initial state are sk_ε.
- Sign^*: on input a message $m \in \{0,1\}^n$, carry out the following.
  1. For $i = 0$ to $n-1$:
     - If $pk_{m|_i 0}, pk_{m|_i 1}$, and $\sigma_{m|_i}$ are not in the state, compute $(pk_{m|_i 0}, sk_{m|_i 0}) \leftarrow \mathsf{Gen}(1^n)$, $(pk_{m|_i 1}, sk_{m|_i 1}) \leftarrow \mathsf{Gen}(1^n)$, and $\sigma_{m|_i} \leftarrow \mathsf{Sign}_{sk_{m|_i}}(pk_{m|_i 0} \,\|\, pk_{m|_i 1})$. In addition, add all of these values to the state.
  2. If σ_m is not yet included in the state, compute $\sigma_m \leftarrow \mathsf{Sign}_{sk_m}(m)$ and store it as part of the state.
  3. Output the signature $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1},\ \sigma_m\right)$.
- Vrfy^*: on input a public key pk_ε, message m, and signature $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1},\ \sigma_m\right)$, output 1 if and only if:
  1. $\mathsf{Vrfy}_{pk_{m|_i}}(pk_{m|_i 0} \,\|\, pk_{m|_i 1}, \sigma_{m|_i}) \stackrel{?}{=} 1$ for all $i \in \{0, \ldots, n-1\}$.
  2. $\mathsf{Vrfy}_{pk_m}(m, \sigma_m) \stackrel{?}{=} 1$.

A "tree-based" signature scheme.

Note that ℓ upper bounds the number of public keys from Π that are needed to generate ℓ^* signatures using Π^*. This is because each signature in Π^* requires at most $2n$ new keys from Π (in the worst case), and one additional key from Π is used as the actual public key pk_ε.

Consider the following PPT adversary $\mathcal{A}$ attacking the one-time-secure signature scheme Π:

Adversary $\mathcal{A}$:
$\mathcal{A}$ is given as input a public key pk (the security parameter n is implicit).

- Choose a uniform index $i^* \in \{1, \ldots, \ell\}$. Construct a list $pk^1, \ldots, pk^\ell$ of keys as follows:
  - Set $pk^{i^*} := pk$.
  - For $i \neq i^*$, compute $(pk^i, sk^i) \leftarrow \mathsf{Gen}(1^n)$.
- Run $\mathcal{A}^*$ on input public key $pk_\varepsilon = pk^1$. When $\mathcal{A}^*$ requests a signature on a message m do:
  1. For $i = 0$ to $n-1$:

– If the values $pk_{m|_i 0}, pk_{m|_i 1}$, and $\sigma_{m|_i}$ have not yet been defined, then set $pk_{m|_i 0}$ and $pk_{m|_i 1}$ equal to the next two unused public keys pk^j and pk^{j+1}, and compute a signature $\sigma_{m|_i}$ on $pk_{m|_i 0} \,\|\, pk_{m|_i 1}$ with respect to $pk_{m|_i}$.[6]

2. If σ_m is not yet defined, compute a signature σ_m on m with respect to pk_m (see footnote 6).
3. Give $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1},\ \sigma_m\right)$ to $\mathcal{A}^*$.

- Say $\mathcal{A}^*$ outputs a message m (for which it had not previously requested a signature) and a signature $\left(\{\ \sigma'_{m|_i}, pk'_{m|_i 0}, pk'_{m|_i 1}\ \}_{i=0}^{n-1},\ \sigma'_m\right)$. If this is a valid signature on m, then:

  Case 1: Say there exists a $j \in \{0, \ldots, n-1\}$ for which $pk'_{m|_j 0} \neq pk_{m|_j 0}$ or $pk'_{m|_j 1} \neq pk_{m|_j 1}$; this includes the case when $pk_{m|_j 0}$ or $pk_{m|_j 1}$ were never defined by $\mathcal{A}$. Take the minimal such j, and let i be such that $pk^i = pk_{m|_j} = pk'_{m|_j}$ (such an i exists by the minimality of j). If $i = i^*$, output $(pk'_{m|_j 0} \| pk'_{m|_j 1},\ \sigma'_{m|_j})$.

  Case 2: If case 1 does not hold, then $pk'_m = pk_m$. Let i be such that $pk^i = pk_m$. If $i = i^*$, output (m, σ'_m).

In experiment $\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n)$, the view of $\mathcal{A}^*$ being run as a subroutine by $\mathcal{A}$ is distributed identically to the view of $\mathcal{A}^*$ in experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A}^*,\Pi^*}(n)$.[7] Thus, the probability that $\mathcal{A}^*$ outputs a forgery is exactly $\delta(n)$ when it is run as a subroutine by $\mathcal{A}$ in this experiment. Given that $\mathcal{A}^*$ outputs a forgery, consider each of the two possible cases described above:

Case 1: Since i^* is uniform and independent of the view of $\mathcal{A}^*$, the probability that $i = i^*$ is exactly $1/\ell$. If $i = i^*$, then $\mathcal{A}$ requested a signature on the message $pk_{m|_j 0} \| pk_{m|_j 1}$ with respect to the public key $pk = pk^{i^*} = pk_{m|_j}$ that it was given (and requested no other signatures). Moreover,

$$pk'_{m|_j 0} \| pk'_{m|_j 1} \neq pk_{m|_j 0} \| pk_{m|_j 1}$$

and yet $\sigma'_{m|_j}$ is a valid signature on $pk'_{m|_j 0} \| pk'_{m|_j 1}$ with respect to pk. Thus, $\mathcal{A}$ outputs a forgery in this case.

Case 2: Again, since i^* was chosen uniformly at random and is independent of the view of $\mathcal{A}^*$, the probability that $i = i^*$ is exactly $1/\ell$. If $i = i^*$, then $\mathcal{A}$

[6] If $i \neq i^*$ then $\mathcal{A}$ can compute a signature with respect to pk^i by itself. $\mathcal{A}$ can also obtain a (single) signature with respect to pk^{i^*} by making the appropriate query to its signing oracle. This is what is meant here.

[7] As we have mentioned, $\mathcal{A}$ never "runs out" of public keys. A signing query of $\mathcal{A}^*$ uses $2n$ public keys; thus, even if new public keys were required to answer *every* signing query of $\mathcal{A}^*$ (which will in general not be the case), only $2n\ell^*(n)$ public keys would be needed by $\mathcal{A}$ in addition to the "root" public key pk_ε.

did not request any signatures with respect to the public key $pk = pk^i = pk_m$ and yet σ'_m is a valid signature on m with respect to pk.

We see that, conditioned on $\mathcal{A}^*$ outputting a forgery, $\mathcal{A}$ outputs a forgery with probability exactly $1/\ell$. This means that

$$\Pr[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1] = \Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}^*,\Pi^*}(n) = 1]/\ell(n).$$

Because Π is a one-time-secure signature scheme, there is a negligible function negl for which

$$\Pr[\mathsf{Sig\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Since ℓ is polynomial, this means $\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}^*,\Pi^*}(n) = 1]$ is negligible. ■

A Stateless Solution

As described, the signer generates state on-the-fly as needed. However, we can imagine having the signer generate the necessary information for all the nodes in the entire tree *in advance*, at the time of key generation. (That is, at the time of key generation the signer could generate the keys $\{(pk_w, sk_w)\}$ and the signatures $\{\sigma_w\}$ for all binary strings w of length at most n.) If key generation were done in this way, the signer would not have to update its state at all; these values could all be stored as part of a (huge) private key, and we would obtain a stateless scheme. The problem with this approach, of course, is that generating all these values would require *exponential* time, and storing them all would require exponential memory.

An alternative is to store some *randomness* that can be used to generate the values $\{(pk_w, sk_w)\}$ and $\{\sigma_w\}$, as needed, rather than storing the values themselves. That is, the signer could store a random string r_w for each w, and whenever the values pk_w, sk_w are needed the signer can compute $(pk_w, sk_w) := \mathsf{Gen}(1^n; r_w)$, where this denotes the generation of a length-n key using random coins r_w. Similarly, if the signing procedure is probabilistic, the signer can store r'_w and then set $\sigma_w := \mathsf{Sign}_{sk_w}(pk_{w0} \| pk_{w1}; r'_w)$ (assuming here that $|w| < n$). Generating and storing sufficiently many random strings, however, still requires exponential time and memory.

A simple modification of this alternative gives a polynomial-time solution. Instead of storing random r_w and r'_w as suggested above, the signer can store two keys k, k' for a pseudorandom function F. When needed, the values pk_w, sk_w can now be generated by the following two-step process:

1. Compute $r_w := F_k(w)$.[8]
2. Compute $(pk_w, sk_w) := \mathsf{Gen}(1^n; r_w)$ (as before).

[8] We assume that the output length of F is sufficiently long, and that w is padded to some fixed-length string in a one-to-one fashion. We ignore these technicalities here.

In addition, the key k' is used to generate the value r'_w that is used to compute the signature σ_w. This gives a *stateless* scheme in which key generation (as well as signing and verifying) can be done in polynomial time. Intuitively, this is secure because storing a random function is equivalent to storing all the r_w and r'_w values that are needed, and storing a pseudorandom function is "just as good." We leave it as an exercise to give a formal proof that this modified scheme remains secure.

Since the existence of collision-resistant hash functions implies the existence of one-way functions (cf. Exercise 7.4), and the latter implies the existence of pseudorandom functions (see Chapter 7), we have:

THEOREM 12.22 *If collision-resistant hash functions exist, then there exists a (stateless) secure signature scheme.*

We remark that it is possible to construct signature schemes satisfying Definition 12.2 from the (minimal) assumption that one-way functions exist; a proof of this result is beyond the scope of this book.

12.7 *Certificates and Public-Key Infrastructures

In this section we briefly discuss one of the primary applications of digital signatures: the secure distribution of public keys. This brings us full circle in our discussion of public-key cryptography. In this and the previous chapter we have seen how to *use* public-key cryptography once public keys are securely distributed. Now we show how public-key cryptography itself can be used to securely distribute public keys. This may sound circular, but it is not. What we will show is that once a *single* public key, belonging to a trusted party, is distributed in a secure fashion, that key can be used to "bootstrap" the secure distribution of arbitrarily many other public keys. Thus, at least in principle, the problem of secure key distribution need only be solved *once*.

The key notion here is a *digital certificate*, which is simply a signature binding an entity to some public key. To be concrete, say a party Charlie has generated a key-pair (pk_C, sk_C) for a secure digital signature scheme (in this section, we will only be concerned with signature schemes satisfying Definition 12.2). Assume further that another party Bob has also generated a key-pair (pk_B, sk_B) (in the present discussion, these may be keys for either a signature scheme or a public-key encryption scheme), and that Charlie *knows* that pk_B is Bob's public key. Then Charlie can compute the signature

$$\mathsf{cert}_{C\to B} \stackrel{\text{def}}{=} \mathsf{Sign}_{sk_C}(\texttt{`Bob's key is } pk_B\texttt{'})$$

and give this signature to Bob. We call $\mathsf{cert}_{C\to B}$ a *certificate* for Bob's key

issued by Charlie. In practice a certificate should unambiguously identify the party holding a particular public key and so a more uniquely descriptive term than "Bob" would be used, for example, Bob's full name and email address, or the URL of Bob's website.

Now say Bob wants to communicate with some other party Alice who already knows pk_C. Bob can send $(pk_B, \mathsf{cert}_{C \to B})$ to Alice, who can then verify that $\mathsf{cert}_{C \to B}$ is indeed a valid signature on the message `'Bob's key is` pk_B`'` with respect to pk_C. Assuming verification succeeds, Alice now knows that Charlie has signed the indicated message. If Alice trusts Charlie, she can accept pk_B as Bob's legitimate public key.

All communication between Bob and Alice can occur over an *insecure* and *unauthenticated* channel. If an active adversary interferes with the transmission of $(pk_B, \mathsf{cert}_{C \to B})$ from Bob to Alice, that adversary will be unable to generate a valid certificate linking Bob to any *other* public key pk'_B unless Charlie had previously signed some other certificate linking Bob with pk'_B (in which case this is anyway not much of an attack). This all assumes that Charlie is not dishonest and that his private key has not been compromised.

We have omitted many details in the above description. Most prominently, we have not discussed how Alice learns pk_C in the first place; how Charlie can be sure that pk_B is Bob's public key; and how Alice decides whether to trust Charlie. Fully specifying such details (and others) defines a *public-key infrastructure* (PKI) that enables the widespread distribution of public keys. A variety of different PKI models have been suggested, and we mention a few of the more popular ones now. Our treatment here will be kept at a relatively high level, and the reader interested in further details is advised to consult the references at the end of this chapter.

A single certificate authority. The simplest PKI assumes a single *certificate authority* (CA) who is completely trusted by everybody and who issues certificates for everyone's public key. A certificate authority would not typically be a person, but would more likely be a company whose business it is to certify public keys, a government agency, or perhaps a department within an organization (although in this latter case the CA would likely only be used by people within the organization). Anyone who wants to rely on the services of the CA would have to obtain a legitimate copy of the CA's public key pk_{CA}. Clearly, this step must be carried out in a secure fashion since if some party obtains an incorrect version of pk_{CA} then that party may not be able to obtain an authentic copy of anyone else's public key. This means that pk_{CA} must be distributed over an *authenticated* channel. The easiest way of doing this is via physical means: for example, if the CA is within an organization then any employee can obtain an authentic copy of pk_{CA} directly from the CA on their first day of work. If the CA is a company, then other users would have to go to this company at some point and, say, pick up a USB stick that contains the CA's public key. This inconvenient step need only be carried out once.

A common way for a CA to distribute its public key in practice is to "bun-

dle" this public key with some other software. For example, this occurs today in many popular web browsers: a CA's public key is provided together with the browser, and the browser is programmed to automatically verify certificates as they arrive. (Actually, modern web browsers have public keys of *multiple* CAs hard-wired into their code, and so more accurately fall into the "multiple CA" model discussed below.)

The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled, although the details may vary from CA to CA. As one example, Bob may have to show up in person with a copy of his public key pk_B along with identification proving that his name (or his email address) is what he claims. Only then would the CA issue the certificate.

In the model where there is a single CA, parties completely trust this CA to issue certificates only when appropriate; this is why it is crucial that a detailed verification process be used before a certificate is issued. As a consequence, if Alice receives a certificate $\mathsf{cert}_{CA \to B}$ certifying that pk_B is Bob's public key, Alice will accept this assertion as valid, and use pk_B as Bob's public key.

Multiple certificate authorities. While the model in which there is only one CA is simple and appealing, it is not very practical. For one thing, outside of a single organization it is unlikely for *everyone* to trust the same CA. This need not imply that anyone thinks the CA is corrupt; it could simply be the case that someone finds the CA's verification process to be insufficient (say, the CA asks for only one form of identification when generating a certificate but Alice would prefer that two be used instead). Moreover, the CA is a single point of failure for the entire system. If the CA is corrupt, or can be bribed, or even if the CA is merely lax with the way it protects its private key, the legitimacy of issued certificates may be called into question. It is also inconvenient for all parties who want certificates to have to contact this CA.

One approach to alleviating these issues is to rely on multiple CAs. A party Bob who wants to obtain a certificate on his public key can choose which CA(s) it wants to issue a certificate, and a party Alice who is presented with a certificate, or even multiple certificates issued by different CAs, can choose which CA's certificates she trusts. There is no harm in having Bob obtain a certificate from more than one CA (apart from some inconvenience and expense for Bob), but Alice must be more careful since the security of her communication is ultimately only as good as the least-secure CA that she trusts. That is, say Alice trusts two CAs CA_1 and CA_2, and CA_2 is corrupted by an adversary. Then, although this adversary will not be able to forge certificates issued by CA_1, it will be able to issue fake certificates in the name of CA_2 for any identity/public key of its choice. This is a real problem in current systems. As mentioned earlier, operating systems/web browsers typically come pre-configured with many CAs' public keys, and the default setting is for all these CAs to be treated as equally trustworthy. Essentially any company willing to pay, however, can be included as a CA. So the list of pre-configured CAs includes some reputable, well-established companies

along with other, newer companies whose trustworthiness cannot be easily established. It is left to the user to manually configure their settings so as to only accept certificates from CAs the user trusts.

Delegation and certificate chains. Another approach which alleviates some of the burden on a single CA (but does not address the security concerns of having a single point of failure) is to use *certificate chains*. We present the idea for certificate chains of length 2, although it is easy to see that everything we say generalizes to chains of arbitrary length.

Say Charlie, acting as a CA, issues a certificate for Bob as in our original discussion. Assume further that Bob's key pk_B is a public key for a signature scheme. Bob, in turn, can issue his own certificates for other parties. For example, Bob may issue a certificate for Alice of the form

$$\mathsf{cert}_{B\to A} \stackrel{\text{def}}{=} \mathsf{Sign}_{sk_B}(\texttt{`Alice's key is } pk_A\texttt{'}).$$

Now, if Alice wants to communicate with some fourth party Dave who knows Charlie's public key (but not Bob's), then Alice can send

$$pk_A,\ \mathsf{cert}_{B\to A},\ pk_B,\ \mathsf{cert}_{C\to B},$$

to Dave. What can Dave deduce from this? Well, he can first verify that Charlie, whom he trusts and whose public key is already in his possession, has signed a certificate $\mathsf{cert}_{C\to B}$ indicating that pk_B indeed belongs to someone named Bob. Dave can also verify that this person named Bob has signed a certificate $\mathsf{cert}_{B\to A}$ indicating that pk_A indeed belongs to Alice. If Dave trusts Charlie to issue certificates only to trustworthy people, then Dave may accept pk_A as being the authentic key of Alice.

We highlight that in this example stronger semantics are associated with a certificate $\mathsf{cert}_{C\to B}$. In our prior discussion, a certificate of this form was only an assertion that Bob holds public key pk_B. Now, a certificate asserts that Bob holds public key pk_B *and Bob is trusted to issue other certificates*. When Charlie signs a certificate for Bob having these stronger semantics, Charlie is, in effect, *delegating* his ability to issue certificates to Bob. Bob can now act as a proxy for Charlie, issuing certificates on Charlie's behalf.

Coming back to a CA-based PKI, we can imagine one "root" CA and n "second-level" CAs $\mathrm{CA}_1, \ldots, \mathrm{CA}_n$. The root CA can issue certificates for each of the second-level CAs, who can then in turn issue certificates for other principles holding public keys. This eases the burden on the root CA, and also makes it more convenient for parties to obtain certificates (since they may now contact the second-level CA who is closest to them, for example). On the other hand, managing these second-level CAs may be difficult, and their presence means that there are now more points of attack in the system.

The "web of trust" model. The last example of a PKI we will discuss is a fully distributed model, with no central points of trust, called the "web of

trust." A variant of this model is used by the PGP ("Pretty Good Privacy") email-encryption software for distribution of public keys.

In the "web of trust" model, anyone can issue certificates to anyone else and each user has to make their own decision about how much trust to place in certificates issued by other users. As an example of how this might work, say a user Alice is already in possession of public keys pk_1, pk_2, pk_3 for some users C_1, C_2, C_3. (We discuss below how these public keys might initially be obtained by Alice.) Another user Bob who wants to communicate with Alice might have certificates $\mathsf{cert}_{C_1 \to B}$, $\mathsf{cert}_{C_3 \to B}$, and $\mathsf{cert}_{C_4 \to B}$, and will send these certificates (along with his public key pk_B) to Alice. Alice cannot verify $\mathsf{cert}_{C_4 \to B}$ (since she doesn't have C_4's public key), but she can verify the other two certificates. Now she has to decide how much trust she places in C_1 and C_3. She may decide to accept pk_B if she unequivocally trusts C_1, or if she trusts both C_1 and C_3 to a lesser extent. (She may, for example, consider it likely that either C_1 or C_3 is corrupt, but consider it unlikely for them *both* to be corrupt.)

In this model, as described, users are expected to collect both public keys of other parties, as well as certificates on their own public key. In the context of PGP, this used to be done at "key-signing parties" where PGP users got together (say, at a conference), gave each other authentic copies of their public keys, and issued certificates for each other. In general the users at a key-signing party may not know each other, but they can check a driver's license, say, before accepting or issuing a certificate for someone's public key.

Public keys and certificates can also be stored in a central database, and this is done for PGP (see `http://pgp.mit.edu`). When Alice wants to send an encrypted message to Bob, she can search for Bob's public key in this database; along with Bob's public key, the database will return a list of all certificates it holds that have been issued for Bob's public key. It is also possible that multiple public keys for Bob will be found in the database, and each of these public keys may be certified by certificates issued by a different set of parties. Once again, Alice then needs to decide how much trust to place in any of these public keys before using them.

The web of trust model is attractive because it does not require trust in any central authority. On the other hand, while it may work well for the average user encrypting their email, it does not seem appropriate for settings where security is more critical, or for the distribution of organizational public keys (e.g., for e-commerce on the web). If a user wants to communicate with his bank, for example, it is unlikely that he would trust people he met at a conference to certify his bank's public key, and also unlikely that a bank representative will go to a key-signing party to get the bank's key certified.

Invalidating Certificates

One important issue we have not yet touched upon at all is the fact that certificates should generally not be valid indefinitely. An employee may leave

a company, in which case he or she is no longer allowed to receive encrypted communication from others within the company; a user's private key might also be stolen, at which point the user (assuming they know about the theft) will want to generate a new key-pair and have the old public key removed from circulation. In either of these scenarios, we need a way to render previously issued certificates invalid.

Approaches for handling these issues are varied and complex, and we will only mention two relatively simple ideas that, in some sense, represent opposite extremes. (Improving these methods is an active area of real-world network-security research.)

Expiration. One method for preventing certificates from being used indefinitely is to include an *expiry date* as part of the certificate. A certificate issued by a CA Charlie for Bob's public key might now have the form

$$\mathsf{cert}_{C\to B} \stackrel{\text{def}}{=} \mathsf{Sign}_{sk_C}(\texttt{`Bob's key is } pk_B\texttt{', date}),$$

where `date` is some date in the future at which point the certificate becomes invalid. (For example, one year from the day the certificate is issued.) When another user verifies this certificate, they need to know not only pk_B but also the expiry date, and they now need to check not only that the signature is valid, but also that the expiry date has not passed. A user who holds a certificate must contact the CA to get a new certificate issued whenever their current one expires; at this point, the CA verifies the identity/credentials of the user again before issuing another certificate.

Using expiry dates provides a very coarse-grained solution to the problems mentioned earlier. If an employee leaves a company the day after getting a certificate, and the certificate expires one year after its issuance date, then this employee can use his or her public key illegitimately for an entire year until the expiry date passes. For this reason, this approach is typically used in conjunction with other methods such as the one we describe next.

Revocation. When an employee leaves an organization, or a user's private key is stolen, we would like the certificates that have been issued for their public keys to become invalid immediately, or at least as soon as possible. This can be achieved by having the CA explicitly *revoke* the certificate. For simplicity we assume a single CA, but everything we say applies more generally if the user had certificates issued by multiple CAs.

There are many different ways revocation can be handled. One possibility (the only one we will discuss) is for the CA to include a serial number in every certificate it issues; that is, a certificate will now have the form

$$\mathsf{cert}_{C\to B} \stackrel{\text{def}}{=} \mathsf{Sign}_{sk_C}(\texttt{`Bob's key is } pk_B\texttt{', ###}),$$

where "`###`" represents the serial number of this certificate. Each certificate should have a unique serial number, and the CA will store the information $(\texttt{Bob}, pk_B, \texttt{###})$ for each certificate it generates.

If a user Bob's private key corresponding to a public key pk_B is stolen, Bob can alert the CA to this fact. (The CA must verify Bob's identity here, to prevent another user from falsely revoking a certificate issued to Bob.) The CA will then search its database to find the serial number associated with the certificate issued for Bob and pk_B. At the end of each day, say, the CA will then generate a *certificate revocation list* (CRL) with the serial numbers of all revoked certificates, and sign the CRL and the current date. The signed CRL is then widely distributed or otherwise made available to potential verifiers. Verification of a certificate now requires checking that the signature in the certificate is valid, checking that the serial number does not appear on the most current revocation list, and verifying the CA's signature on the revocation list itself.

In this approach the way we have described it, there is a gap of at most one day before a certificate becomes invalid. This offers more flexibility than an approach based only on expiry dates.

12.8 Putting It All Together – SSL/TLS

As a culmination of what we have covered in the book so far, we discuss the *Transport Layer Security* (TLS) protocol that is used extensively to secure communication over the web; TLS is the protocol used by your browser any time you connect to a website using `https` rather than `http`. We should be clear that our goal here is to focus on the underlying cryptography used in the core TLS protocol, and not various other aspects that, although interesting from a network-security standpoint, are irrelevant for our concerns. As usual, we have slightly simplified and abstracted parts of the protocol in order to convey the main point, and our description should not be relied upon for an implementation. Finally, we do not formally define or claim security for the protocol; indeed, a formal analysis of TLS is the subject of active research.

TLS is a standardized protocol based on a precursor called SSL (or *Secure Socket Layer*) that was developed by Netscape in the mid-1990s; the last version available was SSL 3.0. TLS version 1.0 was released in 1999, updated to version 1.1 in 2006, and updated again to version 1.2 (the current version) in 2008. As of the time of this writing, approximately 50% of websites still use TLS 1.0 rather than a more recent version; all major web browsers support TLS 1.2, although in some cases earlier versions of TLS are used by default. For the most part, our description below is at a sufficiently high level that the differences between the versions are unimportant for our purposes. We caution, however, that there are several known attacks on earlier versions.

TLS allows a client (e.g., a web browser) and a server (e.g., a website) to agree on a set of shared keys and then to use those keys to encrypt and

authenticate their subsequent communication. It consists of two parts: a *handshake protocol* that performs authenticated key exchange to establish the shared keys, and a *record-layer protocol* that uses those shared keys to encrypt/authenticate the parties' communication. Although TLS allows for clients to authenticate to servers, it is primarily used only for authentication of servers to clients because only servers typically have certificates. (After a TLS session is established, user-to-server authentication—if desired—can be done at the application layer of the network stack by, e.g., sending a password.)

The handshake protocol. We now describe the basic flow of the handshake protocol. At the outset of the protocol, the client C holds a set of CA's public keys $\{pk_1, \ldots, pk_n\}$, and the server S has a key-pair (pk_S, sk_S) for a KEM[9] along with a certificate $\mathsf{cert}_{i \to S}$ issued by one of the CAs whose public key C knows. To connect to S, the parties run the following steps.

1. C begins by sending a message to S that includes information about versions of the protocol supported by the client, the *ciphersuites* supported by the client (e.g., which hash functions or block ciphers the client allows), and a uniform value (a "nonce") N_C.

2. S responds by selecting the latest version of the protocol it supports as well as an appropriate ciphersuite. In addition, it sends its public key pk_S, its certificate $\mathsf{cert}_{i \to S}$, and its own uniform value N_S.

3. C checks whether one of the CA's public keys it holds, say pk_i, matches the CA who issued S's certificate. If so, C verifies the certificate (and also checks that it has not expired or been revoked) and, if successful, learns that pk_S is S's public key. It then runs $(c, \mathsf{pmk}) \leftarrow \mathsf{Encaps}_{pk_S}(1^n)$ (see Section 11.3) to obtain a ciphertext c and what is called a *pre-master key* pmk. It sends c to the server.

   pmk is used to derive a *master key* mk using a key-derivation function (cf. Section 5.6.4) applied to pmk, N_C, and N_S. The client then applies a pseudorandom generator to mk to derive four keys k_C, k'_C, k_S, k'_S.

   Finally, C computes $\tau_C \leftarrow \mathsf{Mac}_{\mathsf{mk}}(\mathsf{transcript})$, where $\mathsf{transcript}$ denotes all messages exchanged between C and S thus far. The client then sends τ_C to S. (In fact, τ_C is itself encrypted and authenticated as done for communication in the record-layer, described below.)

4. S computes $\mathsf{pmk} := \mathsf{Decaps}_{sk_S}(c)$, from which it can derive mk and k_C, k'_C, k_S, k'_S as the client did. If $\mathsf{Vrfy}_{\mathsf{mk}}(\mathsf{transcript}, \tau_C) \neq 1$, then S aborts. Otherwise, it sets $\tau_S \leftarrow \mathsf{Mac}_{\mathsf{mk}}(\mathsf{transcript}')$, where $\mathsf{transcript}'$ denotes all messages exchanged between C and S thus far (i.e., including

[9] It is also possible for S to have a public key for a signature scheme and to use an *ephemeral* KEM public key in the protocol below. We do not discuss this more complicated option.

the last message received from C). S then sends τ_S to C. (Again, τ_S is actually encrypted and authenticated as record-layer traffic is.)

5. If $\mathsf{Vrfy}_{\mathsf{mk}}(\mathsf{transcript}', \tau_S) \neq 1$, the client aborts.

At the end of a successful execution of the handshake protocol, C and S share a set of four symmetric keys k_C, k'_C, k_S, k'_S.

TLS 1.2 supports two KEMs: a CDH/DDH-based KEM as in Construction 11.19, or the RSA-based encryption scheme PKCS #1 v1.5. To prevent Bleichenbacher-style attacks on the scheme in the latter case, the server should not report a decryption error if Decaps fails in step 4. Instead, if decryption fails it should choose a uniform pmk and then continue running the protocol using that value. (In that case, of course, the value τ_C will not verify and S will abort; the point is that the attacker cannot distinguish whether this was due to a decryption failure or not.)

The intuition for the security of the handshake protocol is that since C verifies the certificate, it knows that only the legitimate server S can learn pmk and hence mk. Thus, if the protocol terminates successfully, C knows that it shares keys k_C, k'_C, k_S, k'_S with the legitimate S, and that no adversary learned anything about those keys. This is supposed to hold even in the presence of an active attacker, although a formal proof of this is quite involved. We remark, however, that the message authentication code on the transcript is used to prevent a man-in-the-middle attacker from changing the protocol version and ciphersuites sent at the beginning of the handshake protocol. This prevents an attacker from causing S or C to use old, weaker versions of the protocol, or weak block ciphers and short keys.

The record-layer protocol. Once keys have been agreed upon by C and S, the parties use those keys to encrypt and authenticate all their subsequent communication. C uses k_C (resp., k'_C) to encrypt (resp., authenticate) all messages it sends to S; similarly, S uses k_S and k'_S to encrypt and authenticate all the messages it sends. Sequence numbers are used to prevent replay attacks, as discussed in Section 4.5.3. TLS 1.2 uses an authenticate-then-encrypt approach, which, as we have seen in Section 4.5.2, can be problematic.

12.9 *Signcryption

To close this chapter, we briefly and informally discuss the issue of joint secrecy and integrity in the public-key setting. While this parallels our treatment from Section 4.5, the fact that we are now in the public-key setting introduces several additional complications.

For simplicity, we consider a network in which all relevant parties have public/private key-pairs for *both* encrypting and signing. We let (ek, dk) denote

a (public) encryption key and (private) decryption key, and use (vk, sk) for a (public) verification key and (private) signing key. We assume all parties know everyone else's public keys.

Informally, our goal is to design a mechanism that allows a sender S to send a message m to a receiver R while ensuring that (1) no other party in the network can learn any information about m (i.e., secrecy) and (2) R is assured that the message came from S (i.e., integrity). We will want to consider both of these security properties even against active (e.g., chosen-ciphertext) attacks by other parties in the network.

Following our discussion in Section 4.5, a natural idea is to use an "encrypt-then-authenticate" approach in which S sends $\langle S, c, \mathsf{Sign}_{sk_S}(c)\rangle$ to R, where c is an encryption of m using R's encryption key ek_R. (We explicitly include the sender's identity here for convenience.) However, there is a clever chosen-ciphertext attack here regardless of the encryption scheme used. Having observed a transmission as above, another (adversarial) party A can strip off S's signature and replace it with its own, sending $\langle A, c, \mathsf{Sign}_{sk_A}(c)\rangle$ to R. In this case, R would not detect anything wrong, and would mistakenly think that A has sent it the message m. If R replies to A, or otherwise behaves toward A in a way that depends on the contents of the message, then A can potentially learn the unknown message m.

(Another problem with this scheme, although somewhat independent of our discussion here, is that it no longer provides *non-repudiation*. That is, R cannot easily prove to a third party that S has signed the message m, at least not without divulging its own decryption key dk_R.)

One could instead try an "authenticate-then-encrypt" approach. Here, S would first compute a signature $\sigma \leftarrow \mathsf{Sign}_{sk_S}(m)$ and then send

$$\langle S, \mathsf{Enc}_{ek_R}(m\|\sigma)\rangle.$$

(Note that this solves the non-repudiation issue mentioned above.) If the encryption scheme is only CPA-secure then problems just like those mentioned in Section 4.5 apply, so let us assume a CCA-secure encryption scheme is used instead. Even then, there is an attack that can be carried by a malicious R. Upon receiving $\langle S, \mathsf{Enc}_{ek_R}(m\|\sigma)\rangle$ from S, a malicious R can decrypt to obtain $m\|\sigma$, and then re-encrypt and send $\langle S, \mathsf{Enc}_{ek_{R'}}(m\|\sigma)\rangle$ to another receiver R'. This (honest) receiver R' will then think that S sent it the message m. This can have serious consequences, e.g., if m is the message "I owe you \$100."

These attacks can be prevented if parties are more careful about how they handle identifiers. When encrypting, a sender should include its own identity along with the message; when signing, a party should sign the identity of the intended recipient along with what is being signed. For example, the second approach would be modified so that S first computes $\sigma \leftarrow \mathsf{Sign}_{sk_S}(m\|R)$, and then sends $\langle S, \mathsf{Enc}_{ek_R}(S\|m\|\sigma)\rangle$ to R. When decrypting, the receiver should check that the resulting decrypted value includes the (purported) sender's identity; when verifying, the receiver should check that what was

signed incorporates its own identity. When including identities in this way, both authenticate-then-encrypt and encrypt-then-authenticate are secure if a CCA-secure encryption scheme and a strongly secure signature scheme (whose definition parallels Definition 4.3 for MACs) are used.

References and Additional Reading

Notable early work on signatures includes that of Diffie and Hellman [58], Rabin [144, 145], Rivest, Shamir, and Adleman [148], and Goldwasser, Micali, and Yao [82]. For an extensive treatment of signature schemes beyond what could be covered here, see [98].

Goldwasser, Micali, and Rivest [81] defined the notion of existential unforgeability under an adaptive chosen-message attack, and also gave the first construction of a stateful signature scheme satisfying this definition. Goldreich [74] suggested an approach to make the Goldwasser–Micali–Rivest scheme stateless, and we have essentially adopted Goldreich's ideas in Section 12.6.3.

Plain RSA signatures date to the original RSA paper [148]. RSA-FDH was proposed by Bellare and Rogaway in their original paper on the random-oracle model [21], although the idea (without proof) of using a cryptographic hash function to prevent algebraic attacks can be traced back to Rabin [145]. A later improvement [23] was standardized as part of PKCS #1 v2.1, available at `http://www.emc.com/emc-plus/rsa-labs`.

The Fiat–Shamir transform [65] and the Schnorr signature scheme [152] both date to the late-1980s, although our proof of Theorem 12.10 is due to [1] and our proof of Theorem 12.11 is inspired by [20]. The DSA and ECDSA standards are described in [132].

Lamport's signature scheme was published in 1979 [111], although it was already described in [58]. A tree-based construction similar in spirit to Construction 12.20 was suggested by Merkle [121, 122], although a tree-based approach was also used in [81]. Naor and Yung [128] showed that one-way permutations suffice for constructing one-time-secure signatures that can sign messages of arbitrary length, and this was improved by Rompel [151], who showed that one-way functions are sufficient. (See also [99].) As we have seen in Section 12.6.3, one-time-secure signatures of this sort can be used to construct secure signature schemes, implying that one-way functions suffice for the existence of (stateless) secure signatures.

The notion of certificates was first described by Kohnfelder [107] in his undergraduate thesis. Public-key infrastructures are discussed in greater detail in [102, Chapter 15]. See also [3, 62]. Further details of the TLS protocol can be found in, e.g., [102, 165]. A formal treatment of combined secrecy and integrity in the public-key setting is given by An et al. [10].

Exercises

12.1 Show that Construction 4.7 for constructing a variable-length MAC from any fixed-length MAC can also be used (with appropriate modifications) to construct a signature scheme for arbitrary-length messages from any signature scheme for messages of fixed length $\ell(n) \geq n$.

12.2 Prove that the existence of a one-time-secure signature scheme for 1-bit messages implies the existence of one-way functions.

12.3 In Section 12.4.1 we showed an attack on the plain RSA signature scheme in which an attacker forges a signature on an arbitrary message using two signing queries. Show how an attacker can forge a signature on an arbitrary message using a *single* signing query.

12.4 Assume the RSA problem is hard. Show that the plain RSA signature scheme satisfies the following weak definition of security: an attacker is given the public key $\langle N, e \rangle$ and a uniform message $m \in \mathbb{Z}_N^*$. The adversary succeeds if it can output a valid signature on m without making any signing queries.

12.5 Another approach (besides hashing) that has been tried to construct secure RSA-based signatures is to *encode* the message before applying the RSA permutation. Here the signer fixes a public encoding function $\mathsf{enc} : \{0,1\}^\ell \to \mathbb{Z}_N^*$ as part of its public key, and the signature on a message m is $\sigma := [\mathsf{enc}(m)^d \bmod N]$.

(a) How is verification performed in encoded RSA?

(b) Discuss why appropriate choice of encoding function for $\ell \ll \|N\|$ prevents the "no-message attack" described in Section 12.4.1.

(c) Show that encoded RSA is insecure if $\mathsf{enc}(m) = \mathtt{0x00} \| m \| 0^{\kappa/10}$ (where $\kappa \stackrel{\text{def}}{=} \|N\|$, $\ell = |m| \stackrel{\text{def}}{=} 4\kappa/5$, and m is not the all-0 message). Assume $e = 3$.

(d) Show that encoded RSA is insecure for $\mathsf{enc}(m) = 0 \| m \| 0 \| m$ (where $\ell = |m| \stackrel{\text{def}}{=} (\|N\| - 1)/2$ and m is not the all-0 message). Assume $e = 3$.

(e) Solve parts (c) and (d) for arbitrary e.

12.6 Consider a variant of the Fiat–Shamir transform in which the signature is (I, s) rather than (r, s) and verification is changed in the natural way. Show that if the underlying identification scheme is secure, then the resulting signature scheme is secure here as well.

12.7 Consider a variant of DSA in which the message space is $\mathbb{Z}_q$ and H is omitted. (So the second component of the signature is now $s := [k^{-1} \cdot (m + xr) \bmod q]$.) Show that this variant is not secure.

12.8 Let f be a one-way permutation. Consider the following signature scheme for messages in the set $\{1, \ldots, n\}$:

- To generate keys, choose uniform $x \in \{0,1\}^n$ and set $y := f^{(n)}(x)$ (where $f^{(i)}(\cdot)$ refers to i-fold iteration of f, and $f^{(0)}(x) \stackrel{\text{def}}{=} x$). The public key is y and the private key is x.
- To sign message $i \in \{1, \ldots, n\}$, output $f^{(n-i)}(x)$.
- To verify signature σ on message i with respect to public key y, check whether $y \stackrel{?}{=} f^{(i)}(\sigma)$.

(a) Show that the above is not a one-time-secure signature scheme. Given a signature on a message i, for what messages j can an adversary output a forgery?

(b) Prove that no PPT adversary given a signature on i can output a forgery on any message $j > i$ except with negligible probability.

(c) Suggest how to modify the scheme so as to obtain a one-time-secure signature scheme.

Hint: Include two values y, y' in the public key.

12.9 A *strong* one-time-secure signature scheme satisfies the following (informally): given a signature σ' on a message m', it is infeasible to output $(m, \sigma) \neq (m', \sigma')$ for which σ is a valid signature on m (note that $m = m'$ is allowed).

(a) Give a formal definition of strong one-time-secure signatures.

(b) Assuming the existence of one-way functions, show a one-way function for which Lamport's scheme is *not* a strong one-time-secure signature scheme.

(c) Construct a strong one-time-secure signature scheme based on any assumption used in this book.

Hint: Use a particular one-way function in Lamport's scheme.

12.10 Consider the Lamport signature scheme. Describe an adversary who obtains signatures on *two* messages of its choice and can then forge signatures on any message it likes.

12.11 The Lamport scheme uses 2ℓ values in the public key to sign messages of length ℓ. Consider the variant in which the private key contains 2ℓ values $x_1, \ldots, x_{2\ell}$ and the public key contains the values $y_1, \ldots, y_{2\ell}$ with $y_i := f(x_i)$. A message $m \in \{0,1\}^{\ell'}$ is mapped in a one-to-one fashion to a subset $S_m \subset \{1, \ldots, 2\ell\}$ of size ℓ. To sign m, the signer reveals $\{x_i\}_{i \in S_m}$. Prove that this gives a one-time-secure signature scheme. What is the maximum message length ℓ' that this scheme supports?

12.12 At the end of Section 12.6.3, we show how a pseudorandom function can be used to make Construction 12.20 stateless. Does a similar approach work for the chain-based scheme described in Section 12.6.2? If so, sketch a construction and proof. If not, explain why and modify the scheme to obtain a stateless variant.

12.13 Prove Theorem 12.22.

12.14 Assume revocation of certificates is handled in the following way: when a user Bob claims that the private key corresponding to his public key pk_B has been stolen, the user sends to the CA a statement of this fact *signed with respect to* pk_B. Upon receiving such a signed message, the CA revokes the appropriate certificate.

Explain why it is not necessary for the CA to check Bob's identity in this case. In particular, explain why it is of no concern that an adversary who has stolen Bob's private key can forge signatures with respect to pk_B.

Chapter 13

*Advanced Topics in Public-Key Encryption

In Chapter 11 we saw several examples of public-key encryption schemes used in practice. Here, we explore some schemes that are currently more of theoretical interest—although in some cases it is possible that these schemes (or variants thereof) will be used more widely in the future.

We begin with a treatment of *trapdoor permutations*, a generalization of one-way permutations, and show how to use them to construct public-key encryption schemes. Trapdoor permutations neatly encapsulate the key characteristics of the RSA permutation that make it so useful. As such, they often provide a useful abstraction for designing new cryptosystems.

Next, we present three schemes based on problems related to factoring:

- The *Paillier encryption scheme* is an example of an encryption scheme that is *homomorphic*. This property turns out to be useful for constructing more-complex cryptographic protocols, something we touch on briefly in Section 13.3.
- The *Goldwasser–Micali encryption scheme* is of historical interest as the first scheme to be proven CPA-secure. It is also homomorphic, and uses some interesting number theory that can be applied in other contexts.
- Finally, we discuss the *Rabin trapdoor permutation*, which can be used to construct a public-key encryption scheme. Although superficially similar to the RSA trapdoor permutation, the Rabin trapdoor permutation is distinguished by the fact that its security is based *directly* on the hardness of factoring. (Recall from Section 8.2.5 that hardness of the RSA problem appears to be a stronger assumption.)

13.1 Encryption from Trapdoor Permutations

In Section 11.5.3 we saw how to construct a CPA-secure public-key encryption scheme based on the RSA assumption. By distilling those properties of

RSA that are used in the construction, and defining an abstract notion that encapsulates those properties, we obtain a general *template* for constructing secure encryption schemes based on any primitive satisfying the same set of properties. *Trapdoor permutations* turn out to be the "right" abstraction here.

In the following section we define (families of) trapdoor permutations and observe that the RSA family of one-way permutations (Construction 8.77) satisfies the additional requirements needed to be a family of *trapdoor* permutations. In Section 13.1.2 we generalize the construction from Section 11.5.3 and show that public-key encryption can be constructed from any trapdoor permutation. These results will be used again in Section 13.5, where we show a second example of a trapdoor permutation, this time based directly on the factoring assumption.

In this section we rely on the material from Section 8.4.1 or, alternately, Chapter 7.

13.1.1 Trapdoor Permutations

Recall the definitions of families of functions and families of one-way permutations from Section 8.4.1. In that section, we showed that the RSA assumption naturally gives rise to a family of one-way permutations. The astute reader may have noticed that the construction we gave (Construction 8.77) has a special property that was not remarked upon there: namely, the parameter-generation algorithm Gen outputs some additional information along with I that *enables efficient inversion of* f_I. We refer to such additional information as a *trapdoor*, and call families of one-way permutations with this additional property *families of trapdoor permutations*. A formal definition follows.

DEFINITION 13.1 *A tuple of polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Samp}, f, \mathsf{Inv})$ *is a* family of trapdoor permutations *(or a* trapdoor permutation*) if:*

- *The probabilistic* parameter-generation algorithm Gen*, on input* 1^n*, outputs* (I, td) *with* $|I| \geq n$*. Each value of* I *defines a set* $\mathcal{D}_I$ *that constitutes the domain and range of a permutation (i.e., bijection)* $f_I : \mathcal{D}_I \to \mathcal{D}_I$*.*
- *Let* Gen_1 *denote the algorithm that results by running* Gen *and outputting only* I*. Then* $(\mathsf{Gen}_1, \mathsf{Samp}, f)$ *is a family of one-way permutations.*
- *Let* (I, td) *be an output of* $\mathsf{Gen}(1^n)$*. The deterministic* inverting algorithm Inv*, on input* td *and* $y \in \mathcal{D}_I$*, outputs* $x \in \mathcal{D}_I$*. We denote this by* $x := \mathsf{Inv}_{\mathsf{td}}(y)$*. It is required that with all but negligible probability over* (I, td) *output by* $\mathsf{Gen}(1^n)$ *and uniform choice of* $x \in \mathcal{D}_I$*, we have*

$$\mathsf{Inv}_{\mathsf{td}}(f_I(x)) = x.$$

As shorthand, we drop explicit mention of Samp and simply refer to trapdoor permutation $(\mathsf{Gen}, f, \mathsf{Inv})$. For (I, td) output by Gen we write $x \leftarrow \mathcal{D}_I$ to

denote uniform selection of $x \in \mathcal{D}_I$ (with the understanding that this is done by algorithm Samp).

The second condition above implies that f_I cannot be efficiently inverted without td, but the final condition means that f_I *can* be efficiently inverted with td. It is immediate that Construction 8.77 can be modified to give a family of trapdoor permutations if the RSA problem is hard relative to GenRSA, and so we refer to that construction as the *RSA trapdoor permutation*.

13.1.2 Public-Key Encryption from Trapdoor Permutations

We now sketch how a public-key encryption scheme can be constructed from an arbitrary family of trapdoor permutations. The construction is simply a generalization of what was already done for the specific RSA trapdoor permutation in Section 11.5.3.

We begin by (re-)introducing the notion of a hard-core predicate. This is the natural adaptation of Definition 7.4 to our context, and also generalizes our previous discussion of one specific hard-core predicate for the RSA trapdoor permutation in Section 11.5.3.

DEFINITION 13.2 *Let $\Pi = (\mathsf{Gen}, f, \mathsf{Inv})$ be a family of trapdoor permutations, and let hc be a deterministic polynomial-time algorithm that, on input I and $x \in \mathcal{D}_I$, outputs a single bit $\mathsf{hc}_I(x)$. We say that hc is a* hard-core predicate of Π *if for every probabilistic polynomial-time algorithm $\mathcal{A}$ there is a negligible function negl such that*

$$\Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the experiment in which $\mathsf{Gen}(1^n)$ is run to generate (I, td) and then x is chosen uniformly from $\mathcal{D}_I$.

The asymmetry provided by trapdoor permutations implies that anyone who knows the trapdoor td associated with I can recover x from $f_I(x)$ and thus compute $\mathsf{hc}_I(x)$ from $f_I(x)$. But given only I, it is infeasible to compute $\mathsf{hc}_I(x)$ from $f_I(x)$ for a uniform x.

The following can be proved by a suitable modification of Theorem 7.5:

THEOREM 13.3 *Given a family of trapdoor permutations Π, there is a family of trapdoor permutations $\widehat{\Pi}$ with a hard-core predicate hc for $\widehat{\Pi}$.*

Given a family of trapdoor permutations $\widehat{\Pi} = (\widehat{\mathsf{Gen}}, f, \mathsf{Inv})$ with hard-core predicate hc, we can construct a single-bit encryption scheme via the following approach (see Construction 13.4 below, and compare to Construction 11.32): To generate keys, run $\widehat{\mathsf{Gen}}(1^n)$ to obtain (I, td); the public key is I and the

private key is td. Given a public key I, encryption of a message $m \in \{0,1\}$ works by choosing uniform $r \in \mathcal{D}_I$ subject to the constraint that $\mathsf{hc}_I(r) = m$, and then setting the ciphertext equal to $f_I(r)$. In order to decrypt, the receiver uses td to recover r from $f_I(r)$ and then outputs the message $m := \mathsf{hc}_I(r)$.

CONSTRUCTION 13.4

Let $\widehat{\Pi} = (\widehat{\mathsf{Gen}}, f, \mathsf{Inv})$ be a family of trapdoor permutations with hard-core predicate hc. Define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\widehat{\mathsf{Gen}}(1^n)$ to obtain (I, td). Output the public key I and the private key td.
- Enc: on input a public key I and a message $m \in \{0,1\}$, choose a uniform $r \in \mathcal{D}_I$ subject to the constraint that $\mathsf{hc}_I(r) = m$. Output the ciphertext $c := f_I(r)$.
- Dec: on input a private key td and a ciphertext c, compute the value $r := \mathsf{Inv}_I(c)$ and output the message $\mathsf{hc}_I(r)$.

Public-key encryption from any family of trapdoor permutations.

A proof of security follows along the lines of the proof of Theorem 11.33.

THEOREM 13.5 *If $\widehat{\Pi}$ is a family of trapdoor permutations with hard-core predicate hc, then Construction 13.4 is CPA-secure.*

PROOF Let Π denote Construction 13.4. We prove that Π has indistinguishable encryptions in the presence of an eavesdropper; by Proposition 11.3, this implies it is CPA-secure.

We first observe that hc must be *unbiased* in the following sense. Let

$$\delta_0(n) \stackrel{\text{def}}{=} \Pr_{(I,\mathsf{td})\leftarrow\widehat{\mathsf{Gen}}(1^n);x\leftarrow\mathcal{D}_I}[\mathsf{hc}_I(x) = 0]$$

and

$$\delta_1(n) \stackrel{\text{def}}{=} \Pr_{(I,\mathsf{td})\leftarrow\widehat{\mathsf{Gen}}(1^n);x\leftarrow\mathcal{D}_I}[\mathsf{hc}_I(x) = 1].$$

Then there is a negligible function negl such that

$$\delta_0(n), \delta_1(n) \geq \frac{1}{2} - \mathsf{negl}(n);$$

if not, then an attacker who simply outputs the more frequently occurring bit would violate Definition 13.2.

Now let $\mathcal{A}$ be a probabilistic polynomial-time adversary. Without loss of generality, we may assume $m_0 = 0$ and $m_1 = 1$ in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$.

We then have

$$\begin{aligned}\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathcal{A}(pk, c) = 0 \mid c \text{ is an encryption of } 0] \\ + \frac{1}{2} \cdot \Pr[\mathcal{A}(pk, c) = 1 \mid c \text{ is an encryption of } 1].\end{aligned}$$

But then

$$\begin{aligned}&\Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)] \\ &= \delta_0(n) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 \mid \mathsf{hc}_I(x) = 0] \\ &\quad + \delta_1(n) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 \mid \mathsf{hc}_I(x) = 1] \\ &\geq \left(\frac{1}{2} - \mathsf{negl}(n)\right) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 \mid \mathsf{hc}_I(x) = 0] \\ &\quad + \left(\frac{1}{2} - \mathsf{negl}(n)\right) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 \mid \mathsf{hc}_I(1) = 1] \\ &\geq \frac{1}{2} \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 \mid \mathsf{hc}_I(x) = 0] \\ &\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 \mid \mathsf{hc}_I(1) = 1] - 2 \cdot \mathsf{negl}(n) \\ &= \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - 2 \cdot \mathsf{negl}(n).\end{aligned}$$

Since hc is a hard-core predicate for $\widehat{\Pi}$, there is a negligible function negl' such that $\mathsf{negl}'(n) \geq \Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)]$; this means that

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}'(n) + 2 \cdot \mathsf{negl}(n),$$

completing the proof. ■

Encrypting longer messages. Using Claim 11.7, we know that we can extend Construction 13.4 to encrypt ℓ-bit messages using ciphertexts ℓ times as long. Better efficiency can be obtained by constructing a KEM, following along the lines of Construction 11.34. We leave the details as an exercise.

13.2 The Paillier Encryption Scheme

In this section we describe the *Paillier encryption scheme*, a public-key encryption scheme whose security is based on an assumption related (but not known to be equivalent) to the hardness of factoring. This encryption scheme is particularly interesting because it possesses some nice *homomorphic* properties, as we will discuss further in Section 13.2.3.

The Paillier encryption scheme utilizes the group $\mathbb{Z}_{N^2}^*$, the multiplicative group of elements in the range $\{1, \ldots, N^2\}$ that are relatively prime to N, for N a product of two distinct primes. To understand the scheme it is helpful to first understand the structure of $\mathbb{Z}_{N^2}^*$. A useful characterization of this group is given by the following proposition, which says, among other things, that $\mathbb{Z}_{N^2}^*$ is isomorphic to $\mathbb{Z}_N \times \mathbb{Z}_N^*$ (cf. Definition 8.23) for N of the form we will be interested in. We prove the proposition in the next section. (The reader willing to accept the proposition on faith can skip to Section 13.2.2.)

PROPOSITION 13.6 *Let $N = pq$, where p, q are distinct odd primes of equal length. Then:*

1. $\gcd(N, \phi(N)) = 1$.
2. *For any integer $a \geq 0$, we have $(1+N)^a = (1+aN) \bmod N^2$.*

   As a consequence, the order of $(1+N)$ in $\mathbb{Z}_{N^2}^*$ is N. That is, $(1+N)^N = 1 \bmod N^2$ and $(1+N)^a \neq 1 \bmod N^2$ for any $1 \leq a < N$.
3. *$\mathbb{Z}_N \times \mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{N^2}^*$, with isomorphism $f\colon \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$ given by*
$$f(a, b) = [(1+N)^a \cdot b^N \bmod N^2].$$

In light of the last part of the above proposition, we introduce some convenient notation. With N understood, and $x \in \mathbb{Z}_{N^2}^*$, $a \in \mathbb{Z}_N$, $b \in \mathbb{Z}_N^*$, we write $x \leftrightarrow (a, b)$ if $f(a, b) = x$ where f is the isomorphism from the proposition above. One way to think about this notation is that it means "x in $\mathbb{Z}_{N^2}^*$ *corresponds to* (a, b) in $\mathbb{Z}_N \times \mathbb{Z}_N^*$." We have used the same notation in this book with regard to the isomorphism $\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ given by the Chinese remainder theorem; we keep the notation because in both cases it refers to an isomorphism of groups. Nevertheless, there should be no confusion since the group $\mathbb{Z}_{N^2}^*$ and the above proposition are only used in this section. We remark that here the isomorphism—but not its inverse—is efficiently computable even without the factorization of N.

13.2.1 The Structure of $\mathbb{Z}_{N^2}^*$

This section is devoted to a proof of Proposition 13.6. Throughout, we let N, p, q be as in the proposition.

CLAIM 13.7 $\gcd(N, \phi(N)) = 1$.

PROOF Recall that $\phi(N) = (p-1)(q-1)$. Assume $p > q$ without loss of generality. Since p is prime and $p > p-1 > q-1$, clearly $\gcd(p, \phi(N)) = 1$. Similarly, $\gcd(q, q-1) = 1$. Now, if $\gcd(q, p-1) \neq 1$ then $\gcd(q, p-1) = q$

since q is prime. But then $(p-1)/q \geq 2$, contradicting the assumption that p and q have the same length. ■

CLAIM 13.8 *For $a \geq 0$ an integer, we have $(1+N)^a = 1 + aN \bmod N^2$. Thus, the order of $(1+N)$ in $\mathbb{Z}_{N^2}^*$ is N.*

PROOF Using the binomial expansion theorem (Theorem A.1):

$$(1+N)^a = \sum_{i=0}^{a} \binom{a}{i} N^i.$$

Reducing the right-hand side modulo N^2, all terms with $i \geq 2$ become 0 and so $(1+N)^a = 1 + aN \bmod N^2$. The smallest nonzero a such that $(1+N)^a = 1 \bmod N^2$ is therefore $a = N$. ■

CLAIM 13.9 *The group $\mathbb{Z}_N \times \mathbb{Z}_N^*$ is isomorphic to the group $\mathbb{Z}_{N^2}^*$, with isomorphism $f\colon \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$ given by $f(a,b) = [(1+N)^a \cdot b^N \bmod N^2]$.*

PROOF Note that $(1+N)^a \cdot b^N$ does not have a factor in common with N^2 since $\gcd((1+N), N^2) = 1$ and $\gcd(b, N^2) = 1$ (because $b \in \mathbb{Z}_N^*$). So $[(1+N)^a \cdot b^N \bmod N^2]$ lies in $\mathbb{Z}_{N^2}^*$. We now prove that f is an isomorphism.

We first show that f is a bijection. Since

$$\begin{aligned} |\mathbb{Z}_{N^2}^*| = \phi(N^2) = p \cdot (p-1) \cdot q \cdot (q-1) &= pq \cdot (p-1)(q-1) \\ &= |\mathbb{Z}_N| \cdot |\mathbb{Z}_N^*| = |\mathbb{Z}_N \times \mathbb{Z}_N^*| \end{aligned}$$

(see Theorem 8.19 for the second equality), it suffices to show that f is one-to-one. Say $a_1, a_2 \in \mathbb{Z}_N$ and $b_1, b_2 \in \mathbb{Z}_N^*$ are such that $f(a_1, b_1) = f(a_2, b_2)$. Then:

$$(1+N)^{a_1 - a_2} \cdot (b_1/b_2)^N = 1 \bmod N^2. \tag{13.1}$$

(Note that $b_2 \in \mathbb{Z}_N^*$ and thus $b_2 \in \mathbb{Z}_{N^2}^*$, and so b_2 has a multiplicative inverse modulo N^2.) Raising both sides to the power $\phi(N)$ and using the fact that the order of $\mathbb{Z}_{N^2}^*$ is $\phi(N^2) = N \cdot \phi(N)$ we obtain

$$\begin{aligned} (1+N)^{(a_1-a_2)\cdot\phi(N)} \cdot (b_1/b_2)^{N\cdot\phi(N)} &= 1 \bmod N^2 \\ \Rightarrow (1+N)^{(a_1-a_2)\cdot\phi(N)} &= 1 \bmod N^2 . \end{aligned}$$

By Claim 13.8, $(1+N)$ has order N modulo N^2. Applying Proposition 8.53, we see that $(a_1 - a_2) \cdot \phi(N) = 0 \bmod N$ and so N divides $(a_1 - a_2) \cdot \phi(N)$. Since $\gcd(N, \phi(N)) = 1$ by Claim 13.7, it follows that $N \mid (a_1 - a_2)$. Since $a_1, a_2 \in \mathbb{Z}_N$, this can only occur if $a_1 = a_2$.

Returning to Equation (13.1) and setting $a_1 = a_2$, we thus have $b_1^N = b_2^N \bmod N^2$. This implies $b_1^N = b_2^N \bmod N$. Since N is relatively prime to $\phi(N)$, the order of $\mathbb{Z}_N^*$, exponentiation to the power N is a bijection in $\mathbb{Z}_N^*$ (cf. Corollary 8.17). This means that $b_1 = b_2 \bmod N$; since $b_1, b_2 \in \mathbb{Z}_N^*$, we have $b_1 = b_2$. We conclude that f is one-to-one, and hence a bijection.

To show that f is an isomorphism, we show that $f(a_1, b_1) \cdot f(a_2, b_2) = f(a_1 + a_2, b_1 \cdot b_2)$. (Note that multiplication on the left-hand side of the equality takes place modulo N^2, while addition/multiplication on the right-hand side takes place modulo N.) We have:

$$\begin{aligned} f(a_1, b_1) \cdot f(a_2, b_2) &= \left((1+N)^{a_1} \cdot b_1^N\right) \cdot \left((1+N)^{a_2} \cdot b_2^N\right) \bmod N^2 \\ &= (1+N)^{a_1+a_2} \cdot (b_1 b_2)^N \bmod N^2. \end{aligned}$$

Since $(1+N)$ has order N modulo N^2 (by Claim 13.8), we can apply Proposition 8.52 and obtain

$$\begin{aligned} f(a_1, b_1) \cdot f(a_2, b_2) &= (1+N)^{a_1+a_2} \cdot (b_1 b_2)^N \bmod N^2 \\ &= (1+N)^{[a_1+a_2 \bmod N]} \cdot (b_1 b_2)^N \bmod N^2. \quad (13.2) \end{aligned}$$

We are not yet done, since $b_1 b_2$ in Equation (13.2) represents multiplication modulo N^2 whereas we would like it to be modulo N. Let $b_1 b_2 = r + \gamma N$, where γ, r are integers with $1 \leq r < N$ (r cannot be 0 since $b_1, b_2 \in \mathbb{Z}_N^*$ and so their product cannot be divisible by N). Note that $r = b_1 b_2 \bmod N$. We also have

$$\begin{aligned} (b_1 b_2)^N &= (r + \gamma N)^N \bmod N^2 \\ &= \sum_{k=0}^{N} \binom{N}{k} r^{N-k} (\gamma N)^k \bmod N^2 \\ &= r^N + N \cdot r^{N-1} \cdot (\gamma N) = r^N = ([b_1 b_2 \bmod N])^N \bmod N^2, \end{aligned}$$

using the binomial expansion theorem as in Claim 13.8. Plugging this in to Equation (13.2) we get the desired result:

$$\begin{aligned} f(a_1, b_1) \cdot f(a_2, b_2) &= (1+N)^{[a_1+a_2 \bmod N]} \cdot (b_1 b_2 \bmod N)^N \bmod N^2 \\ &= f(a_1 + a_2, b_1 b_2), \end{aligned}$$

proving that f is an isomorphism from $\mathbb{Z}_N \times \mathbb{Z}_N^*$ to $\mathbb{Z}_{N^2}^*$. ■

13.2.2 The Paillier Encryption Scheme

Let $N = pq$ be a product of two distinct primes of equal length. Proposition 13.6 says that $\mathbb{Z}_N \times \mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{N^2}^*$, with isomorphism given by $f(a, b) = [(1+N)^a \cdot b^N \bmod N^2]$. A consequence is that a uniform element

$y \in \mathbb{Z}_{N^2}^*$ corresponds to a uniform element $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}_N^*$ or, in other words, an element (a, b) with uniform $a \in \mathbb{Z}_N$ and uniform $b \in \mathbb{Z}_N^*$.

Call $y \in \mathbb{Z}_{N^2}^*$ an *Nth residue modulo N^2* if y is an Nth power, that is, if there exists an $x \in \mathbb{Z}_{N^2}^*$ with $y = x^N \bmod N^2$. We denote the set of Nth residues modulo N^2 by $\mathsf{Res}(N^2)$. Let us characterize the Nth residues in $\mathbb{Z}_{N^2}^*$. Taking any $x \in \mathbb{Z}_{N^2}^*$ with $x \leftrightarrow (a, b)$ and raising it to the Nth power gives:

$$[x^N \bmod N^2] \leftrightarrow (a, b)^N = (N \cdot a \bmod N,\ b^N \bmod N) = (0, b^N \bmod N).$$

(Recall that the group operation in $\mathbb{Z}_N \times \mathbb{Z}_N^*$ is addition modulo N in the first component and multiplication modulo N in the second component.) Moreover, we claim that any element y with $y \leftrightarrow (0, b)$ is an Nth residue. To see this, recall that $\gcd(N, \phi(N)) = 1$ and so $d \stackrel{\text{def}}{=} [N^{-1} \bmod \phi(N)]$ exists. So

$$(a,\ [b^d \bmod N])^N = (Na \bmod N, [b^{dN} \bmod N]) = (0, b) \leftrightarrow y$$

for any $a \in \mathbb{Z}_N$. We have thus shown that $\mathsf{Res}(N^2)$ corresponds to the set

$$\left\{(0, b) \mid b \in \mathbb{Z}_N^*\right\}.$$

The above also demonstrates that the number of Nth roots of any $y \in \mathsf{Res}(N^2)$ is exactly N, and so computing Nth powers is an N-to-1 function. As such, if $r \in \mathbb{Z}_{N^2}^*$ is uniform then $[r^N \bmod N^2]$ is a uniform element of $\mathsf{Res}(N^2)$.

The *decisional composite residuosity problem*, roughly speaking, is to distinguish a uniform element of $\mathbb{Z}_{N^2}^*$ from a uniform element of $\mathsf{Res}(N^2)$. Formally, let $\mathsf{GenModulus}$ be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$, and p and q are n-bit primes (except with probability negligible in n). Then:

DEFINITION 13.10 *The* decisional composite residuosity problem is hard relative to GenModulus *if for all probabilistic polynomial-time algorithms D there is a negligible function* negl *such that*

$$\left|\Pr[D(N, [r^N \bmod N^2]) = 1] - \Pr[D(N, r) = 1]\right| \leq \mathsf{negl}(n),$$

where in each case the probabilities are taken over the experiment in which $\mathsf{GenModulus}(1^n)$ *outputs (N, p, q), and then a uniform $r \in \mathbb{Z}_{N^2}^*$ is chosen. (Recall that $[r^N \bmod N^2]$ is a uniform element of* $\mathsf{Res}(N^2)$.)

The *decisional composite residuosity* (*DCR*) *assumption* is the assumption that there is a $\mathsf{GenModulus}$ relative to which the decisional composite residuosity problem is hard.

As we have discussed, elements of $\mathbb{Z}_{N^2}^*$ have the form (r', r) with r' and r arbitrary (in the appropriate groups), whereas Nth residues have the form $(0, r)$ with $r \in \mathbb{Z}_N^*$ arbitrary. The DCR assumption is that it is hard to

distinguish uniform elements of the first type from uniform elements of the second type. This suggests the following abstract way to encrypt a message $m \in \mathbb{Z}_N$ with respect to a public key N: choose a uniform Nth residue $(0, r)$ and set the ciphertext equal to

$$c \leftrightarrow (m, 1) \cdot (0, r) = (m + 0,\ 1 \cdot r) = (m, r).$$

Without worrying for now how this can be carried out efficiently by the sender, or how the receiver can decrypt, let us simply convince ourselves (on an intuitive level) that this is secure. Since a uniform Nth residue $(0, r)$ cannot be distinguished from a uniform element (r', r), the ciphertext as constructed above is indistinguishable (from the point of an eavesdropper who does not know the factorization of N) from the ciphertext

$$c' \leftrightarrow (m, 1) \cdot (r', r) = ([m + r' \bmod N], r)$$

for uniform $r' \in \mathbb{Z}_N$ and $r \in \mathbb{Z}_N^*$. Lemma 11.15 shows that $[m + r' \bmod N]$ is uniformly distributed in $\mathbb{Z}_N$ and so, in particular, this ciphertext c' is independent of the message m. CPA-security follows. A formal proof that proceeds exactly along these lines is given further below.

Before turning to the formal description and proof of security, we show how encryption and decryption can be performed efficiently.

Encryption. We have described encryption above as though it is taking place in $\mathbb{Z}_N \times \mathbb{Z}_N^*$. In fact it takes place in the isomorphic group $\mathbb{Z}_{N^2}^*$. That is, the sender generates a ciphertext $c \in \mathbb{Z}_{N^2}^*$ by choosing a uniform[1] $r \in \mathbb{Z}_N^*$ and then computing

$$c := [(1 + N)^m \cdot r^N \bmod N^2].$$

Observe that

$$c = \big((1 + N)^m \cdot 1^N\big) \cdot \big((1 + N)^0 \cdot r^N\big) \bmod N^2 \leftrightarrow (m, 1) \cdot (0, r),$$

and so $c \leftrightarrow (m, r)$ as desired.

Decryption. We now describe how decryption can be performed efficiently given the factorization of N. For c constructed as above, we claim that m is recovered by the following steps:

- Set $\hat{c} := [c^{\phi(N)} \bmod N^2]$.
- Set $\hat{m} := (\hat{c} - 1)/N$. (Note that this is carried out over the integers.)
- Set $m := \big[\hat{m} \cdot \phi(N)^{-1} \bmod N\big]$.

[1] We remark that it does not make any difference whether the sender chooses uniform $r \in \mathbb{Z}_N^*$ or uniform $r \in \mathbb{Z}_{N^2}^*$, since in either case the distribution of $[r^N \bmod N^2]$ is the same (as can be verified by looking at what happens in the isomorphic group $\mathbb{Z}_N \times \mathbb{Z}_N^*$).

To see why this works, let $c \leftrightarrow (m, r)$ for an arbitrary $r \in \mathbb{Z}_N^*$. Then

$$\begin{aligned}\hat{c} &\stackrel{\text{def}}{=} [c^{\phi(N)} \bmod N^2] \\ &\leftrightarrow (m,\, r)^{\phi(N)} \\ &= \Big([m \cdot \phi(N) \bmod N],\ [r^{\phi(N)} \bmod N]\Big) \\ &= \big([m \cdot \phi(N) \bmod N],\ 1\big).\end{aligned}$$

By Proposition 13.6(3), this means that $\hat{c} = (1+N)^{[m\cdot\phi(N) \bmod N]} \bmod N^2$. Using Proposition 13.6(2), we know that

$$\hat{c} = (1+N)^{[m\cdot\phi(N) \bmod N]} = (1 + [m \cdot \phi(N) \bmod N] \cdot N) \bmod N^2.$$

Since $1 + [m\cdot\phi(N) \bmod N]\cdot N$ is always less that N^2 we can drop the $\bmod N^2$ at the end and view the above as an equality over the integers. Thus, $\hat{m} \stackrel{\text{def}}{=} (\hat{c} - 1)/N = [m \cdot \phi(N) \bmod N]$ and, finally,

$$m = [\hat{m} \cdot \phi(N)^{-1} \bmod N],$$

as required. (Note that $\phi(N)$ is invertible modulo N since $\gcd(N, \phi(N)) = 1$.)

We give a complete description of the Paillier encryption scheme, followed by an example of the above calculations.

CONSTRUCTION 13.11

Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes (except with probability negligible in n). Define the following encryption scheme:

- Gen: on input 1^n run GenModulus(1^n) to obtain (N, p, q). The public key is N, and the private key is $\langle N, \phi(N)\rangle$.
- Enc: on input a public key N and a message $m \in \mathbb{Z}_N$, choose a uniform $r \leftarrow \mathbb{Z}_N^*$ and output the ciphertext

$$c := [(1+N)^m \cdot r^N \bmod N^2].$$

- Dec: on input a private key $\langle N, \phi(N)\rangle$ and a ciphertext c, compute

$$m := \left[\frac{[c^{\phi(N)} \bmod N^2] - 1}{N} \cdot \phi(N)^{-1} \bmod N\right].$$

The Paillier encryption scheme.

Example 13.12

Let $N = 11 \cdot 17 = 187$ (and so $N^2 = 34969$), and consider encrypting the message $m = 175$ and then decrypting the corresponding ciphertext. Choosing

$r = 83 \in \mathbb{Z}^*_{187}$, we compute the ciphertext

$$c := [(1+187)^{175} \cdot 83^{187} \bmod 34969] = 23911$$

corresponding to $(175, 83)$. To decrypt, note that $\phi(N) = 160$. So we first compute $\hat{c} := [23911^{160} \bmod 34969] = 25620$. Subtracting 1 and dividing by 187 gives $\hat{m} := (25620 - 1)/187 = 137$; since $90 = [160^{-1} \bmod 187]$, the message is recovered as $m := [137 \cdot 90 \bmod 187] = 175$. ♢

THEOREM 13.13 *If the decisional composite residuosity problem is hard relative to* GenModulus*, then the Paillier encryption scheme is CPA-secure.*

PROOF Let Π denote the Paillier encryption scheme. We prove that Π has indistinguishable encryptions in the presence of an eavesdropper; by Theorem 11.6 this implies that it is CPA-secure.

Let $\mathcal{A}$ be an arbitrary probabilistic polynomial-time adversary. Consider the following PPT algorithm D that attempts to solve the decisional composite residuosity problem relative to GenModulus:

> **Algorithm** D:
> The algorithm is given N, y as input.
>
> - Set $pk = N$ and run $\mathcal{A}(pk)$ to obtain two messages m_0, m_1.
> - Choose a uniform bit b and set $c := [(1+N)^{m_b} \cdot y \bmod N^2]$.
> - Give the ciphertext c to $\mathcal{A}$ and obtain an output bit b'. If $b' = b$, output 1; otherwise, output 0.

Let us analyze the behavior of D. There are two cases to consider:

Case 1: Say the input to D was generated by running $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q), choosing uniform $r \in \mathbb{Z}^*_{N^2}$, and setting $y := [r^N \bmod N^2]$. (That is, y is a uniform element of $\mathsf{Res}(N^2)$.) In this case,

$$c = [(1+N)^{m_b} \cdot r^N \bmod N^2]$$

for uniform $r \in \mathbb{Z}^*_{N^2}$. Recalling that the distribution on $[r^N \bmod N^2]$ is the same whether r is chosen uniformly from $\mathbb{Z}^*_N$ or from $\mathbb{Z}^*_{N^2}$, we see that in this case the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to $\mathcal{A}$'s view in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$. Since D outputs 1 exactly when the output b' of $\mathcal{A}$ is equal to b, we have

$$\Pr\left[D(N, [r^N \bmod N^2]) = 1\right] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1],$$

where the first probability is taken over the experiment as in Definition 13.10.

Case 2: Say the input to D was generated by running $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q) and choosing uniform $y \in \mathbb{Z}^*_{N^2}$. We claim that the view of $\mathcal{A}$

in this case is *independent* of the bit b. This follows because y is a uniform element of the group $\mathbb{Z}_{N^2}^*$, and so the ciphertext c is uniformly distributed in $\mathbb{Z}_{N^2}^*$ (see Lemma 11.15), independent of m. Thus, the probability that $b' = b$ in this case is exactly $\frac{1}{2}$. That is,

$$\Pr[D(N, r) = 1] = \frac{1}{2},$$

where the probability is taken over the experiment as in Definition 13.10.

Combining the above, we see that

$$\left| \Pr\left[D(N, [r^N \bmod N^2]) = 1\right] - \Pr[D(N, r) = 1] \right| = \left| \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - \tfrac{1}{2} \right|.$$

By the assumption that the decisional composite residuosity problem is hard relative to GenModulus, there is a negligible function negl such that

$$\left| \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - \tfrac{1}{2} \right| \leq \mathsf{negl}(n).$$

Thus $\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$, completing the proof. ■

13.2.3 Homomorphic Encryption

The Paillier encryption scheme is useful in a number of settings because it is *homomorphic*. Roughly, a homomorphic encryption scheme enables (certain) computations to be performed on encrypted data, yielding a ciphertext containing the encrypted result. In the case of Paillier encryption, the computation that can be performed is (modular) *addition*. Specifically, fix a public key $pk = N$. Then the Paillier scheme has the property that multiplying an encryption of m_1 and an encryption of m_2 (with multiplication done modulo N^2) results in an encryption of $[m_1 + m_2 \bmod N]$; this is because

$$\begin{aligned} &\left((1+N)^{m_1} \cdot r_1^N\right) \cdot \left((1+N)^{m_2} \cdot r_2^N\right) \\ &= (1+N)^{[m_1+m_2 \bmod N]} \cdot (r_1 r_2)^N \bmod N^2. \end{aligned}$$

Although the ability to add encrypted values may not seem very useful, it suffices for several interesting applications including voting, discussed below.

We present a general definition, of which Pailler encryption is a special case.

DEFINITION 13.14 *A public-key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* homomorphic *if for all n and all (pk, sk) output by $\mathsf{Gen}(1^n)$, it is possible to define groups $\mathbb{M}, \mathbb{C}$ (depending on pk only) such that:*

- *The message space is $\mathbb{M}$, and all ciphertexts output by Enc_{pk} are elements of $\mathbb{C}$. For notational convenience, we write $\mathbb{M}$ as an additive group and $\mathbb{C}$ as a multiplicative group.*

- *For any $m_1, m_2 \in \mathbb{M}$, any c_1 output by $\mathsf{Enc}_{pk}(m_1)$, and any c_2 output by $\mathsf{Enc}_{pk}(m_2)$, it holds that*

$$\mathsf{Dec}_{sk}(c_1 \cdot c_2) = m_1 + m_2.$$

Moreover, the distribution on ciphertexts obtained by encrypting m_1, encrypting m_2, and then multiplying the results is identical *to the distribution on ciphertexts obtained by encrypting $m_1 + m_2$.*

The last part of the definition ensures that if ciphertexts $c_1 \leftarrow \mathsf{Enc}_{pk}(m_1)$ and $c_2 \leftarrow \mathsf{Enc}_{pk}(m_2)$ are generated and the result $c_3 := c_1 \cdot c_2$ is computed, then the resulting ciphertext c_3 contains no more information about m_1 or m_2 than the sum m_3.

The Paillier encryption scheme with $pk = N$ is homomorphic with $\mathbb{M} = \mathbb{Z}_N$ and $\mathbb{C} = \mathbb{Z}^*_{N^2}$. This is not the first example of a homomorphic encryption scheme we have seen; El Gamal encryption is also homomorphic. Specifically, for public key $pk = \langle \mathbb{G}, q, g, h \rangle$ we can take $\mathbb{M} = \mathbb{G}$ and $\mathbb{C} = \mathbb{G} \times \mathbb{G}$; then

$$\langle g^{y_1},\ h^{y_1} \cdot m_1 \rangle \cdot \langle g^{y_2},\ h^{y_2} \cdot m_2 \rangle = \langle g^{y_1+y_2},\ h^{y_1+y_2} \cdot m_1 m_2 \rangle,$$

where multiplication of ciphertexts is component-wise. The Goldwasser–Micali encryption scheme we will see later is also homomorphic (see Exercise 13.10).

A nice feature of Paillier encryption is that it is homomorphic over a large additive group (namely, $\mathbb{Z}_N$). To see an application of this, consider the following distributed voting scheme, where ℓ voters can vote "no" or "yes" and the goal is to tabulate the number of "yes" votes:

1. A voting authority generates a public key N for the Paillier encryption scheme and publicizes N.

2. Let 0 stand for a "no," and let 1 stand for a "yes." Each voter casts their vote by encrypting it. That is, voter i casts her vote v_i by computing $c_i := [(1+N)^{v_i} \cdot (r_i)^N \bmod N^2]$ for a uniform $r_i \in \mathbb{Z}^*_N$.

3. Each voter broadcasts their vote c_i. These votes are then publicly *aggregated* by computing

$$c_{\text{total}} := \left[\prod_{i=1}^{\ell} c_i \bmod N^2 \right].$$

4. The authority is given c_{total}. (We assume the authority has not been able to observe what goes on until now.) By decrypting it, the authority obtains the vote total

$$v_{\text{total}} \stackrel{\text{def}}{=} \sum_{i=1}^{\ell} v_i \bmod N.$$

If ℓ is small (so that $v_{\text{total}} \ll N$), there is no wrap-around modulo N and $v_{\text{total}} = \sum_{i=1}^{\ell} v_i$.

Key features of the above are that *no voter learns anyone else's vote*, and calculation of the total is *publicly verifiable* if the authority is trusted to correctly compute v_{total} from c_{total}. Also, the authority obtains the correct total *without learning any individual votes*. (Here, we assume the authority cannot see voters' ciphertexts. In Section 13.3.3 we show a protocol in which votes are kept hidden from authorities even if they see all the communication.) We assume all voters act honestly (and only try to learn others' votes based on information they observe); an entire research area of cryptography is dedicated to addressing potential threats from participants who might be malicious and not follow the protocol.

13.3 Secret Sharing and Threshold Encryption

Motivated by the discussion of distributed voting in the previous section, we briefly consider *secure (interactive) protocols*. Such protocols can be significantly more complicated than the basic cryptographic primitives (e.g., encryption and signature schemes) we have focused on until now, both because they can involve multiple parties exchanging several rounds of messages, as well as because they are intended to realize more-complex security requirements.

The goal of this section is mainly to give the reader a taste of this fascinating area, and no attempt is made at being comprehensive or complete. Although the protocols presented here can be proven secure (with respect to appropriate definitions), we omit formal definitions, details, and proofs and instead rely on informal discussion.

13.3.1 Secret Sharing

Consider the following problem. A *dealer* holds a *secret* s—say, a nuclear-launch code—that it wishes to share among some set of N users $P_1, \ldots, P_N$ by giving each user a *share*. Any t users should be able to pool their shares and reconstruct the secret, but no coalition of fewer than t users should get *any* information about s from their collective shares (beyond whatever information they had about s already). We refer to such a sharing mechanism as a *(t, N)-threshold secret-sharing scheme*. Such a scheme ensures that s is not revealed without sufficient authorization, while also guaranteeing availability of s when needed (since any t users can reconstruct it). Beyond their direct application, secret-sharing schemes are also a building block of many cryptographic protocols.

Consider a simple solution for the case $t = N$, assuming $s \in \{0,1\}^\ell$. The dealer chooses uniform $s_1, \ldots, s_{N-1} \in \{0,1\}^\ell$ and sets $s_N := s \oplus \left(\bigoplus_{i=1}^{N-1} s_i\right)$; the share of user P_i is s_i. Since $\bigoplus_{i=1}^{N} s_i = s$ by construction, clearly all the users together can recover s. However, the shares of any coalition of

$N-1$ users are (jointly) uniform and independent of s, and thus reveal no information about s. This is clear when the coalition is $P_1, \ldots, P_{N-1}$. In the general case, when the coalition includes everyone except for P_j ($j \neq N$), this is true because $s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{N-1}$ are uniform and independent of s by construction, and

$$s_N = s \oplus \left(\bigoplus_{i<N, i \neq j} s_i \right) \oplus s_j;$$

thus, even conditioned on some fixed values for s and the shares of the other members of the coalition, the share s_N of user P_N is uniform because s_j is uniform and independent of s.

We can extend this to obtain a solution for $t < N$. The basic idea is to replicate the above scheme for each subset $T \subset N$ of size t. That is, for each such subset $T = \{P_{i_1}, \ldots, P_{i_t}\}$, we choose uniform shares $s_{T,i_1}, \ldots, s_{T,i_t}$ subject to the constraint that $\oplus_{j=1}^{t} s_{T,i_j} = s$, and give s_{T,i_j} to user P_{i_j}. It is not hard to see that this satisfies the requirements.

Unfortunately, this extension of the original scheme is not *efficient*. Each user now stores a share $s_{T,i}$ for *every* subset T of which she is a member. For each user there are $\binom{N-1}{t-1}$ such subsets, which is exponential in N if $t \approx N/2$.

Shamir's scheme. Fortunately, it is possible to do significantly better using a secret-sharing scheme introduced by Adi Shamir (of RSA fame). This scheme is based on polynomials[2] over a finite field $\mathbb{F}$, where $\mathbb{F}$ is chosen so that $s \in \mathbb{F}$ and $|\mathbb{F}| > N$. (See Appendix A.5 for a brief discussion of finite fields.) Before describing the scheme, we briefly review some background related to polynomials over a field $\mathbb{F}$.

A value $x \in \mathbb{F}$ is a *root* of a polynomial p if $p(x) = 0$. We use the well-known fact that any nonzero, degree-t polynomial over a field has at most t roots. This implies:

COROLLARY 13.15 *Any two distinct degree-t polynomials p and q agree on at most t points.*

PROOF If not, then the nonzero, degree-t polynomial $p - q$ would have more than t roots. ■

Shamir's scheme relies on the fact that for any t pairs of elements (x_1, y_1), $\ldots$, (x_t, y_t) from $\mathbb{F}$ (with the $\{x_i\}$ distinct), there is a *unique* polynomial p of degree $(t-1)$ such that $p(x_i) = y_i$ for $1 \leq i \leq t$. We can prove this quite easily. The fact that there exists such a p uses standard polynomial interpolation.

[2] A degree-t polynomial p over $\mathbb{F}$ is given by $p(X) = \sum_{i=0}^{t} a_i X^i$, where $a_i \in \mathbb{F}$ and X is a formal variable. (Note that we allow $a_t = 0$ and so we really mean a polynomial of degree *at most* t.) Any such polynomial naturally defines a function mapping $\mathbb{F}$ to itself, given by evaluating the polynomial on its input.

In detail: for $i = 1, \ldots, t$, define the degree-$(t-1)$ polynomial

$$\delta_i(X) \stackrel{\text{def}}{=} \frac{\prod_{j=1, j\neq i}^{t}(X - x_j)}{\prod_{j=1, j\neq i}^{t}(x_i - x_j)}.$$

Note that $\delta_i(x_j) = 0$ for any $j \neq i$, and $\delta_i(x_i) = 1$. So $p(X) \stackrel{\text{def}}{=} \sum_{i=1}^{t} \delta_i(X) \cdot y_i$ is a polynomial of degree $(t-1)$ with $p(x_i) = y_i$ for $1 \leq i \leq t$. (We remark that this, in fact, demonstrates that the desired polynomial p can be found *efficiently*.) Uniqueness follows from Corollary 13.15.

We now describe Shamir's (t, N)-threshold secret-sharing scheme. Let $\mathbb{F}$ be a finite field that contains the domain of possible secrets, and with $|\mathbb{F}| > N$. Let $x_1, \ldots, x_N \in \mathbb{F}$ be distinct, nonzero elements that are fixed and publicly known. (Such elements exist since $|\mathbb{F}| > N$.) The scheme works as follows:

Sharing: Given a secret $s \in \mathbb{F}$, the dealer chooses uniform $a_1, \ldots, a_{t-1} \in \mathbb{F}$ and defines the polynomial $p(X) \stackrel{\text{def}}{=} s + \sum_{i=1}^{t-1} a_i X^i$. This is a uniform degree-$(t-1)$ polynomial with constant term s. The share of user P_i is $s_i := p(x_i) \in \mathbb{F}$.

Reconstruction: Say t users $P_{i_1}, \ldots, P_{i_t}$ pool their shares $s_{i_1}, \ldots, s_{i_t}$. Using polynomial interpolation, they compute the unique degree-$(t-1)$ polynomial p' for which $p'(x_{i_j}) = s_{i_j}$ for $1 \leq j \leq t$. The secret is $p'(0)$.

It is clear that reconstruction works since $p' = p$ and $p(0) = s$.

It remains to show that any $t-1$ users learn nothing about the secret s from their shares. By symmetry, it suffices to consider the shares of users $P_1, \ldots, P_{t-1}$. We claim that for *any* secret s, the shares $s_1, \ldots, s_{t-1}$ are (jointly) uniform. Since the dealer chooses $a_1, \ldots, a_{t-1}$ uniformly, this follows if we show that there is a one-to-one correspondence between the polynomial p chosen by the dealer and the shares $s_1, \ldots, s_{t-1}$. But this is a direct consequence of Corollary 13.15.

13.3.2 Verifiable Secret Sharing

So far we have considered *passive* attacks in which $t-1$ users may try to use their shares to learn information about the secret. But we may also be concerned about *active*, malicious behavior. Here there are two separate concerns: First, a corrupted *dealer* may give inconsistent shares to the users, i.e., such that different secrets are recovered depending on which t users pool their shares. Second, in the reconstruction phase a malicious user may present a *different* share from the one given to them by the dealer, and thus affect the recovered secret. (While this could be addressed by having the dealer sign the shares, this does not work when the dealer itself may be dishonest.) *Verifiable secret-sharing (VSS) schemes* prevent both these attacks.

More formally, we allow any $t-1$ users to be corrupted and to collude with each other and, possibly, the dealer. We require (1a) at the end of the

sharing phase, a secret s is defined such that any collection that includes t uncorrupted users (whether or not this collection also includes some corrupted users) will successfully recover s in the reconstruction phase; moreover, (1b) if the dealer is honest, then s corresponds to the dealer's secret. In addition, (2) when the dealer is honest then, as before, the $t-1$ corrupted users learn nothing about the secret from their shares and any public information the dealer publishes. Since we want there to be t uncorrupted users even if $t-1$ users are corrupted, we require $N \geq t + (t-1) > 2(t-1)$; in other words, we assume a majority of the users remain uncorrupted.

We describe a VSS scheme due to Feldman that relies on an algorithm $\mathcal{G}$ relative to which the discrete-logarithm problem is hard. For simplicity, we describe it in the random-oracle model and let H denote a function to be modeled as a random oracle. We also assume that some trusted parameters $(\mathbb{G}, q, g)$, generated using $\mathcal{G}(1^n)$, are published in advance, where q is prime and so $\mathbb{Z}_q$ is a field. Finally, we assume that all users have access to a broadcast channel, such that a message broadcast by any user is heard by everyone.

The sharing phase now involves the N users running an interactive protocol with the dealer that proceeds as follows:

1. To share a secret s, the dealer chooses uniform $a_0 \in \mathbb{Z}_q$ and then shares a_0 as in Shamir's scheme. That is, the dealer chooses uniform $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$ and defines the polynomial $p(X) \stackrel{\text{def}}{=} \sum_{i=0}^{t-1} a_i X^i$. The dealer sends the share $s_i := p(i) = \sum_{j=0}^{t-1} a_i \cdot i^j$ to user P_i.[3]

   In addition, the dealer publicly broadcasts the values $A_0 := g^{a_0}$, $\ldots$, $A_{t-1} := g^{a_{t-1}}$, and the "masked secret" $c := H(a_0) \oplus s$.

2. Each user P_i verifies that its share s_i satisfies

$$g^{s_i} \stackrel{?}{=} \prod_{j=0}^{t-1} (A_j)^{i^j}. \tag{13.3}$$

   If not, P_i publicly broadcasts a complaint.

   Note that if the dealer is honest, we have

$$\prod_{j=0}^{t-1} (A_j)^{i^j} = \prod_{j=0}^{t-1} \left(g^{a_j}\right)^{i^j} = g^{\sum_{j=0}^{t-1} a_j \cdot i^j} = g^{p(i)} = g^{s_i},$$

   and so no honest user will complain. Since there are at most $t-1$ corrupted users, there are at most $t-1$ complaints if the dealer is honest.

3. If more than $t-1$ users complain, the dealer is disqualified and the protocol is aborted. Otherwise, the dealer responds to a complaint from P_i by broadcasting s_i. If this share does not satisfy Equation (13.3) (or if the dealer refuses to respond to a complaint at all), the dealer is disqualified and the protocol is aborted. Otherwise, P_i uses the broadcast value (rather than the value it received in the first round) as its share.

[3] Note that we are now setting $x_i = i$, which is fine since we are using the field $\mathbb{Z}_q$.

In the reconstruction phase, say a group of users (that includes at least t uncorrupted users) pool their shares. A share s_i provided by a user P_i is discarded if it does not satisfy Equation (13.3). Among the remaining shares, any t of them are used to recover a_0 exactly as in Shamir's scheme. The original secret is then computed as $s := c \oplus H(a_0)$.

We now argue that this protocol meets the desired security requirements. We first show that, assuming the dealer is not disqualified, the value recovered in the reconstruction phase is uniquely determined by the public information; specifically, the recovered value is $c \oplus H(\log_g A_0)$. (Combined with the fact that an honest dealer is never disqualified, this proves that conditions (1a) and (1b) hold.) Define $a_i := \log_g A_i$ for $0 \le i \le t-1$; the $\{a_i\}$ cannot be computed efficiently if the discrete-logarithm problem is hard, but they are still well-defined. Define the polynomial $p(X) \stackrel{\text{def}}{=} \sum_{i=0}^{t-1} a_i X^i$. Any share s_i, contributed by party P_i, that is not discarded during the reconstruction phase must satisfy Equation (13.3), and hence satisfies $s_i = p(i)$. It follows that, regardless of which shares are used, the parties will reconstruct polynomial p, compute $a_0 = p(0)$, and then recover $s = c \oplus H(a_0)$.

It is also possible to show that condition (2) holds for computationally bounded adversaries if the discrete-logarithm problem is hard for $\mathcal{G}$. (In contrast to Shamir's secret-sharing scheme, secrecy here is no longer unconditional. Unconditionally secure VSS schemes are possible, but are beyond the scope of our treatment.) Intuitively, this is because the secret s is masked by the random value $H(a_0)$, and the information given to any $t-1$ users in the sharing phase—namely, their shares and the public values $\{A_i\}$—reveals only g^{a_0}, from which it is hard to compute a_0. This intuition can be made rigorous, but we do not do so here.

13.3.3 Threshold Encryption and Electronic Voting

In Section 13.2.3 we introduced the notion of homomorphic encryption schemes and gave the Paillier encryption scheme as an example. Here we show a different homomorphic encryption scheme that is a variant of El Gamal encryption. Specifically, given a public key $pk = \langle \mathbb{G}, q, g, h \rangle$ as in regular El Gamal encryption, we now encrypt a message $m \in \mathbb{Z}_q$ by setting $M := g^m$, choosing a uniform $y \in \mathbb{Z}_q$, and sending the ciphertext $c := \langle g^y,\ h^y \cdot M \rangle$. To decrypt, the receiver recovers M as in standard El Gamal decryption and then computes $m := \log_g M$. Although this is not efficient if m comes from a large domain, if m is from a small domain—as it will be in our application—then the receiver can compute $\log_g M$ efficiently using exhaustive search. The advantage of this variant scheme is that it is homomorphic with respect to *addition* in $\mathbb{Z}_q$. That is,

$$\langle g^{y_1},\ h^{y_1} \cdot g^{m_1} \rangle \cdot \langle g^{y_2},\ h^{y_2} \cdot g^{m_2} \rangle = \langle g^{y_1+y_2},\ h^{y_1+y_2} \cdot g^{m_1+m_2} \rangle.$$

Recall that the basic approach to electronic voting using homomorphic en-

cryption has each voter i encrypt her vote $v_i \in \{0,1\}$ to obtain a ciphertext c_i. Once everyone has voted, the ciphertexts are multiplied to obtain an encryption of the sum $v_{total} \stackrel{\text{def}}{=} \sum_i v_i \bmod q = \sum_i v_i$. (The value q is, in practice, large enough so that no wrap-around modulo q occurs.) Since $0 \leq v_{total} \leq \ell$, where ℓ is the total number of voters, an authority with the private key can efficiently decrypt the final ciphertext and recover v_{total}.

A drawback of this approach is that the authority is trusted, both to (correctly) decrypt the final ciphertext as well as not to decrypt any of the individual voters' ciphertexts. (In Section 13.2.3 we assumed the authority could not see the individual voters' ciphertexts.) We might instead prefer to *distribute* trust among a set of N authorities, such that any set of t authorities is able to jointly decrypt an agreed-upon ciphertext (this ensures availability even if some authorities are down or unwilling to help decrypt), but no collection of $t-1$ authorities is able to decrypt any ciphertext on their own (this ensures privacy as long as fewer than t authorities are corrupted).

At first glance, it may seem that secret sharing solves the problem. If we share the private key among the N authorities, then no set of $t-1$ authorities learns the private key and so they cannot decrypt. On the other hand, any t authorities can pool their shares, recover the private key, and then decrypt any desired ciphertext.

A little thought shows that this does not quite work. If the authorities reconstruct the private key in order to decrypt some ciphertext, then as part of this process *all the authorities learn the private key*! Thus, afterward, any authority could decrypt any ciphertext of its choice, on its own.

We need instead a modified approach in which the "secret" (namely, the private key) is never reconstructed in the clear, yet is implicitly reconstructed only enough to enable decryption of one, agreed-upon ciphertext. We can achieve this for the specific case of El Gamal encryption in the following way. Fix a public key $pk = \langle \mathbb{G}, q, g, h \rangle$, and let $x \in \mathbb{Z}_q$ be the private key, i.e., $g^x = h$. Each authority is given a share $x_i \in \mathbb{Z}_q$ exactly as in Shamir's secret-sharing scheme. That is, a uniform degree-$(t-1)$ polynomial p with $p(0) = x$ is chosen, and the ith authority is given $x_i := p(i)$. (We assume a trusted dealer who knows x and securely deletes it once it is shared. It is possible to eliminate the dealer entirely, but this is beyond our present scope.)

Now, say some t authorities $i_1, \ldots, i_t$ wish to jointly decrypt a ciphertext $\langle c_1, c_2 \rangle$. To do so, authority i_j first publishes the value $w_j := c_1^{x_{i_j}}$. Recall from the previous section that there exist publicly computable polynomials $\{\delta_j(X)\}$ (that depend on the identities of these t authorities) such that $p(X) \stackrel{\text{def}}{=} \sum_{j=1}^{t} \delta_j(X) \cdot x_{i_j}$. Setting $\delta_j \stackrel{\text{def}}{=} \delta_j(0)$, we see that there exist publicly computable values $\delta_1, \ldots, \delta_t \in \mathbb{Z}_q$ for which $x = p(0) = \sum_{j=1}^{t} \delta_j \cdot x_{i_j}$. Any authority can then compute

$$M' := \frac{c_2}{\prod_{j=1}^{t} w_j^{\delta_j}}.$$

(They can then each compute $\log_g M$, if desired.) To see that this correctly recovers the message, say $c_1 = g^y$ and $c_2 = h^y \cdot M$. Then

$$\prod_{j=1}^{t} w_j^{\delta_j} = \prod_{j=1}^{t} c_1^{x_{i_j}\delta_j} = c_1^{\sum_{j=1}^{t} x_{i_j}\delta_j} = c_1^{p(0)} = c_1^x,$$

and so

$$M' \stackrel{\text{def}}{=} \frac{c_2}{\prod_{j=1}^{t} w_j^{\delta_j}} = \frac{h^y \cdot M}{c_1^x} = \frac{(g^x)^y \cdot M}{(g^y)^x} = M.$$

Note that any set of $t-1$ corrupted authorities learns nothing about the private key x from their shares. Moreover, it is possible to show that they learn nothing from the decryption process beyond the recovered value M.

Malicious (active) adversaries. Our treatment above assumes that the authorities decrypting some ciphertext all behave correctly. (If they do not, it would be easy for any of them to cause an incorrect result by publishing an arbitrary value w_j.) We also assume that voters behave honestly, and encrypt a vote of either 0 or 1. (Note that a voter could unfairly sway the election by encrypting a large value or a negative value.) Potential malicious behavior of this sort can be prevented using techniques beyond the scope of this book.

13.4 The Goldwasser–Micali Encryption Scheme

Before we present the Goldwasser–Micali encryption scheme, we need to develop a better understanding of *quadratic residues*. We first explore the easier case of quadratic residues modulo a prime p, and then look at the slightly more complicated case of quadratic residues modulo a composite N.

Throughout this section, p and q denote odd primes, and $N = pq$ denotes a product of two distinct, odd primes.

13.4.1 Quadratic Residues Modulo a Prime

In a group $\mathbb{G}$, an element $y \in \mathbb{G}$ is a *quadratic residue* if there exists an $x \in \mathbb{G}$ with $x^2 = y$. In this case, we call x a *square root* of y. An element that is not a quadratic residue is called a *quadratic non-residue*. In an abelian group, the set of quadratic residues forms a subgroup.

In the specific case of $\mathbb{Z}_p^*$, we have that y is a quadratic residue if there exists an x with $x^2 = y \bmod p$. We begin with an easy observation.

PROPOSITION 13.16 *Let $p > 2$ be prime. Every quadratic residue in $\mathbb{Z}_p^*$ has exactly two square roots.*

PROOF Let $y \in \mathbb{Z}_p^*$ be a quadratic residue. Then there exists an $x \in \mathbb{Z}_p^*$ such that $x^2 = y \bmod p$. Clearly, $(-x)^2 = x^2 = y \bmod p$. Furthermore, $-x \neq x \bmod p$: if $-x = x \bmod p$ then $2x = 0 \bmod p$, which implies $p \,|\, 2x$. Since p is prime, this would mean that either $p \,|\, 2$ (which is impossible since $p > 2$) or $p \,|\, x$ (which is impossible since $0 < x < p$). So, $[x \bmod p\,]$ and $[-x \bmod p\,]$ are distinct elements of $\mathbb{Z}_p^*$, and y has at least two square roots.

Let $x' \in \mathbb{Z}_p^*$ be a square root of y. Then $x^2 = y = (x')^2 \bmod p$, implying that $x^2 - (x')^2 = 0 \bmod p$. Factoring the left-hand side we obtain

$$(x - x')(x + x') = 0 \bmod p\,,$$

so that (by Proposition 8.3) either $p \,|\, (x - x')$ or $p \,|\, (x + x')$. In the first case, $x' = x \bmod p$ and in the second case $x' = -x \bmod p$, showing that y indeed has only $[\pm x \bmod p\,]$ as square roots. ■

Let $\mathsf{sq}_p : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ be the function $\mathsf{sq}_p(x) \stackrel{\text{def}}{=} [x^2 \bmod p\,]$. The above shows that sq_p is a two-to-one function when $p > 2$ is prime. This immediately implies that *exactly half the elements of $\mathbb{Z}_p^*$ are quadratic residues*. We denote the set of quadratic residues modulo p by $\mathcal{QR}_p$, and the set of quadratic non-residues by $\mathcal{QNR}_p$. We have just seen that for $p > 2$ prime

$$|\mathcal{QR}_p| = |\mathcal{QNR}_p| = \frac{|\mathbb{Z}_p^*|}{2} = \frac{p-1}{2}\,.$$

Define $\mathcal{J}_p(x)$, the *Jacobi symbol of x modulo p*, as follows.[4] Let $p > 2$ be prime, and $x \in \mathbb{Z}_p^*$. Then

$$\mathcal{J}_p(x) \stackrel{\text{def}}{=} \begin{cases} +1 \text{ if } x \text{ is a quadratic residue modulo } p \\ -1 \text{ if } x \text{ is } \textit{not} \text{ a quadratic residue modulo } p. \end{cases}$$

The notation can be extended in the natural way for any x relatively prime to p by setting $\mathcal{J}_p(x) \stackrel{\text{def}}{=} \mathcal{J}_p([x \bmod p\,])$.

Can we characterize the quadratic residues in $\mathbb{Z}_p^*$? We begin with the fact that $\mathbb{Z}_p^*$ is a cyclic group of order $p-1$ (see Theorem 8.56). Let g be a generator of $\mathbb{Z}_p^*$. This means that

$$\mathbb{Z}_p^* = \{g^0, g^1, g^2, \ldots, g^{\frac{p-1}{2}-1}, g^{\frac{p-1}{2}}, g^{\frac{p-1}{2}+1}, \ldots, g^{p-2}\}$$

(recall that p is odd, so $p-1$ is even). Squaring each element in this list and reducing modulo $p-1$ in the exponent (cf. Corollary 8.15) yields a list of all the quadratic residues in $\mathbb{Z}_p^*$:

$$\mathcal{QR}_p = \{g^0, g^2, g^4, \ldots, g^{p-3}, g^0, g^2, \ldots, g^{p-3}\}.$$

[4] For p prime, $\mathcal{J}_p(x)$ is also sometimes called the *Legendre symbol* of x and denoted by $\mathcal{L}_p(x)$; we have chosen our notation to be consistent with notation introduced later.

Each quadratic residue appears twice in this list. Therefore, the quadratic residues in $\mathbb{Z}_p^*$ are exactly those elements that can be written as g^i with $i \in \{0, \ldots, p-2\}$ an *even* integer.

The above characterization leads to a simple way to compute the Jacobi symbol and thus tell whether an element $x \in \mathbb{Z}_p^*$ is a quadratic residue or not.

PROPOSITION 13.17 *Let $p > 2$ be a prime. Then $\mathcal{J}_p(x) = x^{\frac{p-1}{2}} \bmod p$.*

PROOF Let g be an arbitrary generator of $\mathbb{Z}_p^*$. If x is a quadratic residue modulo p, our earlier discussion shows that $x = g^i$ for some even integer i. Writing $i = 2j$ with j an integer we then have

$$x^{\frac{p-1}{2}} = \left(g^{2j}\right)^{\frac{p-1}{2}} = g^{(p-1)j} = \left(g^{p-1}\right)^j = 1^j = 1 \bmod p\,,$$

and so $x^{\frac{p-1}{2}} = +1 = \mathcal{J}_p(x) \bmod p$ as claimed.

On the other hand, if x is not a quadratic residue then $x = g^i$ for some odd integer i. Writing $i = 2j + 1$ with j an integer we have

$$x^{\frac{p-1}{2}} = \left(g^{2j+1}\right)^{\frac{p-1}{2}} = \left(g^{2j}\right)^{\frac{p-1}{2}} \cdot g^{\frac{p-1}{2}} = 1 \cdot g^{\frac{p-1}{2}} = g^{\frac{p-1}{2}} \bmod p.$$

Now,

$$\left(g^{\frac{p-1}{2}}\right)^2 = g^{p-1} = 1 \bmod p,$$

and so $g^{\frac{p-1}{2}} = \pm 1 \bmod p$ since $[\pm 1 \bmod p]$ are the two square roots of 1 (cf. Proposition 13.16). Since g is a generator, it has order $p-1$ and so $g^{\frac{p-1}{2}} \neq 1 \bmod p$. It follows that $x^{\frac{p-1}{2}} = -1 = \mathcal{J}_p(x) \bmod p$. ■

Proposition 13.17 directly gives a polynomial-time algorithm (cf. Algorithm 13.18) for testing whether an element $x \in \mathbb{Z}_p^*$ is a quadratic residue.

ALGORITHM 13.18
Deciding quadratic residuosity modulo a prime

Input: A prime p; an element $x \in \mathbb{Z}_p^*$
Output: $\mathcal{J}_p(x)$ (or, equivalently, whether x is a quadratic residue or quadratic non-residue)

$b := \left[x^{\frac{p-1}{2}} \bmod p\right]$
if $b = 1$ **return** "quadratic residue"
else return "quadratic non-residue"

We conclude this section by noting a nice multiplicative property of quadratic residues and non-residues modulo p.

PROPOSITION 13.19 *Let $p > 2$ be a prime, and $x, y \in \mathbb{Z}_p^*$. Then*

$$\mathcal{J}_p(xy) = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y).$$

PROOF Using the previous proposition,

$$\mathcal{J}_p(xy) = (xy)^{\frac{p-1}{2}} = x^{\frac{p-1}{2}} \cdot y^{\frac{p-1}{2}} = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) \bmod p.$$

Since $\mathcal{J}_p(xy), \mathcal{J}_p(x), \mathcal{J}_p(y) = \pm 1$, equality holds over the integers as well. ■

COROLLARY 13.20 *Let $p > 2$ be prime, and say $x, x' \in \mathcal{QR}_p$ and $y, y' \in \mathcal{QNR}_p$. Then:*

1. $[xx' \bmod p\,] \in \mathcal{QR}_p$.
2. $[yy' \bmod p\,] \in \mathcal{QR}_p$.
3. $[xy \bmod p\,] \in \mathcal{QNR}_p$.

13.4.2 Quadratic Residues Modulo a Composite

We now turn our attention to quadratic residues in the group $\mathbb{Z}_N^*$, where $N = pq$. Characterizing the quadratic residues modulo N is easy if we use the results of the previous section in conjunction with the Chinese remainder theorem. Recall that the Chinese remainder theorem says that $\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$, and we let $y \leftrightarrow (y_p, y_q)$ denote the correspondence guaranteed by the theorem (i.e., $y_p = [y \bmod p\,]$ and $y_q = [y \bmod q\,]$). The key observation is:

PROPOSITION 13.21 *Let $N = pq$ with p, q distinct primes, and $y \in \mathbb{Z}_N^*$ with $y \leftrightarrow (y_p, y_q)$. Then y is a quadratic residue modulo N if and only if y_p is a quadratic residue modulo p and y_q is a quadratic residue modulo q.*

PROOF If y is a quadratic residue modulo N then, by definition, there exists an $x \in \mathbb{Z}_N^*$ such that $x^2 = y \bmod N$. Let $x \leftrightarrow (x_p, x_q)$. Then

$$(y_p,\, y_q) \leftrightarrow y = x^2 \leftrightarrow (x_p,\, x_q)^2 = ([x_p^2 \bmod p\,],\, [x_q^2 \bmod q\,]),$$

where $(x_p, x_q)^2$ is simply the square of the element (x_p, x_q) in the group $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$. We have thus shown that

$$y_p = x_p^2 \bmod p \quad \text{and} \quad y_q = x_q^2 \bmod q \tag{13.4}$$

and y_p, y_q are quadratic residues (with respect to the appropriate moduli).

Conversely, if $y \leftrightarrow (y_p, y_q)$ and y_p, y_q are quadratic residues modulo p and q, respectively, then there exist $x_p \in \mathbb{Z}_p^*$ and $x_q \in \mathbb{Z}_q^*$ such that Equation (13.4) holds. Let $x \in \mathbb{Z}_N^*$ be such that $x \leftrightarrow (x_p, x_q)$. Reversing the above steps shows that x is a square root of y modulo N. ■

The above proposition characterizes the quadratic residues modulo N. A careful examination of the proof yields another important observation: each quadratic residue $y \in \mathbb{Z}_N^*$ has exactly *four* square roots. To see this, let $y \leftrightarrow (y_p, y_q)$ be a quadratic residue modulo N and let x_p, x_q be square roots of y_p and y_q modulo p and q, respectively. Then the four square roots of y are given by the elements in $\mathbb{Z}_N^*$ corresponding to

$$(x_p,\, x_q), \quad (-x_p,\, x_q), \quad (x_p,\, -x_q), \quad (-x_p,\, -x_q). \tag{13.5}$$

Each of these is a square root of y since

$$\begin{aligned}(\pm x_p,\, \pm x_q)^2 &= \Big([(\pm x_p)^2 \bmod p\,],\, [(\pm x_q)^2 \bmod q\,]\Big) \\ &= ([x_p^2 \bmod p\,],\, [x_q^2 \bmod q\,]) = (y_p,\, y_q) \leftrightarrow y\end{aligned}$$

(where again the notation $(\cdot,\cdot)^2$ refers to squaring in the group $\mathbb{Z}_p \times \mathbb{Z}_q$). The Chinese remainder theorem guarantees that the four elements in Equation (13.5) correspond to *distinct* elements of $\mathbb{Z}_N^*$, since x_p and $-x_p$ are unique modulo p (and similarly for x_q and $-x_q$ modulo q).

Example 13.22
Consider $\mathbb{Z}_{15}^*$ (the correspondence given by the Chinese remainder theorem is tabulated in Example 8.25). Element 4 is a quadratic residue modulo 15 with square root 2. Since $2 \leftrightarrow (2, 2)$, the other square roots of 4 are given by

- $\big(2, [-2 \bmod 3]\big) = (2, 1) \leftrightarrow 7$;
- $\big([-2 \bmod 5], 2\big) = (3, 2) \leftrightarrow 8$; and
- $\big([-2 \bmod 5], [-2 \bmod 3]\big) = (3, 1) \leftrightarrow 13$.

One can verify that $7^2 = 8^2 = 13^2 = 4 \bmod 15$. ♢

Let $\mathcal{QR}_N$ denote the set of quadratic residues modulo N. Since squaring modulo N is a four-to-one function, we see that exactly $1/4$ of the elements of $\mathbb{Z}_N^*$ are quadratic residues. Alternately, we could note that since $y \in \mathbb{Z}_N^*$ is a quadratic residue if and only if y_p, y_q are quadratic residues, there is a one-to-one correspondence between $\mathcal{QR}_N$ and $\mathcal{QR}_p \times \mathcal{QR}_q$. Thus, the fraction of quadratic residues modulo N is

$$\frac{|\mathcal{QR}_N|}{|\mathbb{Z}_N^*|} = \frac{|\mathcal{QR}_p| \cdot |\mathcal{QR}_q|}{|\mathbb{Z}_N^*|} = \frac{\frac{p-1}{2} \cdot \frac{q-1}{2}}{(p-1)(q-1)} = \frac{1}{4}\,,$$

in agreement with the above.

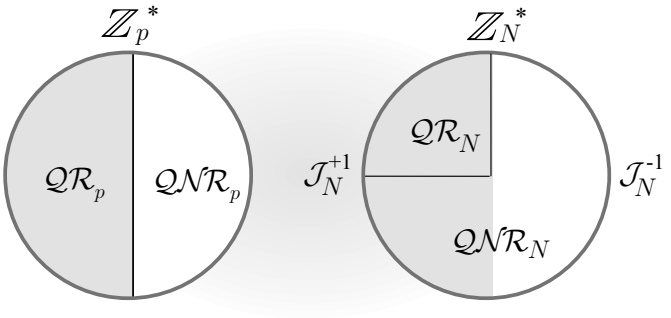

FIGURE 13.1: The structure of $\mathbb{Z}_p^*$ and $\mathbb{Z}_N^*$.

In the previous section, we defined the Jacobi symbol $\mathcal{J}_p(x)$ for $p > 2$ prime. We extend the definition to the case of N a product of distinct, odd primes p and q as follows. For any x relatively prime to $N = pq$,

$$\begin{aligned}\mathcal{J}_N(x) &\stackrel{\text{def}}{=} \mathcal{J}_p(x) \cdot \mathcal{J}_q(x) \\ &= \mathcal{J}_p([x \bmod p\,]) \cdot \mathcal{J}_q([x \bmod q\,]).\end{aligned}$$

We define $\mathcal{J}_N^{+1}$ as the set of elements in $\mathbb{Z}_N^*$ having Jacobi symbol $+1$, and define $\mathcal{J}_N^{-1}$ analogously.

We know from Proposition 13.21 that if x is a quadratic residue modulo N, then $[x \bmod p\,]$ and $[x \bmod q\,]$ are quadratic residues modulo p and q, respectively; that is, $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$. So $\mathcal{J}_N(x) = +1$ and we see that:

If x is a quadratic residue modulo N, then $\mathcal{J}_N(x) = +1$.

However, $\mathcal{J}_N(x) = +1$ can also occur when $\mathcal{J}_p(x) = \mathcal{J}_q(x) = -1$, that is, when *both* $[x \bmod p\,]$ and $[x \bmod q\,]$ are *not* quadratic residues modulo p and q (and so x is not a quadratic residue modulo N). This turns out to be useful for the Goldwasser–Micali encryption scheme, and we therefore introduce the notation $\mathcal{QNR}_N^{+1}$ for the set of elements of this type. That is,

$$\mathcal{QNR}_N^{+1} \stackrel{\text{def}}{=} \left\{ x \in \mathbb{Z}_N^* \;\middle|\; \begin{array}{c} x \text{ is not a quadratic residue modulo } N, \\ \text{but } \mathcal{J}_N(x) = +1 \end{array} \right\}.$$

It is now easy to prove the following (see Figure 13.1):

PROPOSITION 13.23 *Let $N = pq$ with p, q distinct, odd primes. Then:*

1. *Exactly half the elements of $\mathbb{Z}_N^*$ are in $\mathcal{J}_N^{+1}$.*
2. *$\mathcal{QR}_N$ is contained in $\mathcal{J}_N^{+1}$.*
3. *Exactly half the elements of $\mathcal{J}_N^{+1}$ are in $\mathcal{QR}_N$ (the other half are in $\mathcal{QNR}_N^{+1}$).*

PROOF We know that $\mathcal{J}_N(x) = +1$ if either $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$ or $\mathcal{J}_p(x) = \mathcal{J}_q(x) = -1$. We also know (from the previous section) that exactly half the elements of $\mathbb{Z}_p^*$ have Jacobi symbol $+1$, and half have Jacobi symbol -1 (and similarly for $\mathbb{Z}_q^*$). Defining $\mathcal{J}_p^{+1}$, $\mathcal{J}_p^{-1}$, $\mathcal{J}_q^{+1}$, and $\mathcal{J}_q^{-1}$ in the natural way, we thus have

$$\begin{aligned}\left|\mathcal{J}_N^{+1}\right| &= |\mathcal{J}_p^{+1} \times \mathcal{J}_q^{+1}| + |\mathcal{J}_p^{-1} \times \mathcal{J}_q^{-1}| \\ &= |\mathcal{J}_p^{+1}| \cdot |\mathcal{J}_q^{+1}| + |\mathcal{J}_p^{-1}| \cdot |\mathcal{J}_q^{-1}| \\ &= \frac{(p-1)}{2}\frac{(q-1)}{2} + \frac{(p-1)}{2}\frac{(q-1)}{2} = \frac{\phi(N)}{2}.\end{aligned}$$

So $\left|\mathcal{J}_N^{+1}\right| = |\mathbb{Z}_N^*|/2$, proving that half the elements of $\mathbb{Z}_N^*$ are in $\mathcal{J}_N^{+1}$.

We have noted earlier that all quadratic residues modulo N have Jacobi symbol $+1$, showing that $\mathcal{QR}_N \subseteq \mathcal{J}_N^{+1}$.

Since $x \in \mathcal{QR}_N$ if and only if $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$, we have

$$|\mathcal{QR}_N| = |\mathcal{J}_p^{+1} \times \mathcal{J}_q^{+1}| = \frac{(p-1)}{2}\frac{(q-1)}{2} = \frac{\phi(N)}{4},$$

and so $|\mathcal{QR}_N| = \left|\mathcal{J}_N^{+1}\right|/2$. Since $\mathcal{QR}_N$ is a subset of $\mathcal{J}_N^{+1}$, this proves that half the elements of $\mathcal{J}_N^{+1}$ are in $\mathcal{QR}_N$. ■

The next two results are analogues of Proposition 13.19 and Corollary 13.20.

PROPOSITION 13.24 *Let $N = pq$ be a product of distinct, odd primes, and $x, y \in \mathbb{Z}_N^*$. Then $\mathcal{J}_N(xy) = \mathcal{J}_N(x) \cdot \mathcal{J}_N(y)$.*

PROOF Using the definition of $\mathcal{J}_N(\cdot)$ and Proposition 13.19:

$$\begin{aligned}\mathcal{J}_N(xy) = \mathcal{J}_p(xy) \cdot \mathcal{J}_q(xy) &= \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) \cdot \mathcal{J}_q(x) \cdot \mathcal{J}_q(y) \\ &= \mathcal{J}_p(x) \cdot \mathcal{J}_q(x) \cdot \mathcal{J}_p(y) \cdot \mathcal{J}_q(y) = \mathcal{J}_N(x) \cdot \mathcal{J}_N(y).\end{aligned}$$

■

COROLLARY 13.25 *Let $N = pq$ be a product of distinct, odd primes, and say $x, x' \in \mathcal{QR}_N$ and $y, y' \in \mathcal{QNR}_N^{+1}$. Then:*

1. $[xx' \bmod N] \in \mathcal{QR}_N$.
2. $[yy' \bmod N] \in \mathcal{QR}_N$.
3. $[xy \bmod N] \in \mathcal{QNR}_N^{+1}$.

PROOF We prove the final claim; proofs of the others are similar. Since $x \in \mathcal{QR}_N$, we have $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$. Since $y \in \mathcal{QNR}_N^{+1}$, we have $\mathcal{J}_p(y) = \mathcal{J}_q(y) = -1$. Using Proposition 13.19,

$$\mathcal{J}_p(xy) = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) = -1 \quad \text{and} \quad \mathcal{J}_q(xy) = \mathcal{J}_q(x) \cdot \mathcal{J}_q(y) = -1,$$

and so $\mathcal{J}_N(xy) = +1$. But xy is not a quadratic residue modulo N, since $\mathcal{J}_p(xy) = -1$ and so $[xy \bmod p]$ is not a quadratic residue modulo p. We conclude that $xy \in \mathcal{QNR}_N^{+1}$. ■

In contrast to Corollary 13.20, it is *not* true that $y, y' \in \mathcal{QNR}_N$ implies $yy' \in \mathcal{QR}_N$. (Instead, as indicated in the corollary, this is only guaranteed if $y, y' \in \mathcal{QNR}_N^{+1}$.) For example, we could have $\mathcal{J}_p(y) = +1$, $\mathcal{J}_q(y) = -1$ and $\mathcal{J}_p(y') = -1$, $\mathcal{J}_q(y') = +1$, so $\mathcal{J}_p(yy') = \mathcal{J}_q(yy') = -1$ and yy' is not a quadratic residue even though $\mathcal{J}_N(yy') = +1$.

13.4.3 The Quadratic Residuosity Assumption

In Section 13.4.1, we showed an efficient algorithm for deciding whether an input x is a quadratic residue modulo a prime p. Can we adapt the algorithm to work modulo a composite number N? Proposition 13.21 gives an easy solution to this problem *provided the factorization of N is known.* See Algorithm 13.26.

ALGORITHM 13.26
Deciding quadratic residuosity modulo a composite of known factorization

Input: Composite $N = pq$; the factors p and q; element $x \in \mathbb{Z}_N^*$
Output: A decision as to whether $x \in \mathcal{QR}_N$

compute $\mathcal{J}_p(x)$ and $\mathcal{J}_q(x)$
if $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$ **return** "quadratic residue"
else return "quadratic non-residue"

(As always, we assume the factors of N are distinct odd primes.) A simple modification of the above algorithm allows for computing $\mathcal{J}_N(x)$ when the factorization of N is known.

When the factorization of N is *unknown*, however, there is no known polynomial-time algorithm for deciding whether a given x is a quadratic residue modulo N or not. Somewhat surprisingly, a polynomial-time algorithm *is* known for computing $\mathcal{J}_N(x)$ without the factorization of N. (Although the algorithm itself is not that complicated, its proof of correctness is beyond the scope of this book and we therefore do not present the algorithm at all. The interested reader can refer to the references listed at the

end of this chapter.) This leads to a partial test of quadratic residuosity: if, for a given input x, it holds that $\mathcal{J}_N(x) = -1$, then x cannot possibly be a quadratic residue. (See Proposition 13.23.) This test says nothing when $\mathcal{J}_N(x) = +1$, and there is *no* known polynomial-time algorithm for deciding quadratic residuosity in that case (that does better than random guessing).

We now formalize the assumption that this problem is hard. Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$, and p and q are n-bit primes except with probability negligible in n.

DEFINITION 13.27 *We say* deciding quadratic residuosity is hard relative to GenModulus *if for all probabilistic polynomial-time algorithms D there exists a negligible function* negl *such that*

$$\left| \Pr[D(N, \mathsf{qr}) = 1] - \Pr[D(N, \mathsf{qnr}) = 1] \right| \leq \mathsf{negl}(n),$$

where in each case the probabilities are taken over the experiment in which $\mathsf{GenModulus}(1^n)$ *is run to give (N, p, q),* qr *is chosen uniformly from $\mathcal{QR}_N$, and* qnr *is chosen uniformly from $\mathcal{QNR}_N^{+1}$.*

It is crucial in the above that qnr is chosen from $\mathcal{QNR}_N^{+1}$ rather than $\mathcal{QNR}_N$; if qnr were chosen from $\mathcal{QNR}_N$ then with probability $2/3$ it would be the case that $\mathcal{J}_N(x) = -1$ and so distinguishing qnr from a uniform quadratic residue would be easy. (Recall that $\mathcal{J}_N(x)$ can be computed efficiently even without the factorization of N.)

The quadratic residuosity assumption is simply the assumption that there exists a GenModulus relative to which deciding quadratic residuosity is hard. It is easy to see that if deciding quadratic residuosity is hard relative to GenModulus, then factoring must be hard relative to GenModulus as well.

13.4.4 The Goldwasser–Micali Encryption Scheme

The preceding section immediately suggests a public-key encryption scheme for single-bit messages based on the quadratic residuosity assumption:

- The public key is a modulus N, and the private key is its factorization.
- To encrypt a '0,' send a uniform quadratic residue; to encrypt a '1,' send a uniform quadratic non-residue with Jacobi symbol $+1$.
- The receiver can decrypt a ciphertext c with its private key by using the factorization of N to decide whether c is a quadratic residue or not.

CPA-security of this scheme follows almost trivially from the hardness of the quadratic residuosity problem as formalized in Definition 13.27.

One thing missing from the above description is a specification of how the sender, who does not know the factorization of N, can choose a uniform

element of $\mathcal{QR}_N$ (to encrypt a 0) or a uniform element of $\mathcal{QNR}_N^{+1}$ (to encrypt a 1). The first of these is easy, while the second requires some ingenuity.

Choosing a uniform quadratic residue. Choosing a uniform element $y \in \mathcal{QR}_N$ is easy: simply pick a uniform $x \in \mathbb{Z}_N^*$ (see Appendix B.2.5) and set $y := x^2 \bmod N$. Clearly $y \in \mathcal{QR}_N$. The fact that y is uniformly distributed in $\mathcal{QR}_N$ follows from the facts that squaring modulo N is a 4-to-1 function (see Section 13.4.2) and that x is chosen uniformly from $\mathbb{Z}_N^*$. In more detail, fix any $\hat{y} \in \mathcal{QR}_N$ and let us compute the probability that $y = \hat{y}$ after the above procedure. Denote the four square roots of $\hat{y}$ by $\pm\hat{x}, \pm\hat{x}'$. Then:

$$\begin{aligned}\Pr[y = \hat{y}] &= \Pr[x \text{ is a square root of } \hat{y}] \\ &= \Pr\left[x \in \{\pm\hat{x}, \pm\hat{x}'\}\right] \\ &= \frac{4}{|\mathbb{Z}_N^*|} = \frac{1}{|\mathcal{QR}_N|}.\end{aligned}$$

Since the above holds for every $\hat{y} \in \mathcal{QR}_N$, we see that y is distributed uniformly in $\mathcal{QR}_N$.

Choosing a uniform element of $\mathcal{QNR}_N^{+1}$. In general, it is not known how to choose a uniform element of $\mathcal{QNR}_N^{+1}$ if the factorization of N is unknown. What saves us in the present context is that *the receiver can help* by including certain information in the public key. Specifically, we modify the scheme so that the receiver additionally chooses a uniform $z \in \mathcal{QNR}_N^{+1}$ and includes z as part of its public key. (This is easy for the receiver to do since it knows the factorization of N; see Exercise 13.6.) The sender can choose a uniform element $y \in \mathcal{QNR}_N^{+1}$ by choosing a uniform $x \in \mathbb{Z}_N^*$ (as above) and setting $y := [z \cdot x^2 \bmod N]$. It follows from Corollary 13.25 that $y \in \mathcal{QNR}_N^{+1}$. We leave it as an exercise to show that y is uniformly distributed in $\mathcal{QNR}_N^{+1}$; we do not use this fact directly in the proof of security given below.

We give a complete description of the Goldwasser–Micali encryption scheme, implementing the above ideas, in Construction 13.28.

THEOREM 13.29 *If the quadratic residuosity problem is hard relative to* GenModulus*, then the Goldwasser–Micali encryption scheme is CPA-secure.*

PROOF Let Π denote the Goldwasser–Micali encryption scheme. We prove that Π has indistinguishable encryptions in the presence of an eavesdropper; by Theorem 11.6 this implies that it is CPA-secure.

Let $\mathcal{A}$ be an arbitrary probabilistic polynomial-time adversary. Consider the following PPT adversary D that attempts to solve the quadratic residuosity problem relative to GenModulus:

> **Algorithm** D:
> The algorithm is given N and z as input, and its goal is to determine if $z \in \mathcal{QR}_N$ or $z \in \mathcal{QNR}_N^{+1}$.

CONSTRUCTION 13.28

Let GenModulus be as usual. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n, run GenModulus(1^n) to obtain (N, p, q), and choose a uniform $z \in \mathcal{QNR}_N^{+1}$. The public key is $pk = \langle N, z \rangle$ and the private key is $sk = \langle p, q \rangle$.
- Enc: on input a public key $pk = \langle N, z \rangle$ and a message $m \in \{0, 1\}$, choose a uniform $x \in \mathbb{Z}_N^*$ and output the ciphertext

$$c := [z^m \cdot x^2 \bmod N].$$

- Dec: on input a private key sk and a ciphertext c, determine whether c is a quadratic residue modulo N using, e.g., Algorithm 13.26. If yes, output 0; otherwise, output 1.

The Goldwasser–Micali encryption scheme.

- Set $pk = \langle N, z \rangle$ and run $\mathcal{A}(pk)$ to obtain two single-bit messages m_0, m_1.
- Choose a uniform bit b and a uniform $x \in \mathbb{Z}_N^*$, and then set $c := [z^{m_b} \cdot x^2 \bmod N]$.
- Give the ciphertext c to $\mathcal{A}$, who in turn outputs a bit b'. If $b' = b$, output 1; otherwise, output 0.

Let us analyze the behavior of D. There are two cases to consider:

Case 1: Say the input to D was generated by running GenModulus(1^n) to obtain (N, p, q) and then choosing a uniform $z \in \mathcal{QNR}_N^{+1}$. Then D runs $\mathcal{A}$ on a public key constructed exactly as in Π, and we see that in this case the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to $\mathcal{A}$'s view in experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$. Since D outputs 1 exactly when the output b' of $\mathcal{A}$ is equal to b, we have

$$\Pr[D(N, \mathsf{qnr}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1],$$

where qnr represents a uniform element of $\mathcal{QNR}_N^{+1}$ as in Definition 13.27.

Case 2: Say the input to D was generated by running GenModulus(1^n) to obtain (N, p, q) and then choosing a uniform $z \in \mathcal{QR}_N$. We claim that the view of $\mathcal{A}$ in this case is *independent* of the bit b. To see this, note that the ciphertext c given to $\mathcal{A}$ is a uniform quadratic residue regardless of whether a 0 or a 1 is encrypted:

- When a 0 is encrypted, $c = [x^2 \bmod N]$ for a uniform $x \in \mathbb{Z}_N^*$, and so c is a uniform quadratic residue.
- When a 1 is encrypted, $c = [z \cdot x^2 \bmod N]$ for a uniform $x \in \mathbb{Z}_N^*$. Let $\hat{x} \stackrel{\text{def}}{=} [x^2 \bmod N]$, and note that $\hat{x}$ is a uniformly distributed element

of the group $\mathcal{QR}_N$. Since $z \in \mathcal{QR}_N$, we can apply Lemma 11.15 to conclude that c is uniformly distributed in $\mathcal{QR}_N$ as well.

Since $\mathcal{A}$'s view is independent of b, the probability that $b' = b$ in this case is exactly $\frac{1}{2}$. That is,

$$\Pr[D(N, \mathsf{qr}) = 1] = \frac{1}{2},$$

where qr represents a uniform element of $\mathcal{QR}_N$ as in Definition 13.27.

Thus,

$$\left|\Pr[D(N, \mathsf{qr}) = 1] - \Pr[D(N, \mathsf{qnr}) = 1]\right| = \left|\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - \tfrac{1}{2}\right|.$$

By the assumption that the quadratic residuosity problem is hard relative to $\mathsf{GenModulus}$, there is a negligible function negl such that

$$\left|\varepsilon(n) - \tfrac{1}{2}\right| \leq \mathsf{negl}(n);$$

thus, $\varepsilon(n) \leq \frac{1}{2} + \mathsf{negl}(n)$. This completes the proof. ■

13.5 The Rabin Encryption Scheme

As mentioned at the beginning of this chapter, the Rabin encryption scheme is attractive because its security is *equivalent* to the assumption that factoring is hard. An analogous result is *not* known for RSA-based encryption, and the RSA problem may potentially be easier than factoring. (The same is true of the Goldwasser–Micali encryption scheme, and it is possible that deciding quadratic residuosity modulo N is easier than factoring N.)

Interestingly, the Rabin encryption scheme is (superficially, at least) very similar to the RSA encryption scheme yet has the advantage of being based on a potentially weaker assumption. The fact that RSA is more widely used than the former seems to be due more to historical factors than technical ones; we discuss this further at the end of this section.

We begin with some preliminaries about computing modular square roots. We then introduce a trapdoor permutation that can be based directly on the assumption that factoring is hard. The Rabin encryption scheme (or, at least, one instantiation of it) is then obtained by applying the results from Section 13.1. Throughout this section, we continue to let p and q denote odd primes, and let $N = pq$ denote a product of two distinct, odd primes.

13.5.1 Computing Modular Square Roots

The Rabin encryption scheme requires the receiver to compute modular square roots, and so in this section we explore the algorithmic complexity of

this problem. We first show an efficient algorithm for computing square roots modulo a prime p, and then extend this algorithm to enable computation of square roots modulo a composite N of *known* factorization. The reader willing to accept the existence of these algorithms on faith can skip to the following section, where we show that computing square roots modulo a composite N with *unknown* factorization is equivalent to factoring N.

Let p be an odd prime. Computing square roots modulo p is relatively simple when $p = 3 \bmod 4$, but much more involved when $p = 1 \bmod 4$. (The easier case is all we need for the Rabin encryption scheme as presented in Section 13.5.3; we include the second case for completeness.) In both cases, we show how to compute one of the square roots of a quadratic residue $a \in \mathbb{Z}_p^*$. Note that if x is one of the square roots of a, then $[-x \bmod p]$ is the other.

We tackle the easier case first. Say $p = 3 \bmod 4$, meaning we can write $p = 4i + 3$ for some integer i. Since $a \in \mathbb{Z}_p^*$ is a quadratic residue, we have $\mathcal{J}_p(a) = 1 = a^{\frac{p-1}{2}} \bmod p$ (see Proposition 13.17). Multiplying both sides by a we obtain

$$a = a^{\frac{p-1}{2}+1} = a^{2i+2} = \left(a^{i+1}\right)^2 \bmod p\,,$$

and so $a^{i+1} = a^{\frac{p+1}{4}} \bmod p$ is a square root of a. That is, we obtain a square root of a modulo p by simply computing $x := [a^{\frac{p+1}{4}} \bmod p\,]$.

It is crucial above that $(p+1)/2$ is *even* because this ensures that $(p+1)/4$ is an *integer* (this is necessary in order for $a^{\frac{p+1}{4}} \bmod p$ to be well-defined; recall that the exponent must be an integer). This approach does not succeed when $p = 1 \bmod 4$, in which case $p+1$ is an integer that is *not* divisible by 4.

When $p = 1 \bmod 4$ we proceed slightly differently. Motivated by the above approach, we might hope to find an odd integer r for which it holds that $a^r = 1 \bmod p$. Then, as above, $a^{r+1} = a \bmod p$ and $a^{\frac{r+1}{2}} \bmod p$ would be a square root of a with $(r+1)/2$ an integer. Although we will not be able to do this, we *can* do something just as good: we will find an odd integer r along with an element $b \in \mathbb{Z}_p^*$ and an *even* integer r' such that

$$a^r \cdot b^{r'} = 1 \bmod p.$$

Then $a^{r+1} \cdot b^{r'} = a \bmod p$ and $a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p$ is a square root of a (with the exponents $(r+1)/2$ and $r'/2$ being integers).

We now describe the general approach to finding r, b, and r' with the stated properties. Let $\frac{p-1}{2} = 2^\ell \cdot m$ where ℓ, m are integers with $\ell \geq 1$ and m odd.[5] Since a is a quadratic residue, we know that

$$a^{2^\ell m} = a^{\frac{p-1}{2}} = 1 \bmod p. \tag{13.6}$$

This means that $a^{2^\ell m/2} = a^{2^{\ell-1} m} \bmod p$ is a square root of 1. The square roots of 1 modulo p are $\pm 1 \bmod p$, so $a^{2^{\ell-1} m} = \pm 1 \bmod p$. If $a^{2^{\ell-1} m} = 1 \bmod p$, we

[5] The integers ℓ and m can be computed easily by taking out factors of 2 from $(p-1)/2$.

are in the same situation as in Equation (13.6) except that the exponent of a is now divisible by a smaller power of 2. This is progress in the right direction: if we can get to the point where the exponent of a is not divisible by *any* power of 2 (as would be the case here if $\ell = 1$), then the exponent of a is *odd* and we can compute a square root as discussed earlier. We give an example, and discuss in a moment how to deal with the case when $a^{2^{\ell-1}m} = -1 \bmod p$.

Example 13.30

Take $p = 29$ and $a = 7$. Since 7 is a quadratic residue modulo 29, we have $7^{14} \bmod 29 = 1$ and we know that $7^7 \bmod 29$ is a square root of 1. In fact,

$$7^7 = 1 \bmod 29,$$

and the exponent 7 is odd. So $7^{(7+1)/2} = 7^4 = 23 \bmod 29$ is a square root of 7 modulo 29. ♢

To summarize the algorithm so far: we begin with $a^{2^\ell m} = 1 \bmod p$ and we pull out factors of 2 from the exponent until one of two things happen: either $a^m = 1 \bmod p$, or $a^{2^{\ell'} m} = -1 \bmod p$ for some $\ell' < \ell$. In the first case, since m is odd we can immediately compute a square root of a as in Example 13.30. In the second case, we will "restore" the $+1$ on the right-hand side of the equation by multiplying each side of the equation by $-1 \bmod p$. However, as motivated at the beginning of this discussion, we want to achieve this by multiplying the left-hand side of the equation by some element b raised to an *even* power. If we have available a quadratic *non*-residue $b \in \mathbb{Z}_p^*$, this is easy: since $b^{2^\ell m} = b^{\frac{p-1}{2}} = -1 \bmod p$, we have

$$a^{2^{\ell'} m} \cdot b^{2^\ell m} = (-1)(-1) = +1 \bmod p.$$

With this we can proceed as before, taking a square root of the left-hand side to reduce the largest power of 2 dividing the exponent of a, and multiplying by $b^{2^\ell m}$ (as needed) so the right-hand side is always $+1$. Observe that the exponent of b is always divisible by a larger power of 2 than the exponent of a (and so we can indeed take square roots by dividing by 2 in both exponents). We continue performing these steps until the exponent of a is odd, and can then compute a square root of a as described earlier. Pseudocode for this algorithm, which gives another way of viewing what is going on, is given below in Algorithm 13.31. It can be verified that the algorithm runs in polynomial time given a quadratic non-residue b since the number of iterations of the inner loop is $\ell = \mathcal{O}(\log p)$.

One point we have not yet addressed is how to find b in the first place. In fact, no *deterministic* polynomial-time algorithm for finding a quadratic non-residue modulo p is known. Fortunately, it is easy to find a quadratic non-residue probabilistically: simply choose uniform elements of $\mathbb{Z}_p^*$ until a

ALGORITHM 13.31
Computing square roots modulo a prime

Input: Prime p; quadratic residue $a \in \mathbb{Z}_p^*$
Output: A square root of a

case $p = 3 \bmod 4$:
  return $[a^{\frac{p+1}{4}} \bmod p]$
case $p = 1 \bmod 4$:
  let b be a quadratic non-residue modulo p
  compute ℓ and m odd with $2^\ell \cdot m = \frac{p-1}{2}$
  $r := 2^\ell \cdot m,\ r' := 0$
  for $i = \ell$ to 1 {
  // maintain the invariant $a^r \cdot b^{r'} = 1 \bmod p$
    $r := r/2,\ r' := r'/2$
    if $a^r \cdot b^{r'} = -1 \bmod p$
      $r' := r' + 2^\ell \cdot m$
  }
  // now $r = m,\ r'$ is even, and $a^r \cdot b^{r'} = 1 \bmod p$
  return $\left[a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p\right]$

quadratic non-residue is found. This works because exactly half the elements of $\mathbb{Z}_p^*$ are quadratic non-residues, and because a polynomial-time algorithm for deciding quadratic residuosity modulo a prime is known (see Section 13.4.1 for proofs of both these statements). This means that the algorithm we have shown is actually randomized when $p = 1 \bmod 4$; a deterministic polynomial-time algorithm for computing square roots in this case is not known.

Example 13.32
Here we consider the "worst case," when taking a square root always gives -1. Let $a \in \mathbb{Z}_p^*$ be the element whose square root we are trying to compute; let $b \in \mathbb{Z}_p^*$ be a quadratic non-residue; and let $\frac{p-1}{2} = 2^3 \cdot m$ where m is odd.

In the first step, we have $a^{2^3 m} = 1 \bmod p$. Since $a^{2^3 m} = \left(a^{2^2 m}\right)^2$ and the square roots of 1 are ± 1, this means that $a^{2^2 m} = \pm 1 \bmod p$; assuming the worst case, $a^{2^2 m} = -1 \bmod p$. So, we multiply by $b^{\frac{p-1}{2}} = b^{2^3 m} = -1 \bmod p$ to obtain

$$a^{2^2 m} \cdot b^{2^3 m} = 1 \bmod p.$$

In the second step, we observe that $a^{2m} \cdot b^{2^2 m}$ is a square root of 1; again assuming the worst case, we thus have $a^{2m} \cdot b^{2^2 m} = -1 \bmod p$. Multiplying by $b^{2^3 m}$ to "correct" this gives

$$a^{2m} \cdot b^{2^2 m} \cdot b^{2^3 m} = 1 \bmod p.$$

In the third step, taking square roots and assuming the worst case (as

above) we obtain $a^m \cdot b^{2m} \cdot b^{2^2 m} = -1 \bmod p$; multiplying by the "correction factor" $b^{2^3 m}$ we get

$$a^m \cdot b^{2m} \cdot b^{2^2 m} \cdot b^{2^3 m} = 1 \bmod p.$$

We are now where we want to be. To conclude the algorithm, multiply both sides by a to obtain

$$a^{m+1} \cdot b^{2m+2^2 m+2^3 m} = a \bmod p.$$

Since m is odd, $(m+1)/2$ is an integer and $a^{\frac{m+1}{2}} \cdot b^{m+2m+2^2 m} \bmod p$ is a square root of a. ♢

Example 13.33

Here we work out a concrete example. Let $p = 17$, $a = 2$, and $b = 3$. Note that here $(p-1)/2 = 2^3$ and $m = 1$.

We begin with $2^{2^3} = 1 \bmod 17$. So 2^{2^2} should be equal to $\pm 1 \bmod 17$; by calculation, $2^{2^2} = -1 \bmod 17$. Multiplying by 3^{2^3} gives $2^{2^2} \cdot 3^{2^3} = 1 \bmod 17$.

Continuing, we know that $2^2 \cdot 3^{2^2}$ is a square root of 1 and so must be equal to $\pm 1 \bmod 17$; calculation gives $2^2 \cdot 3^{2^2} = 1 \bmod 17$. So no correction term is needed here.

Halving the exponents again we find that $2 \cdot 3^2 = 1 \bmod 17$. We are now almost done: multiplying both sides by 2 gives $2^2 \cdot 3^2 = 2 \bmod 17$, and so $2 \cdot 3 = 6 \bmod 17$ is a square root of 2. ♢

Computing Square Roots Modulo N

It is not hard to see that the algorithm we have shown for computing square roots modulo a prime extends easily to the case of computing square roots modulo a composite $N = pq$ of known factorization. Specifically, let $a \in \mathbb{Z}_N^*$ be a quadratic residue with $a \leftrightarrow (a_p, a_q)$ via the Chinese remainder theorem. Computing the square roots x_p, x_q of a_p, a_q modulo p and q, respectively, gives a square root (x_p, x_q) of a (see Section 13.4.2). Given x_p and x_q, the representation x corresponding to (x_p, x_q) can be recovered as discussed in Section 8.1.5. That is, to compute a square root of a modulo an integer $N = pq$ of known factorization:

- Compute $a_p := [a \bmod p]$ and $a_q := [a \bmod q]$.
- Using Algorithm 13.31, compute a square root x_p of a_p modulo p and a square root x_q of a_q modulo q.
- Convert from the representation $(x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ to $x \in \mathbb{Z}_N^*$ with $x \leftrightarrow (x_p, x_q)$. Output x, which is a square root of a modulo N.

It is easy to modify the algorithm so that it returns all four square roots of a.

13.5.2 A Trapdoor Permutation Based on Factoring

We have seen that computing square roots modulo N can be carried out in polynomial time if the factorization of N is known. We show here that, in contrast, computing square roots modulo a composite N of *unknown* factorization is as hard as factoring N.

More formally, let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes except with probability negligible in n. Consider the following experiment for a given algorithm $\mathcal{A}$ and parameter n:

The square-root computation experiment $\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n)$:

1. *Run* $\mathsf{GenModulus}(1^n)$ *to obtain output* N, p, q.
2. *Choose a uniform* $y \in \mathcal{QR}_N$.
3. $\mathcal{A}$ *is given* (N, y), *and outputs* $x \in \mathbb{Z}_N^*$.
4. *The output of the experiment is defined to be* 1 *if* $x^2 = y \bmod N$, *and* 0 *otherwise.*

DEFINITION 13.34 *We say that* computing square roots is hard relative to GenModulus *if for all probabilistic polynomial-time algorithms* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n).$$

It is easy to see that if computing square roots is hard relative to GenModulus then factoring must be hard relative to GenModulus too: if moduli N output by GenModulus could be factored easily, then it would be easy to compute square roots modulo N by first factoring N and then applying the algorithm discussed in the previous section. Our goal now is to show the converse: that if factoring is *hard* relative to GenModulus then so is the problem of computing square roots. We emphasize again that an analogous result is not known for the RSA problem or the problem of deciding quadratic residuosity.

The key is the following lemma, which says that two "unrelated" square roots of any element in $\mathbb{Z}_N^*$ can be used to factor N.

LEMMA 13.35 *Let* $N = pq$ *with* p, q *distinct, odd primes. Given* $x, \hat{x}$ *such that* $x^2 = y = \hat{x}^2 \bmod N$ *but* $x \neq \pm\hat{x} \bmod N$, *it is possible to factor* N *in time polynomial in* $\|N\|$.

PROOF We claim that either $\gcd(N, x + \hat{x})$ or $\gcd(N, x - \hat{x})$ is equal to one of the prime factors of N.[6] Since gcd computations can be carried out in polynomial time (see Appendix B.1.2), this proves the lemma.

[6] In fact, both of these are equal to one of the prime factors of N.

If $x^2 = \hat{x}^2 \bmod N$ then

$$0 = x^2 - \hat{x}^2 = (x - \hat{x}) \cdot (x + \hat{x}) \bmod N,$$

and so $N \mid (x - \hat{x})(x + \hat{x})$. Then $p \mid (x - \hat{x})(x + \hat{x})$ and so p divides one of these terms. Say $p \mid (x + \hat{x})$ (the proof proceeds similarly if $p \mid (x - \hat{x})$). If $q \mid (x + \hat{x})$ then $N \mid (x + \hat{x})$, but this cannot be the case since $x \neq -\hat{x} \bmod N$. So $q \nmid (x + \hat{x})$ and $\gcd(N, x + \hat{x}) = p$. ■

An alternative way of proving the above is to look at what happens in the Chinese remaindering representation. Say $x \leftrightarrow (x_p, x_q)$. Then, because x and $\hat{x}$ are square roots of the same value y, we know that $\hat{x}$ corresponds to either $(-x_p, x_q)$ or $(x_p, -x_q)$. (It cannot correspond to (x_p, x_q) or $(-x_p, -x_q)$ since the first corresponds to x while the second corresponds to $[-x \bmod N]$, and both possibilities are ruled out by the assumption of the lemma.) Say $\hat{x} \leftrightarrow (-x_p, x_q)$. Then

$$[x + \hat{x} \bmod N] \leftrightarrow (x_p, x_q) + (-x_p, x_q) = (0, [2x_q \bmod q]),$$

and we see that $x + \hat{x} = 0 \bmod p$ while $x + \hat{x} \neq 0 \bmod q$. It follows that $\gcd(N, x + \hat{x}) = p$, a factor of N.

We can now prove the main result of this section.

THEOREM 13.36 *If factoring is hard relative to* GenModulus*, then computing square roots is hard relative to* GenModulus.

PROOF Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm computing square roots (as in Definition 13.34). Consider the following probabilistic polynomial-time algorithm $\mathcal{A}_{\text{fact}}$ for factoring moduli output by GenModulus:

Algorithm $\mathcal{A}_{\text{fact}}$:
The algorithm is given a modulus N as input.

- Choose a uniform $x \in \mathbb{Z}_N^*$ and compute $y := [x^2 \bmod N]$.
- Run $\mathcal{A}(N, y)$ to obtain output $\hat{x}$.
- If $\hat{x}^2 = y \bmod N$ and $\hat{x} \neq \pm x \bmod N$, then factor N using Lemma 13.35.

By Lemma 13.35, we know that $\mathcal{A}_{\text{fact}}$ succeeds in factoring N exactly when $\hat{x} \neq \pm x \bmod N$ and $\hat{x}^2 = y \bmod N$. That is,

$$\begin{aligned}
&\Pr[\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n) = 1] \\
&= \Pr\left[\hat{x} \neq \pm x \bmod N \wedge \hat{x}^2 = y \bmod N\right] \\
&= \Pr\left[\hat{x} \neq \pm x \bmod N \,\middle|\, \hat{x}^2 = y \bmod N\right] \cdot \Pr\left[\hat{x}^2 = y \bmod N\right], \quad (13.7)
\end{aligned}$$

where the above probabilities all refer to experiment $\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n)$ (see Section 8.2.3 for a description of this experiment). In the experiment, the modulus N given as input to $\mathcal{A}_{\text{fact}}$ is generated by $\mathsf{GenModulus}(1^n)$, and y is a uniform quadratic residue modulo N since x was chosen uniformly from $\mathbb{Z}_N^*$ (see Section 13.4.4). So the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}_{\text{fact}}$ is distributed exactly as $\mathcal{A}$'s view in experiment $\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n)$. Therefore,

$$\Pr\left[\hat{x}^2 = y \bmod N\right] = \Pr\left[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1\right]. \tag{13.8}$$

Conditioned on the value of the quadratic residue y used in experiment $\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n)$, the value x is equally likely to be any of the four possible square roots of y. This means that from the point of view of algorithm $\mathcal{A}$ (being run as a subroutine by $\mathcal{A}_{\text{fact}}$), x is equally likely to be each of the four square roots of y. This in turn means that, conditioned on $\mathcal{A}$ outputting *some* square root $\hat{x}$ of y, the probability that $\hat{x} = \pm x \bmod N$ is exactly $1/2$. (We stress that we do not make any assumption about how $\hat{x}$ is distributed among the square roots of y, and in particular are not assuming here that $\mathcal{A}$ outputs a uniform square root of y. Rather we are using the fact that x is uniformly distributed among the square roots of y.) That is,

$$\Pr\left[\hat{x} \neq \pm x \bmod N \;\middle|\; \hat{x}^2 = y \bmod N\right] = \frac{1}{2}. \tag{13.9}$$

Combining Equations (13.7)–(13.9), we see that

$$\Pr\left[\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n) = 1\right] = \frac{1}{2} \cdot \Pr\left[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1\right].$$

Since factoring is hard relative to $\mathsf{GenModulus}$, there is a negligible function negl such that

$$\Pr[\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n),$$

which implies $\Pr\left[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1\right] \leq 2 \cdot \mathsf{negl}(n)$. Since $\mathcal{A}$ was arbitrary, this completes the proof. ∎

The previous theorem leads directly to a family of one-way functions (see Definition 8.76) based on any $\mathsf{GenModulus}$ relative to which factoring is hard:

- Algorithm Gen, on input 1^n, runs $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q) and outputs $I = N$. The domain $\mathcal{D}_I$ is $\mathbb{Z}_N^*$ and the range $\mathcal{R}_I$ is $\mathcal{QR}_N$.
- Algorithm Samp, on input N, chooses a uniform element $x \in \mathbb{Z}_N^*$.
- Algorithm f, on input N and $x \in \mathbb{Z}_N^*$, outputs $[x^2 \bmod N]$.

The preceding theorem shows that this family is one-way if factoring is hard relative to $\mathsf{GenModulus}$.

We can turn this into a family of one-way *permutations* by using moduli N of a special form and letting $\mathcal{D}_I$ be a subset of $\mathbb{Z}_N^*$. (See Exercise 13.19 for another way to make this a permutation.) Call $N = pq$ a *Blum integer* if p and q are distinct primes with $p = q = 3 \bmod 4$. The key to building a permutation is the following proposition.

PROPOSITION 13.37 *Let N be a Blum integer. Then every quadratic residue modulo N has exactly one square root that is also a quadratic residue.*

PROOF Say $N = pq$ with $p = q = 3 \bmod 4$. Using Proposition 13.17, we see that -1 is not a quadratic residue modulo p or q. This is because for $p = 3 \bmod 4$ it holds that $p = 4i + 3$ for some i and so

$$(-1)^{\frac{p-1}{2}} = (-1)^{2i+1} = -1 \bmod p$$

(because $2i+1$ is odd). Now let $y \leftrightarrow (y_p, y_q)$ be an arbitrary quadratic residue modulo N with the four square roots

$$(x_p, x_q), \quad (-x_p, x_q), \quad (x_p, -x_q), \quad (-x_p, -x_q).$$

We claim that exactly one of these is a quadratic residue modulo N. To see this, assume $\mathcal{J}_p(x_p) = +1$ and $\mathcal{J}_q(x_q) = -1$ (the proof proceeds similarly in any other case). Using Proposition 13.19, we have

$$\mathcal{J}_q(-x_q) = \mathcal{J}_q(-1) \cdot \mathcal{J}_q(x_q) = +1\,,$$

and so $(x_p, -x_q)$ corresponds to a quadratic residue modulo N (using Proposition 13.21). Similarly, $\mathcal{J}_p(-x_p) = -1$ and so none of the other square roots of y are quadratic residues modulo N. ■

Expressed differently, the above proposition says that when N is a Blum integer, the function $f_N : \mathcal{QR}_N \to \mathcal{QR}_N$ given by $f_N(x) = [x^2 \bmod N]$ is a permutation over $\mathcal{QR}_N$. Modifying the sampling algorithm Samp above to choose a uniform $x \in \mathcal{QR}_N$ (which, as we have already seen, can be done easily by choosing uniform $r \in \mathbb{Z}_N^*$ and setting $x := [r^2 \bmod N]$) gives a family of one-way *permutations*. Finally, because square roots modulo N can be computed in polynomial time given the factorization of N, a straightforward modification yields a family of *trapdoor* permutations based on any GenModulus relative to which factoring is hard. This is sometimes called the *Rabin* family of trapdoor permutations. In summary:

THEOREM 13.38 *Let* GenModulus *be an algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are distinct primes (except possibly with negligible probability) with $p = q = 3 \bmod 4$. If factoring is hard relative to* GenModulus*, then there exists a family of trapdoor permutations.*

13.5.3 The Rabin Encryption Scheme

We can apply the results of Section 13.1.2 to the Rabin trapdoor permutation to obtain a public-key encryption scheme whose security is based on factoring. To do this, we first need to identify a hard-core predicate for this trapdoor permutation. Although we could appeal to Theorem 13.3, which states that a suitable hard-core predicate always exists, it turns out that the least significant bit lsb is a hard-core predicate for the Rabin trapdoor permutation just as it is for the case of RSA (see Section 11.5.3). Using this as our hard-core predicate, we obtain the scheme of Construction 13.39.

CONSTRUCTION 13.39

Let $\mathsf{GenModulus}$ be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes (except with probability negligible in n) with $p = q = 3 \bmod 4$. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q). The public key is N, and the private key is $\langle p, q \rangle$.
- Enc: on input a public-key N and message $m \in \{0,1\}$, choose a uniform $x \in \mathcal{QR}_N$ subject to the constraint that $\mathsf{lsb}(x) = m$. Output the ciphertext $c := [x^2 \bmod N]$.
- Dec: on input a private key $\langle p, q \rangle$ and a ciphertext c, compute the unique $x \in \mathcal{QR}_N$ such that $x^2 = c \bmod N$, and output $\mathsf{lsb}(x)$.

The Rabin encryption scheme.

Theorems 13.5 and 13.38 imply the following result.

THEOREM 13.40 *If factoring is hard relative to* $\mathsf{GenModulus}$*, then Construction 13.39 is CPA-secure.*

Rabin Encryption vs. RSA Encryption

It is worthwhile to remark on the similarities and differences between the Rabin and RSA cryptosystems. (The discussion here applies to any cryptographic construction—not necessarily a public-key encryption scheme—based on the Rabin or RSA trapdoor permutations.)

The RSA and Rabin trapdoor permutations appear quite similar, with squaring in the case of Rabin corresponding to taking $e = 2$ in the case of RSA. (Of course, 2 is *not* relatively prime to $\phi(N)$ and so Rabin is not a special case of RSA.) In terms of the security offered by each construction, hardness of computing modular square roots is equivalent to hardness of factoring, whereas hardness of solving the RSA problem is not known to

be implied by the hardness of factoring. The Rabin trapdoor permutation is thus based on a potentially *weaker* assumption: it is theoretically possible that someone might develop an efficient algorithm for solving the RSA problem, yet computing square roots will remain hard. Or, someone may develop an algorithm that solves the RSA problem faster than known factoring algorithms. Lemma 13.35 ensures, however, that computing square roots modulo N can never be much faster than the best available algorithm for factoring N.

In terms of their efficiency, the RSA and Rabin permutations are essentially the same. Actually, if a large exponent e is used in the case of RSA then computing eth powers (as in RSA) is slightly slower than squaring (as in Rabin). On the other hand, a bit more care is required when working with the Rabin permutation since it is only a permutation over a *subset* of $\mathbb{Z}_N^*$, in contrast to RSA, which gives a permutation over all of $\mathbb{Z}_N^*$.

A "plain Rabin" encryption scheme, constructed in a manner analogous to plain RSA encryption, is vulnerable to a chosen-ciphertext attack that enables an adversary to learn the entire private key (see Exercise 13.17). Although plain RSA is not CCA-secure either, known chosen-ciphertext attacks on plain RSA are less damaging since they recover the message but not the private key. Perhaps the existence of such an attack on "plain Rabin" influenced cryptographers, early on, to reject the use of Rabin encryption entirely.

In summary, the RSA permutation is much more widely used in practice than the Rabin permutation, but in light of the above this appears to be due more to historical accident than to any compelling technical justification.

References and Additional Reading

The existence of public-key encryption based on arbitrary trapdoor permutations was shown by Yao [179], and the efficiency improvement discussed at the end of Section 13.1.2 is due to Blum and Goldwasser [36].

Childs [44] and Shoup [159] provide further coverage of the (computational) number theory used in this chapter. A good description of the algorithm for computing the Jacobi symbol modulo a composite of unknown factorization, along with a proof of correctness, is given in [57].

The Paillier encryption scheme was introduced in [136]. Shoup [159, Section 7.5] gives a characterization of $\mathbb{Z}_{N^e}^*$ for arbitrary integers N, e (and not just $N = pq$, $e = 2$ as done here).

The problem of deciding quadratic residuosity modulo a composite of unknown factorization goes back to Gauss [71] and is related to other (conjectured) hard number-theoretic problems. The Goldwasser–Micali encryption scheme [80], introduced in 1982, was the first public-key encryption scheme with a proof of security.

Rabin [145] showed that computing square roots modulo a composite is equivalent to factoring. The results of Section 13.5.2 are due to Blum [35]. Hard-core predicates for the Rabin trapdoor permutation are discussed in [8, 86, 7] and references therein.

Exercises

13.1 Construct and prove CPA-security for a KEM based on any trapdoor permutation by suitably generalizing Construction 11.34.

13.2 Show that the isomorphism of Proposition 13.6 can be efficiently inverted when the factorization of N is known.

13.3 Let $\Phi(N^2)$ denote the set $\{(a,1) \mid a \in \mathbb{Z}_N\} \subset \mathbb{Z}^*_{N^2}$. Show that it is *not* hard to decide whether a given element $y \in \mathbb{Z}^*_{N^2}$ is in $\Phi(N^2)$.

13.4 Let $\mathbb{G}$ be an abelian group. Show that the set of quadratic residues in $\mathbb{G}$ forms a subgroup.

13.5 This question concerns the quadratic residues in the additive group $\mathbb{Z}_N$. (An element $y \in \mathbb{Z}_N$ is a quadratic residue if and only if there exists an $x \in \mathbb{Z}_N$ with $2x = y \bmod N$.)

(a) Let p be an odd prime. How many elements of $\mathbb{Z}_p$ are quadratic residues?

(b) Let $N = pq$ be a product of two odd primes p and q. How many elements of $\mathbb{Z}_N$ are quadratic residues?

(c) Let N be an even integer. How many elements of $\mathbb{Z}_N$ are quadratic residues?

13.6 Let $N = pq$ with p, q distinct, odd primes. Show a PPT algorithm for choosing a uniform element of $\mathcal{QNR}^{+1}_N$ *when the factorization of* N *is known.* (Your algorithm can have failure probability negligible in $\|N\|$.)

13.7 Let $N = pq$ with p, q distinct, odd primes. Prove that if $x \in \mathcal{QR}_N$ then $[x^{-1} \bmod N] \in \mathcal{QR}_N$, and if $x \in \mathcal{QNR}^{+1}_N$ then $[x^{-1} \bmod N] \in \mathcal{QNR}^{+1}_N$.

13.8 Let $N = pq$ with p, q distinct, odd primes, and fix $z \in \mathcal{QNR}^{+1}_N$. Show that choosing uniform $x \in \mathcal{QR}_N$ and setting $y := [z \cdot x \bmod N]$ gives a y that is uniformly distributed in $\mathcal{QNR}^{+1}_N$. That is, for any $\hat{y} \in \mathcal{QNR}^{+1}_N$

$$\Pr[z \cdot x = \hat{y} \bmod N] = 1/|\mathcal{QNR}^{+1}_N|,$$

where the probability is taken over uniform choice of $x \in \mathcal{QR}_N$.

Hint: Use the previous exercise.

13.9 Let N be the product of 5 distinct, odd primes. If $y \in \mathbb{Z}_N^*$ is a quadratic residue, how many solutions are there to the equation $x^2 = y \bmod N$?

13.10 Show that the Goldwasser–Micali encryption scheme is homomorphic if the message space $\{0,1\}$ is viewed as the group $\mathbb{Z}_2$.

13.11 Consider the following variation of the Goldwasser–Micali encryption scheme: $\mathsf{GenModulus}(1^n)$ is run to obtain (N,p,q) where $N = pq$ and $p = q = 3 \bmod 4$. (I.e., N is a Blum integer.) The public key is N and the private key is $\langle p,q\rangle$. To encrypt $m \in \{0,1\}$, the sender chooses uniform $x \in \mathbb{Z}_N$ and computes the ciphertext $c := [(-1)^m \cdot x^2 \bmod N]$.

(a) Prove that for N of the stated form, $[-1 \bmod N] \in \mathcal{QNR}_N^{+1}$.

(b) Prove that the scheme described has indistinguishable encryptions under a chosen-plaintext attack if deciding quadratic residuosity is hard relative to $\mathsf{GenModulus}$.

13.12 Assume deciding quadratic residuosity is hard for $\mathsf{GenModulus}$. Show that this implies the hardness of distinguishing a uniform element of $\mathcal{QR}_N$ from a uniform element of $\mathcal{J}_N^{+1}$.

13.13 Show that plain RSA encryption of a message m leaks $\mathcal{J}_N(m)$.

13.14 Consider the following variation of the Goldwasser–Micali encryption scheme: $\mathsf{GenModulus}(1^n)$ is run to obtain (N,p,q). The public key is N and the private key is $\langle p,q\rangle$. To encrypt a 0, the sender chooses n uniform elements $c_1,\ldots,c_n \in \mathcal{QR}_N$. To encrypt a 1, the sender chooses n uniform elements $c_1,\ldots,c_n \in \mathcal{J}_N^{+1}$. In each case, the resulting ciphertext is $c^* = \langle c_1,\ldots,c_n\rangle$.

(a) Show how the sender can generate a uniform element of $\mathcal{J}_N^{+1}$ in polynomial time, where failing with negligible probability.

(b) Suggest a way for the receiver to decrypt efficiently, although with negligible error probability.

(c) Prove that if deciding quadratic residuosity is hard relative to $\mathsf{GenModulus}$, this scheme is CPA-secure.

Hint: Use the previous exercise.

13.15 Let $\mathcal{G}$ be a polynomial-time algorithm that, on input 1^n, outputs a prime p with $\|p\| = n$ and a generator g of $\mathbb{Z}_p^*$. Prove that the DDH problem is *not* hard relative to $\mathcal{G}$.

Hint: Use the fact that quadratic residuosity can be decided efficiently modulo a prime.

13.16 The discrete logarithm problem is believed to be hard for $\mathcal{G}$ as in the previous exercise. This means that the function (family) $f_{p,g}$ where $f_{p,g}(x) \stackrel{\text{def}}{=} [g^x \bmod p]$ is one-way. Let $\mathsf{lsb}(x)$ denote the least-significant bit of x. Show that lsb is *not* a hard-core predicate for $f_{p,g}$.

13.17 Consider the plain Rabin encryption scheme in which a message $m \in \mathcal{QR}_N$ is encrypted relative to a public key N (where N is a Blum integer) by computing the ciphertext $c := [m^2 \bmod N]$. Show a chosen-ciphertext attack on this scheme that recovers the entire private key.

13.18 The plain Rabin signature scheme is like the plain RSA signature scheme, except using the Rabin trapdoor permutation. Show an attack on plain Rabin signatures by which the attacker learns the signer's private key.

13.19 Let N be a Blum integer.

(a) Define the set $S \stackrel{\text{def}}{=} \{x \in \mathbb{Z}_N^* \mid x < N/2 \text{ and } \mathcal{J}_N(x) = +1\}$. Define the function $f_N : S \to \mathbb{Z}_N^*$ by:

$$f_N(x) = \begin{cases} [x^2 \bmod N] & \text{if } [x^2 \bmod N] < N/2 \\ [-x^2 \bmod N] & \text{if } [x^2 \bmod N] > N/2 \end{cases}$$

Show that f_N is a permutation over S.

(b) Define a family of trapdoor permutations based on factoring using f_N as defined above.

13.20 Let N be a Blum integer. Define the function $\mathsf{half}_N : \mathbb{Z}_N^* \to \{0, 1\}$ as

$$\mathsf{half}_N(x) = \begin{cases} -1 \text{ if } x < N/2 \\ +1 \text{ if } x > N/2 \end{cases}$$

Show that the function $f : \mathbb{Z}_N^* \to \mathcal{QR}_N \times \{-1, +1\}^2$ defined as

$$f(x) = \big([x^2 \bmod N],\, \mathcal{J}_N(x),\, \mathsf{half}_N(x)\big)$$

is one-to-one.

Index of Common Notation

General notation:

- $:=$ refers to deterministic assignment
- If S is a set, then $x \leftarrow S$ denotes that x is chosen uniformly from S
- If A is a randomized algorithm, then $y \leftarrow A(x)$ denotes running A on input x with a uniform random tape and assigning the output to y. We write $y := A(x; r)$ to denote running A on input x using random tape r and assigning the output to y
- $\wedge$ denotes Boolean conjunction (the AND operator)
- $\vee$ denotes Boolean disjunction (the OR operator)
- $\oplus$ denotes the exclusive-or (XOR) operator; this operator can be applied to single bits or entire strings (in the latter case, the XOR is bitwise)
- $\{0,1\}^n$ is the set of all bit-strings of length n
- $\{0,1\}^{\leq n}$ is the set of all bit-strings of length at most n
- $\{0,1\}^*$ is the set of all finite bit-strings
- 0^n (resp., 1^n) denotes the string comprised of n zeroes (resp., n ones)
- $\|x\|$ denotes the length of the binary representation of the (positive) integer x, written with leading bit 1. Note that $\log x < \|x\| \leq \log x + 1$
- $|x|$ denotes the length of the binary string x (which may have leading 0s), or the absolute value of the real number x
- $\mathcal{O}(\cdot), \Theta(\cdot), \Omega(\cdot), \omega(\cdot)$ see Appendix A.2
- `0x` denotes that digits are being represented in hexadecimal
- $x \| y$ denotes umambiguous concatenation of the strings x and y ("unambiguous" means that x and y can be recovered from $x \| y$)
- $\Pr[X]$ denotes the probability of event X
- $\log x$ denotes the base-2 logarithm of x

Crypto-specific notation:

- n is the security parameter
- PPT stands for "probabilistic polynomial time"
- $\mathcal{A}^{\mathcal{O}(\cdot)}$ denotes the algorithm $\mathcal{A}$ with oracle access to $\mathcal{O}$
- k typically denotes a secret key (as in private-key encryption and MACs)
- (pk, sk) denotes a public/private key-pair (for public-key encryption and digital signatures)
- negl denotes a negligible function; see Definition 3.4
- $\mathsf{poly}(n)$ denotes an arbitrary polynomial
- $\mathsf{polylog}(n)$ denotes $\mathsf{poly}(\log(n))$
- Func_n denotes the set of functions mapping n-bit strings to n-bit strings
- Perm_n denotes the set of bijections on n-bit strings
- IV denotes an initialization vector (used for modes of operation and collision-resistant hash functions)

Algorithms and procedures:

- G denotes a pseudorandom generator
- F denotes a keyed function that is typically a pseudorandom function or permutation
- $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote the key-generation, encryption, and decryption procedures, respectively, for both private- and public-key encryption. For the case of private-key encryption, when Gen is unspecified then $\mathsf{Gen}(1^n)$ outputs a uniform $k \in \{0,1\}^n$
- $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ denote the key-generation, tag-generation, and verification procedures, respectively, for a message authentication code. When Gen is unspecified then $\mathsf{Gen}(1^n)$ outputs a uniform $k \in \{0,1\}^n$
- $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ denote the key-generation, signature-generation, and verification procedures, respectively, for a digital signature scheme
- $\mathsf{GenPrime}$ denotes a PPT algorithm that, on input 1^n, outputs an n-bit prime except with probability negligible in n
- $\mathsf{GenModulus}$ denotes a PPT algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and (except with negligible probability) p and q are n-bit primes

- GenRSA denotes a PPT algorithm that, on input 1^n, outputs (except with negligible probability) a modulus N, an integer $e > 0$ with $\gcd(e, \phi(N)) = 1$, and an integer d satisfying $ed = 1 \bmod \phi(N)$
- $\mathcal{G}$ denotes a PPT algorithm that, on input 1^n, outputs (except with negligible probability) a description of a cyclic group $\mathbb{G}$, the group order q (with $\|q\| = n$), and a generator $g \in \mathbb{G}$.

Number theory:

- $\mathbb{Z}$ denotes the set of integers
- $a \mid b$ means a divides b
- $a \nmid b$ means that a does not divide b
- $\gcd(a, b)$ denotes the greatest common divisor of a and b
- $[a \bmod b]$ denotes the remainder of a when divided by b. Note that $0 \leq [a \bmod b] < b$.
- $x_1 = x_2 = \cdots = x_n \bmod N$ means that $x_1, \ldots, x_n$ are all congruent modulo N

  Note: $x = y \bmod N$ means that x and y are congruent modulo N, whereas $x = [y \bmod N]$ means that x is equal to the remainder of y when divided by N.
- $\mathbb{Z}_N$ denotes the additive group of integers modulo N as well as the set $\{0, \ldots, N-1\}$
- $\mathbb{Z}_N^*$ denotes the multiplicative group of invertible integers modulo N (i.e., those that are relatively prime to N)
- $\phi(N)$ denotes the size of $\mathbb{Z}_N^*$
- $\mathbb{G}$ and $\mathbb{H}$ denote groups
- $\mathbb{G}_1 \simeq \mathbb{G}_2$ means that groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are isomorphic. If this isomorphism is given by f and $f(x_1) = x_2$ then we write $x_1 \leftrightarrow x_2$
- g is typically a generator of a group
- $\log_g h$ denotes the discrete logarithm of h to the base g
- $\langle g \rangle$ denotes the group generated by g
- p and q usually denote primes
- N typically denotes the product of two distinct primes p and q of equal length

- $\mathcal{QR}_p$ is the set of quadratic residues modulo p
- $\mathcal{QNR}_p$ is the set of quadratic non-residues modulo p
- $\mathcal{J}_p(x)$ is the Jacobi symbol of x modulo p
- $\mathcal{J}_N^{+1}$ is the set of elements with Jacobi symbol $+1$ modulo N
- $\mathcal{J}_N^{-1}$ is the set of elements with Jacobi symbol -1 modulo N
- $\mathcal{QNR}_N^{+1}$ is the set of quadratic non-residues modulo N having Jacobi symbol $+1$

Appendix A

Mathematical Background

A.1 Identities and Inequalities

We list some standard identities and inequalities that are used at various points throughout the text.

THEOREM A.1 (Binomial expansion theorem) *Let x, y be real numbers, and let n be a positive integer. Then*

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i \, y^{n-i}.$$

PROPOSITION A.2 *For all $x \geq 1$ it holds that $(1-1/x)^x \leq e^{-1}$.*

PROPOSITION A.3 *For all x it holds that $1 - x \leq e^{-x}$.*

PROPOSITION A.4 *For all x with $0 \leq x \leq 1$ it holds that*

$$e^{-x} \leq 1 - \left(1 - \frac{1}{e}\right) \cdot x \leq 1 - \frac{x}{2}\,.$$

A.2 Asymptotic Notation

We use standard notation for expressing asymptotic behavior of functions.

DEFINITION A.5 *Let $f(n), g(n)$ be functions from non-negative integers to non-negative reals. Then:*

- *$f(n) = \mathcal{O}(g(n))$ means that there exist positive integers c and n' such that for all $n > n'$ it holds that $f(n) \leq c \cdot g(n)$.*

- $f(n) = \Omega(g(n))$ *means that there exist positive integers* c *and* n' *such that for all* $n > n'$ *it holds that* $f(n) \geq c \cdot g(n)$.
- $f(n) = \Theta(g(n))$ *means that there exist positive integers* c_1, c_2, *and* n' *such that for all* $n > n'$ *it holds that* $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.
- $f(n) = o(g(n))$ *means that* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ *means that* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Example A.6
Let $f(n) = n^4 + 3n + 500$. Then:

- $f(n) = \mathcal{O}(n^4)$.
- $f(n) = \mathcal{O}(n^5)$. In fact, $f(n) = o(n^5)$.
- $f(n) = \Omega(n^3 \log n)$. In fact, $f(n) = \omega(n^3 \log n)$.
- $f(n) = \Theta(n^4)$.

♢

A.3 Basic Probability

We assume the reader is familiar with basic probability theory, on the level of what is covered in a typical undergraduate course on discrete mathematics. Here we simply remind the reader of some notation and basic facts.

If E is an event, then $\bar{E}$ denotes the complement of that event; i.e., $\bar{E}$ is the event that E does *not* occur. By definition, $\Pr[E] = 1 - \Pr[\bar{E}]$. If E_1 and E_2 are events, then $E_1 \wedge E_2$ denotes their conjunction; i.e., $E_1 \wedge E_2$ is the event that *both* E_1 and E_2 occur. By definition, $\Pr[E_1 \wedge E_2] \leq \Pr[E_1]$. Events E_1 and E_2 are said to be *independent* if $\Pr[E_1 \wedge E_2] = \Pr[E_1] \cdot \Pr[E_2]$.

If E_1 and E_2 are events, then $E_1 \vee E_2$ denotes the disjunction of E_1 and E_2; that is, $E_1 \vee E_2$ is the event that *either* E_1 *or* E_2 occurs. It follows from the definition that $\Pr[E_1 \vee E_2] \geq \Pr[E_1]$. The *union bound* is often a very useful upper bound of this quantity.

PROPOSITION A.7 (Union Bound)

$$\Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2].$$

Repeated application of the union bound for any events $E_1, \ldots, E_k$ gives

$$\Pr\left[\bigvee_{i=1}^{k} E_i\right] \leq \sum_{i=1}^{k} \Pr[E_i].$$

The *conditional probability of* E_1 *given* E_2, denoted $\Pr[E_1 \mid E_2]$, is defined as

$$\Pr[E_1 \mid E_2] \stackrel{\text{def}}{=} \frac{\Pr[E_1 \wedge E_2]}{\Pr[E_2]}$$

as long as $\Pr[E_2] \neq 0$. (If $\Pr[E_2] = 0$ then $\Pr[E_1 \mid E_2]$ is undefined.) This represents the probability that event E_1 occurs, given that event E_2 has occurred. It follows immediately from the definition that

$$\Pr[E_1 \wedge E_2] = \Pr[E_1 \mid E_2] \cdot \Pr[E_2]\,;$$

equality holds even if $\Pr[E_2] = 0$ as long as we interpret multiplication by zero on the right-hand side in the obvious way.

We can now easily derive Bayes' theorem.

THEOREM A.8 (Bayes' Theorem) *If* $\Pr[E_2] \neq 0$ *then*

$$\Pr[E_1 \mid E_2] = \frac{\Pr[E_2 \mid E_1] \cdot \Pr[E_1]}{\Pr[E_2]}.$$

PROOF This follows because

$$\Pr[E_1 \mid E_2] = \frac{\Pr[E_1 \wedge E_2]}{\Pr[E_2]} = \frac{\Pr[E_2 \wedge E_1]}{\Pr[E_2]} = \frac{\Pr[E_2 \mid E_1] \cdot \Pr[E_1]}{\Pr[E_2]}.$$

■

Let $E_1, \ldots, E_n$ be events such that $\Pr[E_1 \vee \cdots \vee E_n] = 1$ and $\Pr[E_i \wedge E_j] = 0$ for all $i \neq j$. That is, the $\{E_i\}$ *partition* the space of all possible events, so that with probability 1 exactly one of the events E_i occurs. Then for any F

$$\Pr[F] = \sum_{i=1}^{n} \Pr[F \wedge E_i].$$

A special case is when $n = 2$ and $E_2 = \bar{E}_1$, giving

$$\begin{aligned}\Pr[F] &= \Pr[F \wedge E_1] + \Pr[F \wedge \bar{E}_1] \\ &= \Pr[F \mid E_1] \cdot \Pr[E_1] + \Pr[F \mid \bar{E}_1] \cdot \Pr[\bar{E}_1].\end{aligned}$$

Taking $F = E_1 \vee E_2$, we get a tighter version of the union bound:

$$\begin{aligned}\Pr[E_1 \vee E_2] &= \Pr[E_1 \vee E_2 \mid E_1] \cdot \Pr[E_1] + \Pr[E_1 \vee E_2 \mid \bar{E}_1] \cdot \Pr[\bar{E}_1] \\ &\leq \Pr[E_1] + \Pr[E_2 \mid \bar{E}_1].\end{aligned}$$

Extending this to events $E_1, \ldots, E_n$ we obtain

PROPOSITION A.9

$$\Pr\left[\bigvee_{i=1}^{k} E_i\right] \leq \Pr[E_1] + \sum_{i=2}^{k} \Pr[E_i \mid \bar{E}_1 \wedge \cdots \wedge \bar{E}_{i-1}].$$

* Useful Probability Bounds

We review some terminology and state probability bounds that are standard, but may not be encountered in a basic discrete mathematics course. The material here is used only in Section 7.3.

A (discrete, real-valued) random variable X is a variable whose value is assigned probabilistically from some finite set S of real numbers. X is non-negative if it does not take negative values; it is a 0/1-random variable if $S = \{0, 1\}$. The 0/1-random variables $X_1, \ldots, X_k$ are *independent* if for all $b_1, \ldots, b_k$ it holds that $\Pr[X_1 = b_1 \wedge \cdots \wedge X_k = b_k] = \prod_{i=1}^{k} \Pr[X_i = b_i]$.

We let $\mathsf{Exp}[X]$ denote the expectation of a random variable X; if X takes values in a set S then $\mathsf{Exp}[X] \stackrel{\text{def}}{=} \sum_{s \in S} s \cdot \Pr[X = s]$. One of the most important facts is that expectation is *linear*; for random variables $X_1, \ldots, X_k$ (with arbitrary dependencies) we have $\mathsf{Exp}[\sum_i X_i] = \sum_i \mathsf{Exp}[X_i]$. If X_1, X_2 are independent, then $\mathsf{Exp}[X_i \cdot X_j] = \mathsf{Exp}[X_i] \cdot \mathsf{Exp}[X_j]$.

Markov's inequality is useful when little is known about X.

PROPOSITION A.10 (Markov's inequality) *Let X be a non-negative random variable and $v > 0$. Then $\Pr[X \geq v] \leq \mathsf{Exp}[X]/v$.*

PROOF Say X takes values in a set S. We have

$$\begin{aligned}
\mathsf{Exp}[X] &= \sum_{s \in S} s \cdot \Pr[X = s] \\
&\geq \sum_{x \in S,\ x < v} \Pr[X = s] \cdot 0 + \sum_{x \in S,\ x \geq v} v \cdot \Pr[X = s] \\
&= v \cdot \Pr[X \geq v].
\end{aligned}$$

■

The variance of X, denoted $\mathsf{Var}[X]$, measures how much X deviates from its expectation. We have $\mathsf{Var}[X] \stackrel{\text{def}}{=} \mathsf{Exp}[(X - \mathsf{Exp}[X])^2] = \mathsf{Exp}[X^2] - \mathsf{Exp}[X]^2$, and one can easily show that $\mathsf{Var}[aX + b] = a^2\mathsf{Var}[X]$. For a 0/1-random variable X_i, we have $\mathsf{Var}[X_i] \leq 1/4$ because in this case $\mathsf{Exp}[X_i] = \mathsf{Exp}[X_i^2]$ and so $\mathsf{Var}[X_i] = \mathsf{Exp}[X_i](1 - \mathsf{Exp}[X_i])$, which is maximized when $\mathsf{Exp}[X_i] = \frac{1}{2}$.

PROPOSITION A.11 (Chebyshev's inequality) *Let X be a random variable and $\delta > 0$. Then:*

$$\Pr[|X - \mathsf{Exp}[X]| \geq \delta] \leq \frac{\mathsf{Var}[X]}{\delta^2}.$$

PROOF Define the non-negative random variable $Y \stackrel{\text{def}}{=} (X - \mathsf{Exp}[X])^2$ and then apply Markov's inequality. So,

$$\begin{aligned}\Pr[|X - \mathsf{Exp}[X]| \geq \delta] &= \Pr[(X - \mathsf{Exp}[X])^2 \geq \delta^2] \\ &\leq \frac{\mathsf{Exp}[(X - \mathsf{Exp}[X])^2]}{\delta^2} \quad = \quad \frac{\mathsf{Var}[X]}{\delta^2}.\end{aligned}$$

■

The 0/1-random variables $X_1, \ldots, X_m$ are *pairwise independent* if for every $i \neq j$ and every $b_i, b_j \in \{0, 1\}$ it holds that

$$\Pr[X_i = b_i \ \wedge \ X_j = b_j] = \Pr[X_i = b_i] \cdot \Pr[X_j = b_j].$$

If $X_1, \ldots, X_m$ are pairwise independent then $\mathsf{Var}[\sum_{i=1}^m X_i] = \sum_{i=1}^m \mathsf{Var}[X_i]$. (This follows since $\mathsf{Exp}[X_i \cdot X_j] = \mathsf{Exp}[X_i] \cdot \mathsf{Exp}[X_j]$ when $i \neq j$, using pairwise independence.) An important corollary of Chebyshev's inequality follows.

COROLLARY A.12 *Let $X_1, \ldots, X_m$ be pairwise-independent random variables with the same expectation μ and variance σ^2. Then for every $\delta > 0$,*

$$\Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \mu\right| \geq \delta\right] \leq \frac{\sigma^2}{\delta^2 m}.$$

PROOF By linearity of expectation, $\mathsf{Exp}[\sum_{i=1}^m X_i/m] = \mu$. Applying Chebyshev's inequality to the random variable $\sum_{i=1}^m X_i/m$, we have

$$\Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \mu\right| \geq \delta\right] \leq \frac{\mathsf{Var}\left[\frac{1}{m} \cdot \sum_{i=1}^m X_i\right]}{\delta^2}.$$

Using pairwise independence, it follows that

$$\mathsf{Var}\left[\frac{1}{m} \cdot \sum_{i=1}^m X_i\right] = \frac{1}{m^2} \sum_{i=1}^m \mathsf{Var}[X_i] = \frac{1}{m^2} \sum_{i=1}^m \sigma^2 = \frac{\sigma^2}{m}.$$

The inequality is obtained by combining the above two equations. ■

Say 0/1-random variables $X_1, \ldots, X_m$ each provides an estimate of some fixed (unknown) bit b. That is, $\Pr[X_i = b] \geq 1/2 + \varepsilon$ for all i, where $\varepsilon > 0$.

We can estimate b by looking at the value of X_1; this estimate will be correct with probability $\Pr[X_1 = b]$. A better estimate can be obtained by looking at the values of $X_1, \ldots, X_m$ and taking the value that occurs the majority of the time. We analyze how well this does when $X_1, \ldots, X_m$ are pairwise independent.

PROPOSITION A.13 *Fix $\varepsilon > 0$ and $b \in \{0,1\}$, and let $\{X_i\}$ be pairwise-independent, 0/1-random variables for which $\Pr[X_i = b] \geq \frac{1}{2} + \varepsilon$ for all i. Consider the process in which m values $X_1, \ldots, X_m$ are recorded and X is set to the value that occurs a strict majority of the time. Then*

$$\Pr[X \neq b] \leq \frac{1}{4 \cdot \varepsilon^2 \cdot m}.$$

PROOF Assume $b = 1$; by symmetry, this is without loss of generality. Then $\mathsf{Exp}[X_i] = \frac{1}{2} + \varepsilon$. Let X denote the strict majority of the $\{X_i\}$ as in the proposition, and note that $X \neq 1$ if and only if $\sum_{i=1}^m X_i \leq m/2$. So

$$\begin{aligned}
\Pr[X \neq 1] &= \Pr\left[\sum_{i=1}^m X_i \leq m/2\right] \\
&= \Pr\left[\frac{\sum_{i=1}^m X_i}{m} - \frac{1}{2} \leq 0\right] \\
&= \Pr\left[\frac{\sum_{i=1}^m X_i}{m} - \left(\frac{1}{2} + \varepsilon\right) \leq -\varepsilon\right] \\
&\leq \Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \left(\frac{1}{2} + \varepsilon\right)\right| \geq \varepsilon\right].
\end{aligned}$$

Since $\mathsf{Var}[X_i] \leq 1/4$ for all i, applying the previous corollary shows that $\Pr[X \neq 1] \leq \frac{1}{4\varepsilon^2 m}$ as claimed. ■

A better bound is obtained if the $\{X_i\}$ are independent:

PROPOSITION A.14 (Chernoff bound) *Fix $\varepsilon > 0$ and $b \in \{0,1\}$, and let $\{X_i\}$ be independent 0/1-random variables with $\Pr[X_i = b] = \frac{1}{2} + \varepsilon$ for all i. The probability that their majority value is not b is at most $e^{-\varepsilon^2 m/2}$.*

A.4 The "Birthday" Problem

If we choose q elements $y_1, \ldots, y_q$ uniformly from a set of size N, what is the probability that there exist distinct i, j with $y_i = y_j$? We refer to the stated

event as a *collision*, and denote the probability of this event by $\mathsf{coll}(q, N)$. This problem is related to the so-called *birthday problem*, which asks what size group of people we need such that with probability 1/2 some pair of people in the group share a birthday. To see the relationship, let y_i denote the birthday of the ith person in the group. If there are q people in the group then we have q values $y_1, \ldots, y_q$ chosen uniformly from $\{1, \ldots, 365\}$, making the simplifying assumption that birthdays are uniformly and independently distributed among the 365 days of a non-leap year. Furthermore, matching birthdays correspond to a collision, i.e., distinct i, j with $y_i = y_j$. So the desired solution to the birthday problem is given by the minimal (integer) value of q for which $\mathsf{coll}(q, 365) \geq 1/2$. (The answer may surprise you—taking $q = 23$ people suffices!)

In this section, we prove lower and upper bounds on $\mathsf{coll}(q, N)$. Taken together and summarized at a high level, they show that if $q < \sqrt{N}$ then the probability of a collision is $\Theta(q^2/N)$; alternately, for $q = \Theta(\sqrt{N})$ the probability of a collision is constant.

An upper bound for the collision probability is easy to obtain.

LEMMA A.15 *Fix a positive integer N, and say q elements $y_1, \ldots, y_q$ are chosen uniformly and independently at random from a set of size N. Then the probability that there exist distinct i, j with $y_i = y_j$ is at most $\frac{q^2}{2N}$. That is,*

$$\mathsf{coll}(q, N) \leq \frac{q^2}{2N}.$$

PROOF The proof is a simple application of the union bound (Proposition A.7). Recall that a *collision* means that there exist distinct i, j with $y_i = y_j$. Let Coll denote the event of a collision, and let $\mathsf{Coll}_{i,j}$ denote the event that $y_i = y_j$. It is immediate that $\Pr[\mathsf{Coll}_{i,j}] = 1/N$ for any distinct i, j. Furthermore, $\mathsf{Coll} = \bigvee_{i \neq j} \mathsf{Coll}_{i,j}$ and so repeated application of the union bound implies that

$$\begin{aligned} \Pr[\mathsf{Coll}] &= \Pr\left[\bigvee_{i \neq j} \mathsf{Coll}_{i,j}\right] \\ &\leq \sum_{i \neq j} \Pr[\mathsf{Coll}_{i,j}] = \binom{q}{2} \cdot \frac{1}{N} \leq \frac{q^2}{2N}. \end{aligned}$$

■

LEMMA A.16 *Fix a positive integer N, and say $q \leq \sqrt{2N}$ elements $y_1, \ldots, y_q$ are chosen uniformly and independently at random from a set of size N. Then the probability that there exist distinct i, j with $y_i = y_j$ is at least $\frac{q(q-1)}{4N}$. In fact,*

$$\mathsf{coll}(q, N) \geq 1 - e^{-q(q-1)/2N} \geq \frac{q(q-1)}{4N}.$$

PROOF Recall that a *collision* means that there exist distinct i, j with $y_i = y_j$. Let Coll denote this event. Let NoColl_i be the event that there is *no* collision among $y_1, \ldots, y_i$; that is, $y_j \neq y_k$ for all $j < k \leq i$. Then $\mathsf{NoColl}_q = \overline{\mathsf{Coll}}$ is the event that there is no collision at all.

If NoColl_q occurs then NoColl_i must also have occurred for all $i \leq q$. Thus,

$$\Pr[\mathsf{NoColl}_q] = \Pr[\mathsf{NoColl}_1] \cdot \Pr[\mathsf{NoColl}_2 \mid \mathsf{NoColl}_1] \cdots \Pr[\mathsf{NoColl}_q \mid \mathsf{NoColl}_{q-1}].$$

Now, $\Pr[\mathsf{NoColl}_1] = 1$ since y_1 cannot collide with itself. Furthermore, if event NoColl_i occurs then $\{y_1, \ldots, y_i\}$ contains i distinct values; so, the probability that y_{i+1} collides with one of these values is $\frac{i}{N}$ and hence the probability that y_{i+1} does *not* collide with any of these values is $1 - \frac{i}{N}$. This means

$$\Pr[\mathsf{NoColl}_{i+1} \mid \mathsf{NoColl}_i] = 1 - \frac{i}{N},$$

and so

$$\Pr[\mathsf{NoColl}_q] = \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right).$$

Since $i/N < 1$ for all i, we have $1 - \frac{i}{N} \leq e^{-i/N}$ (by Inequality A.3) and so

$$\Pr[\mathsf{NoColl}_q] \leq \prod_{i=1}^{q-1} e^{-i/N} = e^{-\sum_{i=1}^{q-1}(i/N)} = e^{-q(q-1)/2N}.$$

We conclude that

$$\Pr[\mathsf{Coll}] = 1 - \Pr[\mathsf{NoColl}_q] \geq 1 - e^{-q(q-1)/2N} \geq \frac{q(q-1)}{4N},$$

using Inequality A.4 in the last step (note that $q(q-1)/2N < 1$). ■

A.5 *Finite Fields

We use finite fields only sparingly in the book, but we include a definition and some basic facts for completeness. Further details can be found in any textbook on abstract algebra.

DEFINITION A.17 *A* (finite) field *is a (finite) set* $\mathbb{F}$ *along with two binary operations* $+, \cdot$ *for which the following hold:*

- $\mathbb{F}$ *is an abelian group with respect to the operation '*$+$*.' We let* 0 *denote the identity element of this group.*
- $\mathbb{F} \setminus \{0\}$ *is an abelian group with respect to the operation '*$\cdot$*.' We let* 1 *denote the identity element of this group.*

  As usual, we often write ab *in place of* $a \cdot b$*.*
- **(Distributivity:)** *For all* $a, b, c \in \mathbb{F}$*, we have* $a \cdot (b + c) = ab + ac$*.*

The *additive inverse* of $a \in \mathbb{F}$, denoted by $-a$, is the unique element satisfying $a + (-a) = 0$; we write $b - a$ in place of $b + (-a)$. The *multiplicative inverse* of $a \in \mathbb{F} \setminus \{0\}$, denoted by a^{-1}, is the unique element satisfying $aa^{-1} = 1$; we often write b/a in place of ba^{-1}.

Example A.18
It follows from the results of Section 8.1.4 that for any prime p the set $\{0, \ldots, p-1\}$ is a finite field with respect to addition and multiplication modulo p. We denote this field by $\mathbb{F}_p$. ♢

Finite fields have a rich theory. For our purposes, we need only a few basic facts. The *order* of $\mathbb{F}$ is the number of elements in $\mathbb{F}$ (assuming it is finite). Recall also that q is a prime power if $q = p^r$ for some prime p and integer $r \geq 1$.

THEOREM A.19 *If* $\mathbb{F}$ *is a finite field, then the order of* $\mathbb{F}$ *is a prime power. Conversely, for every prime power* q *there is a finite field of order* q*, which is moreover the unique such field (up to relabeling of the elements).*

For $q = p^r$ with p prime, we let $\mathbb{F}_q$ denote the (unique) field of order q. We call p the *characteristic* of $\mathbb{F}_q$. The preceding theorem tells us that the characteristic of any finite field is prime.

As in the case of groups, if n is a positive integer and $a \in \mathbb{F}$ then

$$n \cdot a \stackrel{\text{def}}{=} \underbrace{a + \cdots + a}_{n \text{ times}} \quad \text{and} \quad a^n \stackrel{\text{def}}{=} \underbrace{a \cdots a}_{n \text{ times}}.$$

The notation is extended for $n \leq 0$ in the natural way.

THEOREM A.20 *Let* $\mathbb{F}_q$ *be a finite field of characteristic* p*. Then for all* $a \in \mathbb{F}_q$ *we have* $p \cdot a = 0$*.*

Let $q = p^r$ with p prime. For $r = 1$, we have seen in Example A.18 that $\mathbb{F}_q = \mathbb{F}_p$ can be taken to be the set $\{0, \ldots, p-1\}$ under addition and multiplication

modulo p. We caution, however, that for $r > 1$ the set $\{0, \ldots, q-1\}$ is *not* a field under addition and multiplication modulo q. For example, if we take $q = 3^2 = 9$ then the element 3 does not have a multiplicative inverse modulo 9.

Finite fields of characteristic p can be represented using polynomials over $\mathbb{F}_p$. We give an example to demonstrate the flavor of the construction, without discussing why the construction works or describing the general case. We construct the field $\mathbb{F}_4$ by working with polynomials over $\mathbb{F}_2$. Fix the polynomial $r(x) = x^2+x+1$, and note that $r(x)$ has no roots over $\mathbb{F}_2$ since $r(0) = r(1) = 1$ (recall that we are working in $\mathbb{F}_2$, which means that all operations are carried out modulo 2). In the same way that we can introduce the imaginary number i to be a root of $x^2 + 1$ over the reals, we can introduce a value ω to be a root of $r(x)$ over $\mathbb{F}_2$; that is, $\omega^2 = -\omega - 1$. We then define $\mathbb{F}_4$ to be the set of all degree-1 polynomials in ω over $\mathbb{F}_2$; that is, $\mathbb{F}_4 = \{0, 1, \omega, \omega + 1\}$. Addition in $\mathbb{F}_4$ will just be regular polynomial addition, remembering that operations on the coefficients are done in $\mathbb{F}_2$ (that is, modulo 2). Multiplication in $\mathbb{F}_4$ will be polynomial multiplication (again, with operations on the coefficients carried out modulo 2) followed by the substitution $\omega^2 = -\omega - 1$; this also ensures that the result lies in $\mathbb{F}_4$. So, for example,

$$\omega + (\omega + 1) = 2\omega + 1 = 1$$

and

$$(\omega + 1) \cdot (\omega + 1) = \omega^2 + 2\omega + 1 = (-\omega - 1) + 1 = -\omega = \omega.$$

Although not obvious, one can check that this is a field; the only difficult condition to verify is that every nonzero element has a multiplicative inverse.

We need only one other result.

THEOREM A.21 *Let $\mathbb{F}_q$ be a finite field of order q. Then the abelian group $\mathbb{F}_q \setminus \{0\}$ with respect to '$\cdot$' is a* **cyclic** *group of order $q - 1$.*

Appendix B

Basic Algorithmic Number Theory

For the cryptographic constructions given in this book to be efficient (i.e., to run in time polynomial in the lengths of their inputs), it is necessary for these constructions to utilize efficient (that is, polynomial-time) algorithms for performing basic number-theoretic operations. Although in some cases there exist "trivial" algorithms that would work, it is still worthwhile to carefully consider their efficiency since for cryptographic applications it is not uncommon to use integers that are *thousands* of bits long. In other cases obtaining any polynomial-time algorithm requires a bit of cleverness, and an analysis of their performance may rely on non-trivial group-theoretic results.

In Appendix B.1 we describe basic algorithms for integer arithmetic. Here we cover the familiar algorithms for addition, subtraction, etc., as well as the Euclidean algorithm for computing greatest common divisors. We also discuss the extended Euclidean algorithm, assuming there that the reader has covered the material in Section 8.1.1.

In Appendix B.2 we show various algorithms for modular arithmetic. In addition to a brief discussion of basic modular operations (i.e., modular reduction, addition, multiplication, and inversion), we also describe *Montgomery multiplication*, which can greatly simplify (and speed up) implementations of modular arithmetic. We then discuss algorithms for problems that are less common outside the field of cryptography: exponentiation modulo N (as well as in arbitrary groups) and choosing a uniform element of $\mathbb{Z}_N$ or $\mathbb{Z}_N^*$ (or in an arbitrary group). This section assumes familiarity with the basic group theory covered in Section 8.1.

The material above is used implicitly throughout the second half of the book, although it is not absolutely necessary to read this material in order to follow the book. (In particular, the reader willing to accept the results of this Appendix without proof can simply read the summary of those results in the theorems below.) Appendix B.3, which discusses finding generators in cyclic groups (when the factorization of the group order is known) and assumes the results of Section 8.3.1, contains material that is hardly used at all; it is included for completeness and reference.

Since our goal is only to establish that certain problems can be solved in polynomial time, we have opted for *simplicity* rather than *efficiency* in our selection of algorithms and their descriptions (as long as the algorithms run in polynomial time). For this reason, we generally will not be interested in

the exact running times of the algorithms we present beyond establishing that they indeed run in polynomial time. The reader who is seriously interested in implementing these algorithms is forewarned to look at other sources for more efficient alternatives as well as various techniques for speeding up the necessary computations.

The results in this Appendix are summarized by the theorems that follow. Throughout, we assume that any integer a provided as input is written using exactly $\|a\|$ bits; i.e., the high-order bit is 1. In Appendix B.1 we show:

THEOREM B.1 (Integer operations) *Given integers a and b, it is possible to perform the following operations in time polynomial in $\|a\|$ and $\|b\|$:*

1. *Computing the sum $a + b$ and the difference $a - b$;*
2. *Computing the product ab;*
3. *Computing positive integers q and $r < b$ such that $a = qb + r$ (i.e., computing* division with remainder*);*
4. *Computing the greatest common divisor of a and b, $\gcd(a, b)$;*
5. *Computing integers X, Y with $Xa + Yb = \gcd(a, b)$.*

The following results are proved in Appendix B.2:

THEOREM B.2 (Modular operations) *Given integers $N > 1$, a, and b, it is possible to perform the following operations in time polynomial in $\|a\|$, $\|b\|$, and $\|N\|$:*

1. *Computing the modular reduction $[a \bmod N]$;*
2. *Computing the sum $[(a+b) \bmod N]$, the difference $[(a-b) \bmod N]$, and the product $[ab \bmod N]$;*
3. *Determining whether a is invertible modulo N;*
4. *Computing the multiplicative inverse $[a^{-1} \bmod N]$, assuming a is invertible modulo N;*
5. *Computing the exponentiation $[a^b \bmod N]$.*

The following generalizes Theorem B.2(5) to arbitrary groups:

THEOREM B.3 (Group exponentiation) *Let $\mathbb{G}$ be a group, written multiplicatively. Let g be an element of the group and let b be a non-negative integer. Then g^b can be computed using $\mathsf{poly}(\|b\|)$ group operations.*

THEOREM B.4 (Choosing uniform elements) *There exists a randomized algorithm with the following properties: on input N,*

- *The algorithm runs in time polynomial in $\|N\|$;*
- *The algorithm outputs* fail *with probability negligible in $\|N\|$; and*
- *Conditioned on not outputting* fail*, the algorithm outputs a uniformly distributed element of $\mathbb{Z}_N$.*

An algorithm with analogous properties exists for $\mathbb{Z}_N^*$ as well.

Since the probability that either algorithm referenced in the above theorem outputs fail is negligible, we ignore this possibility (and instead leave it implicit). In Appendix B.2 we also discuss generalizations of the above to the case of selecting a uniform element from any finite group (subject to certain requirements on the representation of group elements).

A proof of the following is in Appendix B.3:

THEOREM B.5 (Testing and finding generators) *Let $\mathbb{G}$ be a cyclic group of order q, and assume that the group operation and selection of a uniform group element can be carried out in unit time.*

1. *There is an algorithm that on input q, the prime factorization of q, and an element $g \in \mathbb{G}$, runs in $\mathsf{poly}(\|q\|)$ time and decides whether g is a generator of $\mathbb{G}$.*
2. *There is a randomized algorithm that on input q and the prime factorization of q, runs in $\mathsf{poly}(\|q\|)$ time and outputs a generator of $\mathbb{G}$ except with probability negligible in $\|q\|$. Conditioned on the output being a generator, it is uniformly distributed among the generators of $\mathbb{G}$.*

B.1 Integer Arithmetic

B.1.1 Basic Operations

We begin our exploration of algorithmic number theory with a discussion of integer addition/subtraction, multiplication, and division with remainder. A little thought shows that all these operations can be carried out in time polynomial in the input length using the standard "grade-school" algorithms for these problems. For example, addition of two positive integers a and b with $a > b$ can be done in time linear in $\|a\|$ by stepping one-by-one through the bits of a and b, starting with the low-order bits, and computing the corresponding output bit and a "carry bit" at each step. (Details are omitted.) Multiplication of two n-bit integers a and b, to take another example, can

be done by first generating a list of n integers of length at most $2n$ (each of which is equal to $a \cdot 2^{i-1} \cdot b_i$, where b_i is the ith bit of b) and then adding these n integers together to obtain the final result. (Division with remainder is trickier to implement efficiently, but can also be done.)

Although these grade-school algorithms suffice to demonstrate that the aforementioned problems can be solved in polynomial time, it is interesting to note that these algorithms are in some cases not the best ones available. As an example, the simple algorithm for multiplication given above multiplies two n-bit numbers in time $\mathcal{O}(n^2)$, but there exists a better algorithm running in time $\mathcal{O}(n^{\log_2 3})$ (and even that is not the best possible). While the difference is insignificant for numbers of the size we encounter daily, it becomes noticeable when the numbers are large. In cryptographic applications it is not uncommon to use integers that are thousands of bits long (i.e., $n > 1000$), and a judicious choice of which algorithms to use then becomes critical.

B.1.2 The Euclidean and Extended Euclidean Algorithms

Recall from Section 8.1 that $\gcd(a, b)$, the *greatest common divisor* of two integers a and b, is the largest integer d that divides both a and b. We state an easy proposition regarding the greatest common divisor, and then show how this leads to an efficient algorithm for computing gcd's.

PROPOSITION B.6 *Let $a, b > 1$ with $b \nmid a$. Then*

$$\gcd(a, b) = \gcd(b, [a \bmod b]).$$

PROOF If $b > a$ the stated claim is immediate. So assume $a > b$. Write $a = qb + r$ for q, r positive integers and $r < b$ (cf. Proposition 8.1); note that $r > 0$ because $b \nmid a$. Since $r = [a \bmod b]$, we prove the proposition by showing that $\gcd(a, b) = \gcd(b, r)$.

Let $d = \gcd(a, b)$. Then d divides both a and b, and so d also divides $r = a - qb$. By definition of the greatest common divisor, we thus have $\gcd(b, r) \geq d = \gcd(a, b)$.

Let $d' = \gcd(b, r)$. Then d' divides both b and r, and so d' also divides $a = qb + r$. By definition of the greatest common divisor, we thus have $\gcd(a, b) \geq d' = \gcd(b, r)$.

Since $d \geq d'$ and $d' \geq d$, we conclude that $d = d'$. ■

The above suggests the recursive *Euclidean algorithm* (Algorithm B.7) for computing the greatest common divisor $\gcd(a, b)$ of two integers a and b. Correctness of the algorithm follows readily from Proposition B.6. As for its running time, we show below that on input (a, b) the algorithm makes fewer than $2 \cdot \|b\|$ recursive calls. Since checking whether b divides a and computing

ALGORITHM B.7
The Euclidean algorithm GCD

Input: Integers a, b with $a \geq b > 0$
Output: The greatest common divisor of a and b

if b divides a
  return b
else return $\mathsf{GCD}(b, [a \bmod b])$

$[a \bmod b]$ can both be done in time polynomial in $\|a\|$ and $\|b\|$, this implies that the entire algorithm runs in polynomial time.

PROPOSITION B.8 *Consider an execution of* $\mathsf{GCD}(a_0, b_0)$*, and let* a_i, b_i *(for* $i = 1, \ldots, \ell$*) denote the arguments to the* i*th recursive call of* GCD*. Then* $b_{i+2} \leq b_i/2$ *for* $0 \leq i \leq \ell - 2$*.*

PROOF First note that for any $a > b$ we have $[a \bmod b] < a/2$. To see this, consider the two cases: If $b \leq a/2$ then $[a \bmod b] < b \leq a/2$ is immediate. On the other hand, if $b > a/2$ then $[a \bmod b] = a - b < a/2$.

Now fix arbitrary i with $0 \leq i \leq \ell - 2$. Then $b_{i+2} = [a_{i+1} \bmod b_{i+1}] < a_{i+1}/2 = b_i/2$. ∎

COROLLARY B.9 *In an execution of algorithm* $\mathsf{GCD}(a, b)$*, there are at most* $2\|b\| - 2$ *recursive calls to* GCD*.*

PROOF Let a_i, b_i (for $i = 1, \ldots, \ell$) denote the arguments to the ith recursive call of GCD. The $\{b_i\}$ are always greater than zero, and the algorithm makes no further recursive calls if it ever happens that $b_i = 1$ (since then $b_i \mid a_i$). The previous proposition indicates that the $\{b_i\}$ decrease by a multiplicative factor of (at least) 2 in every two iterations. It follows that the number of recursive calls to GCD is at most $2 \cdot (\|b\| - 1)$. ∎

The Extended Euclidean Algorithm

By Proposition 8.2, we know that for positive integers a, b there exist integers X, Y with $Xa + Yb = \gcd(a, b)$. A simple modification of the Euclidean algorithm, called the *extended Euclidean algorithm*, can be used to find X, Y in addition to computing $\gcd(a, b)$; see Algorithm B.10. You are asked to show correctness of the extended Euclidean algorithm in Exercise B.1, and to prove that the algorithm runs in polynomial time in Exercise B.2.

ALGORITHM B.10
The extended Euclidean algorithm eGCD

Input: Integers a, b with $a \geq b > 0$
Output: (d, X, Y) with $d = \gcd(a, b)$ and $Xa + Yb = d$

if b divides a
  return $(b, 0, 1)$
else
  Compute integers q, r with $a = qb + r$ and $0 < r < b$
  $(d, X, Y) := \mathsf{eGCD}(b, r)$ // note that $Xb + Yr = d$
  return $(d, Y, X - Yq)$

B.2 Modular Arithmetic

We now turn our attention to basic arithmetic operations modulo $N > 1$. We will use $\mathbb{Z}_N$ to refer both to the set $\{0, \ldots, N-1\}$ as well as to the group that results by considering addition modulo N among the elements of this set.

B.2.1 Basic Operations

Efficient algorithms for the basic arithmetic operations over the integers immediately imply efficient algorithms for the corresponding arithmetic operations modulo N. For example, computing the modular reduction $[a \bmod N]$ can be done in time polynomial in $\|a\|$ and $\|N\|$ by computing division-with-remainder over the integers. Next consider modular operations on two elements $a, b \in \mathbb{Z}_N$ where $\|N\| = n$. (Note that a, b have length at most n. Actually, it is convenient to simply assume that all elements of $\mathbb{Z}_N$ have length *exactly* n, padding to the left with 0s if necessary.) Addition of a and b modulo N can be done by first computing $a+b$, an integer of length at most $n+1$, and then reducing this intermediate result modulo N. Similarly, multiplication modulo N can be performed by first computing the integer ab of length at most $2n$ and then reducing the result modulo N. Since addition, multiplication, and division-with-remainder can all be done in polynomial time, these give polynomial-time algorithms for addition and multiplication modulo N.

B.2.2 Computing Modular Inverses

Our discussion thus far has shown how to add, subtract, and multiply modulo N. One operation we are missing is "division" or, equivalently, computing multiplicative inverses modulo N. Recall from Section 8.1.2 that the multiplicative inverse (modulo N) of an element $a \in \mathbb{Z}_N$ is an element $a^{-1} \in \mathbb{Z}_N$ such that $a \cdot a^{-1} = 1 \bmod N$. Proposition 8.7 shows that a has an inverse if and only if $\gcd(a, N) = 1$, i.e., if and only if $a \in \mathbb{Z}_N^*$. Thus, using the

Euclidean algorithm we can easily determine whether a given element a has a multiplicative inverse modulo N.

Given N and $a \in \mathbb{Z}_N$ with $\gcd(a, N) = 1$, Proposition 8.2 tells us that there exist integers X, Y with $Xa + YN = 1$. This means that $[X \bmod N]$ is the multiplicative inverse of a. Integers X and Y satisfying $Xa + YN = 1$ can be found efficiently using the extended Euclidean algorithm eGCD shown in Section B.1.2. This leads to the following polynomial-time algorithm for computing multiplicative inverses:

ALGORITHM B.11
Computing modular inverses

Input: Modulus N; element a
Output: $[a^{-1} \bmod N]$ (if it exists)

$(d, X, Y) := \mathsf{eGCD}(a, N)$ // note that $Xa + YN = \gcd(a, N)$
if $d \neq 1$ **return** "a is not invertible modulo N"
else return $[X \bmod N]$

B.2.3 Modular Exponentiation

A more challenging task is that of *exponentiation* modulo N, that is, computing $[a^b \bmod N]$ for base $a \in \mathbb{Z}_N$ and integer exponent $b > 0$. (When $b = 0$ the problem is easy. When $b < 0$ and $a \in \mathbb{Z}_N^*$ then $a^b = (a^{-1})^{-b} \bmod N$ and the problem is reduced to the case of exponentiation with a positive exponent given that we can compute inverses, as discussed in the previous section.) Notice that the basic approach used in the case of addition and multiplication (i.e., computing the integer a^b and then reducing this intermediate result modulo N) does *not* work here: the integer a^b has length $\|a^b\| = \Theta(\log a^b) = \Theta(b \cdot \|a\|)$, and so even storing the intermediate result a^b would require time exponential in $\|b\| = \Theta(\log b)$.

We can address this problem by reducing modulo N at all intermediate steps of the computation, rather than only reducing modulo N at the end. This has the effect of keeping the intermediate results "small" throughout the computation. Even with this important initial observation, it is still non-trivial to design a polynomial-time algorithm for modular exponentiation. Consider the naïve approach of Algorithm B.12, which simply performs b multiplications by a. This still runs in time that is *exponential* in $\|b\|$.

This naïve algorithm can be viewed as relying on the following recurrence:

$$[a^b \bmod N] = [a \cdot a^{b-1} \bmod N] = [a \cdot a \cdot a^{b-2} \bmod N] = \cdots$$

Any algorithm based on this relationship will require $\Theta(b)$ time. We can do

ALGORITHM B.12
A naïve algorithm for modular exponentiation

Input: Modulus N; base $a \in \mathbb{Z}_N$; integer exponent $b > 0$
Output: $[a^b \bmod N]$

$x := 1$
for $i = 1$ to b:
  $x := [x \cdot a \bmod N]$
return x

better by relying on the following recurrence:

$$[a^b \bmod N] = \begin{cases} \left[\left(a^{\frac{b}{2}}\right)^2 \bmod N\right] & \text{when } b \text{ is even} \\ \left[a \cdot \left(a^{\frac{b-1}{2}}\right)^2 \bmod N\right] & \text{when } b \text{ is odd.} \end{cases}$$

Doing so leads to an algorithm—called, for obvious reasons, "square-and-multiply" (or "repeated squaring")—that requires only $\mathcal{O}(\log b) = \mathcal{O}(\|b\|)$ modular squarings/multiplications; see Algorithm B.13. In this algorithm, the length of b decreases by 1 in each iteration; it follows that the number of iterations is $\|b\|$, and so the overall algorithm runs in time polynomial in $\|a\|$, $\|b\|$, and $\|N\|$. More precisely, the number of modular squarings is exactly $\|b\|$, and the number of additional modular multiplications is exactly the Hamming weight of b (i.e., the number of 1s in the binary representation of b). This explains the preference, discussed in Section 8.2.4, for choosing the public RSA exponent e to have small length/Hamming weight.

ALGORITHM B.13
Algorithm ModExp for efficient modular exponentiation

Input: Modulus N; base $a \in \mathbb{Z}_N$; integer exponent $b > 0$
Output: $[a^b \bmod N]$

$x := a$
$t := 1$
// maintain the invariant that the answer is $[t \cdot x^b \bmod N]$
while $b > 0$ do:
  if b is odd
    $t := [t \cdot x \bmod N]$, $b := b - 1$
  $x := [x^2 \bmod N]$, $b := b/2$
return t

Fix a and N and consider the modular exponentiation function given by $f_{a,N}(b) = [a^b \bmod N]$. We have just seen that computing $f_{a,N}$ is easy. In contrast, computing the *inverse* of this function—that is, computing b given a, N, and $[a^b \bmod N]$—is believed to be hard for appropriate choice of a

and N. Inverting this function requires solving the *discrete-logarithm problem*, something we discuss in detail in Section 8.3.2.

Using precomputation. If the base a is known in advance, and there is a bound on the length of the exponent b, then one can use precomputation and a small amount of memory to speed up computation of $[a^b \bmod N]$. Say $\|b\| \leq n$. Then we precompute and store the n values

$$x_0 := a, \quad x_1 := [a^2 \bmod N], \quad \ldots, \quad x_{n-1} := [a^{2^{n-1}} \bmod N].$$

Given exponent b with binary representation $b_{n-1} \cdots b_0$ (written from most to least significant bit), we then have

$$a^b = a^{\sum_{i=0}^{n-1} 2^i \cdot b_i} = \prod_{i=0}^{n-1} x_i^{b_i} \bmod N.$$

Since $b_i \in \{0, 1\}$, the number of multiplications needed to compute the result is exactly one less than the Hamming weight of b.

Exponentiation in Arbitrary Groups

The efficient modular exponentiation algorithm given above carries over in a straightforward way to enable efficient exponentiation in any group, as long as the underlying group operation can be performed efficiently. Specifically, if $\mathbb{G}$ is a group and g is an element of $\mathbb{G}$, then g^b can be computed using at most $2 \cdot \|b\|$ applications of the underlying group operation. Precomputation could also be used, exactly as described above.

If the order q of $\mathbb{G}$ is known, then $a^b = a^{[b \bmod q]}$ (cf. Proposition 8.52) and this can be used to speed up the computation by reducing b modulo q first.

Considering the (additive) group $\mathbb{Z}_N$, the group exponentiation algorithm just described gives a method for computing the "exponentiation"

$$[b \cdot g \bmod N] \stackrel{\text{def}}{=} [\underbrace{g + \cdots + g}_{b \text{ times}} \bmod N]$$

that differs from the method discussed earlier that relies on standard integer multiplication followed by a modular reduction. In comparing the two approaches to solving the same problem, note that the original algorithm uses specific information about $\mathbb{Z}_N$; in particular, it (essentially) treats the "exponent" b as an element of $\mathbb{Z}_N$ (possibly by reducing b modulo N first). In contrast, the "square-and-multiply" algorithm just presented treats $\mathbb{Z}_N$ only as an abstract group. (Of course, the group operation of addition modulo N relies on the specifics of $\mathbb{Z}_N$.) The point of this discussion is merely to illustrate that some group algorithms are *generic* (i.e., they apply equally well to all groups) while some group algorithms rely on specific properties of a *particular* group or class of groups. We saw some examples of this phenomenon in Chapter 9.

B.2.4 *Montgomery Multiplication

Although division over the integers (and hence modular reduction) can be done in polynomial time, algorithms for integer division are slow in comparison to, say, algorithms for integer multiplication. Montgomery multiplication provides a way to perform modular multiplication *without* carrying out any expensive modular reductions. Since pre- and postprocessing is required, the method is advantageous only when several modular multiplications will be done in sequence as, e.g., when computing a modular exponentiation.

Fix an odd modulus N with respect to which modular operations are to be done. Let $R > N$ be a power of two, say $R = 2^w$, and note that $\gcd(R, N) = 1$. The key property we will exploit is that division by R is fast: the quotient of x upon division by R is obtained by simply shifting x to the right w positions, and $[x \bmod R]$ is just the w least-significant bits of x.

Define the *Montgomery representation* of $x \in \mathbb{Z}_N^*$ by $\bar{x} \stackrel{\text{def}}{=} [xR \bmod N]$. Montgomery multiplication of $\bar{x}, \bar{y} \in \mathbb{Z}_N^*$ is defined as

$$\mathsf{Mont}(\bar{x}, \bar{y}) \stackrel{\text{def}}{=} [\bar{x}\bar{y}R^{-1} \bmod N].$$

(We show below how this can be computed without any expensive modular reductions.) Note that

$$\mathsf{Mont}(\bar{x}, \bar{y}) = \bar{x}\bar{y}R^{-1} = (xR)(yR)R^{-1} = (xy)R = \overline{xy} \bmod N.$$

This means we can multiply several values in $\mathbb{Z}_N$ by (1) converting to the Montgomery representation, (2) carrying out all multiplications using Montgomery multiplication to obtain the final result, and then (3) converting the result from Montgomery representation back to the standard representation.

Let $\alpha \stackrel{\text{def}}{=} [-N^{-1} \bmod R]$, a value which can be precomputed. (Computation of α, and conversion to/from Montgomery representation, can also be done without any expensive modular reductions; details are beyond our scope.) To compute $c \stackrel{\text{def}}{=} \mathsf{Mont}(x, y)$ without any expensive modular reductions do:

1. Let $z := x \cdot y$ (over the integers).
2. Set $c' := (z + [z\alpha \bmod R] \cdot N)/R$.
3. If $c' < N$ then set $c := c'$; else set $c := c' - N$.

To see that this works, we first need to verify that step 2 is well-defined, namely, that the numerator is divisible by R. This follows because

$$z + [z\alpha \bmod R] \cdot N = z + z\alpha N = z - zN^{-1}N = 0 \bmod R.$$

Next, note that $c' = z/R \bmod N$ after step 2; moreover, since $z < N^2 < RN$ we have $0 < c' < (z + RN)/R < 2RN/R = 2N$. But then $[c' \bmod N] = c'$ if $c' < N$, and $[c' \bmod N] = c' - N$ if $c' > N$. We conclude that

$$c = [c' \bmod N] = [z/R \bmod N] = [xyR^{-1} \bmod N],$$

as desired.

B.2.5 Choosing a Uniform Group Element

For cryptographic applications, it is often necessary to choose a uniform element of a group $\mathbb{G}$. We first treat the problem in an abstract setting, and then focus specifically on the cases of $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$.

Note that if $\mathbb{G}$ is a cyclic group of order q, and a generator $g \in \mathbb{G}$ is known, then choosing a uniform element $h \in \mathbb{G}$ reduces to choosing a uniform integer $x \in \mathbb{Z}_q$ and setting $h := g^x$. In what follows we make no assumptions on $\mathbb{G}$.

Elements of a group $\mathbb{G}$ must be specified using some *representation* of these elements as bit-strings, where we assume without any real loss of generality that all elements are represented using strings of the same length. (It is also crucial that there is a *unique* string representing each group element.) For example, if $\|N\| = n$ then elements of $\mathbb{Z}_N$ can all be represented as strings of length n, where the integer $a \in \mathbb{Z}_N$ is padded to the left with 0s if $\|a\| < n$.

We do not focus much on the issue of representation, since for all the groups considered in this text the representation can simply be taken to be the "natural" one (as in the case of $\mathbb{Z}_N$, above). Note, however, that different representations of the same group can affect the complexity of performing various computations, and so choosing the "right" representation for a given group is often important in practice. Since our goal is only to show *polynomial-time* algorithms for each of the operations we need (and not to show the most efficient algorithms known), the exact representation used is less important for our purposes. Moreover, most of the "higher-level" algorithms we present use the group operation in a "black-box" manner, so that as long as the group operation can be performed in polynomial time (in some parameter), the resulting algorithm will run in polynomial time as well.

Given a group $\mathbb{G}$ where elements are represented by strings of length ℓ, a uniform group element can be selected by choosing uniform ℓ-bit strings until the first string that corresponds to a group element is found. (Note this assumes that testing group membership can be done efficiently.) To obtain an algorithm with bounded running time, we introduce a parameter t bounding the maximum number of times this process is repeated; if all t iterations fail to find an element of $\mathbb{G}$, then the algorithm outputs fail. (An alternative is to output an arbitrary element of $\mathbb{G}$.) That is:

ALGORITHM B.14
Choosing a uniform group element

Input: A (description of a) group $\mathbb{G}$; length-parameter ℓ; parameter t
Output: A uniform element of $\mathbb{G}$

for $i = 1$ to t:
  Choose uniform $x \in \{0,1\}^\ell$
  if $x \in \mathbb{G}$ **return** x
return "fail"

It is clear that whenever the above algorithm does *not* output fail, it outputs a uniformly distributed element of $\mathbb{G}$. This is simply because each element of $\mathbb{G}$ is equally likely to be chosen in any iteration. Formally, if we let Fail be the event that the algorithm outputs fail, then for any element $g \in \mathbb{G}$ we have

$$\Pr\left[\text{output of the algorithm equals } g \mid \overline{\mathsf{Fail}}\right] = \frac{1}{|\mathbb{G}|}.$$

What is the probability that the algorithm outputs fail? In any iteration the probability that $x \in \mathbb{G}$ is exactly $|\mathbb{G}|/2^\ell$, and so the probability that x does *not* lie in $\mathbb{G}$ in any of the t iterations is

$$\left(1 - \frac{|\mathbb{G}|}{2^\ell}\right)^t. \tag{B.1}$$

There is a trade-off between the running time of Algorithm B.14 and the probability that the algorithm outputs fail: increasing t decreases the probability of failure but increases the worst-case running time. For cryptographic applications we need an algorithm where the worst-case running time is polynomial in the security parameter n, while the failure probability is negligible in n. Let $K \stackrel{\text{def}}{=} 2^\ell/|\mathbb{G}|$. If we set $t := K \cdot n$ then the probability that the algorithm outputs fail is:

$$\left(1 - \frac{1}{K}\right)^{K \cdot n} = \left(\left(1 - \frac{1}{K}\right)^K\right)^n \leq \left(e^{-1}\right)^n = e^{-n},$$

using Proposition A.2. Thus, if $K = \mathsf{poly}(n)$ (we assume some group-generation algorithm that depends on the security parameter n, and so both $|\mathbb{G}|$ and ℓ are functions of n), we obtain an algorithm with the desired properties.

The case of $\mathbb{Z}_N$. Consider the group $\mathbb{Z}_N$, with $n = \|N\|$. Checking whether an n-bit string x (interpreted as a positive integer of length at most n) is an element of $\mathbb{Z}_N$ simply requires checking whether $x < N$. Furthermore,

$$\frac{2^n}{|\mathbb{Z}_N|} = \frac{2^n}{N} \leq \frac{2^n}{2^{n-1}} = 2,$$

and so we can sample a uniform element of $\mathbb{Z}_N$ in $\mathsf{poly}(n)$ time and with failure probability negligible in n.

The case of $\mathbb{Z}_N^*$. Consider next the group $\mathbb{Z}_N^*$, with $n = \|N\|$ as before. Determining whether an n-bit string x is an element of $\mathbb{Z}_N^*$ is also easy (see the exercises). Moreover,

$$\frac{2^n}{|\mathbb{Z}_N^*|} = \frac{2^n}{\phi(N)} = \frac{2^n}{N} \cdot \frac{N}{\phi(N)} \leq 2 \cdot \frac{N}{\phi(N)}.$$

A $\mathsf{poly}(n)$ upper-bound is a consequence of the following theorem.

THEOREM B.15 *For $N \geq 3$ of length n, we have $\frac{N}{\phi(N)} < 2n$.*

(Stronger bounds are known, but the above suffices for our purpose.) The theorem can be proved using Bertrand's Postulate (Theorem 8.32), but we content ourselves with a proof in two special cases: when N is prime and when N is a product of two equal-length (distinct) primes.

The analysis is easy when N is an odd prime. Here $\phi(N) = N - 1$ and so

$$\frac{N}{\phi(N)} \leq \frac{2^n}{\phi(N)} = \frac{2^n}{N-1} \leq \frac{2^n}{2^{n-1}} = 2$$

(using the fact that N is odd for the second inequality). Consider next the case of $N = pq$ for p and q distinct, odd primes. Then

$$\frac{N}{\phi(N)} = \frac{pq}{(p-1)(q-1)} = \frac{p}{p-1} \cdot \frac{q}{q-1} < \left(\frac{3}{2}\right) \cdot \left(\frac{5}{4}\right) < 2.$$

We conclude that when N is prime or the product of two distinct, odd primes, there is an algorithm for generating a uniform element of $\mathbb{Z}_N^*$ that runs in time polynomial in $n = \|N\|$ and outputs fail with probability negligible in n.

Throughout this book, when we speak of sampling a uniform element of $\mathbb{Z}_N$ or $\mathbb{Z}_N^*$ we simply ignore the negligible probability of outputting fail with the understanding that this has no significant effect on the analysis.

B.3 *Finding a Generator of a Cyclic Group

In this section we address the problem of finding a generator of an arbitrary cyclic group $\mathbb{G}$ of order q. Here, q does not necessarily denote a prime number; indeed, finding a generator when q is prime is trivial by Corollary 8.55.

We actually show how to sample a *uniform* generator, proceeding in a manner very similar to that of Section B.2.5. Here, we repeatedly sample uniform elements of $\mathbb{G}$ until we find an element that is a generator. As in Section B.2.5, an analysis of this method requires understanding two things:

- How to efficiently test whether a given element is a generator; and
- the fraction of group elements that are generators.

In order to understand these issues, we first develop a bit of additional group-theoretic background.

B.3.1 Group-Theoretic Background

We tackle the second issue first. Recall that the order of an element h is the smallest positive integer i for which $h^i = 1$. Let g be a generator of a

group $\mathbb{G}$ of order $q > 1$; this means the order of g is q. Consider an element $h \in \mathbb{G}$ that is not the identity (the identity cannot be a generator of $\mathbb{G}$), and let us ask whether h might also be a generator of $\mathbb{G}$. Since g generates $\mathbb{G}$, we can write $h = g^x$ for some $x \in \{1, \ldots, q-1\}$ (note $x \neq 0$ since h is not the identity). Consider two cases:

Case 1: $\gcd(x, q) = r > 1$. Write $x = \alpha \cdot r$ and $q = \beta \cdot r$ with α, β non-zero integers less than q. Then:

$$h^\beta = (g^x)^\beta = g^{\alpha r \beta} = (g^q)^\alpha = 1.$$

So the order of h is at most $\beta < q$, and h cannot be a generator of $\mathbb{G}$.

Case 2: $\gcd(x, q) = 1$. Let $i \leq q$ be the order of h. Then

$$g^0 = 1 = h^i = (g^x)^i = g^{xi},$$

implying $xi = 0 \bmod q$ by Proposition 8.53. This means that $q \,|\, xi$. Since $\gcd(x, q) = 1$, however, Proposition 8.3 shows that $q \,|\, i$ and so $i = q$. We conclude that h is a generator of $\mathbb{G}$.

Summarizing the above, we see that for $x \in \{1, \ldots, q-1\}$ the element $h = g^x$ is a generator of $\mathbb{G}$ exactly when $\gcd(x, q) = 1$. We have thus proved the following:

THEOREM B.16 *Let $\mathbb{G}$ be a cyclic group of order $q > 1$ with generator g. There are $\phi(q)$ generators of $\mathbb{G}$, and these are exactly given by $\{g^x \mid x \in \mathbb{Z}_q^*\}$.*

In particular, if $\mathbb{G}$ is a group of prime order q, then it has $\phi(q) = q - 1$ generators—exactly in agreement with Corollary 8.55.

We turn next to the first issue, that of deciding whether a given element h is a generator of $\mathbb{G}$. Of course, one way to check whether h generates $\mathbb{G}$ is to enumerate $\{h^0, h^1, \ldots, h^{q-1}\}$ and see whether this list includes every element of $\mathbb{G}$. This requires time linear in q (i.e., exponential in $\|q\|$) and is therefore unacceptable for our purposes. Another approach, if we already know a generator g, is to compute the discrete logarithm $x = \log_g h$ and then apply the previous theorem; in general, however, we may not have such a g, and anyway computing the discrete logarithm may itself be a hard problem.

If we know the factorization of q, we can do better.

PROPOSITION B.17 *Let $\mathbb{G}$ be a group of order q, and let $q = \prod_{i=1}^{k} p_i^{e_i}$ be the prime factorization of q, where the $\{p_i\}$ are distinct primes and $e_i \geq 1$. Set $q_i = q/p_i$. Then $h \in \mathbb{G}$ is a generator of $\mathbb{G}$ if and only if*

$$h^{q_i} \neq 1 \quad \textit{for } i = 1, \ldots, k.$$

PROOF One direction is easy. Say $h^{q_i} = 1$ for some i. Then the order of h is at most $q_i < q$, and so h cannot be a generator.

Conversely, say h is not a generator but instead has order $q' < q$. By Proposition 8.54, we know $q' \mid q$. This implies that q' can be written as $q' = \prod_{i=1}^{k} p_i^{e'_i}$, where $e'_i \geq 0$ and for at least one index j we have $e'_j < e_j$. But then q' divides $q_j = p_j^{e_j - 1} \cdot \prod_{i \neq j} p_i^{e_i}$, and so (using Proposition 8.53) $h^{q_j} = h^{[q_j \bmod q']} = h^0 = 1$. ∎

The proposition does not require $\mathbb{G}$ to be cyclic; if $\mathbb{G}$ is not cyclic then every element $h \in \mathbb{G}$ will satisfy $h^{q_i} = 1$ for some i and there are no generators.

B.3.2 Efficient Algorithms

Armed with the results of the previous section, we show how to efficiently *test* whether a given element is a generator, as well as how to efficiently *find* a generator in an arbitrary group.

Testing if an element is a generator. Proposition B.17 immediately suggests an efficient algorithm for deciding whether a given element h is a generator or not.

ALGORITHM B.18
Testing whether an element is a generator

Input: Group order q; prime factors $\{p_i\}_{i=1}^{k}$ of q; element $h \in \mathbb{G}$
Output: A decision as to whether h is a generator of $\mathbb{G}$

for $i = 1$ to k:
  if $h^{q/p_i} = 1$ **return** "h is not a generator"
return "h is a generator"

Correctness of the algorithm is evident from Proposition B.17. We now show that the algorithm terminates in time polynomial in $\|q\|$. Since, in each iteration, h^{q/p_i} can be computed in polynomial time, we need only show that the number of iterations k is polynomial. This is the case since an integer q can have no more than $\log_2 q = \mathcal{O}(\|q\|)$ prime factors; this is because

$$q = \prod_{i=1}^{k} p_i^{e_i} \geq \prod_{i=1}^{k} p_i \geq \prod_{i=1}^{k} 2 = 2^k$$

and so $k \leq \log_2 q$.

Algorithm B.18 requires the prime factors of the group order q to be provided as input. Interestingly, there is no known efficient algorithm for testing whether an element of an arbitrary group is a generator when the factors of the group order are *not* known.

The fraction of elements that are generators. As shown in Theorem B.16, the fraction of elements of a group $\mathbb{G}$ of order q that are generators is $\phi(q)/q$. Theorem B.15 says that $\phi(q)/q = \Omega(1/\|q\|)$. The fraction of elements that are generators is thus sufficiently high to ensure that sampling a polynomial number of elements from the group will yield a generator with all but negligible probability. (The analysis is the same as in Section B.2.5.)

Concrete examples in $\mathbb{Z}_p^*$. Putting everything together, we see there is an efficient probabilistic algorithm for finding a generator of a group $\mathbb{G}$ *as long as the factorization of the group order is known.* When selecting a group for cryptographic applications, it is therefore important that the group is chosen in such a way that this holds. This explains again the preference, discussed extensively in Section 8.3.2, for working in an appropriate prime-order subgroup of $\mathbb{Z}_p^*$. Another possibility is to use $\mathbb{G} = \mathbb{Z}_p^*$ for p a strong prime (i.e., $p = 2q+1$ with q also prime), in which case the prime factorization of the group order $p-1$ is known. One final possibility is to generate a prime p in such a way that the factorization of $p-1$ is known. Further details are beyond the scope of this book.

References and Additional Reading

The book by Shoup [159] is highly recommended for those seeking to explore the topics of this chapter in further detail. In particular, bounds on $\phi(N)/N$ (and an asymptotic version of Theorem B.15) can be found in [159, Chapter 5]. Hankerson et al. [83] also provide extensive detail on the implementation of number-theoretic algorithms for cryptography.

Exercises

B.1 Prove correctness of the extended Euclidean algorithm.

B.2 Prove that the extended Euclidean algorithm runs in time polynomial in the lengths of its inputs.

Hint: First prove a proposition analogous to Proposition B.8.

B.3 Show how to determine that an n-bit string is in $\mathbb{Z}_N^*$ in polynomial time.

References

[1] M. Abdalla, J.H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Necessary and sufficient conditions for security and forward-security. *IEEE Trans. Information Theory*, 54(8):3631–3646, 2008.

[2] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Cryptographers' Track—RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, 2001. See `http://cseweb.ucsd.edu/~mihir/papers/dhies.html`.

[3] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison Wesley, 2nd edition, 2002.

[4] L.M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *20th Annual Symposium on Foundations of Computer Science*, pages 55–60. IEEE, 1979.

[5] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[6] W. Aiello, M. Bellare, G. Di Crescenzo, and R. Venkatesan. Security amplification by composition: The case of doubly-iterated, ideal ciphers. In *Advances in Cryptology—Crypto '98*, volume 1462 of *LNCS*, pages 390–407. Springer, 1998.

[7] A. Akavia, S. Goldwasser, and S. Safra. Proving hard-core predicates using list decoding. In *Proc. 44th Annual Symposium on Foundations of Computer Science*, pages 146–157. IEEE, 2003.

[8] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, 1988.

[9] N.J. AlFardan, D.J. Bernstein, K.G. Paterson, B. Poettering, and J.C.N. Schuldt. On the security of RC4 in TLS and WPA. In *USENIX Security Symposium*, 2013.

[10] J.H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *Advances in Cryptology—Eurocrypt 2002*, volume 2332 of *LNCS*, pages 83–107. Springer, 2002.

[11] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *Advances in Cryptology—Eurocrypt 2014*, volume 8441 of *LNCS*, pages 1–16. Springer, 2014.

[12] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay. Efficient padding oracle attacks on cryptographic hardware. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 608–625. Springer, 2012.

[13] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management—part 1: General (revision 3), July 2012. NIST Special Publication 800-57.

[14] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology—Crypto '96*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.

[15] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proc. 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.

[16] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology—Crypto '98*, volume 1462 of *LNCS*, pages 26–45. Springer, 1998.

[17] M. Bellare, O. Goldreich, and A. Mityagin. The power of verification queries in message authentication and authenticated encryption. Available at `http://eprint.iacr.org/2004/309`.

[18] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[19] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology—Asiacrypt 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, 2000.

[20] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *13th ACM Conf. on Computer and Communications Security*, pages 390–399. ACM Press, 2006.

[21] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conf. on Computer and Communications Security*, pages 62–73. ACM, 1993.

[22] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Eurocrypt '94*, volume 950 of *LNCS*, pages 92–111. Springer, 1994.

[23] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology—Eurocrypt '96*, volume 1070 of *LNCS*, pages 399–416. Springer, 1996.

[24] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology—Eurocrypt 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, 2006. A full version of the paper is available at `http://eprint.iacr.org`.

[25] S.M. Bellovin. Frank Miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203–222, 2011.

[26] D.J. Bernstein. A short proof of the unpredictability of cipher block chaining. Available at `http://cr.yp.to/papers.html#easycbc`.

[27] D.J. Bernstein. How to stretch random functions: The security of protected counter sums. *Journal of Cryptology*, 12(3):185–192, 1999.

[28] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology—Eurocrypt 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, 2008.

[29] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[30] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[31] A. Biryukov and A. Shamir. Structural cryptanalysis of SASAS. *J. Cryptology*, 23(4):505–518, 2010.

[32] J. Black and P. Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. *Journal of Cryptology*, 18(2):111–131, 2005.

[33] J. Black, P. Rogaway, T. Shrimpton, and M. Stam. An analysis of the blockcipher-based hash functions from PGV. *J. Cryptology*, 23(4):519–545, 2010.

[34] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1. In *Advances in Cryptology—Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[35] M. Blum. Coin flipping by telephone. In *Proc. IEEE COMPCOM*, pages 133–137, 1982.

[36] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Advances in Cryptology—Crypto '84*, volume 196 of *Lecture Notes in Computer Science*, pages 289–302. Springer, 1985.

[37] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[38] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society*, 46(2):203–213, 1999.

[39] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In *Advances in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 275–291. Springer, 2001.

[40] D. Boneh, A. Joux, and P.Q. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In *Advances in Cryptology—Asiacrypt 2000*, volume 1976 of *LNCS*, pages 30–43. Springer, 2000.

[41] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[42] L. Carter and M.N. Wegman. Universal classes of hash functions. *J. Computer and System Sciences*, 18(2):143–154, 1979.

[43] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In *Advances in Cryptology—Crypto '91*, volume 576 of *LNCS*, pages 470–484. Springer, 1992.

[44] L.N. Childs. *A Concrete Introduction to Higher Algebra*. Undergraduate Texts in Mathematics. Springer, 2nd edition, 2000.

[45] D. Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, 1994.

[46] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[47] D. Coppersmith, M.K. Franklin, J. Patarin, and M.K. Reiter. Low-exponent RSA with related messages. In *Advances in Cryptology—Eurocrypt '96*, volume 1070 of *LNCS*, pages 1–9. Springer, 1996.

[48] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology—Crypto 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.

[49] J.-S. Coron, M. Joye, D. Naccache, and P. Paillier. New attacks on PKCS #1 v1.5 encryption. In *Advances in Cryptology—Eurocrypt 2000*, volume 1807 of *LNCS*, pages 369–381. Springer, 2000.

[50] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[51] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer, 2nd edition, 2005.

[52] I. Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, volume 304 of *LNCS*, pages 203–216. Springer, 1988.

[53] I. Damgård. A design principle for hash functions. In *Advances in Cryptology—Crypto '89*, volume 435 of *LNCS*, pages 416–427. Springer, 1990.

[54] J.P. Degabriele and K.G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In *17th ACM Conf. on Computer and Communications Security*, pages 493–504. ACM Press, 2010.

[55] J. DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8:253–259, 1984.

[56] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and non-interactive non-malleable commitment. In *Advances in Cryptology—Eurocrypt 2001*, volume 2045 of *LNCS*, pages 40–59. Springer, 2001.

[57] M. Dietzfelbinger. *Primality Testing in Polynomial Time*. Springer, 2004.

[58] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[59] W. Diffie and M. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, pages 74–84, June 1977.

[60] J.D. Dixon. Asymptotically fast factorization of integers. *Mathematics of Computation*, 36:255–260, 1981.

[61] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM J. Computing*, 30(2):391–437, 2000. Preliminary version in STOC '91.

[62] C. Ellison and B. Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[63] S. Even and Y. Mansour. A construction of a cipher from a single pseudorandom permutation. *J. Cryptology*, 10(3):151–162, 1997.

[64] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973.

[65] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—Crypto '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[66] R. Fischlin and C.-P. Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, 2000.

[67] J.B. Fraleigh. *A First Course in Abstract Algebra*. Addison Wesley, 7th edition, 2002.

[68] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *Journal of Cryptology*, 17(2):81–104, 2004.

[69] S.D. Galbraith. *The Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[70] T. El Gamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Info. Theory*, 31(4):469–472, 1985.

[71] C.F. Gauss. *Disquisitiones Arithmeticae*. Springer, 1986. (English edition).

[72] R. Gennaro, Y. Gertner, and J. Katz. Lower bounds on the efficiency of encryption and digital signature schemes. In *35th Annual ACM Symposium on Theory of Computing*, pages 417–425. ACM Press, 2003.

[73] E.N. Gilbert, F.J. MacWilliams, and N.J.A. Sloane. Codes which detect deception. *Bell Systems Technical Journal*, 53(3):405–424, 1974.

[74] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In *Advances in Cryptology—Crypto '86*, volume 263 of *LNCS*, pages 104–110. Springer, 1987.

[75] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*. Cambridge University Press, 2001.

[76] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, 2004.

[77] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In *Advances in Cryptology—Crypto '84*, volume 196 of *LNCS*, pages 276–288. Springer, 1985.

[78] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[79] O. Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32. ACM Press, 1989.

[80] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[81] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, 1988.

[82] S. Goldwasser, S. Micali, and A.C.-C. Yao. Strong signature schemes. In *Proc. 15th Annual ACM Symposium on Theory of Computing*, pages 431–439. ACM, 1983.

[83] D. Hankerson, A.J. Menezes, and S.A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[84] J. Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, 1988.

[85] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[86] J. Håstad and M. Näslund. The security of all RSA and discrete log bits. *Journal of the ACM*, 51(2):187–230, 2004.

[87] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory*, 26(4):401–406, 1980.

[88] N. Heninger, Z. Durumeric, E. Wustrow, and J.A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proc. 21st USENIX Security Symposium*, 2012.

[89] I.N. Herstein. *Abstract Algebra*. Wiley, 3rd edition, 1996.

[90] H.M. Heys. *The Design of Substitution-Permutation Network Ciphers Resistant to Cryptanalysis*. PhD thesis, Queen's University, 1994.

[91] H.M. Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002. Also available at `http://www.engr.mun.ca/~howard/Research/Papers/`.

[92] D. Hofheinz and E. Kiltz. Practical chosen ciphertext secure encryption from factoring. In *Advances in Cryptology—Eurocrypt 2009*, volume 5479 of *LNCS*, pages 313–332. Springer, 2009.

[93] R. Impagliazzo and M. Luby. One-way functions are essential for complexity-based cryptography. In *30th Annual Symposium on Foundations of Computer Science*, pages 230–235. IEEE, 1989.

[94] ISO/IEC 9797. Data cryptographic techniques—data integrity mechanism using a cryptographic check function employing a block cipher algorithm, 1989.

[95] T. Iwata and K. Kurosawa. OMAC: One-key CBC MAC. In *Fast Software Encryption—FSE 2003*, volume 2887 of *LNCS*, pages 129–153. Springer, 2003.

[96] A. Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC Press, 2009.

[97] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[98] J. Katz. *Digital Signatures*. Springer, 2010.

[99] J. Katz and C.-Y. Koo. On constructing universal one-way hash functions from arbitrary one-way functions. *J. Cryptology*, to appear. Available at `http://eprint.iacr.org/2005/328`.

[100] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *Fast Software Encryption—FSE 2000*, volume 1978 of *LNCS*, pages 284–299. Springer, 2000.

[101] J. Katz and M. Yung. Characterization of security notions for probabilistic private-key encryption. *Journal of Cryptology*, 19(1):67–96, 2006.

[102] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, 2nd edition, 2002.

[103] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, IX:5–38, January 1883. A copy of the paper is available at `http://www.petitcolas.net/fabien/kerckhoffs`.

[104] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, IX:161–191, February 1883. A copy of the paper is available at `http://www.petitcolas.net/fabien/kerckhoffs`.

[105] J. Kilian and P. Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *Journal of Cryptology*, 14(1):17–35, 2001.

[106] L. Knudsen and M.J.B. Robshaw. *The Block Cipher Companion*. Springer, 2011.

[107] L.M. Kohnfelder. Towards a practical public-key cryptosystem, 1978. Undergraduate thesis, MIT.

[108] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *Advances in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 310–331. Springer, 2001.

[109] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology—Crypto 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, 2010.

[110] T. Krovetz and P. Rogaway. The software performance of authenticated-encryption modes. In *Fast Software Encryption—FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, 2011.

[111] L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, 1978.

[112] A.K. Lenstra, J.P. Hughes, M. Augier, J.W. Bos, T. Kleinjung, and C. Wachter. Public keys. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 626–642. Springer, 2012.

[113] A.K. Lenstra and E.R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[114] S. Levy. *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*. Viking, 2001.

[115] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.

[116] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, 1988.

[117] J. Manger. A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS #1 v2.0. In *Advances in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 230–238. Springer, 2001.

[118] M. Matsui. Linear cryptoanalysis method for DES cipher. In *Advances in Cryptology—Eurocrypt '93*, volume 765 of *LNCS*, pages 386–397. Springer, 1993.

[119] A. May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD thesis, University of Paderborn, 2003.

[120] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRS Press, 1997.

[121] R.C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—Crypto '87*, volume 293 of *LNCS*, pages 369–378. Springer, 1988.

[122] R.C. Merkle. A certified digital signature. In *Advances in Cryptology—Crypto '89*, volume 435 of *LNCS*, pages 218–238. Springer, 1990.

[123] R.C. Merkle. One way hash functions and DES. In *Advances in Cryptology—Crypto '89*, volume 435 of *LNCS*, pages 428–446. Springer, 1990.

[124] R.C. Merkle and M. Hellman. On the security of multiple encryption. *Communications of the ACM*, 24(7):465–467, 1981.

[125] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Computing*, 17(2):412–426, 1988.

[126] E. Miles and E. Viola. Substitution-permutation networks, pseudorandom functions, and natural proofs. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 68–85. Springer, 2012.

[127] G.L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[128] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 33–43. ACM, 1989.

[129] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437. ACM Press, 1990.

[130] National Bureau of Standards. Federal information processing standard publication 81: DES modes of operation, 1980.

[131] National Institute of Standards and Technology. Federal information processing standard publication 198-1: The keyed-hash message authentication code (HMAC), July 2008.

[132] National Institute of Standards and Technology. Federal information processing standards publication 186-4: Digital signature standard (DSS), July 2013.

[133] V.I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[134] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology—Crypto 2003*, volume 2729 of *LNCS*, pages 617–630. Springer, 2003.

[135] C. Paar and J. Pelzl. *Understanding Cryptography.* Springer, 2010.

[136] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology—Eurocrypt '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[137] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, 13(3):315–338, 2000.

[138] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Information Theory*, 24(1):106–110, 1978.

[139] J.M. Pollard. Theorems of factorization and primality testing. *Proc. Cambridge Philosophical Society*, 76:521–528, 1974.

[140] J.M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[141] J.M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[142] C. Pomerance. The quadratic sieve factoring algorithm. In *Advances in Cryptology—Eurocrypt '84*, volume 209 of *LNCS*, pages 169–182. Springer, 1985.

[143] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology—Crypto '93*, volume 773 of *LNCS*, pages 368–378. Springer, 1994.

[144] M.O. Rabin. Digitalized signatures. In R.A. Demillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, editors, *Foundations of Security Computation*, pages 155–168. Academic Press, 1978.

[145] M.O. Rabin. Digitalized signatures as intractable as factorization. Technical Report TR-212, MIT/LCS, 1979.

[146] M.O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[147] C. Rackoff and D.R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology—Crypto '91*, volume 576 of *LNCS*, pages 433–444. Springer, 1992.

[148] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[149] P. Rogaway. The security of DESX. *RSA Laboratories' CryptoBytes*, Summer 1996.

[150] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption—FSE 2004*, volume 3017 of *LNCS*, pages 371–388. Springer, 2004.

[151] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 387–394. ACM, 1990.

[152] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—Crypto '89*, volume 435 of *LNCS*, pages 239–252. Springer, 1990.

[153] D. Shanks. Class number, a theory of factorization, and genera. In *Proc. Symposia in Pure Mathematics 20*, pages 415–440. American Mathematical Society, 1971.

[154] C.E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[155] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—Eurocrypt '97*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.

[156] V. Shoup. Why chosen ciphertext security matters. Technical Report RZ 3076, IBM Zurich, November 1998. Available at `http://shoup.net/papers/expo.pdf`.

[157] V. Shoup. A proposal for an ISO standard for public key encryption, 2001. Available at `http://eprint.iacr.org/201/112`.

[158] V. Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, 2002.

[159] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2nd edition, 2009. Also available at `http://www.shoup.net/ntb`.

[160] J.H. Silverman and J. Tate. *Rational Points on Elliptic Curves*. Undergraduate Texts in Mathematics. Springer, 1994.

[161] G. Simmons. A "weak" privacy protocol using the RSA crypto algorithm. *Cryptologia*, 7:180–182, 1983.

[162] G. Simmons. A survey of information authentication. In G. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 379–419. IEEE Press, 1992.

[163] S. Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor Books, 2000.

[164] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.

[165] W. Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall, 5th edition, 2013.

[166] D.R. Stinson. Universal hashing and authentication codes. *Designs, Codes, and Cryptography*, 4(4):369–380, 1994.

[167] D.R. Stinson. *Cryptography: Theory and Practice*. Chapman & Hall/CRC Press, 1st edition, 1995.

[168] D.R. Stinson. *Cryptography: Theory and Practice*. Chapman & Hall/CRC Press, 3rd edition, 2005.

[169] W. Trappe and L. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, 2nd edition, 2005.

[170] P.C. van Oorschot and M.J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[171] S. Vaudenay. Security flaws induced by CBC padding—applications to SSL, IPSEC, WTLS, ... In *Advances in Cryptology—Eurocrypt 2002*, volume 2332 of *LNCS*, pages 534–546. Springer, 2002.

[172] G.S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Journal of the American Institute for Electrical Engineers*, 55:109–115, 1926.

[173] S.S. Wagstaff, Jr. *Cryptanalysis of Number Theoretic Ciphers*. Chapman & Hall/CRC Press, 2003.

[174] X. Wang, Y.L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology—Crypto 2005*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

[175] X. Wang and H. Yu. How to break MD5 and other hash functions. In *Advances in Cryptology—Eurocrypt 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

[176] L. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman & Hall/CRC Press, 2003.

[177] M.N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *J. Computer and System Sciences*, 22(3):265–279, 1981.

[178] ANSI X9.9. American national standard for financial institution message authentication (wholesale), 1981.

[179] A.C.-C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91. IEEE, 1982.

[180] G. Yuval. How to swindle Rabin. *Cryptologia*, 3:187–189, 1979.

Computer Science/Mathematics

Cryptography is ubiquitous and plays a key role in ensuring data secrecy and integrity as well as in securing computer systems more broadly. **Introduction to Modern Cryptography** provides a rigorous yet accessible treatment of this fascinating subject.

The authors introduce the core principles of modern cryptography, with an emphasis on formal definitions, clear assumptions, and rigorous proofs of security. The book begins by focusing on private-key cryptography, including an extensive treatment of private-key encryption, message authentication codes, and hash functions. The authors also present design principles for widely used stream ciphers and block ciphers including RC4, DES, and AES, plus they provide provable constructions of stream ciphers and block ciphers from lower-level primitives. The second half of the book covers public-key cryptography, beginning with a self-contained introduction to the number theory needed to understand the RSA, Diffie–Hellman, and El Gamal cryptosystems (and others), followed by a thorough treatment of several standardized public-key encryption and digital signature schemes.

Integrating a more practical perspective without sacrificing rigor, this widely anticipated **Second Edition** offers improved treatment of

- Stream ciphers and block ciphers, including modes of operation and design principles
- Authenticated encryption and secure communication sessions
- Hash functions, including hash-function applications and design principles
- Attacks on poorly implemented cryptography, including attacks on chained-CBC encryption, padding-oracle attacks, and timing attacks
- The random-oracle model and its application to several standardized, widely used public-key encryption and signature schemes
- Elliptic-curve cryptography and associated standards such as DSA/ECDSA and DHIES/ECIES

Containing updated exercises and worked examples, **Introduction to Modern Cryptography, Second Edition** can serve as a textbook for undergraduate- or graduate-level courses in cryptography, a valuable reference for researchers and practitioners, or a general introduction suitable for self-study.

CRC Press
Taylor & Francis Group
an informa business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
711 Third Avenue
New York, NY 10017
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

K16475
ISBN: 978-1-4665-7026-9
90000
9 781466 570269
www.crcpress.com