5

Other Public-key Cryptosystems

In this chapter, we look at several other public-key cryptosystems. The **ElGamal Cryptosystem** is based on the **Discrete Logarithm** problem, which we will have occasion to use in numerous cryptographic protocols throughout the rest of the text. Thus we devote a considerable amount of time to discussion of this important problem. In later sections, we give relatively brief treatments of some other well-known public-key cryptosystems. These include ElGamal-type systems based on finite fields and elliptic curves, the (broken) **Merkle-Hellman Knapsack Cryptosystem** and the **McEliece Cryptosystem**.

5.1 The ElGamal Cryptosystem and Discrete Logs

The **ElGamal Cryptosystem** is based on the **Discrete Logarithm** problem. We begin by describing this problem in the setting of a finite field $\mathbb{Z}_p$, where p is prime, in Figure 5.1. (Recall that the multiplicative group $\mathbb{Z}_p{}^*$ is cyclic, and a generator of $\mathbb{Z}_p{}^*$ is called a primitive element.)

The **Discrete Logarithm** problem in $\mathbb{Z}_p$ has been the object of much study. The problem is generally regarded as being difficult if p is carefully chosen. In particular, there is no known polynomial-time algorithm for the **Discrete Logarithm** problem. To thwart known attacks, p should have at least 150 digits, and $p - 1$ should have at at least one "large" prime factor. The utility of the **Discrete Logarithm** problem in a cryptographic setting is that finding discrete logs is (probably) difficult, but the inverse operation of exponentiation can be computed efficiently by using the square-and-multiply method described earlier. Stated another way, exponentiation modulo p is a one-way function for suitable primes p.

ElGamal has developed a public-key cryptosystem based on the **Discrete Logarithm** problem. This system is presented in Figure 5.2.

The **ElGamal Cryptosystem** is non-deterministic, since the ciphertext depends on both the plaintext x and on the random value k chosen by Alice. So there will

FIGURE 5.1
The discrete logarithm problem in $\mathbb{Z}_p$

Problem Instance $I = (p, \alpha, \beta)$, where p is prime, $\alpha \in \mathbb{Z}_p$ is a primitive element, and $\beta \in \mathbb{Z}_p^*$.

Objective Find the unique integer a, $0 \leq a \leq p-2$, such that

$$\alpha^a \equiv \beta \pmod{p}.$$

We will denote this integer a by $\log_\alpha \beta$.

FIGURE 5.2
ElGamal Public-key Cryptosystem in $\mathbb{Z}_p^*$

Let p be a prime such that the discrete log problem in $\mathbb{Z}_p$ is intractible, and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*$, $\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\}.$$

The values p, α and β are public, and a is secret.

For $K = (p, \alpha, a, \beta)$, and for a (secret) random number $k \in \mathbb{Z}_{p-1}$, define

$$e_K(x, k) = (y_1, y_2),$$

where

$$y_1 = \alpha^k \bmod p$$

and

$$y_2 = x\beta^k \bmod p.$$

For $y_1, y_2 \in \mathbb{Z}_p^*$, define

$$d_K(y_1, y_2) = y_2(y_1{}^a)^{-1} \bmod p.$$

be many ciphertexts that are encryptions of the same plaintext.

Informally, this is how the **ElGamal Cryptosystem** works. The plaintext x is "masked" by multiplying it by β^k, yielding y_2. The value α^k is also transmitted as part of the ciphertext. Bob, who knows the secret exponent a, can compute β^k from α^k. Then he can "remove the mask" by dividing y_2 by β^k to obtain x.

A small example will illustrate.

Example 5.1
Suppose $p = 2579$, $\alpha = 2$, $a = 765$, and hence

$$\beta = 2^{765} \bmod 2579 = 949.$$

Now, suppose that Alice wishes to send the message $x = 1299$ to Bob. Say $k = 853$ is the random integer she chooses. Then she computes

$$\begin{aligned} y_1 &= 2^{853} \bmod 2579 \\ &= 435 \end{aligned}$$

and

$$\begin{aligned} y_2 &= 1299 \times 949^{853} \bmod 2579 \\ &= 2396. \end{aligned}$$

When Bob receives the ciphertext $y = (435, 2396)$, he computes

$$\begin{aligned} x &= 2396 \times (435^{765})^{-1} \bmod 2579 \\ &= 1299, \end{aligned}$$

which was plaintext that Alice encrypted. □

5.1.1 Algorithms for the Discrete Log Problem

Throughout this section, we assume that p is prime and α is a primitive element modulo p. We take p and α to be fixed. Hence the **Discrete Logarithm** problem can be phrased in the following form: Given $\beta \in \mathbb{Z}_p^*$, find the unique exponent a, $0 \leq a \leq p-2$, such that $\alpha^a \equiv \beta \pmod{p}$.

Clearly, the **Discrete Logarithm** problem can be solved by exhasutive search in $O(p)$ time and $O(1)$ space (neglecting logarithmic factors). By precomputing all possible values α^a, and sorting the ordered pairs $(a, \alpha^a \bmod p)$ with respect to their second coordinates, we can solve the discrete log problem in $O(1)$ time with $O(p)$ precomputation and $O(p)$ memory (again, neglecting logarithmic factors). The first non-trivial algorithm we describe is a time-memory trade-off due to Shanks.

FIGURE 5.3
Shanks' algorithm for the discrete logarithm problem

1. Compute $\alpha^{mj} \bmod p$, $0 \leq j \leq m-1$
2. Sort the m ordered pairs $(j, \alpha^{mj} \bmod p)$ with respect to their second coordinates, obtaining a list L_1
3. Compute $\beta\alpha^{-i} \bmod p$, $0 \leq i \leq m-1$
4. Sort the m ordered pairs $(i, \beta\alpha^{-i} \bmod p)$ with respect to their second coordinates, obtaining a list L_2
5. Find a pair $(j, y) \in L_1$ and a pair $(i, y) \in L_2$ (i.e., a pair having identical second coordinates)
6. define $\log_\alpha \beta = mj + i \bmod (p-1)$.

Shanks' Algorithm

Denote $m = \lceil \sqrt{p-1} \rceil$. Shanks' algorithm is presented in Figure 5.3. Some comments are in order. First, steps 1 and 2 can be precomputed, if desired (this will not affect the asymptotic running time, however). Next, observe that if $(j, y) \in L_1$ and $(i, y) \in L_2$, then

$$\alpha^{mj} = y = \beta\alpha^{-i},$$

so

$$\alpha^{mj+i} = \beta,$$

as desired. Conversely, for any β, we can write

$$\log_\alpha \beta = mj + i,$$

where $0 \leq j, i \leq m-1$. Hence, the search in step 5 will be successful.

It is not difficult to implement the algorithm to run in $O(m)$ time with $O(m)$ memory (neglecting logarithmic factors). Note that step 5 can be done with one (simultaneous) pass through each of the two lists L_1 and L_2.

Here is a small example to illustrate.

Example 5.2
Suppose $p = 809$, and we wish to find $\log_3 525$. So we have $\alpha = 3$, $\beta = 525$ and $m = \lceil \sqrt{808} \rceil = 29$. Then

$$\alpha^{29} \bmod 809 = 99.$$

First, we compute the ordered pairs $(j, 99^j \bmod 809)$ for $0 \leq j \leq 28$. We obtain the list

(0, 1)	(1, 99)	(2, 93)	(3, 308)	(4, 559)
(5, 329)	(6, 211)	(7, 664)	(8, 207)	(9, 268)
(10, 644)	(11, 654)	(12, 26)	(13, 147)	(14, 800)
(15, 727)	(16, 781)	(17, 464)	(18, 632)	(19, 275)
(20, 528)	(21, 496)	(22, 564)	(23, 15)	(24, 676)
(25, 586)	(26, 575)	(27, 295)	(28, 81)	

which is then sorted to produce L_1.

The second list contains the ordered pairs $(i, 525 \times (3^i)^{-1} \bmod 809)$, $0 \leq j \leq 28$. It is as follows:

(0, 525)	(1, 175)	(2, 328)	(3, 379)	(4, 396)
(5, 132)	(6, 44)	(7, 554)	(8, 724)	(9, 511)
(10, 440)	(11, 686)	(12, 768)	(13, 256)	(14, 355)
(15, 388)	(16, 399)	(17, 133)	(18, 314)	(19, 644)
(20, 754)	(21, 521)	(22, 713)	(23, 777)	(24, 259)
(25, 356)	(26, 658)	(27, 489)	(28, 163)	

After sorting this list, we get L_2.

Now, if we proceed simultaneously through the two sorted lists, we find (10, 644) in L_1 and (19, 644) in L_2. Hence, we can compute

$$\begin{aligned} \log_3 525 &= 29 \times 10 + 19 \\ &= 309. \end{aligned}$$

As a check, it can be verified that indeed $3^{309} \equiv 525 \pmod{809}$. □

The Pohlig-Hellman Algorithm

The next algorithm we study is the Pohlig-Hellman algorithm. Suppose

$$p - 1 = \prod_{i=1}^{k} {p_i}^{c_i},$$

where the p_i's are distinct primes. The value $a = \log_\alpha \beta$ is determined (uniquely) modulo $p - 1$. We first observe that if we can compute $a \bmod {p_i}^{c_i}$ for each i, $1 \leq i \leq k$, then we can compute $a \bmod (p-1)$ by the Chinese remainder theorem. So, let's suppose that q is prime,

$$p - 1 \equiv 0 \pmod{q^c}$$

and

$$p - 1 \not\equiv 0 \pmod{q^{c+1}}.$$

We will show how to compute the value

$$x = a \bmod q^c,$$

where $0 \leq x \leq q^c - 1$. We can express x in radix q representation as

$$x = \sum_{i=0}^{c-1} a_i q^i,$$

where $0 \leq a_i \leq q - 1$ for $0 \leq i \leq c - 1$. Also, observe that we can express a as

$$a = x + q^c s$$

for some integer s.

The first step of the algorithm is to compute a_0. The main observation is that

$$\beta^{(p-1)/q} \equiv \alpha^{(p-1)a_0/q} \pmod{p}.$$

To see this, note that

$$\beta^{(p-1)/q} \equiv \alpha^{(p-1)(x+q^c s)/q} \pmod{p},$$

so it suffices to show that

$$\alpha^{(p-1)(x+q^c s)/q} \equiv \alpha^{(p-1)a_0/q} \pmod{p}.$$

This will be true if and only if

$$\frac{(p-1)(x+q^c s)}{q} \equiv \frac{(p-1)a_0}{q} \pmod{p-1}.$$

However, we have

$$\begin{aligned}
\frac{(p-1)(x+q^c s)}{q} - \frac{(p-1)a_0}{q} &= \frac{p-1}{q}(x + q^c s - a_0) \\
&= \frac{p-1}{q}\left(\sum_{i=0}^{c-1} a_i q^i + q^c s - a_0\right) \\
&= \frac{p-1}{q}\left(\sum_{i=1}^{c-1} a_i q^i + q^c s\right) \\
&= (p-1)\left(\sum_{i=1}^{c-1} a_i q^{i-1} + q^{c-1} s\right) \\
&\equiv 0 \pmod{p-1},
\end{aligned}$$

which was what we wanted to prove.

Hence, we begin by computing $\beta^{(p-1)/q} \bmod p$. If

$$\beta^{(p-1)/q} \equiv 1 \pmod{p},$$

then $a_0 = 0$. Otherwise, we successively compute

$$\gamma = \alpha^{(p-1)/q} \bmod p, \gamma^2 \bmod p, \ldots,$$

until

$$\gamma^i \equiv \beta^{(p-1)/q} \pmod{p}$$

for some i. When this happens, we have $a_0 = i$.

Now, if $c = 1$, we're done. Otherwise $c > 1$, and we proceed to determine a_1. To do this, we define

$$\beta_1 = \beta\alpha^{-a_0}$$

and denote

$$x_1 = \log_\alpha \beta_1 \bmod q^c.$$

It is not hard to see that

$$x_1 = \sum_{i=1}^{c-1} a_i q^i.$$

Hence, it follows that

$$\beta_1{}^{(p-1)/q^2} \equiv \alpha^{(p-1)a_1/q} \pmod{p}.$$

So, we will compute $\beta_1{}^{(p-1)/q^2} \bmod p$, and then find i such that

$$\gamma^i \equiv \beta_1{}^{(p-1)/q^2} \pmod{p}.$$

Then we have $a_1 = i$.

If $c = 2$, we are now finished; otherwise, we repeat this process $c - 2$ more times, obtaining $a_2, \ldots, a_{c-1}$.

A pseudo-code description of the Pohlig-Hellman algorithm is given in Figure 5.4. In this algorithm, α is a primitive element modulo p, q is prime,

$$p - 1 \equiv 0 \pmod{q^c}$$

and

$$p - 1 \not\equiv 0 \pmod{q^{c+1}}.$$

The algorithm calculates $a_0, \ldots, a_{c-1}$, where

$$\log_\alpha \beta \bmod q^c = \sum_{i=0}^{c-1} a_i q^i.$$

FIGURE 5.4
Pohlig-Hellman algorithm to compute $\log_\alpha \beta \bmod q^c$

1. compute $\gamma_i = \alpha^{(p-1)i/q} \bmod p$ for $0 \le i \le q-1$
2. set $j = 0$ and $\beta_j = \beta$
3. **while** $j \le c-1$ **do**
4. compute $\delta = \beta_j^{(p-1)/q^{j+1}} \bmod p$
5. find i such that $\delta = \gamma_i$
6. $a_j = i$
7. $\beta_{j+1} = \beta_j \alpha^{-a_j q^j} \bmod p$
8. $j = j+1$

We illustrate the Pohlig-Hellman algorithm with a small example.

Example 5.3
Suppose $p = 29$; then

$$n = p - 1 = 28 = 2^2 7^1.$$

Suppose $\alpha = 2$ and $\beta = 18$, so we want to determine $a = \log_2 18$. We proceed by first computing $a \bmod 4$ and then computing $a \bmod 7$.

We start by setting $q = 2$ and $c = 2$. First,

$$\gamma_0 = 1$$

and

$$\begin{aligned}\gamma_1 &= \alpha^{28/2} \bmod 29 \\ &= 2^{14} \bmod 29 \\ &= 28.\end{aligned}$$

Next,

$$\begin{aligned}\delta &= \beta^{28/2} \bmod 29 \\ &= 18^{14} \bmod 29 \\ &= 28.\end{aligned}$$

Hence, $a_0 = 1$. Next, we compute

$$\begin{aligned}\beta_1 &= \beta_0 \alpha^{-1} \bmod 29 \\ &= 9.\end{aligned}$$

and

$$\begin{aligned}{\beta_1}^{28/4} \bmod 29 &= 9^7 \bmod 29 \\ &= 28.\end{aligned}$$

Since

$$\gamma_1 \equiv 28 \bmod 29,$$

we have $a_1 = 1$. Hence, $a \equiv 3 \pmod 4$.

Next, we set $q = 7$ and $c = 1$. We have

$$\begin{aligned}\beta^{28/7} \bmod 29 &= 18^4 \bmod 29 \\ &= 25\end{aligned}$$

and

$$\begin{aligned}\gamma_1 &= \alpha^{28/7} \bmod 29 \\ &= 2^4 \bmod 29 \\ &= 16.\end{aligned}$$

Then we would compute

$$\begin{aligned}\gamma_2 &= 24 \\ \gamma_3 &= 7 \\ \gamma_4 &= 25.\end{aligned}$$

Hence, $a_0 = 4$ and $a \equiv 4 \pmod 7$.

Finally, solving the system

$$\begin{aligned}a &\equiv 3 \pmod 4 \\ a &\equiv 4 \pmod 7\end{aligned}$$

using the Chinese remainder theorem, we get $a \equiv 11 \pmod{28}$. That is, we have computed $\log_2 18$ in $\mathbb{Z}_{29}$ to be 11. □

The Index Calculus Method

The index calculus method for computing discrete logs bears considerable resemblence to many of the best factoring algorithms. We give a very brief overview in this section. The method uses a *factor base*, which, as before, is a set $\mathcal{B}$ of "small" primes. Suppose $\mathcal{B} = \{p_1, p_2, \ldots, p_B\}$. The first step (a preprocessing step) is to find the logarithms of the B primes in the factor base. The second step is to compute a discrete log of a desired element β, using the knowledge of the discrete logs of the elements in the factor base.

In the precomputation, we construct $C = B + 10$ congruences modulo p, as follows:

$$\alpha^{x_j} \equiv {p_1}^{a_{1j}} {p_2}^{a_{2j}} \ldots {p_B}^{a_{Bj}} \pmod{p},$$

$1 \leq j \leq C$. Notice these congruences can be written equivalently as

$$x_j \equiv a_{1j} \log_\alpha p_1 + \ldots + a_{Bj} \log_\alpha p_B \pmod{p-1},$$

$1 \leq j \leq C$. Given C congruences in the B "unknowns" $\log_\alpha p_i$ $(1 \leq i \leq B)$, we hope that there is a unique solution modulo $p-1$. If this is the case, then we can compute the logarithms of the elements in the factor base.

How do we generate congruences of the desired form? One elementary way is to take a random value x, compute $\alpha^x \bmod p$, and then determine if $\alpha^x \bmod p$ has all its factors in $\mathcal{B}$ (using trial division, for example).

Now, given that we have already successfully carried out the precomputation step, we compute a desired logarithm $\log_\alpha \beta$ by means of a Las Vegas type probabilistic algorithm. Choose a random integer s $(1 \leq s \leq p-2)$ and compute

$$\gamma = \beta \alpha^s \bmod p.$$

Now attempt to factor γ over the factor base $\mathcal{B}$. If this can be done, then we obtain a congruence of the form

$$\beta \alpha^s \equiv {p_1}^{c_1} {p_2}^{c_2} \ldots {p_B}^{c_B} \pmod{p}.$$

This can be written equivalently as

$$\log_\alpha \beta + s \equiv c_1 \log_\alpha p_1 + \ldots + c_B \log_\alpha p_B \pmod{p-1}.$$

Since everything is now known except $\log_\alpha \beta$, we can easily solve for $\log_\alpha \beta$.

Here is a small, very artificial, example to illustrate the two steps in the algorithm.

Example 5.4

Suppose $p = 10007$ and $\alpha = 5$ is the primitive element used as the base of logarithms modulo p. Suppose we take $\mathcal{B} = \{2, 3, 5, 7\}$ as the factor base. Of course $\log_5 5 = 1$, so there are three logs of factor base elements to be determined.

Some examples of "lucky" exponents that might be chosen are 4063, 5136 and 9865.

With $x = 4063$, we compute

$$5^{4063} \bmod 10007 = 42 = 2 \times 3 \times 7.$$

This yields the congruence

$$\log_5 2 + \log_5 3 + \log_5 7 \equiv 4063 \pmod{10006}.$$

Similarly, since

$$5^{5136} \bmod 10007 = 54 = 2 \times 3^3$$

and

$$5^{9865} \bmod 10007 = 189 = 3^3 \times 7,$$

we obtain two more congruences:

$$\log_5 2 + 3\log_5 3 \equiv 5136 \pmod{10006}$$

and

$$3\log_5 3 + \log_5 7 \equiv 9865 \pmod{10006}.$$

We now have three congruences in three unknowns, and there happens to be a unique solution modulo 10006, namely $\log_5 2 = 6578$, $\log_5 3 = 6190$ and $\log_5 7 = 1301$.

Now, let's suppose that we wish to find $\log_5 9451$. Suppose we choose the "random" exponent $s = 7736$, and compute

$$9451 \times 5^{7736} \bmod 10007 = 8400.$$

Since $8400 = 2^4 3^1 5^2 7^1$ factors over $\mathcal{B}$, we obtain

$$\begin{aligned} \log_5 9451 &= 4\log_5 2 + \log_5 3 + 2\log_5 5 + \log_5 7 - s \bmod 10006 \\ &= 4 \times 6578 + 6190 + 2 \times 1 + 1301 - 7736 \bmod 10006 \\ &= 6057. \end{aligned}$$

To verify, we can check that $5^{6057} \equiv 9451 \pmod{10007}$. □

Heuristic analyses of various versions of the algorithm have been done. Under reasonable assumptions, the asymptotic running time of the precomputation phase is $O\left(e^{(1+o(1))\sqrt{\ln p \ln\ln p}}\right)$, and the time to find an individual discrete log is $O\left(e^{(1/2+o(1))\sqrt{\ln p \ln\ln p}}\right)$.

FIGURE 5.5
***i*th bit of discrete logarithm**

Problem Instance $I = (p, \alpha, \beta, i)$, where p is prime, $\alpha \in \mathbb{Z}_p^*$ is a primitive element, $\beta \in \mathbb{Z}_p^*$, and i is an integer such that $1 \leq i \leq \lceil \log_2(p-1) \rceil$.
Objective Compute $L_i(\beta)$, which (for the specified α and p) denotes the ith least significant bit of $\log_\alpha \beta$.

5.1.2 Bit Security of Discrete Logs

We now look at the question of partial information about discrete logs. In particular, we consider whether individual bits of a discrete logarithm are easy or hard to compute. To be precise, consider the problem presented in Figure 5.5, which we call the ***i*th Bit** problem.

We will first show that computing the least significant bit of a discrete logarithm is easy. In other words, if $i = 1$, the ***i*th Bit** problem can be solved efficiently. This follows from Euler's criterion concerning quadratic residues modulo p, where p is prime.

Consider the mapping $f : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ defined by

$$f(x) = x^2 \bmod p.$$

Denote by $\mathrm{QR}(p)$ the set of quadratic residues modulo p; then

$$\mathrm{QR}(p) = \{x^2 \bmod p : x \in \mathbb{Z}_p^*\}.$$

First, observe that $f(x) = f(p-x)$. Next note that

$$w^2 \equiv x^2 \pmod p$$

if and only if

$$p \mid (w-x)(w+x),$$

which happens if and only if

$$w \equiv \pm x \pmod p.$$

It follows that

$$|f^{-1}(y)| = 2$$

for every $y \in \mathrm{QR}(p)$, and hence

$$|\mathrm{QR}(p)| = \frac{p-1}{2}.$$

That is, exactly half the residues in $\mathbb{Z}_p^*$ are quadratic residues and half are not.

Now, suppose α is a primitive element of $\mathbb{Z}_p$. Then $\alpha^a \in \mathrm{QR}(p)$ if a is even. Since the $(p-1)/2$ elements $\alpha^0 \bmod p, \alpha^2 \bmod p, \ldots, \alpha^{p-3} \bmod p$ are all distinct, it follows that

$$\mathrm{QR}(p) = \{\alpha^{2i} \bmod p : 0 \leq i \leq (p-3)/2\}.$$

Hence, β is a quadratic residue if and only if $\log_\alpha \beta$ is even, that is, if and only if $L_1(\beta) = 0$. But we already know, by Euler's criterion, that β is a quadratic residue if and only if

$$\beta^{(p-1)/2} \equiv 1 \pmod{p}.$$

So we have the following efficient formula to calculate $L_1(\beta)$:

$$L_1(\beta) = \begin{cases} 0 & \text{if } \beta^{(p-1)/2} \equiv 1 \pmod{p} \\ 1 & \text{otherwise.} \end{cases}$$

Let's now consider the computation of $L_i(\beta)$ for values of i exceeding 1. Suppose

$$p - 1 = 2^s t$$

where t is odd. Then it can be shown that it is easy to compute $L_i(\beta)$ if $i \leq s$. On the other hand, computing $L_{s+1}(\beta)$ is (probably) difficult, in the sense that any hypothetical algorithm (or oracle) to compute $L_{s+1}(\beta)$ could be used to find discrete logarithms in $\mathbb{Z}_p$.

We shall prove this result in the case $s = 1$. More precisely, if $p \equiv 3 \pmod 4$ is prime, then we show how any oracle for computing $L_2(\beta)$ can be used to solve the **Discrete Log** problem in $\mathbb{Z}_p$.

Recall that, if β is a quadratic residue in $\mathbb{Z}_p$ and $p \equiv 3 \pmod 4$, then $\pm\beta^{(p+1)/4} \bmod p$ are the two square roots of β modulo p. It is also important that, for any $\beta \neq 0$,

$$L_1(\beta) \neq L_1(p - \beta)$$

if $p \equiv 3 \pmod 4$. We see this as follows. Suppose

$$\alpha^a \equiv \beta \pmod{p};$$

then

$$\alpha^{a+(p-1)/2} \equiv -\beta \pmod{p}.$$

Since $p \equiv 3 \pmod 4$, the integer $(p-1)/2$ is odd, and the result follows.

Now, suppose that $\beta = \alpha^a$ for some (unknown) even exponent a. Then either

$$\beta^{(p+1)/4} \equiv \alpha^{a/2} \pmod{p}$$

or

$$-\beta^{(p+1)/4} \equiv \alpha^{a/2} \pmod{p}.$$

FIGURE 5.6
Computing discrete logs in $\mathbb{Z}_p$ for $p \equiv 3 \pmod 4$, given an oracle for $L_2(\beta)$

```
1.  x_0 = L_1(β)
2.  β = β/α^{x_0} mod p
3.  i = 1
4.  while β ≠ 1 do
5.      x_i = L_2(β)
6.      γ = β^{(p+1)/4} mod p
7.      if L_1(γ) = x_i then
8.          β = γ
9.      else
10.         β = p − γ
11.     β = β/α^{x_i} mod p
12.     i = i + 1
```

We can determine which of these two possibilities is correct if we know the value $L_2(\beta)$, since

$$L_2(\beta) = L_1(\alpha^{a/2}).$$

This fact is exploited in our algorithm, which we present in Figure 5.6.

At the end of the algorithm, the x_i's comprise the bits in the binary representation of $\log_\alpha \beta$; that is,

$$\log_\alpha \beta = \sum_{i \geq 0} x_i 2^i.$$

We will work out a small example to illustrate the algorithm.

Example 5.5

Suppose $p = 19$, $\alpha = 2$ and $\beta = 6$. Since the example is so small, we can tabulate the values of $L_1(\gamma)$ and $L_2(\gamma)$ for all $\gamma \in \mathbb{Z}_{19}^*$. (In general, L_1 can be computed efficiently using Euler's criterion and L_2 is an oracle.) These values are given in Table 5.1. The algorithm now proceeds as shown in Figure 5.7.

Hence, $\log_2 6 = 1110_2 = 14$, as can easily be verified. □

TABLE 5.1
Values of L_1 and L_2 for $p = 19, \alpha = 2$

γ	$L_1(\gamma)$	$L_2(\gamma)$	γ	$L_1(\gamma)$	$L_2(\gamma)$	γ	$L_1(\gamma)$	$L_2(\gamma)$
1	0	0	7	0	1	13	1	0
2	1	0	8	1	1	14	1	1
3	1	0	9	0	0	15	1	1
4	0	1	10	1	0	16	0	0
5	0	0	11	0	0	17	0	1
6	0	1	12	1	1	18	1	0

FIGURE 5.7
Computation of $\log_2 6$ in $\mathbb{Z}_{19}$

1.	$x_0 = 0$
2.	$\beta = 6$
3.	$i = 1$
5.	$x_1 = L_2(6) = 1$
6.	$\gamma = 5$
7.	$L_1(5) = 0 \neq x_1$
10.	$\beta = 14$
11.	$\beta = 7$
12.	$i = 2$
5.	$x_2 = L_2(7) = 1$
6.	$\gamma = 11$
7.	$L_1(11) = 0 \neq x_2$
10.	$\beta = 8$
11.	$\beta = 4$
12.	$i = 3$
5.	$x_3 = L_2(4) = 1$
6.	$\gamma = 17$
7.	$L_1(17) = 0 \neq x_3$
10.	$\beta = 2$
11.	$\beta = 1$
12.	$i = 4$
4.	DONE

It is possible to give formal proof of the algorithm's correctness using mathematical induction. Denote

$$x = \log_\alpha \beta = \sum_{i \geq 0} x_i 2^i.$$

For $i \geq 0$, define

$$Y_i = \left\lfloor \frac{x}{2^{i+1}} \right\rfloor.$$

Also, define β_0 to be the value of β in step 2 of the algorithm; and, for $i \geq 1$, define β_i to be the value of β in step 11 during the ith iteration of the **while** loop. It can be proved by induction that

$$\beta_i \equiv \alpha^{2Y_i} \pmod{p}$$

for all $i \geq 0$. Now, with the observation that

$$2Y_i = Y_{i-1} - x_i,$$

it follows that

$$x_{i+1} = L_2(\beta_i),$$

$i \geq 0$. Since

$$x_{i+1} = L_2(\beta),$$

the algorithm is correct. The details are left to the reader.

5.2 Finite Field and Elliptic Curve Systems

We have spent a considerable amount of time looking at the **Discrete Logarithm** problem and the factoring. We will see these two problems again and again, underlying various types of cryptosystems and cryptographic protocols. So far, we have considered the **Discrete Logarithm** problem in the finite field $\mathbb{Z}_p$, but it is also useful to consider the problem in other settings. This is the theme of this section.

The **ElGamal Cryptosystem** can be implemented in any group where the **Discrete Log** problem is intractible. We used the multiplicative group $\mathbb{Z}_p{}^*$, but other groups are also suitable candidates. First, we phrase the **Discrete Logarithm** problem in a general (finite) group G, where we will denote the group operation by $\circ$. This generalized version of the problem is presented in Figure 5.8.

It is easy to define an **ElGamal Cryptosystem** in the subgroup H in a similar fashion as it was originally described in $\mathbb{Z}_p{}^*$. This is done in Figure 5.9. Note that encryption requires the use of a random integer k such that $0 \leq k \leq |H| - 1$. However, if Alice does not know the order of the subgroup H, she can generate

FIGURE 5.8
The discrete logarithm problem in $(G, \circ)$

Problem Instance $I = (G, \alpha, \beta)$, where G is a finite group with group operation $\circ$, $\alpha \in G$ and $\beta \in H$, where $H = \{\alpha^i : i \geq 0\}$ is the subgroup generated by α.

Objective Find the unique integer a such that $0 \leq a \leq |H| - 1$ and $\alpha^a = \beta$, where the notation α^a means

$$\underbrace{\alpha \circ \ldots \circ \alpha}_{a \text{ times}}.$$

We will denote this integer a by $\log_\alpha \beta$.

an integer k such that $0 \leq k \leq |G| - 1$, and encryption and decryption will work without any changes. Also note that the group G need not be an abelian group (of course H is abelian since it is cyclic).

Let's now turn to the "generalized" **Discrete Log** problem. The subgroup H generated by any $\alpha \in G$ is of course a cyclic group of order $|H|$. So any version of the problem is equivalent, in some sense, to the **Discrete Log** problem in a cyclic group. However, the difficulty of the **Discrete Log** problem seems to depend in an essential way on the representation of the group that is used.

As an example to illustrate a representation where the problem is easy to solve, consider the additive cyclic group $\mathbb{Z}_n$, and suppose $\gcd(\alpha, n) = 1$, so α is a generator of $\mathbb{Z}_n$. Since the group operation is addition modulo n, an "exponentiation" operation, α^a, corresponds to muliplication by a modulo n. Hence, in this setting, the **Discrete Log** problem is to find the integer a such that

$$\alpha a \equiv \beta \pmod{n}.$$

Since $\gcd(\alpha, n) = 1$, α has a multiplicative inverse modulo n, and we can compute $\alpha^{-1} \bmod n$ easily using the Euclidean algorithm. Then we can solve for a, obtaining

$$\log_\alpha \beta = \beta\alpha^{-1} \bmod n.$$

We previously discussed the **Discrete Log** problem in the multiplicative group $\mathbb{Z}_p^*$, where p is prime. This group is a cyclic group of order $p - 1$, and hence it is isomorphic to the additive group $\mathbb{Z}_{p-1}$. By the discussion above, we know how to compute discrete logs efficiently in this additive group. This suggests that we could solve the **Discrete Log** problem in $\mathbb{Z}_p^*$ by "reducing" the problem to the the easily solved formulation in $\mathbb{Z}_{p-1}$.

Let us think about how this could be done. The statement that $(\mathbb{Z}_p^*, \times)$ is

FIGURE 5.9
Generalized ElGamal Public-key Cryptosystem

Let G be a finite group with group operation $\circ$, and let $\alpha \in G$ be an element such that the discrete log problem in H is intractible, where $H = \{\alpha^i : i \geq 0\}$ is the subgroup generated by α. Let $\mathcal{P} = G$, $\mathcal{C} = G \times G$, and define

$$\mathcal{K} = \{(G, \alpha, a, \beta) : \beta = \alpha^a\}.$$

The values α and β are public, and a is secret.

For $K = (G, \alpha, a, \beta)$, and for a (secret) random number $k \in \mathbb{Z}_{|H|}$, define

$$e_K(x, k) = (y_1, y_2),$$

where

$$y_1 = \alpha^k$$

and

$$y_2 = x \circ \beta^k.$$

For a ciphertext $y = (y_1, y_2)$, define

$$d_K(y) = y_2 \circ (y_1{}^a)^{-1}.$$

isomorphic to $(\mathbb{Z}_{p-1}, +)$ means that there is a bijection

$$\phi : \mathbb{Z}_p{}^* \to \mathbb{Z}_{p-1}$$

such that

$$\phi(xy \bmod p) = (\phi(x) + \phi(y)) \bmod (p-1).$$

It follows easily that

$$\phi(\alpha^a \bmod p) = a\phi(\alpha) \bmod (p-1),$$

so we have that

$$\beta \equiv \alpha^a \pmod p \Leftrightarrow a\phi(\alpha) \equiv \phi(\beta) \pmod{p-1}.$$

Hence, solving for a as described above, we have that

$$\log_\alpha \beta = \phi(\beta)(\phi(\alpha))^{-1} \bmod (p-1).$$

Consequently, if we have an efficient method of computing the isomorphism ϕ, then we would have an efficient algorithm to compute discrete logs in $\mathbb{Z}_p^*$. The catch is that there is no known general method to efficiently compute the isomorphism ϕ for an arbitrary prime p. Even though we know the two groups in question are isomorphic, we do not know an efficient algorithm to explicitly describe the isomorphism.

This method can be applied to the **Discrete Log** problem in any group G. If there is an efficient method of computing the isomorphism between H and $\mathbb{Z}_{|H|}$, then the discrete log problem in G described above can be solved efficiently. Conversely, it is not hard to see that an efficient method of computing discrete logs yields an efficient algorithm to compute the isomorphism between the two groups.

This discussion has shown that the **Discrete Log** problem may be easy or (apparently) difficult, depending on the representation of the (cyclic) group that is used. So it may be useful to look at other groups in the hope of finding other settings where the **Discrete Log** problem seems to be intractible.

Two such classes of groups are

1. the multiplicative group of the Galois field $\mathrm{GF}(p^n)$
2. the group of an elliptic curve defined over a finite field.

We will discuss these two classes of groups in the next subsections.

5.2.1 Galois Fields

We have already discussed the fact that $\mathbb{Z}_p$ is a field if p is prime. However, there are other examples of finite fields not of this form. In fact, there is a finite field with q elements if $q = p^n$ where p is prime and $n \geq 1$ is an integer. We will now describe very briefly how to construct such a field. First, we need several definitions.

DEFINITION 5.1 *Suppose p is prime. Define $\mathbb{Z}_p[x]$ to be the set of all polynomials in the indeterminate x. By defining addition and multiplication of polynomials in the usual way (and reducing coefficients modulo p), we construct a ring.*

*For $f(x), g(x) \in \mathbb{Z}_p[x]$, we say that $f(x)$ **divides** $g(x)$ (notation: $f(x) \mid g(x)$) if there exists $q(x) \in \mathbb{Z}_p[x]$ such that*

$$g(x) = q(x)f(x).$$

*For $f(x) \in \mathbb{Z}_p[x]$, define **deg**(f), the degree of f, to be the highest exponent in a term of f.*

Suppose $f(x), g(x), h(x) \in \mathbb{Z}_p[x]$, and $\deg(f) = n \geq 1$. We define

$$g(x) \equiv h(x) \pmod{f(x)}$$

if

$$f(x) \mid (g(x) - h(x)).$$

Notice the resemblance of the definition of congruence of polynomials to that of congruence of integers.

We are now going to define a ring of polynomials "modulo $f(x)$" which we denote by $\mathbb{Z}_p[x]/(f(x))$. The construction of $\mathbb{Z}_p[x]/(f(x))$ from $\mathbb{Z}_p[x]$ is based on the idea of congruences modulo $f(x)$ and is analogous to the construction of $\mathbb{Z}_m$ from $\mathbb{Z}$.

Suppose $deg(f) = n$. If we divide $g(x)$ by $f(x)$, we obtain a (unique) *quotient* $q(x)$ and *remainder* $r(x)$, where

$$g(x) = q(x)f(x) + r(x)$$

and

$$deg(r) < n.$$

This can be done by usual long division of polynomials. Hence any polynomial in $\mathbb{Z}_p[x]$ is congruent modulo $f(x)$ to a unique polynomial of degree at most $n - 1$.

Now we define the elements of $\mathbb{Z}_p[x]/(f(x))$ to be the p^n polynomials in $\mathbb{Z}_p[x]$ of degree at most $n - 1$. Addition and multiplication in $\mathbb{Z}_p[x]/(f(x))$ is defined as in $\mathbb{Z}_p[x]$, followed by a reduction modulo $f(x)$. Equipped with these operations, $\mathbb{Z}_p[x]/(f(x))$ is a ring.

Recall that $\mathbb{Z}_m$ is a field if and only if m is prime, and multiplicative inverses can be found using the Euclidean algorithm. A similar situation holds for $\mathbb{Z}_p[x]/(f(x))$. The analog of primality for polynomials is irreducibility, which we define as follows:

DEFINITION 5.2 *A polynomial $f(x) \in \mathbb{Z}_p[x]$ is said to be* ***irreducible*** *if there do not exist polynomials $f_1(x), f_2(x) \in \mathbb{Z}_p[x]$ such that*

$$f(x) = f_1(x)f_2(x),$$

where $deg(f_1) > 0$ and $deg(f_2) > 0$.

A very important fact is that $\mathbb{Z}_p[x]/(f(x))$ is a field if and only if $f(x)$ is irreducible. Further, multiplicative inverses in $\mathbb{Z}_p[x]/(f(x))$ can be computed using a straightforward modification of the (extended) Euclidean algorithm.

Here is an example to illustrate the concepts described above.

Example 5.6

Let's attempt to construct a field having eight elements. This can be done by finding an irreducible polynomial of degree three in $\mathbb{Z}_2[x]$. It is sufficient to consider the polynomials having constant term equal to 1, since any polynomial with constant term 0 is divisible by x and hence is reducible. There are four such polynomials:

$$f_1(x) = x^3 + 1$$
$$f_2(x) = x^3 + x + 1$$
$$f_3(x) = x^3 + x^2 + 1$$
$$f_4(x) = x^3 + x^2 + x + 1.$$

Now, $f_1(x)$ is reducible, since

$$x^3 + 1 = (x + 1)(x^2 + x + 1)$$

(remember that all coefficients are to be reduced modulo 2). Also, f_4 is reducible since

$$x^3 + x^2 + x + 1 = (x + 1)(x^2 + 1).$$

However, $f_2(x)$ and $f_3(x)$ are both irreducible, and either one can be used to construct a field having eight elements.

Let us use $f_2(x)$, and thus construct the field $\mathbb{Z}_2[x]/(x^3 + x + 1)$. The eight field elements are the eight polynomials 0, 1, x, $x + 1$, x^2, $x^2 + 1$, $x^2 + x$ and $x^2 + x + 1$.

To compute a product of two field elements, we multiple the two polynomials together, and reduce modulo $x^3 + x + 1$ (i.e., divide by $x^3 + x + 1$ and find the remainder polynomial). Since we are dividing by a polynomial of degree three, the remainder will have degree at most two and hence is an element of the field.

For example, to compute $(x^2 + 1)(x^2 + x + 1)$ in $\mathbb{Z}_2[x]/(x^3 + x + 1)$, we first compute the product in $\mathbb{Z}_2[x]$, which is $x^4 + x^3 + x + 1$. Then we divide by $x^3 + x + 1$, obtaining the expression

$$x^4 + x^3 + x + 1 = (x + 1)(x^3 + x + 1) + x^2 + x.$$

Hence, in the field $\mathbb{Z}_2[x]/(x^3 + x + 1)$, we have that

$$(x^2 + 1)(x^2 + x + 1) = x^2 + x.$$

Below, we present a complete multiplication table for the non-zero field elements. To save space, we write a polynomial $a_2x^2+a_1x+a_0$ as the ordered triple $a_2a_1a_0$.

	001	010	011	100	101	110	111
001	001	010	011	100	101	110	111
010	010	100	110	011	001	111	101
011	011	110	101	111	100	001	010
100	100	011	111	110	010	101	001
101	101	001	100	010	111	011	110
110	110	111	001	101	011	010	100
111	111	101	010	001	110	100	011

Computation of inverses can be done by using a straightforward adaptation of the extended Euclidean algorithm.

Finally, the multiplicative group of the non-zero polynomials in the field is a cyclic group of order seven. Since 7 is prime, it follows that any non-zero field element is a generator of this group, i.e., a primitive element of the field.

For example, if we compute the powers of x, we obtain

$$\begin{aligned} x^1 &= x \\ x^2 &= x^2 \\ x^3 &= x+1 \\ x^4 &= x^2+x \\ x^5 &= x^2+x+1 \\ x^6 &= x^2+1 \\ x^7 &= 1, \end{aligned}$$

which comprise all the non-zero field elements. □

It remains to discuss existence and uniqueness of fields of this type. It can be shown that there is at least one irreducible polynomial of any given degree $n \geq 1$ in $\mathbb{Z}_p[x]$. Hence, there is a finite field with p^n elements for all primes p and all integers $n \geq 1$. There are usually many irreducible polynomials of degree n in $\mathbb{Z}_p[x]$. But the finite fields constructed from any two irreducible polynomials of degree n can be shown to be isomorphic. Thus there is a unique finite field of any size p^n (p prime, $n \geq 1$), which is denoted by $\mathrm{GF}(p^n)$. In the case $n = 1$, the resulting field $\mathrm{GF}(p)$ is the same thing as $\mathbb{Z}_p$. Finally, it can be shown that there does not exist a finite field with r elements unless $r = p^n$ for some prime p and some integer $n \geq 1$.

We have already noted that the multiplicative group $\mathbb{Z}_p{}^*$ (p prime) is a cyclic group of order $p-1$. In fact, the multiplicative group of any finite field is cyclic:

$GF(p^n)\backslash\{0\}$ is a cyclic group of order $p^n - 1$. This provides further examples of cyclic groups in which the discrete log problem can be studied.

In practice, the finite fields $GF(2^n)$ have been most studied. Both the Shanks and Pohlig-Hellman discrete logarithm algorithms work for fields $GF(2^n)$. The index calculus method can be modified to work in these fields. The precomputation time of the index calculus algorithm turns out to be $O\left(e^{(1.405+o(1))n^{1/3}(\ln n)^{2/3}}\right)$, and the time to find an individual discrete log is $O\left(e^{(1.098+o(1))n^{1/3}(\ln n)^{2/3}}\right)$. However, for large values of n (say $n > 800$), the discrete log problem in $GF(2^n)$ is thought to be intractible provided 2^n has at least one "large" prime factor (in order to thwart a Pohlig-Hellman attack).

5.2.2 Elliptic Curves

We begin by defining the concept of an elliptic curve.

DEFINITION 5.3 *Let $p > 3$ be prime. The **elliptic curve** $y^2 = x^3 + ax + b$ over $\mathbb{Z}_p$ is the set of solutions $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ to the congruence*

$$y^2 \equiv x^3 + ax + b \pmod{p}, \tag{5.1}$$

*where $a, b \in \mathbb{Z}_p$ are constants such that $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, together with a special point $\mathcal{O}$ called the **point at infinity**.*[1]

An elliptic curve E can be made into an abelian group by defining a suitable operation on its points. The operation is written additively, and is defined as follows (where all arithmetic operations are performed in $\mathbb{Z}_p$): Suppose

$$P = (x_1, y_1)$$

and

$$Q = (x_2, y_2)$$

are points on E. If $x_2 = x_1$ and $y_2 = -y_1$, then $P + Q = \mathcal{O}$; otherwise $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1,$$

[1] Equation 5.1 can be used to define an elliptic curve over any field $GF(p^n)$, for $p > 3$ prime. An elliptic curve over $GF(2^n)$ or $GF(3^n)$ is defined by a slightly different equation.

and

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{if } P = Q. \end{cases}$$

Finally, define

$$P + \mathcal{O} = \mathcal{O} + P = P$$

for all $P \in E$. With this definition of addition, it can be shown that E is an abelian group with identity element $\mathcal{O}$ (most of the verifications are tedious but straightforward, but proving associativity is quite difficult).

Note that inverses are very easy to compute. The inverse of (x, y) (which we write as $-(x, y)$ since the group operation is additive) is $(x, -y)$, for all $(x, y) \in E$.

Let us look at a small example.

Example 5.7

Let E be the elliptic curve $y^2 = x^3 + x + 6$ over $\mathbb{Z}_{11}$. Let's first determine the points on E. This can be done by looking at each possible $x \in \mathbb{Z}_{11}$, computing $x^3 + x + 6 \bmod 11$, and then trying to solve Equation 5.1 for y. For a given x we can test to see if $z = x^3 + x + 6 \bmod 11$ is a quadratic residue by applying Euler's criterion. Recall that there is an explicit formula to compute square roots of quadratic residues modulo p for primes $p \equiv 3 \pmod 4$. Applying this formula, we have that the square roots of a quadratic residue z are

$$\pm z^{(11+1)/4} \bmod 11 = \pm z^3 \bmod 11.$$

The results of these computations are tabulated in Table 5.2.

Thus E has 13 points on it. Since any group of prime order is cyclic, it follows that E is isomorphic to $\mathbb{Z}_{13}$, and any point other than the point at infinity is a generator of E. Suppose we take the generator $\alpha = (2, 7)$. Then we can compute the "powers" of α (which we will write as multiples of α, since the group operation is additive). To compute $2\alpha = (2, 7) + (2, 7)$, we first compute

$$\begin{aligned} \lambda &= (3 \times 2^2 + 1)(2 \times 7)^{-1} \bmod 11 \\ &= 2 \times 3^{-1} \bmod 11 \\ &= 2 \times 4 \bmod 11 \\ &= 8. \end{aligned}$$

Then we have

$$\begin{aligned} x_3 &= 8^2 - 2 - 2 \bmod 11 \\ &= 5 \end{aligned}$$

TABLE 5.2
Points on the elliptic curve $y^2 = x^3 + x + 6$ over $\mathbb{Z}_{11}$

x	$x^3 + x + 6 \bmod 11$	in QR(11)?	y
0	6	no	
1	8	no	
2	5	yes	4,7
3	3	yes	5,6
4	8	no	
5	4	yes	2,9
6	8	no	
7	4	yes	2,9
8	9	yes	3,8
9	7	no	
10	4	yes	2,9

and

$$\begin{aligned} y_3 &= 8(2-5) - 7 \bmod 11 \\ &= 2, \end{aligned}$$

so $2\alpha = (5, 2)$.

The next multiple would be $3\alpha = 2\alpha + \alpha = (5, 2) + (2, 7)$. Again, we begin by computing λ, which in this situation is done as follows:

$$\begin{aligned} \lambda &= (7-2)(2-5)^{-1} \bmod 11 \\ &= 5 \times 8^{-1} \bmod 11 \\ &= 5 \times 7 \bmod 11 \\ &= 2. \end{aligned}$$

Then we have

$$\begin{aligned} x_3 &= 2^2 - 5 - 2 \bmod 11 \\ &= 8 \end{aligned}$$

and

$$\begin{aligned} y_3 &= 2(5-8) - 2 \bmod 11 \\ &= 3, \end{aligned}$$

so $3\alpha = (8, 3)$.

Continuing in this fashion, the remaining multiples can be computed to be the following:

$$\begin{array}{lllllllll} \alpha & = & (2,7) & 2\alpha & = & (5,2) & 3\alpha & = & (8,3) \\ 4\alpha & = & (10,2) & 5\alpha & = & (3,6) & 6\alpha & = & (7,9) \\ 7\alpha & = & (7,2) & 8\alpha & = & (3,5) & 9\alpha & = & (10,9) \\ 10\alpha & = & (8,8) & 11\alpha & = & (5,9) & 12\alpha & = & (2,4) \end{array}$$

Hence $\alpha = (2,7)$ is indeed a primitive element. □

An elliptic curve E defined over $\mathbb{Z}_p$ (p prime, $p > 3$)) will have roughly p points on it. More precisely, a well-known theorem due to Hasse asserts that the number of points on E, which we denote by $\#E$, satisfies the following inequality

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}.$$

Computing the exact value of $\#E$ is more difficult, but there is an efficient algorithm to do this, due to Schoof. (By "efficient" we mean that it has a running time that is polynomial in $\log p$. Schoof's algorithm has a running time of $O((\log p)^8)$ bit operations and is practical for primes p having several hundred digits.)

Now, given that we can compute $\#E$, we further want to find a cyclic subgroup of E in which the discrete log problem is intractible. So we would like to know something about the structure of the group E. The following theorem gives a considerable amount of information on the group structure of E.

THEOREM 5.1
Let E be an elliptic curve defined over $\mathbb{Z}_p$, where p is prime, $p > 3$. Then there exist integers n_1 and n_2 such that E is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. Further, $n_2 \mid n_1$ and $n_2 \mid (p-1)$.

Hence, if the integers n_1 and n_2 can be computed, then we know that E has a cyclic subgroup isomorphic to $\mathbb{Z}_{n_1}$ that can potentially be used as a setting for an **ElGamal Cryptosystem**.

Note that if $n_2 = 1$, then E is a cyclic group. Also, if $\#E$ is a prime, or the product of distinct primes, then E must be a cyclic groupindexcyclic group.

The Shanks and Pohlig-Hellman algorithms apply to the elliptic curve logarithm problem, but there is no known adaptation of the index calculus method to elliptic curves. However, there is a method of exploiting an explicit isomorphism between elliptic curves and finite fields that leads to efficient algorithms for certain classes of elliptic curves. This technique, due to Menezes, Okamoto and Vanstone, can be applied to some particular examples within a special class of elliptic curves called supersingular curves that were suggested for use in cryptosystems. If the supersingular curves are avoided, however, then it appears that an elliptic curve having a cyclic subgroup of size 2^{160} will provide a secure setting for a

cryptosystem, provided that the order of the subgroup is divisible by at least one large prime factor (again, to guard against a Pohlig-Hellman attack).

Let's now look an example of **ElGamal** encryption using the elliptic curve of Example 5.7.

Example 5.8
Suppose that $\alpha = (2,7)$ and Bob's secret "exponent" is $a = 7$, so

$$\beta = 7\alpha = (7,2).$$

Thus the encryption operaton is

$$e_K(x,k) = (k(2,7), x + k(7,2)),$$

where $x \in E$ and $0 \leq k \leq 12$, and the decryption operation is

$$d_K(y_1,y_2) = y_2 - 7y_1,$$

Suppose that Alice wishes to encrypt the message $x = (10,9)$ (which is a point on E). If she chooses the random value $k = 3$, then she will compute

$$\begin{aligned} y_1 &= 3(2,7) \\ &= (8,3) \end{aligned}$$

and

$$\begin{aligned} y_2 &= (10,9) + 3(7,2) \\ &= (10,9) + (3,5) \\ &= (10,2). \end{aligned}$$

Hence, $y = ((8,3),(10,2))$. Now, if Bob receives the ciphertext y, he decrypts it as follows:

$$\begin{aligned} x &= (10,2) - 7(8,3) \\ &= (10,2) - (3,5) \\ &= (10,2) + (3,6) \\ &= (10,9). \end{aligned}$$

Hence, the decryption yields the correct plaintext. □

There are some practical difficulties in implementing an **ElGamal Cryptosystem** on an elliptic curve. This system, when implemented in $\mathbb{Z}_p$ (or in GF(p^n) with $n > 1$) has a message expansion factor of two. An elliptic curve implementation has a message expansion factor of (about) four. This happens since there

FIGURE 5.10
Menezes-Vanstone Elliptic Curve Cryptosystem

Let E be an elliptic curve defined over $\mathbb{Z}_p$ ($p > 3$ prime) such that E contains a cyclic subgroup H in which the discrete log problem is intractible.
Let $\mathcal{P} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $\mathcal{C} = E \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and define

$$\mathcal{K} = \{(E, \alpha, a, \beta) : \beta = a\alpha\},$$

where $\alpha \in E$. The values α and β are public, and a is secret.

For $K = (E, \alpha, a, \beta)$, for a (secret) random number $k \in \mathbb{Z}_{|H|}$, and for $x = (x_1, x_2) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, define

$$e_K(x, k) = (y_0, y_1, y_2),$$

where

$$\begin{aligned} y_0 &= k\alpha, \\ (c_1, c_2) &= k\beta, \\ y_1 &= c_1 x_1 \bmod p, \qquad \text{and} \\ y_2 &= c_2 x_2 \bmod p. \end{aligned}$$

For a ciphertext $y = (y_0, y_1, y_2)$, define

$$d_K(y) = (y_1 {c_1}^{-1} \bmod p, y_2 {c_2}^{-1} \bmod p),$$

where

$$ay_0 = (c_1, c_2).$$

are approximately p plaintexts, but each ciphertext consists of four field elements. A more serious problem is that the plaintext space consists of the points on the curve E, and there is no convenient method known of deterministically generating points on E.

A more efficient variation has been found by Menezes and Vanstone. In this variation, the elliptic curve is used for "masking," and plaintexts and ciphertexts are allowed to be arbitrary ordered pairs of (nonzero) field elements (i.e., they are not required to be points on E). This yields a message expansion factor of two, the same as in the original **ElGamal Cryptosystem**. The **Menezes-Vanstone Cryptosystem** is presented in Figure 5.10.

If we return to the curve $y^2 = x^3 + x + 6$ over $\mathbb{Z}_{11}$, we see that the **Menezes-Vanstone Cryptosystem** allows $10 \times 10 = 100$ plaintexts, as compared to 13 in the original system. We illustrate encryption and decryption in this system using this same curve.

Example 5.9
As in the previous example, suppose that $\alpha = (2, 7)$ and Bob's secret "exponent" is $a = 7$, so

$$\beta = 7\alpha = (7, 2).$$

Suppose Alice wants to encrypt the plaintext

$$x = (x_1, x_2) = (9, 1)$$

(note that x is not a point on E), and she chooses the random value $k = 6$. First, she computes

$$y_0 = k\alpha = 6(2, 7) = (7, 9)$$

and

$$k\beta = 6(7, 2) = (8, 3),$$

so $c_1 = 8$ and $c_2 = 3$.

Next, she calculates

$$y_1 = c_1 x_1 \bmod p = 8 \times 9 \bmod 11 = 6$$

and

$$y_2 = c_2 x_2 \bmod p = 3 \times 1 \bmod 11 = 3.$$

The ciphertext she sends to Bob is

$$y = (y_0, y_1, y_2) = ((7, 9), 6, 3).$$

When Bob receives the ciphertext y, he first computes

$$(c_1, c_2) = ay_0 = 7(7, 9) = (8, 3),$$

and then

$$\begin{aligned} x &= (y_1 {c_1}^{-1} \bmod p, y_2 {c_2}^{-1} \bmod p) \\ &= (6 \times 8^{-1} \bmod 11, 3 \times 3^{-1} \bmod 11) \\ &= (6 \times 7 \bmod 11, 3 \times 4 \bmod 11) \\ &= (9, 1). \end{aligned}$$

Hence, the decryption yields the correct plaintext. □

FIGURE 5.11
Subset sum problem

Problem Instance $I = (s_1, \ldots s_n, T)$, where $s_1, \ldots s_n$ and T are positive integers. The s_i's are called *sizes* and T is called the *target sum.*
Question Is there a 0-1 vector $\mathbf{x} = (x_1, \ldots, x_n)$ such that $$\sum_{i=1}^{n} x_i s_i = T?$$

5.3 The Merkle-Hellman Knapsack System

The well-known **Merkle-Hellman Knapsack Cryptosystem** was first described by Merkle and Hellman in 1978. Although this cryptosystem, and several variants of it, were broken in the early 1980's, it is still worth studying for its conceptual elegance and for the underlying design technique.

The term "knapsack" is actually a misnomer[2]; the system is based on the **Subset Sum** problem which is presented in Figure 5.11.

The **Subset Sum** problem, as phrased in Figure 5.11, is a *decision problem* (i.e., we are required only to answer "yes" or "no"). If we rephrase the problem slightly, so that in any instance where the answer is "yes" we are required to find the desired vector $\mathbf{x}$ (which may not be unique), then we have a *search problem.*

The **Subset Sum** (decision) problem is one of the so-called NP-complete problems. Among other things, this means that there is no known polynomial-time algorithm that solves it. This is also the case for the **Subset Sum** search problem. But even if a problem has no polynomial-time algorithm to solve it in general, this does not rule out the possibility that certain special cases can be solved in polynomial time. This is indeed the situation with the **Subset Sum** problem.

We define a list of sizes, $(s_1, \ldots, s_n)$ to be *superincreasing* if

$$s_j > \sum_{i=1}^{j-1} s_i$$

for $2 \leq j \leq n$. If the list of sizes is superincreasing, then the search version of the **Subset Sum** problem can be solved very easily in time $O(n)$, and a solution $\mathbf{x}$ (if it exists) must be unique. The algorithm to do this is presented in Figure 5.12.

[2]The **Knapsack** problem, as it is usually defined, is a problem involving selecting objects with given weights and profits in such a way that a specified capacity is not exceeded and a specified target profit is attained.

FIGURE 5.12
Algorithm for solving a superincreasing instance of the subset sum problem

```
1.  for i = n downto 1 do
2.      if T ≥ s_i then
3.          T = T − s_i
4.          x_i = 1
5.      else
6.          x_i = 0
7.  if Σ_{i=1}^{n} x_i s_i = T then
8.      X = (x_1, ..., x_n) is the solution
9.  else
10.     there is no solution.
```

Suppose $\mathbf{s} = (s_1, \ldots, s_n)$ is superincreasing, and consider the function

$$e_{\mathbf{s}} : \{0,1\}^n \rightarrow \left\{0, \ldots, \sum_{i=1}^{n} s_i\right\}$$

defined by the rule

$$e_{\mathbf{s}}(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i s_i.$$

Is $e_{\mathbf{s}}$ a possible candidate for an encryption rule? Since $\mathbf{s}$ is superincreasing, $e_{\mathbf{s}}$ is an injection, and the algorithm presented in Figure 5.12 would be the corresponding decryption algorithm. However, such a system would be completely insecure since anyone (including Oscar) can decrypt a message that is encrypted in this way.

The strategy therefore is to transform the list of sizes in such a way that it is no longer superincreasing. Bob will be able to apply an inverse transformation to restore the superincreasing list of sizes. On the other hand Oscar, who does not know the transformation that was applied, is faced with what looks like a general, apparently difficult, instance of the subset sum problem when he tries to decrypt a ciphertext.

One suitable type of transformation is a *modular transformation*. That is, a prime modulus p is chosen such that

$$p > \sum_{i=1}^{n} s_i,$$

FIGURE 5.13
Merkle-Hellman Knapsack Cryptosystem

Let $\mathbf{s} = (s_1, \ldots, s_n)$ be a superincreasing list of integers, let $p > \sum_{i=1}^{n} s_i$ be prime, and let $1 \leq a \leq p-1$. For $1 \leq i \leq n$, define

$$t_i = as_i \bmod p,$$

and denote $\mathbf{t} = (t_1, \ldots, t_n)$. Let $\mathcal{P} = \{0,1\}^n$, $\mathcal{C} = \{0, \ldots, n(p-1)\}$, and let

$$\mathcal{K} = \{(\mathbf{s}, p, a, \mathbf{t})\},$$

where $\mathbf{s}$, p, a, and $\mathbf{t}$ are constructed as described above. $\mathbf{t}$ is public, and p, a and $\mathbf{s}$ are secret.

For $K = (\mathbf{s}, p, a, \mathbf{t})$, define

$$e_K(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i t_i.$$

For $0 \leq y \leq n(p-1)$, define $z = a^{-1}y \bmod p$ and solve the subset problem $(s_1, \ldots, s_n, z)$, obtaining $d_K(y) = (x_1, \ldots, x_n)$.

as well as a multiplier a, where $1 \leq a \leq p-1$. Then we define

$$t_i = as_i \bmod p,$$

$1 \leq i \leq n$. The list of sizes $\mathbf{t} = (t_1, \ldots, t_n)$ will be the public key used for encryption. The values a, p used to define the modular transformation are secret. The complete description of the **Merkle-Hellman Knapsack Cryptosystem** is given in Figure 5.13.

The following small example illustrates the encryption and decryption operations in the **Merkle-Hellman Cryptosystem**.

Example 5.10
Suppose

$$\mathbf{s} = (2, 5, 9, 21, 45, 103, 215, 450, 946)$$

is the secret superincreasing list of sizes. Suppose $p = 2003$ and $a = 1289$. Then the public list of sizes is

$$\mathbf{t} = (575, 436, 1586, 1030, 1921, 569, 721, 1183, 1570).$$

Now, if Alice wants to encrypt the plaintext $x = (1,0,1,1,0,0,1,1,1)$, she computes

$$y = 575 + 1586 + 1030 + 721 + 1183 + 1570 = 6665.$$

When Bob receives the ciphertext y, he first computes

$$\begin{aligned} z &= a^{-1}y \bmod p \\ &= 317 \times 6665 \bmod 2003 \\ &= 1643. \end{aligned}$$

Then Bob solves the instance $I = (\mathbf{s}, z)$ of the **Subset Sum** problem using the algorithm presented in Figure 5.12. The plaintext $(1,0,1,1,0,0,1,1,1)$ is obtained. □

By the early 1980's, the **Merkle-Hellman Knapsack Cryptosystem** had been broken by Shamir. Shamir was able to use an integer programming algorithm of Lenstra to break the system. This allows Bob's trapdoor (or an equivalent trapdoor) to be discovered by Oscar, the cryptanalyst. Then Oscar can decrypt messages exactly as Bob does.

5.4 The McEliece System

The **McEliece Cryptosystem** uses the same design principle as the **Merkle-Hellman Cryptosystem**: decryption is an easy special case of an NP-complete problem, disguised so that it looks like a general instance of the problem. In this system, the NP-complete problem that is employed is decoding a general linear (binary) error-correcting code. However, for many special classes of codes, polynomial-time algorithms are known to exist. One such class of codes, the Goppa codes, are used as the basis of the **McEliece Cryptosystem**.

We begin with some essential definitions.

DEFINITION 5.4 *Let k, n be positive integers, $k \leq n$. An $[n,k]$* ***code***, **C**, *is a k-dimensional subspace of $(\mathbb{Z}_2)^n$, the vector space of all binary n-tuples.*

A ***generating matrix*** *for an $[n,k]$ code,* **C**, *is a $k \times n$ binary matrix whose rows form a basis for* **C**.

Let $\mathbf{x}, \mathbf{y} \in (\mathbb{Z}_2)^n$, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$. Define the ***Hamming distance***

$$d(\mathbf{x}, \mathbf{y}) = |\{i : 1 \leq i \leq n, x_i \neq y_i\}|,$$

i.e., the number of coordinates in which **x** *and* **y** *differ.*

Let $\mathbf{C}$ *be an* $[n, k]$ *code. Define the* ***distance*** *of* $\mathbf{C}$ *to be the quantity*

$$d(\mathbf{C}) = \min\{d(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathbf{C}, \mathbf{x} \neq \mathbf{y}\}.$$

An $[n, k]$ *code with distance* d *is denoted as an* $[n, k, d]$ ***code.***

The purpose of an error-correcting code is to correct random errors that occur in the transmission of (binary) data through a noisy channel. Briefly, this is done as follows. Let G be a generating matrix for an $[n, k, d]$ code. Suppose $\mathbf{x}$ is the binary k-tuple we wish to transmit. Then Alice *encodes* $\mathbf{x}$ as the n-tuple $\mathbf{y} = \mathbf{x}G$, and transmits $\mathbf{y}$ through the channel.

Now, suppose Bob receives the n-tuple $\mathbf{r}$, which may not be the same as $\mathbf{y}$. He will *decode* $\mathbf{r}$ using the strategy of *nearest neighbor decoding*. In nearest neighbor decoding, Bob finds the codeword $\mathbf{y}'$ that has minimum distance to $\mathbf{r}$. Then he decodes $\mathbf{r}$ to $\mathbf{y}'$, and, finally, determines the k-tuple $\mathbf{x}'$ such that $\mathbf{y}' = \mathbf{x}'G$. Bob is hoping that $\mathbf{y}' = \mathbf{y}$, so $\mathbf{x}' = \mathbf{x}$ (i.e., he is hoping that any transmission errors have been corrected).

It is fairly easy to show that if at most $(d-1)/2$ errors occurred during transmission, then nearest neighbor decoding does in fact correct all the errors.

Let us think about how nearest neighbor decoding would be done in practice. $|\mathbf{C}| = 2^k$, so if Bob compares $\mathbf{r}$ to every codeword, he will have to examine 2^k vectors, which is an exponentially large number compared to k. In other words, this obvious algorithm is not a polynomial-time algorithm.

Another approach, which forms the basis for many practical decoding algorithms, is based on the idea of a syndrome. A *parity-check matrix* for an $[n, k, d]$ code $\mathbf{C}$ having generating matrix G is an $(n-k) \times n$ $0-1$ matrix, denoted by H, whose rows form a basis for the orthogonal complement of $\mathbf{C}$, which is denoted by $\mathbf{C}^{\perp}$ and called the *dual code* to $\mathbf{C}$. Stated another way, the rows of H are linearly independent vectors, and GH^T is a $k \times (n-k)$ matrix of zeroes.

Given a vector $\mathbf{r} \in (\mathbb{Z}_2)^n$, we define the *syndrome* of $\mathbf{r}$ to be Hr^T. A syndrome is a column vector with $n-k$ components.

The following basic results follow immediately from linear algebra.

THEOREM 5.2

Suppose $\mathbf{C}$ *is an* $[n, k]$ *code with generating matrix* G *and parity-check matrix* H. *Then* $\mathbf{x} \in (\mathbb{Z}_2)^n$ *is a codeword if and only if*

$$H\mathbf{x}^T = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Further, if $\mathbf{x} \in \mathbf{C}$, $\mathbf{e} \in (\mathbb{Z}_2)^n$, *and* $\mathbf{r} = \mathbf{x} + \mathbf{e}$, *then* $H\mathbf{r}^T = H\mathbf{e}^T$.

Think of $\mathbf{e}$ as being the vector of errors that occur during transmission of a codeword $\mathbf{x}$. Then $\mathbf{r}$ represents the vector that is received. The above theorem is saying that the syndrome depends only on the errors, and not on the particular codeword that was transmitted.

This suggests the following approach to decoding, known as *syndrome decoding*: First, compute $\mathbf{s} = H\mathbf{r}^T$. If $\mathbf{s}$ is a vector of zeroes, then decode $\mathbf{r}$ as $\mathbf{r}$. If not, then generate all possible error vectors or weight 1 in turn. For each such $\mathbf{e}$, compute $H\mathbf{e}^T$. If, for any of these vectors $\mathbf{e}$, it happens that $H\mathbf{e}^T = \mathbf{s}$, then decode $\mathbf{r}$ to $\mathbf{r} - \mathbf{e}$. Otherwise, continue on to generate all error vectors of weight $2, \ldots, \lfloor (d-1)/2 \rfloor$. If at any time $H\mathbf{e}^T = \mathbf{s}$, then we decode $\mathbf{r}$ to $\mathbf{r} - \mathbf{e}$ and quit. If this equation is never satisfied, then we conclude that more than $\lfloor (d-1)/2 \rfloor$ errors have occurred during transmission.

By this approach, we can decode a received vector in at most

$$1 + \binom{n}{1} + \ldots + \binom{n}{\lfloor (d-1)/2 \rfloor}$$

steps.

This method will work on any linear code. For certain specific types of codes, the decoding procedure can be speeded up. However, a decision version of nearest neighbor decoding is in fact an NP-complete problem. Thus no polynomial-time algorithm is known for the general problem of nearest neighbor decoding (when the number of errors is not bounded by $\lfloor (d-1)/2 \rfloor$).

As was the case with the subset sum problem, we can identify an "easy" special case, and then disguise it so that it looks like a "difficult" general case of the problem. It would take us too long to go into the theory here, so we will just summarize the results. The "easy" special case that was suggested by McEliece is to use a code from a class of codes known as the *Goppa codes*. These codes do in fact have efficient decoding algorithms. Also, they are easy to generate, and there are a large number of inequivalent Goppa codes with the same parameters.

The parameters of the Goppa codes have the form $n = 2^m$, $d = 2t + 1$ and $k = n - mt$. For a practical implementation of the public-key cryptosystem, McEliece suggested taking $m = 10$ and $t = 50$. This gives rose to a Goppa code that is a $[1024, 524, 101]$ code. Each plaintext is a binary 524-tuple, and each ciphertext is a binary 1024-tuple. The public key is a 524×1024 binary matrix.

A description of the **McEliece Cryptosystem** is given in Figure 5.14.

We presently a ridiculously small example to illustrate the encoding and decoding procedures.

FIGURE 5.14
McEliece Cryptosystem

Let G be a generating matrix for an $[n, k, d]$ Goppa code $\mathbf{C}$, where $n = 2^m$, $d = 2t + 1$ and $k = n - mt$. Let S be a $k \times k$ matrix that is invertible over $\mathbb{Z}_2$, let P be an $n \times n$ permutation matrix, and let $G' = SGP$. Let $\mathcal{P} = (\mathbb{Z}_2)^k$, $\mathcal{C} = (\mathbb{Z}_2)^n$, and let

$$\mathcal{K} = \{(G, S, P, G')\},$$

where G, S, P, and G' are constructed as described above. G' is public, and G, S, and P are secret.

For $K = (G, S, P, G')$, define

$$e_K(\mathbf{x}, \mathbf{e}) = \mathbf{x}G' + \mathbf{e},$$

where $\mathbf{e} \in (\mathbb{Z}_2)^n$ is a random vector of weight t.

Bob decrypts a ciphertext $\mathbf{y} \in (\mathbb{Z}_2)^n$ by means of the following operations:

1. Compute $\mathbf{y}_1 = \mathbf{y}P^{-1}$.
2. Decode $\mathbf{y}_1$, obtaining $\mathbf{y}_1 = \mathbf{x}_1 + \mathbf{e}_1$, where $\mathbf{x}_1 \in \mathbf{C}$.
3. Compute $\mathbf{x}_0 \in (\mathbb{Z}_2)^k$ such that $\mathbf{x}_0 G = \mathbf{x}_1$.
4. Compute $\mathbf{x} = \mathbf{x}_0 S^{-1}$.

Example 5.11

The matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

is a generating matrix for a $[7, 4, 3]$ code, known as a *Hamming code*. Suppose Bob chooses the matrices

$$S = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

and

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Then, the public generating matrix is

$$G' = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Now, suppose Alice encrypts the plaintext $\mathbf{x} = (1, 1, 0, 1)$ using as the random error vector of weight 1 the vector $\mathbf{e} = (0, 0, 0, 0, 1, 0, 0)$. The ciphertext is computed to be

$$\begin{aligned} \mathbf{y} &= \mathbf{x}G' + \mathbf{e} \\ &= (1, 1, 0, 1) \begin{pmatrix} 1\,1\,1\,1\,0\,0\,0 \\ 1\,1\,0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,1\,0\,1 \\ 0\,1\,0\,1\,1\,1\,0 \end{pmatrix} + (0, 0, 0, 0, 1, 0, 0) \\ &= (0, 1, 1, 0, 0, 1, 0) + (0, 0, 0, 0, 1, 0, 0) \\ &= (0, 1, 1, 0, 1, 1, 0). \end{aligned}$$

When Bob receives the ciphertext $\mathbf{y}$, he first computes

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{y}P^{-1} \\ &= (0, 1, 1, 0, 1, 1, 0) \begin{pmatrix} 0\,0\,0\,1\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0\,0\,0 \end{pmatrix} \\ &= (1, 0, 0, 0, 1, 1, 1). \end{aligned}$$

Next, he decrypts $\mathbf{y}_1$ to get $\mathbf{x}_1 = (1, 0, 0, 0, 1, 1, 0)$ (note that $\mathbf{e}_1 \neq \mathbf{e}$ due to the multiplication by P^{-1}).

Next, Bob forms $\mathbf{x}_0 = (1, 0, 0, 0)$ (the first four components of $\mathbf{x}_1$).

Finally, Bob calculates

$$\begin{aligned}\mathbf{x} &= S^{-1}\mathbf{x}_0 \\ &= \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} (1,0,0,0) \\ &= (1,1,0,1).\end{aligned}$$

This is indeed the plaintext that Alice encrypted. □

5.5 Notes and References

The **ElGamal Cryptosystem** was presented in [EL85]. The Pohlig-Hellman algorithm was published in [PH78], and the material concerning individual bits of the **Discrete Logarithm** problem is based on Peralta [PE86]. For further information on the **Discrete Logarithm** problem, we recommend the articles by LaMacchia and Odlyzko [LO91] and McCurley [MC90].

The main reference book for finite fields is Lidl and Niederreiter [LN83]. McEliece [MC87] is a good textbook on the subject, and a recent research monograph on applications of finite fields has been published by Menezes *et al.* [MBGMVY93]. A recent article on the **Discrete Logarithm** problem in GF(2^n) is Gordon and McCurley [GM93].

The idea of using elliptic curves for public-key cryptosystems is due to Koblitz [KO87A] and Miller [MI86]. Menezes [ME93] is a monograph on elliptic curve cryptosystems. See also Menezes and Vanstone [MV93] and Chapter 6 of Koblitz [KO87]. For an elementary treatment of elliptic curves, see Silverman and Tate [ST92]. The Menezes-Okamoto-Vanstone reduction of discrete logarithms from elliptic curves to finite fields is given in [MOV94] (see also [ME93]).

The **Merkle-Hellman Cryptosystem** was presented in [MH78]. This system was broken by Shamir [SH84], and the "iterated" version of the system was broken by Brickell [BR85]. A different knapsack-type system, due to Chor and Rivest [CR88], has not been broken. For more information, see the survey article by Brickell and Odlyzko [BO92].

The most important reference book for coding theory is MacWilliams and Sloane [MS77]. There are many good textbooks on coding theory, e.g., Hoffman *et al.* [HLLPRW91] and Vanstone and van Oorschot [VV89]. The **McEliece Cryptosystem** was first described in [MC78]. A recent article discussing the security of this cryptosystem is by Chabaud [CH95].

TABLE 5.3
ElGamal Ciphertext

(3781, 14409)	(31552, 3930)	(27214, 15442)	(5809, 30274)
(5400, 31486)	(19936, 721)	(27765, 29284)	(29820, 7710)
(31590, 26470)	(3781, 14409)	(15898, 30844)	(19048, 12914)
(16160, 3129)	(301, 17252)	(24689, 7776)	(28856, 15720)
(30555, 24611)	(20501, 2922)	(13659, 5015)	(5740, 31233)
(1616, 14170)	(4294, 2307)	(2320, 29174)	(3036, 20132)
(14130, 22010)	(25910, 19663)	(19557, 10145)	(18899, 27609)
(26004, 25056)	(5400, 31486)	(9526, 3019)	(12962, 15189)
(29538, 5408)	(3149, 7400)	(9396, 3058)	(27149, 20535)
(1777, 8737)	(26117, 14251)	(7129, 18195)	(25302, 10248)
(23258, 3468)	(26052, 20545)	(21958, 5713)	(346, 31194)
(8836, 25898)	(8794, 17358)	(1777, 8737)	(25038, 12483)
(10422, 5552)	(1777, 8737)	(3780, 16360)	(11685, 133)
(25115, 10840)	(14130, 22010)	(16081, 16414)	(28580, 20845)
(23418, 22058)	(24139, 9580)	(173, 17075)	(2016, 18131)
(19886, 22344)	(21600, 25505)	(27119, 19921)	(23312, 16906)
(21563, 7891)	(28250, 21321)	(28327, 19237)	(15313, 28649)
(24271, 8480)	(26592, 25457)	(9660, 7939)	(10267, 20623)
(30499, 14423)	(5839, 24179)	(12846, 6598)	(9284, 27858)
(24875, 17641)	(1777, 8737)	(18825, 19671)	(31306, 11929)
(3576, 4630)	(26664, 27572)	(27011, 29164)	(22763, 8992)
(3149, 7400)	(8951, 29435)	(2059, 3977)	(16258, 30341)
(21541, 19004)	(5865, 29526)	(10536, 6941)	(1777, 8737)
(17561, 11884)	(2209, 6107)	(10422, 5552)	(19371, 21005)
(26521, 5803)	(14884, 14280)	(4328, 8635)	(28250, 21321)
(28327, 19237)	(15313, 28649)		

Exercises

5.1 Implement Shanks' algorithm for finding discrete logarithms in $\mathbb{Z}_p$, where p is prime and α is a primitive element. Use your program to find $\log_{106} 12375$ in $\mathbb{Z}_{24691}$ and $\log_6 248388$ in $\mathbb{Z}_{458009}$.

5.2 Implement the Pohlig-Hellman algorithm for finding discrete logarithms in $\mathbb{Z}_p$, where p is prime and α is a primitive element. Use your program to find $\log_5 8563$ in $\mathbb{Z}_{28703}$ and $\log_{10} 12611$ in $\mathbb{Z}_{31153}$.

5.3 Find $\log_5 896$ in $\mathbb{Z}_{1103}$ using the algorithm presented in Figure 5.6, given that $L_2(\beta) = 1$ for $\beta = 25$, 219 and 841, and $L_2(\beta) = 0$ for $\beta = 163$, 532, 625 and 656.

5.4 Decrypt the **ElGamal** ciphertext presented in Table 5.3. The parameters of the system are $p = 31847$, $\alpha = 5$, $a = 7899$ and $\beta = 18074$. Each element of $\mathbb{Z}_n$ represents three alphabetic characters as in Exercise 4.6.

The plaintext was taken from "The English Patient," by Michael Ondaatje, Alfred A. Knopf, Inc., New York, 1992.

5.5 Determine which of the following polynomials are irreducible over $\mathbb{Z}_2[x]$: $x^5 +$

$x^4+1, x^5+x^3+1, x^5+x^4+x^2+1$.

5.6 The field GF(2^5) can be constructed as $\mathbb{Z}_2[x]/(x^5+x^2+1)$. Perform the following computations in this field.
- (a) Compute $(x^4+x^2)\times(x^3+x+1)$.
- (b) Using the extended Euclidean algorithm, compute $(x^3+x^2)^{-1}$.
- (c) Using the square-and-multiply algorithm, compute x^{25}.

5.7 We give an example of the **ElGamal Cryptosystem** implemented in GF(3^3). The polynomial x^3+2x^2+1 is irreducible over $\mathbb{Z}_3[x]$ and hence $\mathbb{Z}_3[x]/(x^3+2x^2+1)$ is the field GF(3^3). We can associate the 26 letters of the alphabet with the 26 nonzero field elements, and thus encrypt ordinary text in a convenient way. We will use a lexicographic ordering of the (nonzero) polynomials to set up the correspondence. This correspondence is as follows:

A	$\leftrightarrow$	1	B	$\leftrightarrow$	2	C	$\leftrightarrow$	x
D	$\leftrightarrow$	$x+1$	E	$\leftrightarrow$	$x+2$	F	$\leftrightarrow$	$2x$
G	$\leftrightarrow$	$2x+1$	H	$\leftrightarrow$	$2x+2$	I	$\leftrightarrow$	x^2
J	$\leftrightarrow$	x^2+1	K	$\leftrightarrow$	x^2+2	L	$\leftrightarrow$	x^2+x
M	$\leftrightarrow$	x^2+x+1	N	$\leftrightarrow$	x^2+x+2	O	$\leftrightarrow$	x^2+2x
P	$\leftrightarrow$	x^2+2x+1	Q	$\leftrightarrow$	x^2+2x+2	R	$\leftrightarrow$	$2x^2$
S	$\leftrightarrow$	$2x^2+1$	T	$\leftrightarrow$	$2x^2+2$	U	$\leftrightarrow$	$2x^2+x$
V	$\leftrightarrow$	$2x^2+x+1$	W	$\leftrightarrow$	$2x^2+x+2$	X	$\leftrightarrow$	$2x^2+2x$
Y	$\leftrightarrow$	$2x^x+2x+1$	Z	$\leftrightarrow$	$2x^x+2x+2$			

Suppose Bob uses $\alpha = x$ and $a = 11$ in an ElGamal system; then $\beta = x+2$. Show how Bob will decrypt the following string of ciphertext:

```
(K,H) (P,X) (N,K) (H,R) (T,F) (V,Y) (E,H) (F,A) (T,W) (J,D) (U,J)
```

5.8 Let E be the elliptic curve $y^2 = x^3+x+28$ defined over $\mathbb{Z}_{71}$.
- (a) Determine the number of points on E.
- (b) Show that E is not a cyclic group.
- (c) What is the maximum order of an element in E? Find an element having this order.

5.9 Let E be the elliptic curve $y^2 = x^3+x+13$ defined over $\mathbb{Z}_{31}$. It can be shown that $\#E = 34$ and $(9,10)$ is an element of order 34 in E. The **Menezes-Vanstone Cryptosystem** defined on E will have as its plaintext space $\mathbb{Z}_{34}^* \times \mathbb{Z}_{34}^*$. Suppose Bob's secret exponent is $a = 25$.
- (a) Compute $\beta = a\alpha$.
- (b) Decrypt the following string of ciphertext:

$$((4,9),28,7),((19,28),9,13),((5,22),20,17),((25,16),12,27).$$

- (c) Assuming that each plaintext represents two alphabetic characters, convert the plaintext into an English word. (Here we will use the correspondence $A \leftrightarrow 1, \ldots, Z \leftrightarrow 26$, since 0 is not allowed in a (plaintext) ordered pair.)

5.10 Suppose the **Merkle-Hellman Cryptosystem** has as its public list of sizes the vector

$$\mathbf{t} = (1394, 1256, 1508, 1987, 439, 650, 724, 339, 2303, 810).$$

Suppose Oscar discovers that $p = 2503$.
- (a) By trial and error, determine the value a such that the list $a^{-1}\mathbf{t} \bmod p$ is a permutation of a superincreasing list.
- (b) Show how the ciphertext 5746 would be decrypted.

5.11 It can be shown that the matrix H shown below is a parity-check matrix for a [15, 7, 5] code called a BCH code.

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Decode, if possible, each of the following received vectors $\mathbf{r}$ using the syndrome decoding method.

(a) $\mathbf{r} = (1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$.

(b) $\mathbf{r} = (1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0)$.

(c) $\mathbf{r} = (1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0)$.