CHAPTER 10

Asymmetric-Key Cryptography

Objectives

This chapter has several objectives:

- ❏ To distinguish between symmetric-key and asymmetric-key cryptosystems
- ❏ To introduce trapdoor one-way functions and their use in asymmetric-key cryptosystems
- ❏ To introduce the knapsack cryptosystem as one of the first ideas in asymmetric-key cryptography
- ❏ To discuss the RSA cryptosystem
- ❏ To discuss the Rabin cryptosystem
- ❏ To discuss the ElGamal cryptosystem
- ❏ To discuss the elliptic curve cryptosystem

This chapter discusses several asymmetric-key cryptosystems: RSA, Rabin, ElGamal, and ECC. Discussion of the Diffie-Hellman cryptosystem is postponed until Chapter 15 because it is mainly a key-exchange algorithm rather than an encryption/decryption algorithm.

The Diffie-Hellman cryptosystem is discussed in Chapter 15.

10.1 INTRODUCTION

In Chapters 2 through 8, we emphasized the principles of **symmetric-key cryptography.** In this chapter, we start the discussion of **asymmetric-key cryptography.** Symmetric- and asymmetric-key cryptography will exist in parallel and continue to serve the community. We actually believe that they are complements of each other; the advantages of one can compensate for the disadvantages of the other.

The conceptual differences between the two systems are based on how these systems keep a secret. In symmetric-key cryptography, the secret must be shared between two persons. In asymmetric-key cryptography, the secret is personal (unshared); each person creates and keeps his or her own secret.

In a community of n people, $n(n - 1)/2$ shared secrets are needed for symmetric-key cryptography; only n personal secrets are needed in asymmetric-key cryptography. For a community with a population of 1 million, symmetric-key cryptography would require half a billion shared secrets; asymmetric-key cryptography would require 1 million personal secrets.

Symmetric-key cryptography is based on sharing secrecy; asymmetric-key cryptography is based on personal secrecy.

There are some other aspects of security besides encipherment that need asymmetric-key cryptography. These include authentication and digital signatures. Whenever an application is based on a personal secret, we need to use asymmetric-key cryptography.

Whereas symmetric-key cryptography is based on substitution and permutation of symbols (characters or bits), asymmetric-key cryptography is based on applying mathematical functions to numbers. In symmetric-key cryptography, the plaintext and ciphertext are thought of as a combination of symbols. Encryption and decryption permute these symbols or substitute a symbol for another. In asymmetric-key cryptography, the plaintext and ciphertext are numbers; encryption and decryption are mathematical functions that are applied to numbers to create other numbers.

In symmetric-key cryptography, symbols are permuted or substituted; in asymmetric-key cryptography, numbers are manipulated.

Keys

Asymmetric key cryptography uses two separate keys: one private and one public. If encryption and decryption are thought of as locking and unlocking padlocks with keys, then the padlock that is locked with a public key can be unlocked only with the corresponding private key. Figure 10.1 shows that if Alice locks the padlock with Bob's public key, then only Bob's private key can unlock it.

General Idea

Figure 10.2 shows the general idea of asymmetric-key cryptography as used for encipherment. We will see other applications of asymmetric-key cryptography in future chapters. The figure shows that, unlike symmetric-key cryptography, there are distinctive keys in asymmetric-key cryptography: a **private key** and a **public key.** Although some books use the term *secret key* instead of *private key,* we use the term *secret key* only for symmetric-key and the terms *private key* and *public key* for asymmetric key cryptography. We even use different symbols to show the three keys. One reason is that we believe the nature of the *secret key* used in symmetric-key cryptography is different

Figure 10.1 *Locking and unlocking in asymmetric-key cryptosystem*

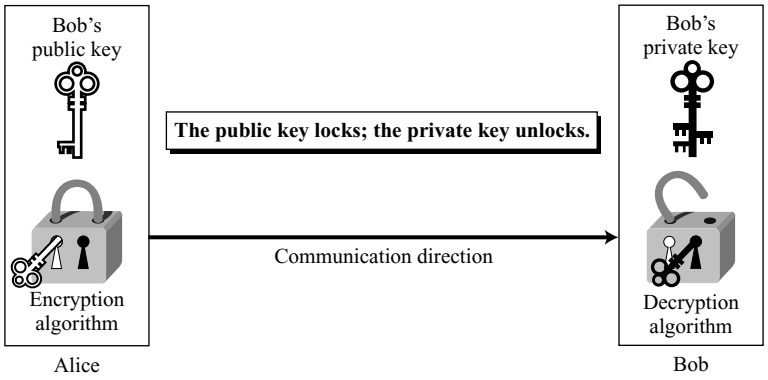
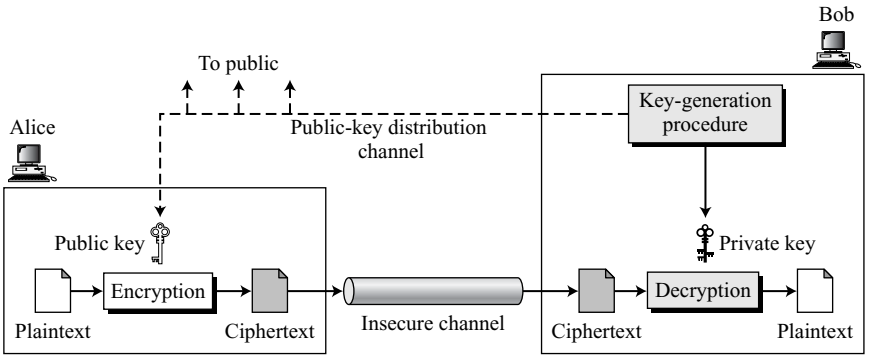


Figure 10.2 *General idea of asymmetric-key cryptosystem*



from the nature of the *private key* used in asymmetric-key cryptography. The first is normally a string of symbols (bits for example), the second is a number or a set of numbers. In other words, we want to show that a *secret key* is not exchangeable with a *private key;* there are two different types of secrets.

Figure 10.2 shows several important facts. First, it emphasizes the asymmetric nature of the cryptosystem. The burden of providing security is mostly on the shoulders of the receiver (Bob, in this case). Bob needs to create two keys: one private and one public. Bob is responsible for distributing the public key to the community. This can be done through a public-key distribution channel. Although this channel is not required to provide secrecy, it must provide authentication and integrity. Eve should not be able to advertise her public key to the community pretending that it is Bob's public key. Issues regarding public-key distribution are discussed in Chapter 15. For the moment, we assume that such a channel exists.

Second, asymmetric-key cryptography means that Bob and Alice cannot use the same set of keys for two-way communication. Each entity in the community should create its own private and public keys. Figure 10.2 shows how Alice can use Bob's public key to send encrypted messages to Bob. If Bob wants to respond, Alice needs to establish her own private and public keys.

Third, asymmetric-key cryptography means that Bob needs only one private key to receive all correspondence from anyone in the community, but Alice needs *n* public keys to communicate with *n* entities in the community, one public key for each entity. In other words, Alice needs a ring of public keys.

Plaintext/Ciphertext

Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers in asymmetric-key cryptography. The message must be encoded as an integer (or a set of integers) before encryption; the integer (or the set of integers) must be decoded into the message after decryption. Asymmetric-key cryptography is normally used to encrypt or decrypt small pieces of information, such as the cipher key for a symmetric-key cryptography. In other words, asymmetric-key cryptography normally is used for ancillary goals instead of message encipherment. However, these ancillary goals play a very important role in cryptography today.

Encryption/Decryption

Encryption and decryption in asymmetric-key cryptography are mathematical functions applied over the numbers representing the plaintext and ciphertext. The ciphertext can be thought of as $C = \boldsymbol{f}(K_{public}, P)$; the plaintext can be thought of as $P = \boldsymbol{g}(K_{private}, C)$. The decryption function $\boldsymbol{f}$ is used only for encryption; the decryption function $\boldsymbol{g}$ is used only for decryption. Next we show that the function $\boldsymbol{f}$ needs to be a *trapdoor one-way function* to allow Bob to decrypt the message but to prevent Eve from doing so.

Need for Both

There is a very important fact that is sometimes misunderstood: The advent of asymmetric-key (public-key) cryptography does not eliminate the need for symmetric-key (secret-key) cryptography. The reason is that asymmetric-key cryptography, which uses mathematical functions for encryption and decryption, is much slower than symmetric-key cryptography. For encipherment of large messages, symmetric-key cryptography is still needed. On the other hand, the speed of symmetric-key cryptography does not eliminate the need for asymmetric-key cryptography. Asymmetric-key cryptography is still needed for authentication, digital signatures, and secret-key exchanges. This means that, to be able to use all aspects of security today, we need both symmetric-key and asymmetric-key cryptography. One complements the other.

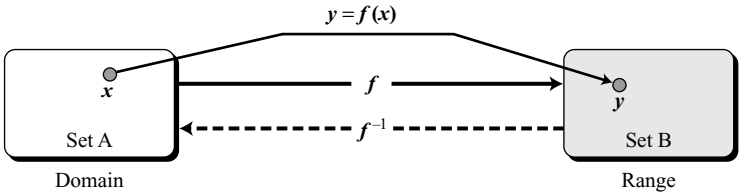
Trapdoor One-Way Function

The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function.

Functions

Although the concept of a function is familiar from mathematics, we give an informal definition here. A **function** is a rule that associates (maps) one element in set A, called the domain, to one element in set B, called the range, as shown in Figure 10.3.

Figure 10.3 *A function as rule mapping a domain to a range*



An **invertible function** is a function that associates each element in the range with exactly one element in the domain.

One-Way Function

A **one-way function (OWF)** is a function that satisfies the following two properties:

1. $\boldsymbol{f}$ is easy to compute. In other words, given x, $y = \boldsymbol{f}(x)$ can be easily computed.
2. $\boldsymbol{f}^{-1}$ is difficult to compute. In other words, given y, it is computationally infeasible to calculate $x = \boldsymbol{f}^{-1}(y)$.

Trapdoor One-Way Function

A **trapdoor one-way function (TOWF)** is a one-way function with a third property:

3. Given y and a **trapdoor** (secret), x can be computed easily.

Example 10.1

When n is large, $n = p \times q$ is a one-way function. Note that in this function x is a tuple (p, q) of two primes and y is n. Given p and q, it is always easy to calculate n; given n, it is very difficult to compute p and q. This is the *factorization problem* that we saw in Chapter 9. There is not a polynomial time solution to the $\boldsymbol{f}^{-1}$ function in this case.

Example 10.2

When n is large, the function $y = x^k \bmod n$ is a trapdoor one-way function. Given x, k, and n, it is easy to calculate y using the fast exponential algorithm we discussed in Chapter 9. Given y, k, and n, it is very difficult to calculate x. This is the *discrete logarithm problem* we discussed in Chapter 9. There is not a polynomial time solution to the $\boldsymbol{f}^{-1}$ function in this case. However, if we know the trapdoor, k' such that $k \times k' = 1 \bmod \phi(n)$, we can use $x = y^{k'} \bmod n$ to find x. This is the famous RSA, which will be discussed later in this chapter.

Knapsack Cryptosystem

The first brilliant idea of public-key cryptography came from Merkle and Hellman, in their **knapsack cryptosystem.** Although this system was found to be insecure with today's standards, the main idea behind this cryptosystem gives an insight into recent public-key cryptosystems discussed later in this chapter.

If we are told which elements, from a predefined set of numbers, are in a knapsack, we can easily calculate the sum of the numbers; if we are told the sum, it is difficult to say which elements are in the knapsack.

Definition

Suppose we are given two k-tuples, $a = [a_1, a_2, \ldots, a_k]$ and $x = [x_1, x_2, \ldots, x_k]$. The first tuple is the predefined set; the second tuple, in which x_i is only 0 or 1, defines which elements of a are to be dropped in the knapsack. The sum of elements in the knapsack is

$$s = knapsackSum\ (a, x) = x_1a_1 + x_2a_2 + \cdots + x_ka_k$$

Given a and x, it is easy to calculate s. However, given s and a it is difficult to find x. In other words, $s = knapsackSum\ (x, a)$ is easy to calculate, but $x = inv\_knapsackSum\ (s, a)$ is difficult. The function *knapsackSum* is a one-way function if a is a general k-tuple.

Superincreasing Tuple

It is easy to compute *knapsackSum* and *inv_knapsackSum* if the k-tuple a is *superincreasing.* In a **superincreasing tuple,** $a_i \geq a_1 + a_2 + \cdots + a_{i-1}$. In other words, each element (except a_1) is greater than or equal to the sum of all previous elements. In this case we calculate *knapsackSum* and *inv_knapsackSum* as shown in Algorithm 10.1. The algorithm *inv_knapsackSum* starts from the largest element and proceeds to the smallest one. In each iteration, it checks to see whether an element is in the knapsack.

Algorithm 10.1 *knapsacksum and inv_knapsackSum for a superincreasing k-tuple*

```
knapsackSum (x [1 … k], a [1 … k])
{
   s ← 0
   for (i = 1 to k)
   {
     s ← s + a_i × x_i
   }
   return s
}
```

```
inv_knapsackSum (s, a [1 … k])
{
   for (i = k down to 1)
   {
      if s ≥ a_i
      {
         x_i ← 1
         s ← s – a_i
      }
      else x_i ← 0
   }
   return x [1 … k]
}
```

Example 10.3

As a very trivial example, assume that a = [17, 25, 46, 94, 201,400] and s = 272 are given. Table 10.1 shows how the tuple x is found using *inv_knapsackSum* routine in Algorithm 10.1.

Table 10.1 *Values of i, a_i, s, and x_i in Example 10.3*

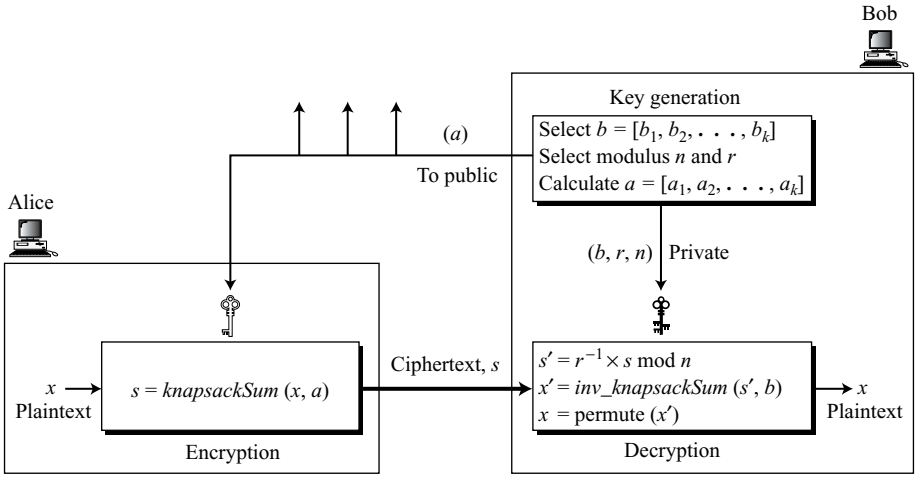
i	a_i	s	$s \geq a_i$	x_i	$s \leftarrow s - a_i \times x_i$
6	400	272	false	$x_6 = 0$	272
5	201	272	true	$x_5 = 1$	71
4	94	71	false	$x_4 = 0$	71
3	46	71	true	$x_3 = 1$	25
2	25	25	true	$x_2 = 1$	0
1	17	0	false	$x_1 = 0$	0

In this case x = [0, 1, 1, 0, 1, 0], which means that 25, 46, and 201 are in the knapsack.

Secret Communication with Knapsacks

Let us see how Alice can send a secret message to Bob using a knapsack cryptosystem. The idea is shown in Figure 10.4.

Figure 10.4 *Secret communication with knapsack cryptosystem*



Key Generation

a. Create a superincreasing k-tuple $b = [b_1, b_2, \ldots, b_k]$
b. Choose a modulus n, such that $n > b_1 + b_2 + \cdots + b_k$

c. Select a random integer r that is relatively prime with n and $1 \le r \le n - 1$.
d. Create a temporary k-tuple $t = [t_1, t_2, \ldots, t_k]$ in which $t_i = r \times b_i \bmod n$.
e. Select a permutation of k objects and find a new tuple $a = permute\ (t)$.
f. The public key is the k-tuple a. The private key is n, r, and the k-tuple b.

Encryption

Suppose Alice needs to send a message to Bob.

a. Alice converts her message to a k-tuple $x = [x_1, x_2, \ldots, x_k]$ in which x_i is either 0 or 1. The tuple x is the plaintext.
b. Alice uses the *knapsackSum* routine to calculate s. She then sends the value of s as the ciphertext.

Decryption

Bob receives the ciphertext s.

a. Bob calculates $s' = r^{-1} \times s \bmod n$.
b. Bob uses *inv_knapsackSum* to create x'.
c. Bob permutes x' to find x. The tuple x is the recovered plaintext.

Example 10.4

This is a trivial (very insecure) example just to show the procedure.

1. Key generation:
   a. Bob creates the superincreasing tuple $b = [7, 11, 19, 39, 79, 157, 313]$.
   b. Bob chooses the modulus $n = 900$ and $r = 37$, and [4 2 5 3 1 7 6] as permutation table.
   c. Bob now calculates the tuple $t = [259, 407, 703, 543, 223, 409, 781]$.
   d. Bob calculates the tuple a = permute $(t) = [543, 407, 223, 703, 259, 781, 409]$.
   e. Bob publicly announces a; he keeps n, r, and b secret.
2. Suppose Alice wants to send a single character "g" to Bob.
   a. She uses the 7-bit ASCII representation of "g", $(1100111)_2$, and creates the tuple $x = [1, 1, 0, 0, 1, 1, 1]$. This is the plaintext.
   b. Alice calculates $s = knapsackSum\ (a, x) = 2165$. This is the ciphertext sent to Bob.
3. Bob can decrypt the ciphertext, $s = 2165$.
   a. Bob calculates $s' = s \times r^{-1} \bmod n = 2165 \times 37^{-1} \bmod 900 = 527$.
   b. Bob calculates $x' = Inv\_knapsackSum\ (s', b) = [1, 1, 0, 1, 0, 1, 1]$.
   c. Bob calculates $x = permute\ (x') = [1, 1, 0, 0, 1, 1, 1]$. He interprets the string $(1100111)_2$ as the character "g".

Trapdoor

Calculating the sum of items in Alice's knapsack is actually the multiplication of the row matrix $\boldsymbol{x}$ by the column matrix $\boldsymbol{a}$. The result is a 1×1 matrix $\boldsymbol{s}$. Matrix multiplication, $\boldsymbol{s} = \boldsymbol{x} \times \boldsymbol{a}$, in which $\boldsymbol{x}$ is a row matrix and $\boldsymbol{a}$ is a column matrix, is a one-way function. Given $\boldsymbol{s}$ and $\boldsymbol{x}$, Eve cannot find $\boldsymbol{a}$ easily. Bob, however, has a trapdoor. Bob uses his $\boldsymbol{s}' = r^{-1} \times \boldsymbol{s}$ and the secret superincreasing column matrix $\boldsymbol{b}$ to find a row matrix $\boldsymbol{x}'$ using the *inv_knapsackSum* routine. The permutation allows Bob to find $\boldsymbol{x}$ from $\boldsymbol{x}'$.

10.2 RSA CRYPTOSYSTEM

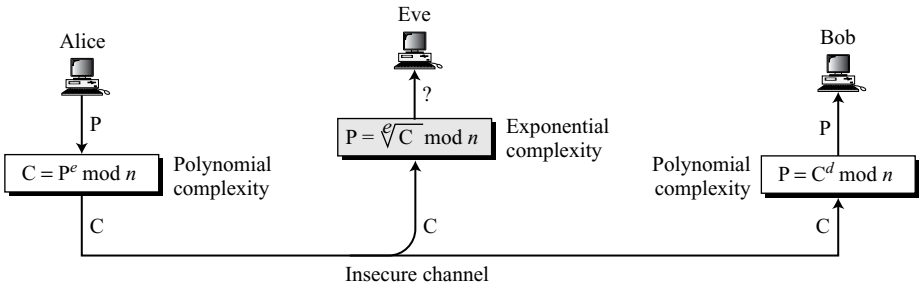
The most common public-key algorithm is the **RSA cryptosystem,** named for its inventors (Rivest, Shamir, and Adleman).

Introduction

RSA uses two exponents, e and d, where e is public and d is private. Suppose P is the plaintext and C is the ciphertext. Alice uses $C = P^e \bmod n$ to create ciphertext C from plaintext P; Bob uses $P = C^d \bmod n$ to retrieve the plaintext sent by Alice. The modulus n, a very large number, is created during the key generation process, as we will discuss later.

Encryption and decryption use modular exponentiation. As we discussed in Chapter 9, modular exponentiation is feasible in polynomial time using the fast exponentiation algorithm. However, modular logarithm is as hard as factoring the modulus, for which there is no polynomial algorithm yet. This means that Alice can encrypt in polynomial time (e is public), Bob also can decrypt in polynomial time (because he knows d), but Eve cannot decrypt because she would have to calculate the eth root of C using modular arithmetic. Figure 10.5 shows the idea.

Figure 10.5 *Complexity of operations in RSA*



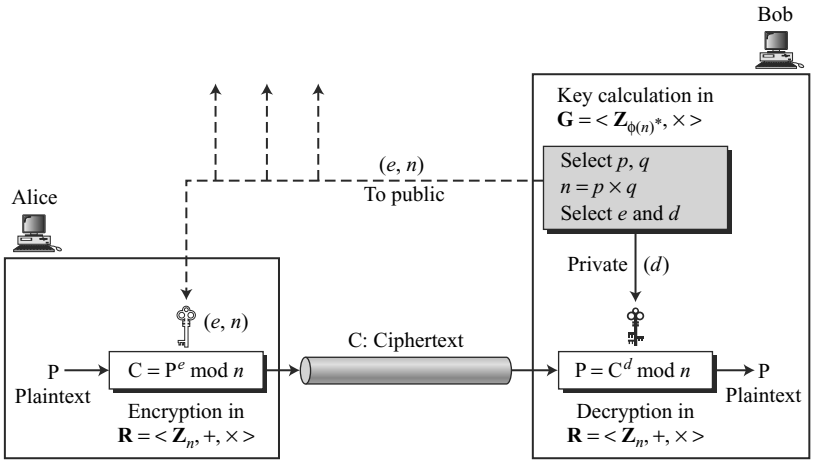
In other words, Alice uses a one-way function (modular exponentiation) with a trapdoor known only to Bob. Eve, who does not know the trapdoor, cannot decrypt the message. If some day, a polynomial algorithm for eth root modulo n calculation is found, modular exponentiation is not a one-way function any more.

Procedure

Figure 10.6 shows the general idea behind the procedure used in RSA.

RSA uses modular exponentiation for encryption/decryption; To attack it, Eve needs to calculate $\sqrt[e]{C} \bmod n$.

Figure 10.6 *Encryption, decryption, and key generation in RSA*



Two Algebraic Structures

RSA uses two algebraic structures: a ring and a group.

Encryption/Decryption Ring Encryption and decryption are done using the commutative ring $\mathbf{R} = <\mathbf{Z}_n, +, \times>$ with two arithmetic operations: addition and multiplication. In RSA, this ring is public because the modulus n is public. Anyone can send a message to Bob using this ring to do encryption.

Key-Generation Group RSA uses a multiplicative group $\mathbf{G} = <\mathbf{Z}_{\phi(n)}^*, \times>$ for key generation. This group supports only multiplication and division (using multiplicative inverses), which are needed for generating public and private keys. This group is hidden from the public because its modulus, $\phi(n)$, is hidden from the public. We will see shortly that if Eve can find this modulus, she can easily attack the cryptosystem.

RSA uses two algebraic structures: a public ring $\mathbf{R} = <\mathbf{Z}_n, +, \times>$ and a private group $\mathbf{G} = <\mathbf{Z}_{\phi(n)}^*, \times>$.

Key Generation

Bob uses the steps shown in Algorithm 10.2 to create his public and private key. After key generation, Bob announces the tuple (e, n) as his public key; Bob keeps the integer d as his private key. Bob can discard p, q, and $\phi(n)$; they will not be needed unless Bob needs to change his private key without changing the modulus (which is not recommended, as we will see shortly). To be secure, the recommended size for each prime, p or q, is 512 bits (almost 154 decimal digits). This makes the size of n, the modulus, 1024 bits (309 digits).

Algorithm 10.2 *RSA Key Generation*

```
RSA_Key_Generation
{
    Select two large primes p and q such that p ≠ q.
    n ← p × q
    φ(n) ← (p − 1) × (q − 1)
    Select e such that 1 < e < φ(n) and e is coprime to φ(n)
    d ← e^−1 mod φ(n)                    // d is inverse of e modulo φ(n)
    Public_key ← (e, n)                  // To be announced publicly
    Private_key ← d                      // To be kept secret
    return Public_key and Private_key
}
```

In RSA, the tuple (*e*, *n*) is the public key; the integer *d* is the private key.

Encryption

Anyone can send a message to Bob using his public key. Encryption in RSA can be done using an algorithm with polynomial time complexity, as shown in Algorithm 10.3. The fast exponentiation algorithm was discussed in Chapter 9. The size of the plaintext must be less than n, which means that if the size of the plaintext is larger than n, it should be divided into blocks.

Algorithm 10.3 *RSA encryption*

```
RSA_Encryption (P, e, n)             // P is the plaintext in Z_n and P < n
{
    C ← Fast_Exponentiation (P, e, n)   // Calculation of (P^e mod n)
    return C
}
```

Decryption

Bob can use Algorithm 10.4 to decrypt the ciphertext message he received. Decryption in RSA can be done using an algorithm with polynomial time complexity. The size of the ciphertext is less than n.

Algorithm 10.4 *RSA decryption*

```
RSA_Decryption (C, d, n)             //C is the ciphertext in Z_n
{
    P ← Fast_Exponentiation (C, d, n)   // Calculation of (C^d mod n)
    return P
}
```

In RSA, *p* and *q* must be at least 512 bits; *n* must be at least 1024 bits.

Proof of RSA

We can prove that encryption and decryption are inverses of each other using the second version of Euler's theorem discussed in Chapter 9:

If $n = p \times q$, $a < n$, and k is an integer, then $a^{k \times \phi(n)+1} \equiv a \pmod{n}$.

Assume that the plaintext retrieved by Bob is P_1 and prove that it is equal to P.

$P_1 = C^d \bmod n = (P^e \bmod n)^d \bmod n = P^{ed} \bmod n$
$ed = k\phi(n) + 1$ // d and e are inverses modulo $\phi(n)$
$P_1 = P^{ed} \bmod n \rightarrow P_1 = P^{k\phi(n)+1} \bmod n$
$P_1 = P^{k\phi(n)+1} \bmod n = P \bmod n$ // Euler's theorem (second version)

Some Trivial Examples

Following are some trivial (insecure) examples of the RSA procedure. The criteria that make the RSA system secure will be discussed in the later sections.

Example 10.5

Bob chooses 7 and 11 as p and q and calculates $n = 7 \times 11 = 77$. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60. Now he chooses two exponents, e and d, from $\mathbf{Z}_{60}$*. If he chooses e to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$ (they are inverses of each other). Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

Plaintext: 5 $C = 5^{13} = 26 \bmod 77$ Ciphertext: 26

Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:

Ciphertext: 26 $P = 26^{37} = 5 \bmod 77$ Plaintext: 5

The plaintext 5 sent by Alice is received as plaintext 5 by Bob.

Example 10.6

Now assume that another person, John, wants to send a message to Bob. John can use the same public key announced by Bob (probably on his website), 13; John's plaintext is 63. John calculates the following:

Plaintext: 63 $C = 63^{13} = 28 \bmod 77$ Ciphertext: 28

Bob receives the ciphertext 28 and uses his private key 37 to decipher the ciphertext:

Ciphertext: 28 $P = 28^{37} = 63 \bmod 77$ Plaintext: 63

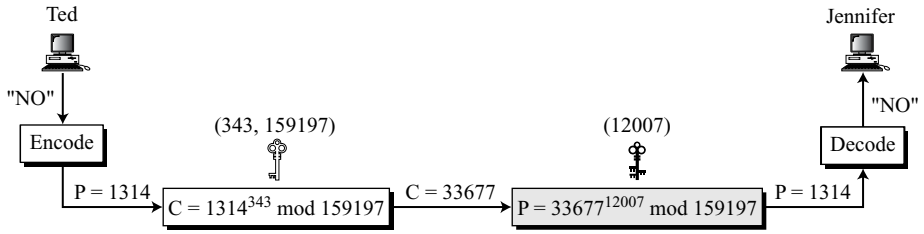
Example 10.7

Jennifer creates a pair of keys for herself. She chooses $p = 397$ and $q = 401$. She calculates $n = 397 \times 401 = 159197$. She then calculates $\phi(n) = 396 \times 400 = 158400$. She then chooses $e = 343$ and $d = 12007$. Show how Ted can send a message to Jennifer if he knows e and n.

Solution

Suppose Ted wants to send the message "NO" to Jennifer. He changes each character to a number (from 00 to 25), with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Ted then uses e and n to encrypt the message. The ciphertext is $1314^{343} = 33677 \bmod 159197$. Jennifer receives the message 33677 and uses the decryption key d to decipher it as $33677^{12007} = 1314 \bmod 159197$. Jennifer then decodes 1314 as the message "NO". Figure 10.7 shows the process.

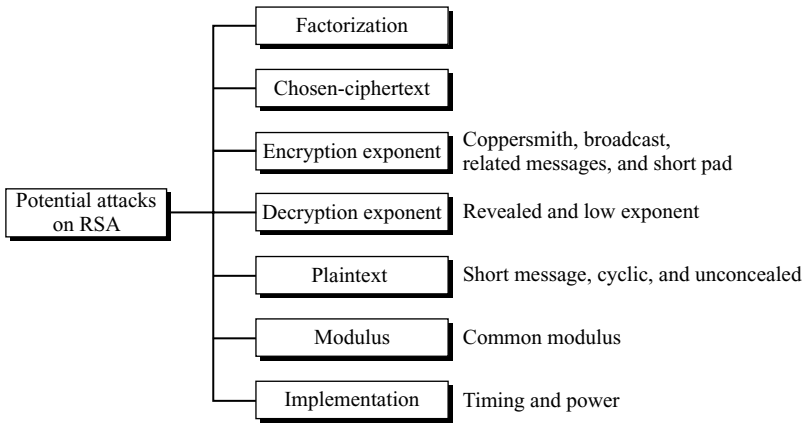
Figure 10.7 *Encryption and decryption in Example 10.7*



Attacks on RSA

No devastating attacks on RSA have been yet discovered. Several attacks have been predicted based on the weak plaintext, weak parameter selection, or inappropriate implementation. Figure 10.8 shows the categories of potential attacks.

Figure 10.8 *Taxonomy of potential attacks on RSA*



Factorization Attack

The security of RSA is based on the idea that the modulus is so large that it is infeasible to factor it in a reasonable time. Bob selects p and q and calculates $n = p \times q$. Although n is public, p and q are secret. If Eve can factor n and obtain p and q, she can calculate $\phi(n) = (p - 1)\ (q - 1)$. Eve then can calculate $d = e^{-1} \bmod \phi(n)$ because e is public. The private exponent d is the trapdoor that Eve can use to decrypt any encrypted message.

As we learned in Chapter 9, there are many factorization algorithms, but none of them can factor a large integer with polynomial time complexity. To be secure, RSA presently requires that n should be more than 300 decimal digits, which means that the modulus must be at least 1024 bits. Even using the largest and fastest computer available today, factoring an integer of this size would take an infeasibly long period of time. This means that RSA is secure as long as an efficient algorithm for factorization has not been found.

Chosen-Ciphertext Attack

A potential attack on RSA is based on the multiplicative property of RSA. Assume that Alice creates the ciphertext $C = P^e \bmod n$ and sends C to Bob. Also assume that Bob will decrypt an arbitrary ciphertext for Eve, other than C. Eve intercepts C and uses the following steps to find P:

a. Eve chooses a random integer X in Z_n^*.
b. Eve calculates $Y = C \times X^e \bmod n$.
c. Eve sends Y to Bob for decryption and get $Z = Y^d \bmod n$; This step is an instance of a chosen-ciphertext attack.
d. Eve can easily find P because

$$Z = Y^d \bmod n = (C \times X^e)^d \bmod n = (C^d \times X^{ed}) \bmod n = (C^d \times X) \bmod n = (P \times X) \bmod n$$
$$Z = (P \times X) \bmod n \quad \rightarrow \quad P = Z \times X^{-1} \bmod n$$

Eve uses the extended Euclidean algorithm to find the multiplicative inverse of X and eventually the value of P.

Attacks on the Encryption Exponent

To reduce the encryption time, it is tempting to use a small encryption exponent e. The common value for e is $e = 3$ (the second prime). However, there are some potential attacks on low encryption exponent that we briefly discuss here. These attacks do not generally result in a breakdown of the system, but they still need to be prevented. To thwart these kinds of attacks, the recommendation is to use $e = 2^{16} + 1 = 65537$ (or a prime close to this value).

Coppersmith Theorem Attack The major low encryption exponent attack is referred to as the **Coppersmith theorem attack.** This theorem states that in a modulo-n polynomial $\boldsymbol{f}(x)$ of degree e, one can use an algorithm of the complexity log n to find the roots if one of the roots is smaller than $n^{1/e}$. This theorem can be applied to the RSA

cryptosystem with $C = \boldsymbol{f}(P) = P^e \bmod n$. If $e = 3$ and only two thirds of the bits in the plaintext P are known, the algorithm can find all bits in the plaintext.

Broadcast Attack The **broadcast attack** can be launched if one entity sends the same message to a group of recipients with the same low encryption exponent. For example, assume the following scenario: Alice wants to send the same message to three recipients with the same public exponent $e = 3$ and the moduli n_1, n_2, and n_3.

$$C_1 = P^3 \bmod n_1 \qquad C_2 = P^3 \bmod n_2 \qquad C_3 = P^3 \bmod n_3$$

Applying the Chinese remainder theorem to these three equations, Eve can find an equation of the form $C' = P^3 \bmod n_1 n_2 n_3$. This means that $P^3 < n_1 n_2 n_3$. This means $C' = P^3$ is in regular arithmetic (not modular arithmetic). Eve can find the value of $C' = P^{1/3}$.

Related Message Attack The **related message attack,** discovered by Franklin Reiter, can be briefly described as follows. Alice encrypts two plaintexts, P_1 and P_2, and encrypts them with $e = 3$ and sends C_1 and C_2 to Bob. If P_1 is related to P_2 by a linear function, then Eve can recover P_1 and P_2 in a feasible computation time.

Short Pad Attack The **short pad attack,** discovered by Coppersmith, can be briefly described as follows. Alice has a message M to send to Bob. She pads the message with r_1, encrypts the result to get C_1, and sends C_1 to Bob. Eve intercepts C_1 and drops it. Bob informs Alice that he has not received the message, so Alice pads the message again with r_2, encrypts it, and sends it to Bob. Eve also intercepts this message. Eve now has C_1 and C_2, and she knows that they both are ciphertexts belonging to the same plaintext. Coppersmith proved that if r_1 and r_2 are short, Eve may be able to recover the original message M.

Attacks on the Decryption Exponent

Two forms of attacks can be launched on the decryption exponent: **revealed decryption exponent attack** and **low decryption exponent attack.** They are discussed briefly.

Revealed Decryption Exponent Attack It is obvious that if Eve can find the decryption exponent, d, she can decrypt the current encrypted message. However, the attack does not stop here. If Eve knows the value of d, she can use a probabilistic algorithm (not discussed here) to factor n and find the value of p and q. Consequently, if Bob changes only the compromised decryption exponent but keeps the same modulus, n, Eve will be able to decrypt future messages because she has the factorization of n. This means that if Bob finds out that the decryption exponent is compromised, he needs to choose new value for p and q, calculate n, and create totally new private and public keys.

In RSA, if d is comprised, then p, q, n, e, and d must be regenerated.

Low Decryption Exponent Attack Bob may think that using a small private-key d, would make the decryption process faster for him. Wiener showed that if $d < 1/3\ n^{1/4}$, a special type of attack based on *continuous fraction,* a topic discussed in number theory, can jeopardize the security of RSA. For this to happen, it must be the case that $q < p < 2q$. If these two conditions exist, Eve can factor n in polynomial time.

In RSA, the recommendation is to have $d \geq 1/3\ n^{1/4}$ to prevent low decryption exponent attack.

Plaintext Attacks

Plaintext and ciphertext in RSA are permutations of each other because they are integers in the same interval (0 to $n - 1$). In other words, Eve already knows something about the plaintext. This characteristic may allow some attacks on the plaintext. Three attacks have been mentioned in the literature: short message attack, cycling attack, and unconcealed attack.

Short Message Attack In the **short message attack,** if Eve knows the set of possible plaintexts, she then knows one more piece of information in addition to the fact that the ciphertext is the permutation of plaintext. Eve can encrypt all of the possible messages until the result is the same as the ciphertext intercepted. For example, if it is known that Alice is sending a four-digit number to Bob, Eve can easily try plaintext numbers from 0000 to 9999 to find the plaintext. For this reason, short messages must be padded with random bits at the front and the end to thwart this type of attack. It is strongly recommended that messages be padded with random bits before encryption using a method called OAEP, which is discussed later in this chapter.

Cycling Attack The **cycling attack** is based on the fact that if the ciphertext is a permutation of the plaintext, the continuous encryption of the ciphertext will eventually result in the plaintext. In other words, if Eve continuously encrypts the intercepted ciphertext C, she will eventually get the plaintext. However, Eve does not know what the plaintext is, so she does not know when to stop. She needs to go one step further. When she gets the ciphertext C again, she goes back one step to find the plaintext.

Intercepted ciphertext: C
$C_1 = C^e \mod n$
$C_2 = C_1^e \mod n$
…
$C_k = C_{k-1}^e \mod n \rightarrow$ If $C_k = C$, stop: the plaintext is $P = C_{k-1}$

Is this a serious attack on RSA? It has been shown that the complexity of the algorithm is equivalent to the complexity of factoring n. In other words, there is no efficient algorithm that can launch this attack in polynomial time if n is large.

Unconcealed Message Attack Another attack that is based on the permutation relationship between plaintext and ciphertext is the **unconcealed message attack.** An

unconcealed message is a message that encrypts to itself (cannot be concealed). It has been proven that there are always some messages that are encrypted to themselves. Because the encryption exponent normally is odd, there are some plaintexts that are encrypted to themselves such as P = 0 and P = 1. Although there are more, if the encrypting exponent is selected carefully, the number of these message is negligible. The encrypting program can always check if the calculated ciphertext is the same as the plaintext and reject the plaintext before submitting the ciphertext.

Attacks on the Modulus

The main attack on RSA, as discussed previously, is the factorization attack**.** The factorization attack can be considered an attack on the low modulus. However, because we have already discussed this attack, we will concentrate on another attack on the modulus: the common modulus attack.

Common Modulus Attack The **common modulus attack** can be launched if a community uses a common modulus, n. For example, people in a community might let a trusted party select p and q, calculate n and $\phi(n)$, and create a pair of exponents (e_i, d_i) for each entity. Now assume Alice needs to send a message to Bob. The ciphertext to Bob is $C = P^{e_B} \bmod n$. Bob uses his private exponent, d_B, to decrypt his message, $P = C^{d_B} \bmod n$. The problem is that Eve can also decrypt the message if she is a member of the community and has been assigned a pair of exponents (e_E and d_E), as we learned in the section "Low Decryption Exponent Attack". Using her own exponents (e_E and d_E), Eve can launch a probabilistic attack to factor n and find Bob's d_B. To thwart this type of attack, the modulus must not be shared. Each entity needs to calculate her or his own modulus.

Attacks on Implementation

Previous attacks were based on the underlying structure of RSA. As Dan Boneh has shown, there are several attacks on the implementation of RSA. We mention two of these attacks: the timing attack and the power attack.

Timing Attack Paul Kocher elegantly demonstrated a ciphertext-only attack, called the **timing attack.** The attack is based on the fast-exponential algorithm discussed in Chapter 9. The algorithm uses only squaring if the corresponding bit in the private exponent d is 0; it uses both squaring and multiplication if the corresponding bit is 1. In other words, the timing required to do each iteration is longer if the corresponding bit is 1. This timing difference allows Eve to find the value of bits in d, one by one.

Assume that Eve has intercepted a large number of ciphertexts, C_1 to C_m. Also assume that Eve has observed how long it takes for Bob to decrypt each ciphertext, T_1 to T_m. Eve, who knows how long it takes for the underlying hardware to calculate a multiplication operation, calculated t_1 to t_m, where t_i is the time required to calculate the multiplication operation Result = Result $\times C_i \bmod n$.

Eve can use Algorithm 10.5, which is a simplified version of the algorithm used in practice, to calculate all bits in d (d_0 to d_{k-1}).

The algorithm sets $d_0 = 1$ (because d should be odd) and calculates new values for T_i's (decryption time related to d_1 to d_{k-1}). The algorithm then assumes the next bit is 1

Algorithm 10.5 *Timing attack on RSA*

```
RSA_Timing_Attack ([T1 … Tm])
{
    d0 ← 1                                          // Because d is odd
    Calculate [t1 … tm]
    [T1 … Tm] ← [T1 … Tm] − [t1 … tm]               // Update Ti for the next bit
    for (j from 1 to k − 1)
    {
        Recalculate [t1 … tm]                       // Recalculate ti assuming the next bit is 1
        [D1 … Dm] ← [T1 … Tm] − [t1 … tm]
        var ← variance ([D1 … Dm]) − variance ([T1 … Tm])
        if (var > 0)  dj ← 1      else  dj ← 0
        [T1 … Tm] ← [T1 … Tm] − dj × [t1 … tm]      // Update Ti for the next bit
    }
}
```

and finds some new values D_1 to D_m based on this assumption. If the assumption is correct, each D_i is probably smaller than the corresponding T_i. However, the algorithm uses the variance (or other correlation criteria) to consider all variations of D_i and T_i. If the difference in variance is positive, the algorithm assumes that the next bit is 1; otherwise, it assumes that the next bit is 0. The algorithm then calculates the new T_i's to be used for remaining bits.

There are two methods to thwart timing attack:

1. Add random delays to the exponentiations to make each exponentiation take the same amount of time.
2. Rivest recommended **blinding.** The idea is to multiply the ciphertext by a random number before decryption. The procedure is as follows:
   a. Select a secret random number r between 1 and $(n - 1)$.
   b. Calculate $C_1 = C \times r^e \bmod n$.
   c. Calculate $P_1 = C_1^{\,d} \bmod n$.
   d. Calculate $P = P_1 \times r^{-1} \bmod n$.

Power Attack The **Power attack** is similar to the timing attack. Kocher showed that if Eve can precisely measure the power consumed during decryption, she can launch a power attack based on the principle discussed for timing attack. An iteration involving multiplication and squaring consumes more power than an iteration that uses only squaring. The same kind of techniques used to prevent timing attacks can be used to thwart power attacks.

Recommendations

The following recommendations are based on theoretical and experimental results.

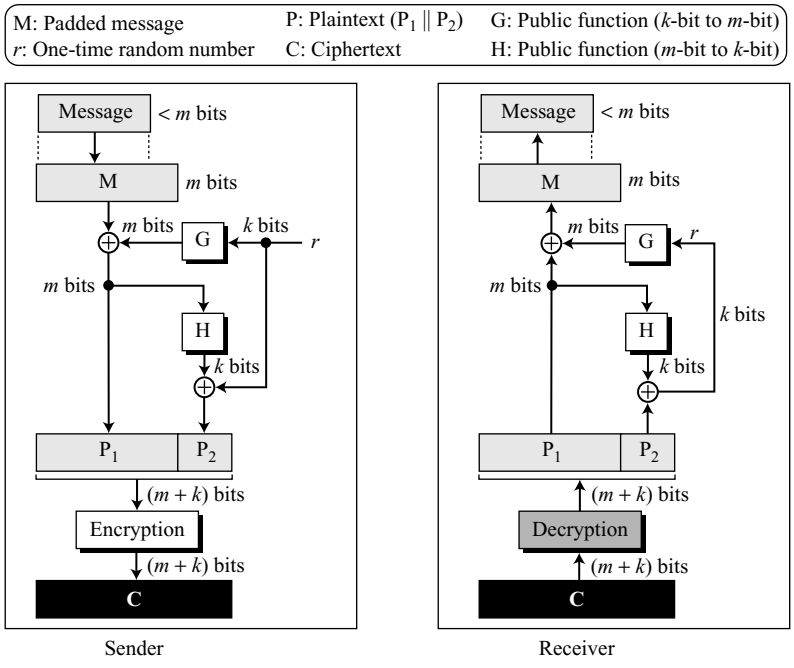
1. The number of bits for n should be at least 1024. This means that n should be around 2^{1024}, or 309 decimal digits.

2. The two primes p and q must each be at least 512 bits. This means that p and q should be around 2^{512} or 154 decimal digits.
3. The values of p and q should not be very close to each other.
4. Both $p - 1$ and $q - 1$ should have at least one large prime factor.
5. The ratio p/q should not be close to a rational number with a small numerator or denominator.
6. The modulus n must not be shared.
7. The value of e should be $2^{16} + 1$ or an integer close to this value.
8. If the private key d is leaked, Bob must immediately change n as well as both e and d. It has been proven that knowledge of n and one pair (e, d) can lead to the discovery of other pairs of the same modulus.
9. Messages must be padded using OAEP, discussed later.

Optimal Asymmetric Encryption Padding (OAEP)

As we mentioned earlier, a short message in RSA makes the ciphertext vulnerable to *short message attacks*. It has been shown that simply adding bogus data (padding) to the message might make Eve's job harder, but with additional efforts she can still attack the ciphertext. The solution proposed by the RSA group and some vendors is to apply a procedure called **optimal asymmetric encryption padding (OAEP).** Figure 10.9

Figure 10.9 *Optimal asymmetric encryption padding (OAEP)*



shows a simple version of this procedure; the implementation may use a more sophisticated version.

The whole idea in Figure 10.9 is that $P = P_1 \parallel P_2$, where P_1 is the masked version of the padded message, M; P_2 is sent to allow Bob to find the mask.

Encryption The following shows the encryption process:

1. Alice pads the message to make an *m*-bit message, which we call M.
2. Alice chooses a random number *r* of *k* bits. Note that *r* is used only once and is then destroyed.
3. Alice uses a public one-way function, G, that takes an *r*-bit integer and creates an *m*-bit integer (*m* is the size of M, and $r < m$). This is the mask.
4. Alice applies the mask G(*r*) to create the first part of the plaintext $P_1 = M \oplus G(r)$. P_1 is the masked message.
5. Alice creates the second part of the plaintext as $P_2 = H(P_1) \oplus r$. The function H is another public function that takes an *m*-bit input and creates an *k*-bit output. This function can be a *cryptographic hash function* (see Chapter 12). P_2 is used to allow Bob to recreate the mask after decryption.
6. Alice creates $C = P^e = (P_1 \parallel P_2)^e$ and sends C to Bob.

Decryption The following shows the decryption process:

1. Bob creates $P = C^d = (P_1 \parallel P_2)$.
2. Bob first recreates the value of *r* using $H(P_1) \oplus P_2 = H(P_1) \oplus H(P_1) \oplus r = r$.
3. Bob uses $G(r) \oplus P = G(r) \oplus G(r) \oplus M = M$ to recreate the value of the padded message.
4. After removing the padding from M, Bob finds the original message.

Error in Transmission

If there is even a single bit error during transmission, RSA will fail. If the received ciphertext is different from what was sent, the receiver cannot determine the original plaintext. The plaintext calculated at the receiver site may be very different from the one sent by the sender. The transmission media must be made error-free by adding error-detecting or error-correcting redundant bits to the ciphertext.

Example 10.8

Here is a more realistic example. We choose a 512-bit *p* and *q*, calculate *n* and $\phi(n)$, then choose *e* and test for relative primeness with $\phi(n)$. We then calculate *d*. Finally, we show the results of encryption and decryption. The integer *p* is a 159-digit number.

p =	961303453135835045741915812806154279093098455949962158225831508796 479404550564706384912571601803475031209866660649242019180878066742 1096063354219926661209

The integer q is a 160-digit number.

q =	120601919572314469182767942044508960015559250546370339360617983217 314821484837646592153894532091752252732268301071206956046025138871 4552496900035966004561 7

The modulus $n = p \times q$. It has 309 digits.

n =	115935041739676149688925098646158875237714573754541447754855261376 147885408326350817276878815968325168468849300625485764111250162414 552339182927162507656772727460097082714127730434960500556347274566 628060099924037102991424472292215772798531727033839381334692684137 327622000966676671831831088373420823444370953

$\phi(n) = (p - 1)(q - 1)$ has 309 digits.

$\phi(n)$ =	115935041739676149688925098646158875237714573754541447754855261376 147885408326350817276878815968325168468849300625485764111250162414 552339182927162507656751054233608492916752034482627988117554787657 013923444405716989581728196098226361075467211864612171359107358640 614008885170265377277264467341066243857664128

Bob chooses e = 35535 (the ideal is 65537) and tests it to make sure it is relatively prime with $\phi(n)$. He then finds the inverse of e modulo $\phi(n)$ and calls it d.

e =	35535
d =	580083028600377639360936612896779175946690620896509621804228661113 805938528223587317062869100300217108590443384021707298690876006115 306202524959884448047568240966247081485817130463240644077704833134 010850947385295645071936774061197326557424237217617674620776371642 076003370853332885321447088595513667029483 1

Alice wants to send the message "THIS IS A TEST", which can be changed to a numeric value using the 00–26 encoding scheme (26 is the *space* character).

P =	19070818260818260026190418 19

The ciphertext calculated by Alice is $C = P^e$, which is

C =	475309123646226827206365550610545180942371796070491716523923905 4 452960613199328566617843418359114151197411252005682979794571736036 101278218847892741566090480023507190715277185914975188465888632101 148354103361657898467968386763733765777465625079280521148141844048 141844308127730590046928742485591664621086 56

Bob can recover the plaintext from the ciphertext using $P = C^d$, which is

P =	19070818260818260026190418 19

The recovered plaintext is "THIS IS A TEST" after decoding.

Applications

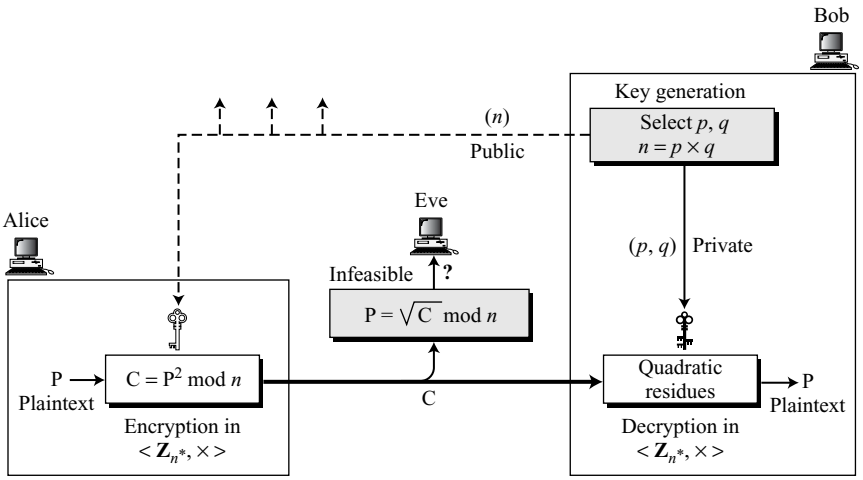
Although RSA can be used to encrypt and decrypt actual messages, it is very slow if the message is long. RSA, therefore, is useful for short messages. In particular, we will see that RSA is used in digital signatures and other cryptosystems that often need to encrypt a small message without having access to a symmetric key. RSA is also used for authentication, as we will see in later chapters.

10.3 RABIN CRYPTOSYSTEM

The **Rabin cryptosystem,** devised by M. Rabin, is a variation of the RSA cryptosystem. RSA is based on the exponentiation congruence; Rabin is based on quadratic congruence. The Rabin cryptosystem can be thought of as an RSA cryptosystem in which the value of e and d are fixed; $e = 2$ and $d = 1/2$. In other words, the encryption is $C \equiv P^2 \pmod{n}$ and the decryption is $P \equiv C^{1/2} \pmod{n}$.

The public key in the Rabin cryptosystem is n; the private key is the tuple (p, q). Everyone can encrypt a message using n; only Bob can decrypt the message using p and q. Decryption of the message is infeasible for Eve because she does not know the values of p and q. Figure 10.10 shows the encryption and decryption.

Figure 10.10 *Encryption, decryption, and key generation in the Rabin cryptosystem*



We need to emphasize a point here. If Bob is using RSA, he can keep d and n and discard p, q, and $\phi(n)$ after key generation. If Bob is using Rabin cryptosystem, he needs to keep p and q.

Procedure

Key generation, encryption, and decryption are described below.

Key Generation

Bob uses the steps shown in Algorithm 10.6 to create his public key and private key.

Algorithm 10.6 *Key generation for Rabin cryptosystem*

```
Rabin_Key_Generation
{
    Choose two large primes p and q in the form 4k + 3 and p ≠ q.
    n ← p × q
    Public_key ← n                    // To be announced publicly
    Private_key ← (p, q)              // To be kept secret
    return Public_key and Private_key
}
```

Although the two primes, p and q, can be in the form $4k + 1$ or $4k + 3$, the decryption process becomes more difficult if the first form is used. It is recommended to use the second form, $4k + 3$, to make the decryption for Alice much easier.

Encryption

Anyone can send a message to Bob using his public key. The encrypting process is shown in Algorithm 10.7.

Algorithm 10.7 *Encryption in Rabin cryptosystem*

```
Rabin_Encryption (n, P)          // n is the public key; P is the ciphertext from Z_n*
{
    C ← P^2 mod n                // C is the ciphertext
    return C
}
```

Although the plaintext P can be chosen from the set $\mathbf{Z}_n$, we have defined the set to be in $\mathbf{Z}_n^*$ to make the decryption easier.

Encryption in the Rabin cryptosystem is very simple. The operation needs only one multiplication, which can be done quickly. This is beneficial when resources are limited. For example, smart cards have limited memory and need to use short CPU time.

Decryption

Bob can use Algorithm 10.8 to decrypt the received ciphertext.

Algorithm 10.8 *Decryption in Rabin cryptosystem*

```
Rabin_Decryption (p, q, C)          // C is the ciphertext; p and q are private keys
{
    a1 ← +(C^((p+1)/4)) mod p
    a2 ← −(C^((p+1)/4)) mod p
    b1 ← +(C^((q+1)/4)) mod q
    b2 ← −(C^((q+1)/4)) mod q
    // The algorithm for the Chinese remainder theorem is called four times.
    P1 ← Chinese_Remainder (a1, b1, p, q)
    P2 ← Chinese_Remainder (a1, b2, p, q)
    P3 ← Chinese_Remainder (a2, b1, p, q)
    P4 ← Chinese_Remainder (a2, b2, p, q)
    return P1, P2, P3, and P4
}
```

Several points should be emphasized here. The decryption is based on the solution of quadratic congruence, discussed in Chapter 9. Because the received ciphertext is the square of the plaintext, it is guaranteed that C has roots (quadratic residues) in $\boldsymbol{Z}_n^*$. The Chinese remainder algorithm is used to find the four square roots.

The most important point about the Rabin system is that it is not deterministic. The decryption has four answers. It is up to the receiver of the message to choose one of the four as the final answer. However, in many situations, the receiver can easily pick up the right answer.

The Rabin cryptosystem is not deterministic: Decryption creates four equally probable plaintexts.

Example 10.9

Here is a very trivial example to show the idea.

1. Bob selects $p = 23$ and $q = 7$. Note that both are congruent to 3 mod 4.
2. Bob calculates $n = p \times q = 161$.
3. Bob announces n publicly; he keeps p and q private.
4. Alice wants to send the plaintext P = 24. Note that 161 and 24 are relatively prime; 24 is in $\boldsymbol{Z}_{161}^*$. She calculates $C = 24^2 = 93 \bmod 161$, and sends the ciphertext 93 to Bob.
5. Bob receives 93 and calculates four values:
   a. $a_1 = +(93^{(23+1)/4}) \bmod 23 = 1 \bmod 23$
   b. $a_2 = -(93^{(23+1)/4}) \bmod 23 = 22 \bmod 23$
   c. $b_1 = +(93^{(7+1)/4}) \bmod 7 = 4 \bmod 7$
   d. $b_2 = -(93^{(7+1)/4}) \bmod 7 = 3 \bmod 7$
6. Bob takes four possible answers, (a_1, b_1), (a_1, b_2), (a_2, b_1), and (a_2, b_2), and uses the Chinese remainder theorem to find four possible plaintexts: 116, **24**, 137, and 45 (all of them relatively prime to 161). Note that only the second answer is Alice's plaintext. Bob needs to make a decision based on the situation. Note also that all four of these answers, when squared modulo n, give the ciphertext 93 sent by Alice.

$116^2 = 93 \bmod 161$ $24^2 = 93 \bmod 161$ $137^2 = 93 \bmod 161$ $45^2 = 93 \bmod 161$

Security of the Rabin System

The Rabin system is secure as long as p and q are large numbers. The complexity of the Rabin system is at the same level as factoring a large number n into its two prime factors p and q. In other words, the Rabin system is as secure as RSA.

10.4 ELGAMAL CRYPTOSYSTEM

Besides RSA and Rabin, another public-key cryptosystem is **ElGamal,** named after its inventor, Taher ElGamal. ElGamal is based on the discrete logarithm problem discussed in Chapter 9.

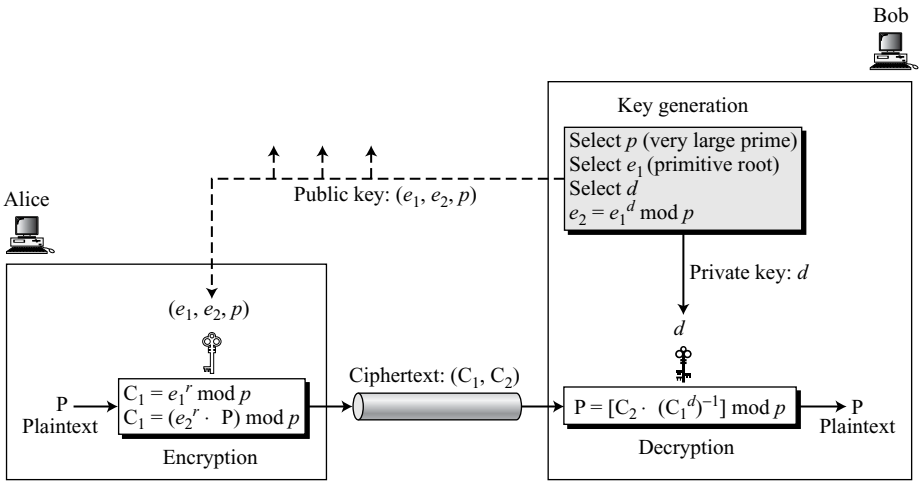
ElGamal Cryptosystem

Recall from Chapter 9 that if p is a very large prime, e_1 is a primitive root in the group $\mathbf{G} = <\mathbf{Z}_p^*, \times >$ and r is an integer, then $e_2 = e_1^{\,r} \bmod p$ is easy to compute using the fast exponential algorithm (square-and-multiply method), but given e_2, e_1, and p, it is infeasible to calculate $r = \log_{e1} e_2 \bmod p$ (discrete logarithm problem).

Procedure

Figure 10.11 shows key generation, encryption, and decryption in ElGamal.

Figure 10.11 *Key generation, encryption, and decryption in ElGamal*



Key Generation

Bob uses the steps shown in Algorithm 10.9 to create his public and private keys.

Algorithm 10.9 *ElGamal key generation*

```
ElGamal_Key_Generation
{
    Select a large prime p
    Select d to be a member of the group G = < Zp*, × > such that 1 ≤ d ≤ p − 2
    Select e1 to be a primitive root in the group G = < Zp*, × >
    e2 ← e1^d mod p
    Public_key ← (e1, e2, p)                    // To be announced publicly
    Private_key ← d                             // To be kept secret
    return Public_key and Private_key
}
```

Encryption

Anyone can send a message to Bob using his public key. The encryption process is shown in Algorithm 10.10. If the fast exponential algorithm (see Chapter 9) is used, encryption in the ElGamal cryptosystem can also be done in polynomial time complexity.

Algorithm 10.10 *ElGamal encryption*

```
ElGamal_Encryption (e1, e2, p, P)               // P is the plaintext
{
    Select a random integer r in the group G = < Zp*, × >
    C1 ← e1^r mod p
    C2 ← (P × e2^r) mod p                       // C1 and C2 are the ciphertexts
    return C1 and C2
}
```

Decryption

Bob can use Algorithm 10.11 to decrypt the ciphertext message received.

Algorithm 10.11 *ElGamal decryption*

```
ElGamal_Decryption (d, p, C1, C2)               // C1 and C2 are the ciphertexts
{
    P ← [C2 (C1^d)^−1] mod p                    // P is the plaintext
    return P
}
```

The bit-operation complexity of encryption or decryption in ElGamal cryptosystem is polynomial.

Proof

The ElGamal decryption expression $C_2 \times (C_1{}^d)^{-1}$ can be verified to be P through substitution:

$$[C_2 \times (C_1{}^d)^{-1}] \bmod p = [(e_2{}^r \times P) \times (e_1{}^{rd})^{-1}] \bmod p = (e_1{}^{dr}) \times P \times (e_1{}^{rd})^{-1} = P$$

Example 10.10

Here is a trivial example. Bob chooses 11 as p. He then chooses $e_1 = 2$. Note that 2 is a primitive root in $\boldsymbol{Z}_{11}$* (see Appendix J). Bob then chooses $d = 3$ and calculates $e_2 = e_1{}^d = 8$. So the public keys are (2, 8, 11) and the private key is 3. Alice chooses $r = 4$ and calculates C_1 and C_2 for the plaintext 7.

Plaintext: 7
$C_1 = e_1{}^r \bmod 11 = 16 \bmod 11 = 5 \bmod 11$
$C_2 = (P \times e_2{}^r) \bmod 11 = (7 \times 4096) \bmod 11 = 6 \bmod 11$
Ciphertext: (5, 6)

Bob receives the ciphertexts (5 and 6) and calculates the plaintext.

$[C_2 \times (C_1{}^d)^{-1}] \bmod 11 = 6 \times (5^3)^{-1} \bmod 11 = 6 \times 3 \bmod 11 = 7 \bmod 11$
Plaintext: 7

Example 10.11

Instead of using $P = [C_2 \times (C_1{}^d)^{-1}] \bmod p$ for decryption, we can avoid the calculation of multiplicative inverse and use $P = [C_2 \times C_1{}^{p-1-d}] \bmod p$ (see Fermat's little theorem in Chapter 9). In Example 10.10, we can calculate $P = [6 \times 5^{11-1-3}] \bmod 11 = 7 \bmod 11$.

Analysis

A very interesting point about the ElGamal cryptosystem is that Alice creates r and keeps it secret; Bob creates d and keeps it secret. The puzzle of this cryptosystem can be solved as follows:

a. Alice sends $C_2 = [e_2{}^r \times P] \bmod p = [(e_1{}^{rd}) \times P] \bmod p$. The expression $(e_1{}^{rd})$ acts as a mask that hides the value of P. To find the value of P, Bob must remove this mask.
b. Because modular arithmetic is being used, Bob needs to create a replica of the mask and invert it (multiplicative inverse) to cancel the effect of the mask.
c. Alice also sends $C_1 = e_1{}^r$ to Bob, which is a part of the mask. Bob needs to calculate $C_1{}^d$ to make a replica of the mask because $C_1{}^d = (e_1{}^r)^d = (e_1{}^{rd})$. In other words, after obtaining the mask replica, Bob inverts it and multiplies the result with C_2 to remove the mask.
d. It might be said that Bob helps Alice make the mask $(e_1{}^{rd})$ without revealing the value of d (d is already included in $e_2 = e_1{}^d$); Alice helps Bob make the mask $(e_1{}^{rd})$ without revealing the value of r (r is already included in $C_1 = e_1{}^r$).

Security of ElGamal

Two attacks have been mentioned for the ElGamal cryptosystem in the literature: attacks based on low modulus and known-plaintext attacks.

Low-Modulus Attacks

If the value of p is not large enough, Eve can use some efficient algorithms (see Chapter 9) to solve the discrete logarithm problem to find d or r. If p is small, Eve can easily find $d = \log_{e1} e_2 \bmod p$ and store it to decrypt any message sent to Bob.This can be done once and used as long as Bob uses the same keys. Eve can also use the value of C_1 to find random number r used by Alice in each transmission $r = \log_{e1} C_1 \bmod p$. Both of these cases emphasize that security of the ElGamal cryptosystem depends on the infeasibility of solving a discrete logarithm problem with a very large modulus. It is recommended that p be at least 1024 bits (300 decimal digits).

Known-Plaintext Attack

If Alice uses the same random exponent r, to encrypt two plaintexts P and P′, Eve discovers P′ if she knows P. Assume that $C_2 = P \times (e_2{}^r) \bmod p$ and $C'_2 = P' \times (e_2{}^r) \bmod p$. Eve finds P′ using the following steps:

1. $(e_2{}^r) = C_2 \times P^{-1} \bmod p$
2. $P' = C'_2 \times (e_2{}^r)^{-1} \bmod p$

It is recommended that Alice use a fresh value of r to thwart the known-plaintext attacks.

For the ElGamal cryptosystem to be secure, *p* must be at least 300 digits and *r* must be new for each encipherment.

Example 10.12

Here is a more realistic example. Bob uses a random integer of 512 bits (the ideal is 1024 bits). The integer p is a 155-digit number (the ideal is 300 digits). Bob then chooses e_1, d, and calculates e_2, as shown below: Bob announces (e_1, e_2, p) as his public key and keeps d as his private key.

p =	115348992725616762449253137170143317404900945326098349598143469219056898698622645932129754737871895144368891765264730936159299937280611659643473534400085 77
e_1 =	2
d =	1007
e_2 =	97886413043009189508766856938097739043880062887337687610022062233255450707415618921231831770461014167336015088413294085724853770315820660100725587074 55

Alice has the plaintext P = 3200 to send to Bob. She chooses $r = 545131$, calculates C_1 and C_2, and sends them to Bob.

P =	3200
***r* =**	545131
C_1 =	887297069383528471022570471492275663120260067256562125018188351429417223599712681114105363661705173051581533189165400973736355080295736788569060619152881
C_2 =	708454333048929944577016012380794999567436021836192446961774506921244696155165800779455593080345889614402408599525919579209721628879681350582779566430295O

Bob calculates the plaintext $P = C_2 \times ((C_1)^d)^{-1} \bmod p = 3200 \bmod p$.

P =	3200

Application

ElGamal can be used whenever RSA can be used. It is used for key exchange, authentication, and encryption and decryption of small messages.

10.5 ELLIPTIC CURVE CRYPTOSYSTEMS

Although RSA and ElGamal are secure asymmetric-key cryptosystems, their security comes with a price, their large keys. Researchers have looked for alternatives that give the same level of security with smaller key sizes. One of these promising alternatives is the **elliptic curve cryptosystem (ECC).** The system is based on the theory of **elliptic curves.** Although the deep involvement of this theory is beyond the scope of this book, this section first gives a very simple introduction to three types of elliptic curves and then suggests a flavor of a cryptosystem that uses some of these curves.

Elliptic Curves over Real Numbers

Elliptic curves, which are not directly related to ellipses, are cubic equations in two variables that are similar to the equations used to calculate the length of a curve in the circumference of an ellipse. The general equation for an elliptic curve is

$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

Elliptic curves over real numbers use a special class of elliptic curves of the form

$$y^2 = x^3 + ax + b$$

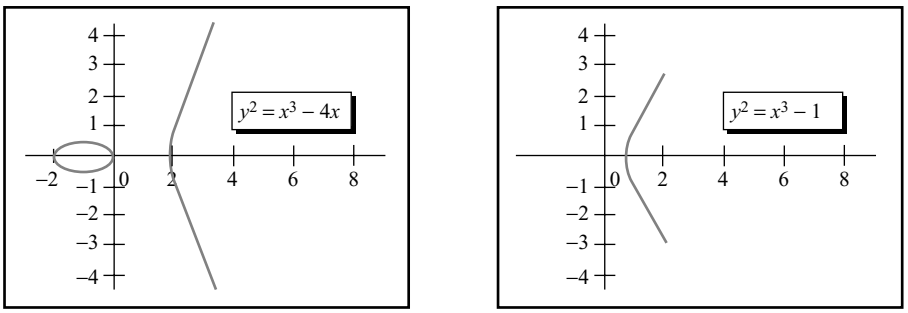
In the above equation, if $4a^3 + 27b^2 \neq 0$, the equation represents a **nonsingular elliptic curve;** otherwise, the equation represented a **singular elliptic curve.** In a nonsingular elliptic curve, the equation $x^3 + ax + b = 0$ has three distinct roots (real or complex); in a singular elliptic curve the equation $x^3 + ax + b = 0$ does not have three distinct roots.

Looking at the equation, we can see that the left-hand side has a degree of 2 while the right-hand side has a degree of 3. This means that a horizontal line can intersects the curve in three points if all roots are real. However, a vertical line can intersects the curve at most in two points.

Example 10.13

Figure 10.12 shows two elliptic curves with equations $y^2 = x^3 - 4x$ and $y^2 = x^3 - 1$. Both are nonsingular. However, the first has three real roots ($x = -2$, $x = 0$, and $x = 2$), but the second has only one real root ($x = 1$) and two imaginary ones.

Figure 10.12 *Two elliptic curves over a real field*



a. Three real roots

b. One real and two imaginary roots

An Abelian Group

Let us define an abelian (commutative) group (see Chapter 4) using points on an elliptic curve. A tuple $P = (x_1, y_1)$ represents a point on the curve if x_1 and y_1 are the coordinates of a point on the curve that satisfy the equation of the curve. For example, the points P = (2.0, 0.0), Q = (0.0, 0.0), R = (–2.0, 0.0), S = (10.0, 30.98), and T = (10.0, –30.98) are all points on the curve $y^2 = x^3 - 4x$. Note that each point is represented by two real numbers. Recall from Chapter 4 that to create an abelian group we need a set, an operation on the set, and five properties that are satisfied by the operation. The group in this case is **G** = <**E**, +>.

Set We define the set as the points on the curve, where each point is a pair of real numbers. For example, the set E for the elliptic curve $y^2 = x^3 - 4x$ is shown as

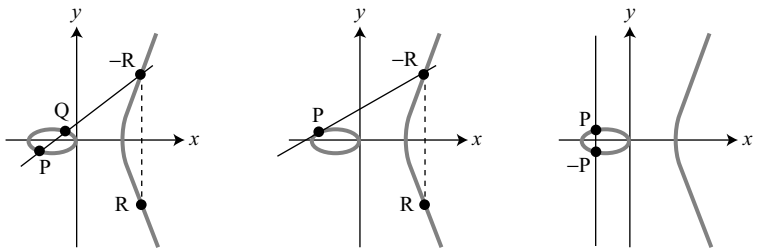
$$\mathbf{E} = \{(2.0, 0.0), (0.0, 0.0), (-2.0, 0.0), (10.0, 30.98), (10.0, -30.98), \ldots\}$$

Operation The specific properties of a nonsingular elliptic curve allows us to define an *addition* operation on the points of the curve. However, we need to remember that the *addition* operation here is different from the operation that has been defined for integers. The operation is the addition of two points on the curve to get another point on the curve

$$R = P + Q, \text{ where } P = (x_1, y_1), Q = (x_2, y_2), \text{ and } R = (x_3, y_3)$$

To find R on the curve, consider three cases as shown in Figure 10.13.

Figure 10.13 *Three adding cases in an elliptic curve*



1. In the first case, the two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ have different x-coordinates and y-coordinates ($x_1 \neq y_1$ and $x_2 \neq y_2$), as shown in Figure 10.13a. The line connecting P and Q intercepts the curve at a point called –R. R is the reflection of –R with respect to the x-axis. The coordinates of the point R, x_3 and y_3, can be found by first finding the slope of the line, λ, and then calculating the values of x_3 and y_3, as shown below:

$$\lambda = (y_2 - y_1) / (x_2 - x_1)$$
$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

2. In the second case, the two points overlap (R = P + P), as shown in Figure 10.13b. In this case, the slope of the line and the coordinates of the point R can be found as shown below:

$$\lambda = (3x_1^2 + a)/(2y_1)$$
$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

3. In the third case, the two points are additive inverses of each other as shown in Figure 10.13c. If the first point is P = (x_1, y_1), the second point is Q = $(x_1, -y_1)$. The line connecting the two points does not intercept the curve at a third point. Mathematicians say that the intercepting point is at infinity; they define a point ***O*** as the *point at infinity* or *zero point,* which is the *additive identity* of the group.

Properties of the Operation The following are brief definitions of the properties of the operation as discussed in Chapter 4:

1. *Closure:* It can be proven that adding two points, using the addition operation defined in the previous section, creates another point on the curve.
2. *Associativity:* It can be proven that (P + Q) + R = P + (Q + R).
3. *Commutativity:* The group made from the points on a non-singular elliptic curve is an abelian group; it can be proven that P + Q = Q + P.
4. *Existence of identity:* The additive identity in this case is the *zero point,* ***O***. In other words P = P + ***O*** = ***O*** + P.
5. *Existence of inverse*: Each point on the curve has an inverse. The inverse of a point is its reflection with respect to the *x*-axis. In other words, the point P = (x_1, y_1) and Q = $(x_1, -y_1)$ are inverses of each other, which means that P + Q = ***O***. Note that the identity element is the inverse of itself.

A Group and a Field

Note that the previous discussion refers to two algebraic structures: a group and a field. The group defines the set of the points on the elliptic curve and the addition operation on the points. The field defines the addition, subtraction, multiplication, and division using operations on real numbers that are needed to find the addition of the points in the group.

Elliptic Curves over GF(*p*)

Our previous elliptic curve group used a real field for calculations involved in adding points. Cryptography requires modular arithmetic. We have defined an elliptic curve group with an addition operation, but the operation on the coordinates of the point are over the **GF**(*p*) field with $p > 3$. In modular arithmetic, the points on the curve do not make nice graphs as seen in the previous figures, but the concept is the same. We use the same addition operation with the calculation done in modulo *p*. We call the resulting elliptic curve $E_p(a, b)$, where *p* defines the modulus and *a* and *b* are the coefficient of the equation $y^2 = x^3 + ax + b$. Note that although the value of *x* in this case ranges from 0 to *p*, normally not all points are on the curve.

Finding an Inverse

The inverse of a point (*x*, *y*) is (*x*, –*y*), where –*y* is the additive inverse of *y.* For example, if $p = 13$, the inverse of (4, 2) is (4, 11).

Finding Points on the Curve

Algorithm 10.12 shows the pseudocode for finding the points on the curve $E_p(a, b)$.

Algorithm 10.12 *Pseudocode for finding points on an elliptic curve*

```
ellipticCurve_points (p, a, b)                          // p is the modulus
{
    x ← 0
    while (x < p)
    {
        w ← (x^3 + ax + b) mod p                        // w is y^2
        if (w is a perfect square in Z_p) output (x, √w) (x, − √w)
        x ← x + 1
    {
}
```

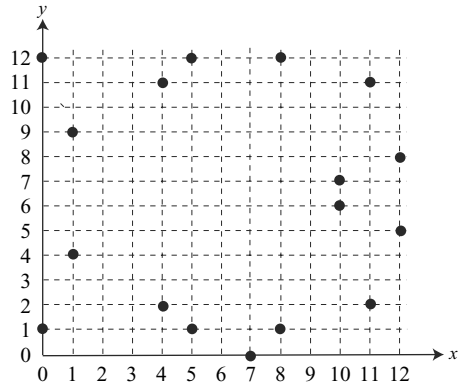
Example 10.14

Define an elliptic curve $E_{13}(1, 1)$. The equation is $y^2 = x^3 + x + 1$ and the calculation is done modulo 13. Points on the curve can be found as shown in Figure 10.14.

Figure 10.14 *Points on an elliptic curve over GF(p)*

(0, 1)	(0, 12)
(1, 4)	(1, 9)
(4, 2)	(4, 11)
(5, 1)	(5, 12)
(7, 0)	(7, 0)
(8, 1)	(8, 12)
(10, 6)	(10, 7)
(11, 2)	(11, 11)
(12, 5)	(12, 8)

Points



Graph

Note the following:

a. Some values of y^2 do not have a square root in modulo 13 arithmetic. These are not points on this elliptic curve. For example, the points with $x = 2$, $x = 3$, $x = 6$, and $x = 9$ are not on the curve.

b. Each point defined for the curve has an inverse. The inverses are listed as pairs. Note that (7, 0) is the inverse of itself.

c. Note that for a pair of inverse points, the y values are additive inverses of each other in $\boldsymbol{Z}_p$. For example, 4 and 9 are additive inverses in $\boldsymbol{Z}_{13}$. So we can say that if 4 is y, then 9 is $-y$.

d. The inverses are on the same vertical lines.

Adding Two Points

We use the elliptic curve group defined earlier, but calculations are done in **GF**(p). Instead of subtraction and division, we use additive and multiplicative inverses.

Example 10.15

Let us add two points in Example 10.14, R = P + Q, where P = (4, 2) and Q = (10, 6).

a. $\lambda = (6 - 2) \times (10 - 4)^{-1} \bmod 13 = 4 \times 6^{-1} \bmod 13 = 5 \bmod 13$.
b. $x = (5^2 - 4 - 10) \bmod 13 = 11 \bmod 13$.
c. $y = [5\ (4 - 11) - 2] \bmod 13 = 2 \bmod 13$.
d. R = (11, 2), which is a point on the curve in Example 10.14.

Multiplying a Point by a Constant

In arithmetic, multiplying a number by a constant k means adding the number to itself k times. The situation here is the same. Multiplying a point P on an elliptic curve by a constant k means adding the point P to itself k times. For example, in E_{13} (1, 1), if the point (1, 4) is multiplied by 4, the result is the point (5, 1). If the point (8, 1) is multiplied by 3, the result is the point (10, 7).

Elliptic Curves over GF(2^n)

Calculation in the elliptic curve group can be defined over the GF(2^n) field. Recall from Chapter 4 that elements of the set in this field are n-bit words that can be interpreted as polynomials with coefficient in **GF**(2). Addition and multiplication on the elements are the same as addition and multiplication on polynomials. To define an elliptic curve over **GF**(2^n), one needs to change the cubic equation. The common equation is

$$y^2 + xy = x^3 + ax^2 + b$$

where $b \neq 0$. Note that the value of x, y, a, and b are polynomials representing n-bit words.

Finding Inverses

If P = (x, y), then –P = (x, $x + y$).

Finding Points on the Curve

We can write an algorithm to find the points on the curve using generators for polynomials discussed in Chapter 4. This algorithm is left as an exercise. Following is a very trivial example.

Example 10.16

We choose **GF**(2^3) with elements {0, 1, g, g^2, g^3, g^4, g^5, g^6} using the irreducible polynomial of $f(x) = x^3 + x + 1$, which means that $g^3 + g + 1 = 0$ or $g^3 = g + 1$. Other powers of g can be calculated accordingly. The following shows the values of the g's.

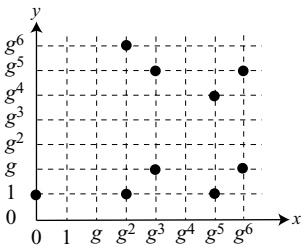
0	000	$g^3 = g + 1$	011
1	001	$g^4 = g^2 + g$	110
g	010	$g^5 = g^2 + g + 1$	111
g^2	100	$g^6 = g^2 + 1$	101

Using the elliptic curve $y^2 + xy = x^3 + g^3x^2 + 1$, with $a = g^3$ and $b = 1$, we can find the points on this curve, as shown in Figure 10.15.

Figure 10.15 *Points on an elliptic curve over GF(2^n)*

(0, 1)	(0, 1)
(g^2, 1)	(g^2, g^6)
(g^3, g^2)	(g^3, g^5)
(g^5, 1)	(g^5, g^4)
(g^6, g)	(g^6, g^5)

Points



Graph

Adding Two Points

The rules for adding points in **GF**(2^n) is slightly different from the rules for **GF**(p).

1. If P = (x_1, y_1), Q = (x_2, y_2), Q ≠ –P, and Q ≠ P, then R = (x_3, y_3) = P + Q can be found as

$$\lambda = (y_2 + y_1) / (x_2 + x_1)$$
$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \qquad y_3 = \lambda (x_1 + x_3) + x_3 + y_1$$

2. If Q = P, then R = P + P (or R = 2P) can be found as

$$\lambda = x_1 + y_1 / x_1$$
$$x_3 = \lambda^2 + \lambda + a \qquad y_3 = {x_1}^2 + (\lambda + 1)\, x_3$$

Example 10.17

Let us find R = P + Q, where P = (0, 1) and Q = (g^2, 1). We have $\lambda = 0$ and R = (g^5, g^4).

Example 10.18

Let us find R = 2P, where P = (g^2, 1). We have $\lambda = g^2 + 1/g^2 = g^2 + g^5 = g + 1$ and R = (g^6, g^5).

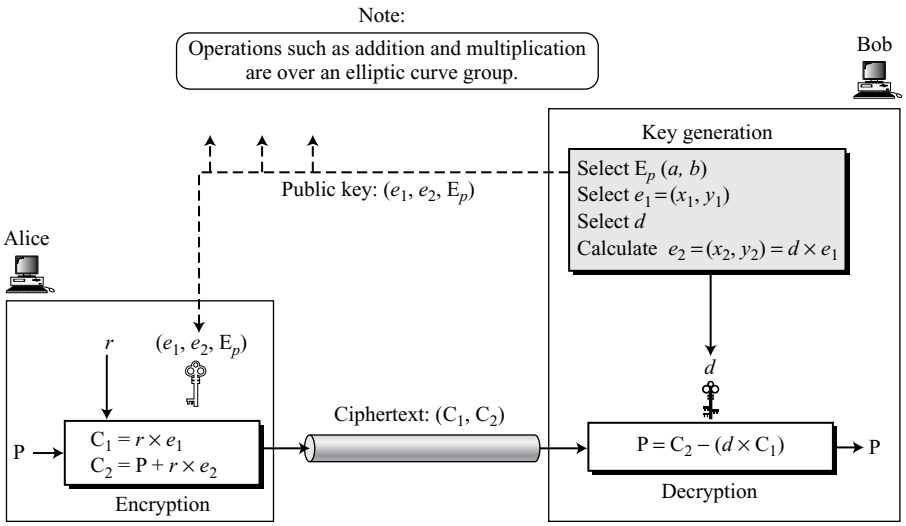
Multiplying a Point by a Constant

To multiply a point by a constant, the points must be added continuously with attention to the rule for R = 2P.

Elliptic Curve Cryptography Simulating ElGamal

Several methods have been used to encrypt and decrypt using elliptic curves. The common one is to simulate the ElGamal cryptosystem using an elliptic curve over **GF**(p) or **GF**(2^n), as shown in Figure 10.16.

Figure 10.16 *ElGamal cryptosystem using the elliptic curve*



Generating Public and Private Keys

1. Bob chooses E(a, b) with an elliptic curve over **GF**(p) or **GF**(2^n).
2. Bob chooses a point on the curve, $e_1(x_1, y_1)$.
3. Bob chooses an integer d.
4. Bob calculates $e_2(x_2, y_2) = d \times e_1(x_1, y_1)$. Note that multiplication here means multiple addition of points as defined before.
5. Bob announces E(a, b), $e_1(x_1, y_1)$, and $e_2(x_2, y_2)$ as his public key; he keeps d as his private key.

Encryption

Alice selects P, a point on the curve, as her plaintext, P. She then calculates a pair of points on the text as ciphertexts:

$$\mathbf{C_1 = r \times e_1} \qquad \mathbf{C_2 = P + r \times e_2}$$

The reader may wonder how an arbitrary plaintext can be a point on the elliptic curve. This is one of the challenging issues in the use of the elliptic curve for simulation. Alice needs to use an algorithm to find a one-to-one correspondence between symbols (or a block of text) and the points on the curve.

Decryption

Bob, after receiving C_1 and C_2, calculates P, the plaintext using the following formula.

$$\mathbf{P = C_2 - (d \times C_1)}$$ The minus sign here means adding with the inverse.

We can prove that the P calculated by Bob is the same as that intended by Alice, as shown below:

$$P + r \times e_2 - (d \times r \times e_1) = P + (r \times d \times e_1) - (r \times d \times e_1) = P + \boldsymbol{O} = P$$

P, C_1, C_2, e_1, and e_2 are all points on the curve. Note that the result of adding two inverse points on the curve is the *zero point*.

Example 10.19

Here is a very trivial example of encipherment using an elliptic curve over **GF**(p).

1. Bob selects $E_{67}(2, 3)$ as the elliptic curve over **GF**(p).
2. Bob selects $e_1 = (2, 22)$ and $d = 4$.
3. Bob calculates $e_2 = (13, 45)$, where $e_2 = d \times e_1$.
4. Bob publicly announces the tuple (E, e_1, e_2).
5. Alice wants to send the plaintext $P = (24, 26)$ to Bob. She selects $r = 2$.
6. Alice finds the point $C_1 = (35, 1)$, where $C_1 = r \times e_1$.
7. Alice finds the point $C_2 = (21, 44)$, where $C_2 = P + r \times e_2$.
8. Bob receives C_1 and C_2. He uses $2 \times C_1$ (35, 1) to get (23, 25).
9. Bob inverts the point (23, 25) to get the point (23, 42).
10. Bob adds (23, 42) with $C_2 = (21, 44)$ to get the original plaintext $P = (24, 26)$.

Comparison

The following shows a quick comparison of the original ElGamal algorithm with its simulation using the elliptic curve.

a. The original algorithm uses a multiplicative group; the simulation uses an elliptic group.
b. The two exponents in the original algorithm are numbers in the multiplicative group; the two multipliers in the simulation are points on the elliptic curve.
c. The private key in each algorithm is an integer.
d. The secret numbers chosen by Alice in each algorithm are integers.
e. The exponentiation in the original algorithm is replaced by the multiplication of a point by a constant.
f. The multiplication in the original algorithm is replaced by addition of points.
g. The inverse in the original algorithm is the multiplicative inverse in the multiplicative group; the inverse in the simulation is the additive inverse of a point on the curve.
h. Calculation is usually easier in the elliptic curve because multiplication is simpler than exponentiation, addition is simpler than multiplication, and finding the inverse is much simpler in the elliptic curve group than in a multiplicative group.

Security of ECC

To decrypt the message, Eve needs to find the value of r or d.

a. If Eve knows r, she can use $P = C_2 - (r \times e_2)$ to find the point P related to the plaintext. But to find r, Eve needs to solve the equation $C_1 = r \times e_1$. This means, given two points on the curve, C_1 and e_1, Eve must find the multiplier that creates C_1 starting from e_1. This is referred to as the **elliptic curve logarithm problem,** and the only method available to solve it is the Polard rho algorithm, which is infeasible if r is large, and p in GF(p) or n in GF(2^n) is large.
b. If Eve knows d, she can use $P = C_2 - (d \times C_1)$ to find the point P related to the plaintext. Because $e_2 = d \times e_1$, this is the same type of problem. Eve knows the value of e_1 and e_2; she needs to find the multiplier d.

The security of ECC depends on the difficulty of solving the elliptic curve logarithm problem.

Modulus Size

For the same level of security (computational effort), the modulus, n, can be smaller in ECC than in RSA. For example, ECC over the **GF**(2^n) with n of 160 bits can provide the same level of security as RSA with n of 1024 bits.

10.6 RECOMMENDED READING

The following books and websites provide more details about subjects discussed in this chapter. The items enclosed in brackets refer to the reference list at the end of the book.

Books

The RSA cryptosystem is discussed in [Sti06], [Sta06], [PHS03], [Vau06], [TW06], and [Mao04]. The Rabin and ElGamal cryptosystems are discussed in [Sti06] and [Mao04]. Elliptic curve cryptography is discussed in [Sti06], [Eng99], and [Bla99].

WebSites

The following websites give more information about topics discussed in this chapter.

http://www1.ics.uci.edu/~mingl/knapsack.html
www.dtc.umn.edu/~odlyzko/doc/arch/**knapsack**.survey.pdf
http://en.wikipedia.org/wiki/RSA
citeseer.ist.psu.edu/boneh99twenty.html
www.mat.uniroma3.it/users/pappa/SLIDES/**RSA**-HRI_05.pdf
http://en.wikipedia.org/wiki/Rabin_cryptosystem
http://en.wikipedia.org/wiki/ElGamal_encryption
ww.cs.purdue.edu/homes/wspeirs/**elgamal**.pdf
http://en.wikipedia.org/wiki/Elliptic_curve_cryptography
www.cs.utsa.edu/~rakbani/publications/Akbani-ECC-IEEESMC03.pdf

10.7 KEY TERMS

asymmetric-key cryptography
blinding
broadcast attack
common modulus attack
Coppersmith theorem attack
cycling attack
ElGamal cryptosystem
elliptic curve
elliptic curve cryptosystem (ECC)
elliptic curve logarithm problem
function
invertible function
knapsack cryptosystem
low decryption exponent attack
low encryption exponent attack
nonsingular elliptic curve
one-way function (OWF)
optimal asymmetric encryption padding (OAEP)
power attack
private key
public key
Rabin cryptosystem
random fault attack
related message attack
revealed decryption exponent attack
RSA (Rivest, Shamir, Adleman) cryptosystem
short message attack
short pad attack
singular elliptic curve
superincreasing tuple
symmetric-key cryptography
timing attack
trapdoor
trapdoor one-way function (TOWF)
unconcealed message attack

10.8 SUMMARY

- There are two ways to achieve secrecy: symmetric-key cryptography and asymmetric-key cryptography. These two will exist in parallel and complement each other; the advantages of one can compensate for the disadvantages of the other.
- The conceptual differences between the two systems are based on how they keep a secret. In symmetric-key cryptography, the secret needs to be shared between two entities; in asymmetric-key cryptography, the secret is personal (unshared).
- Symmetric-key cryptography is based on substitution and permutation of symbols; asymmetric-key cryptography is based on applying mathematical functions to numbers.
- Asymmetric-key cryptography uses two separate keys: one private and one public. Encryption and decryption can be thought of as locking and unlocking padlocks with keys. The padlock that is locked with a public key can be unlocked only with the corresponding private key.
- In asymmetric-key cryptography, the burden of providing security is mostly on the shoulder of the receiver (Bob), who needs to create two keys: one private and one public. Bob is responsible for distributing the private key to the community. This can be done through a public-key distribution channel.

- Unlike in symmetric-key cryptography, in asymmetric-key cryptography plaintexts and ciphertexts are treated as integers. The message must be encoded as an integer (or a set of integers) before encryption; the integer (or the set of integers) must be decoded into the message after decryption. Asymmetric-key cryptography is normally used to encrypt or decrypt small messages, such as a cipher key for symmetric-key cryptography.
- The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function (TOWF), which is a function such that $\boldsymbol{f}$ is easy to compute, but $\boldsymbol{f}^{-1}$ is computationally infeasible unless a trapdoor is used.
- A brilliant idea of public-key cryptography came from Merkle and Hellman in their knapsack cryptosystem. If we are told which elements, from a predefined set of numbers, are in a knapsack, we can easily calculate the sum of the numbers; if we are told the sum, it is difficult to say which elements are in the knapsack unless the knapsack is filled with elements from a superincreasing set.
- The most common public-key algorithm is the RSA cryptosystem. RSA uses two exponents, e and d, where e is public and d is private. Alice uses $C = P^e \bmod n$ to create ciphertext C from plaintext P; Bob uses $P = C^d \bmod n$ to retrieve the plaintext sent by Alice.
- RSA uses two algebraic structures: a ring and a group. Encryption and decryption are done using the commutative ring $\mathbf{R} = <\mathbf{Z}_n, +, \times>$ with two arithmetic operations: addition and multiplication. RSA uses a multiplicative group $\mathbf{G} = <\mathbf{Z}_n^*, \times>$ for key generation.
- No devastating attacks have yet been discovered on RSA. Several attacks have been predicted based on factorization, chosen-ciphertext, decryption exponent, encryption exponent, plaintext, modulus, and implementation.
- The Rabin cryptosystem is a variation of the RSA cryptosystem. RSA is based on the exponentiation congruence; Rabin is based on quadratic congruence. We can think of Rabin as the RSA in which the value of $e = 2$ and $d = 1/2$. The Rabin cryptosystem is secure as long as p and q are large numbers. The complexity of the Rabin cryptosystem is at the same level as factoring a large number n into its two prime factors p and q.
- The ElGamal cryptosystem is based on the discrete logarithm problem. ElGamal uses the idea of primitive roots in $\mathbf{Z}_p^*$. Encryption and decryption in ElGamal use the group $\mathbf{G} = <\mathbf{Z}_p^*, \times>$. The public key is two exponents e_1 and e_2; the private key is an integer d. The security of ElGamal is based on the infeasibility of solving discrete logarithm problems. However, an attack based on low modulus and a known-plaintext attack have been mentioned in the literature.
- Another cryptosystem discussed in this chapter is based on elliptic curves. Elliptic curves are cubic equations in two variables. Elliptic curves over real numbers use a special class of elliptic curves $y^2 = x^3 + ax + b$ where $4a^3 + 27b^2 \neq 0$. An abelian group has been defined over the elliptic curve with an addition operation that shows how two points on the curve can be added to get another point on the curve.

- Elliptic curve cryptography (ECC) uses two algebraic structures, an abelian group and a field. The field can be the nonfinite field of real numbers, **GF**(p) and **GF**(2^n).We have been shown how the ElGamal cryptosystem can be simulated using elliptic curves over finite fields. The security of the ECC depends on the *elliptic curve logarithm problem,* a solution which is infeasible if the modulus is large.

10.9 PRACTICE SET

Review Questions

1. Distinguish between symmetric-key and asymmetric-key cryptosystems.
2. Distinguish between public and private keys in an asymmetric-key cryptosystem. Compare and contrast the keys in symmetric-key and asymmetric-key cryptosystems.
3. Define a trapdoor one-way function and explain its use in asymmetric-key cryptography.
4. Briefly explain the idea behind the knapsack cryptosystem.
   a. What is the one-way function in this system?
   b. What is the trapdoor in this system?
   c. Define the public and private keys in this system.
   d. Describe the security of this system.
5. Briefly explain the idea behind the RSA cryptosystem.
   a. What is the one-way function in this system?
   b. What is the trapdoor in this system?
   c. Define the public and private keys in this system.
   d. Describe the security of this system.
6. Briefly explain the idea behind the Rabin cryptosystem.
   a. What is the one-way function in this system?
   b. What is the trapdoor in this system?
   c. Define the public and private keys in this system.
   d. Describe the security of this system.
7. Briefly explain the idea behind the ElGamal cryptosystem.
   a. What is the one-way function in this system?
   b. What is the trapdoor in this system?
   c. Define the public and private keys in this system.
   d. Describe the security of this system.
8. Briefly explain the idea behind ECC.
   a. What is the one-way function in this system?
   b. What is the trapdoor in this system?
   c. Define the public and private keys in this system.
   d. Describe the security of this system.

9. Define elliptic curves and explain their applications in cryptography.
10. Define the operation used in the abelian group made of points on an elliptic curve.

Exercises

11. Given the superincreasing tuple $b = [7, 11, 23, 43, 87, 173, 357]$, $r = 41$, and modulus $n = 1001$, encrypt and decrypt the letter "a" using the knapsack cryptosystem. Use [7 6 5 1 2 3 4] as the permutation table.
12. In RSA:
    a. Given $n = 221$ and $e = 5$, find d.
    b. Given $n = 3937$ and $e = 17$, find d.
    c. Given $p = 19$, $q = 23$, and $e = 3$, find n, $\phi(n)$, and d.
13. To understand the security of the RSA algorithm, find d if you know that $e = 17$ and $n = 187$.
14. In RSA, given n and $\phi(n)$, calculate p and q.
15. In RSA, given $e = 13$ and $n = 100$
    a. encrypt the message "HOW ARE YOU" using 00 to 25 for letters A to Z and 26 for the space. Use different blocks to make $P < n$.
16. In RSA, given $n = 12091$ and $e = 13$, Encrypt the message "THIS IS TOUGH" using the 00 to 26 encoding scheme. Decrypt the ciphertext to find the original message.
17. In RSA:
    a. Why can't Bob choose 1 as the public key e?
    b. What is the problem in choosing 2 as the public key e?
18. Alice uses Bob's RSA public key ($e = 17$, $n = 19519$) to send a four-character message to Bob using the (A $\leftrightarrow$ 0, B $\leftrightarrow$ 1, … Z $\leftrightarrow$ 25) encoding scheme and encrypting each character separately. Eve intercepts the ciphertext (6625 0 2968 17863) and decrypts the message without factoring the modulus. Find the plaintext and explain why Eve could easily break the ciphertext.
19. Alice uses Bob's RSA public key ($e = 7$, $n = 143$) to send the plaintext P = 8 encrypted as ciphertext C = 57. Show how Eve can use the chosen-ciphertext attack if she has access to Bob's computer to find the plaintext.
20. Alice uses Bob's RSA public key ($e = 3$, $n = 35$) and sends the ciphertext 22 to Bob. Show how Eve can find the plaintext using the *cycling attack*.
21. Suggest how Alice can prevent a *related message attack* on RSA.
22. Using the Rabin cryptosystem with $p = 47$ and $q = 11$:
    a. Encrypt P = 17 to find the ciphertext.
    b. Use the Chinese remainder theorem to find four possible plaintexts.
23. In ElGamal, given the prime $p = 31$:
    a. Choose an appropriate e_1 and d, then calculate e_2.
    b. Encrypt the message "HELLO"; use 00 to 25 for encoding. Use different blocks to make $P < p$.
    c. Decrypt the ciphertext to obtain the plaintext.

24. In ElGamal, what happens if C_1 and C_2 are swapped during the transition?
25. Assume that Alice uses Bob's ElGamal public key ($e_1 = 2$ and $e_2 = 8$) to send two messages P = 17 and P′ = 37 using the same random integer $r = 9$. Eve intercepts the ciphertext and somehow she finds the value of P = 17. Show how Eve can use a known-plaintext attack to find the value of P′.
26. In the elliptic curve E(1, 2) over the **GF**(11) field:
    a. Find the equation of the curve.
    b. Find all points on the curve and make a figure similar to Figure 10.14.
    c. Generate public and private keys for Bob.
    d. Choose a point on the curve as a plaintext for Alice.
    e. Create ciphertext corresponding to the plaintext in part d for Alice.
    f. Decrypt the ciphertext for Bob to find the plaintext sent by Alice.
27. In the elliptic curve E(g^4, 1) over the **GF**(2^4) field:
    a. Find the equation of the curve.
    b. Find all points on the curve and make a figure similar to Figure 10.15.
    c. Generate public and private keys for Bob.
    d. Choose a point on the curve as a plaintext for Alice.
    e. Create ciphertext corresponding to the plaintext in part d for Alice.
    f. Decrypt the ciphertext for Bob to find the plaintext sent by Alice.
28. Using the knapsack cryptosystem:
    a. Write an algorithm for encryption.
    b. Write an algorithm for decryption.
29. In RSA:
    a. Write an algorithm for encryption using OAEP.
    b. Write an algorithm for decryption using OAEP.
30. Write an algorithm for a *cycling attack* on RSA.
31. Write an algorithm to add two points on an elliptic curve over **GF**(p).
32. Write an algorithm to add two points on an elliptic curve over **GF**(2^n).