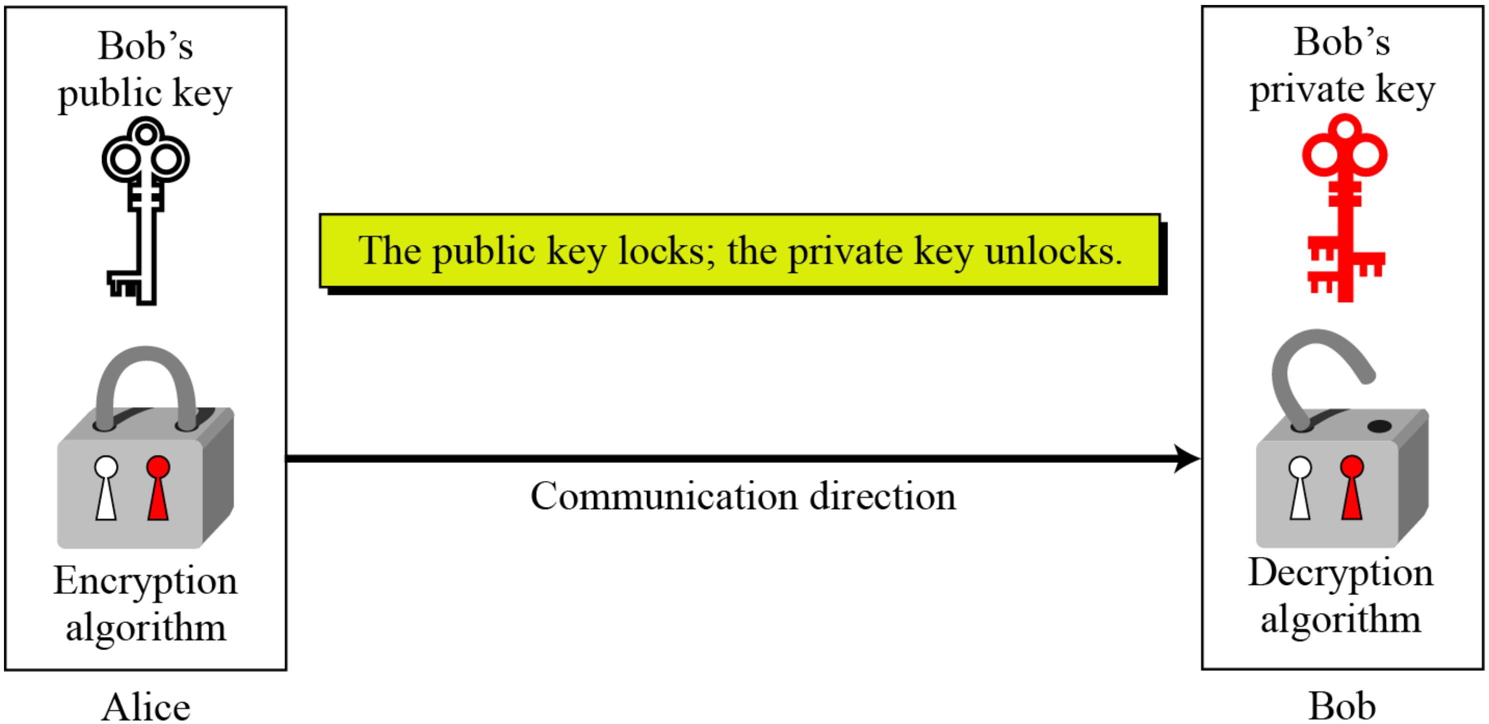
Chapter 10

Asymmetric-Key Cryptography

10.1.1 Keys

Asymmetric key cryptography uses two separate keys: one private and one public.

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



10.1.2 General Idea

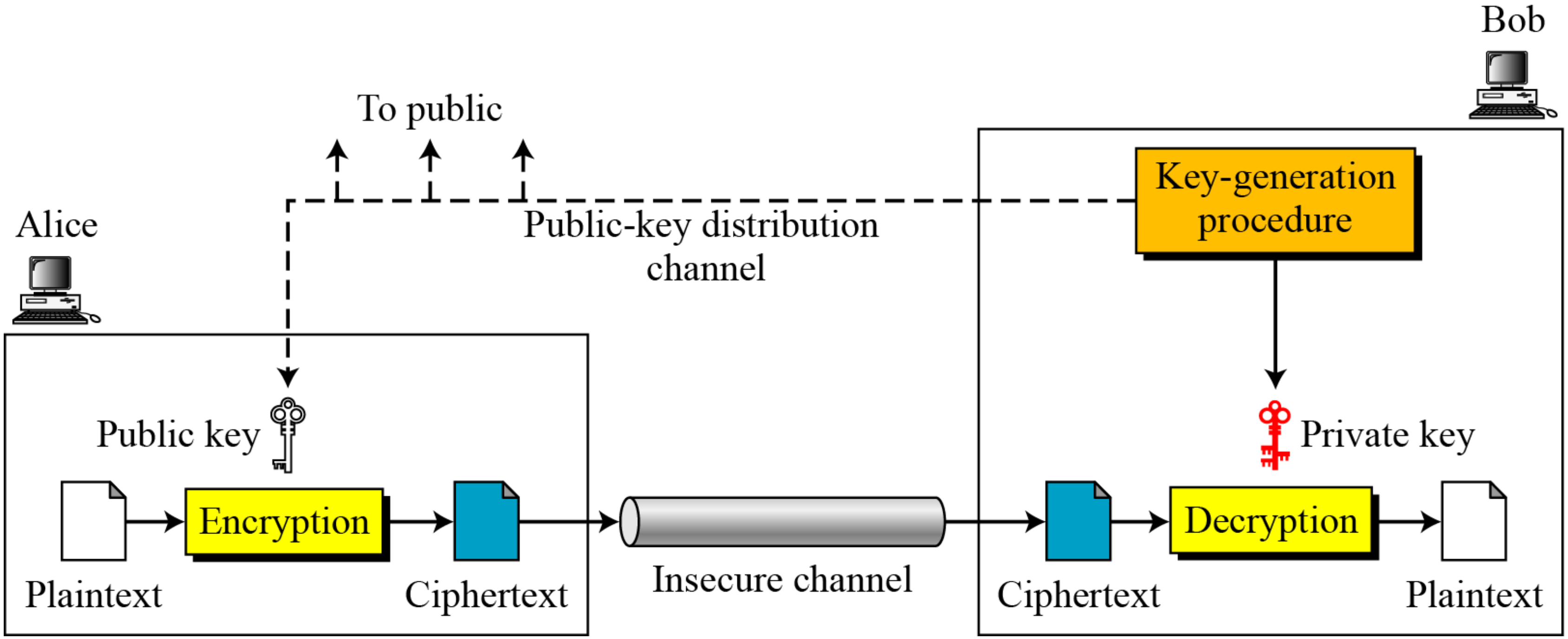


Figure 10.2 *General idea of asymmetric-key cryptosystem*

10.1.2 Continued

Plaintext/Ciphertext

Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers in asymmetric-key cryptography.

Encryption/Decryption

$$C = f\,(K_{public}\,,\, P) \qquad P = g(K_{private}\,,\, C)$$

10.1.3 Need for Both

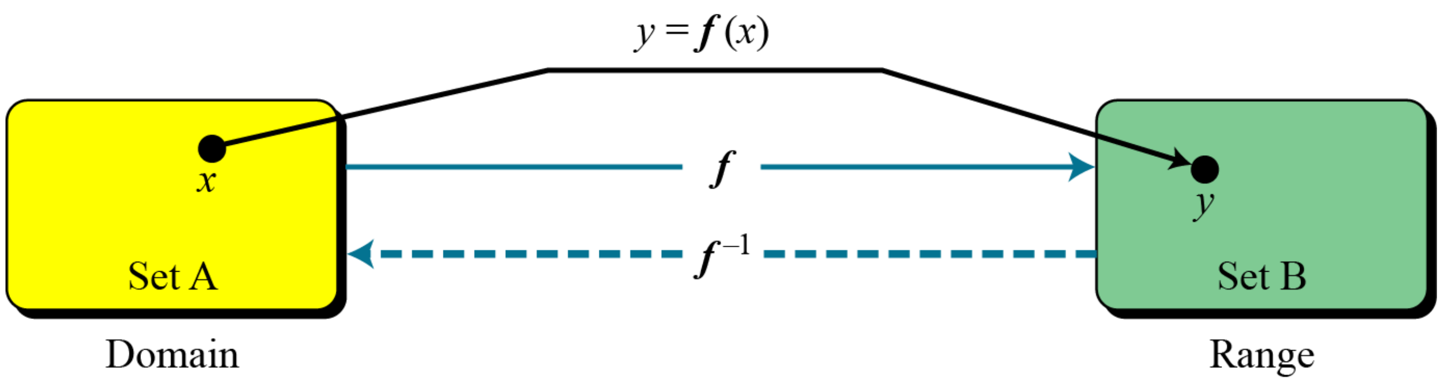
There is a very important fact that is sometimes misunderstood: The advent of asymmetric-key cryptography does not eliminate the need for symmetric-key cryptography.

10.1.4 Trapdoor One-Way Function

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

Functions

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



10.1.4 Continued

One-Way Function (OWF)

1. *f* is easy to compute.
2. f^{-1} is difficult to compute.

Trapdoor One-Way Function (TOWF)

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

10.1.4 Continued

Example 10. 1

When n is large, $n = p \times q$ is a one-way function. 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.

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. Given y, k, and n, it is very difficult to calculate x. This is the discrete logarithm problem. 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.

10.1.5 Knapsack Cryptosystem

Definition

$a = [a_1, a_2, \ldots, a_k]$ and $x = [x_1, x_2, \ldots, x_k]$.

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

Given a and x, it is easy to calculate s. However, given s and a it is difficult to find x.

Superincreasing Tuple

$$a_i \geq a_1 + a_2 + \ldots + a_{i-1}$$

10.1.5 Continued

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

knapsackSum (x [1 … k], a [1 … k]) { $s \leftarrow 0$ for (i = 1 to k) { $s \leftarrow s + a_i \times x_i$ } return s }	**inv_knapsackSum** (s, a [1 … k]) { for (i = k down to 1) { if $s \geq a_i$ { $x_i \leftarrow 1$ $s \leftarrow s - a_i$ } else $x_i \leftarrow 0$ } return x [1 … k] }

10.1.5 Continued

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

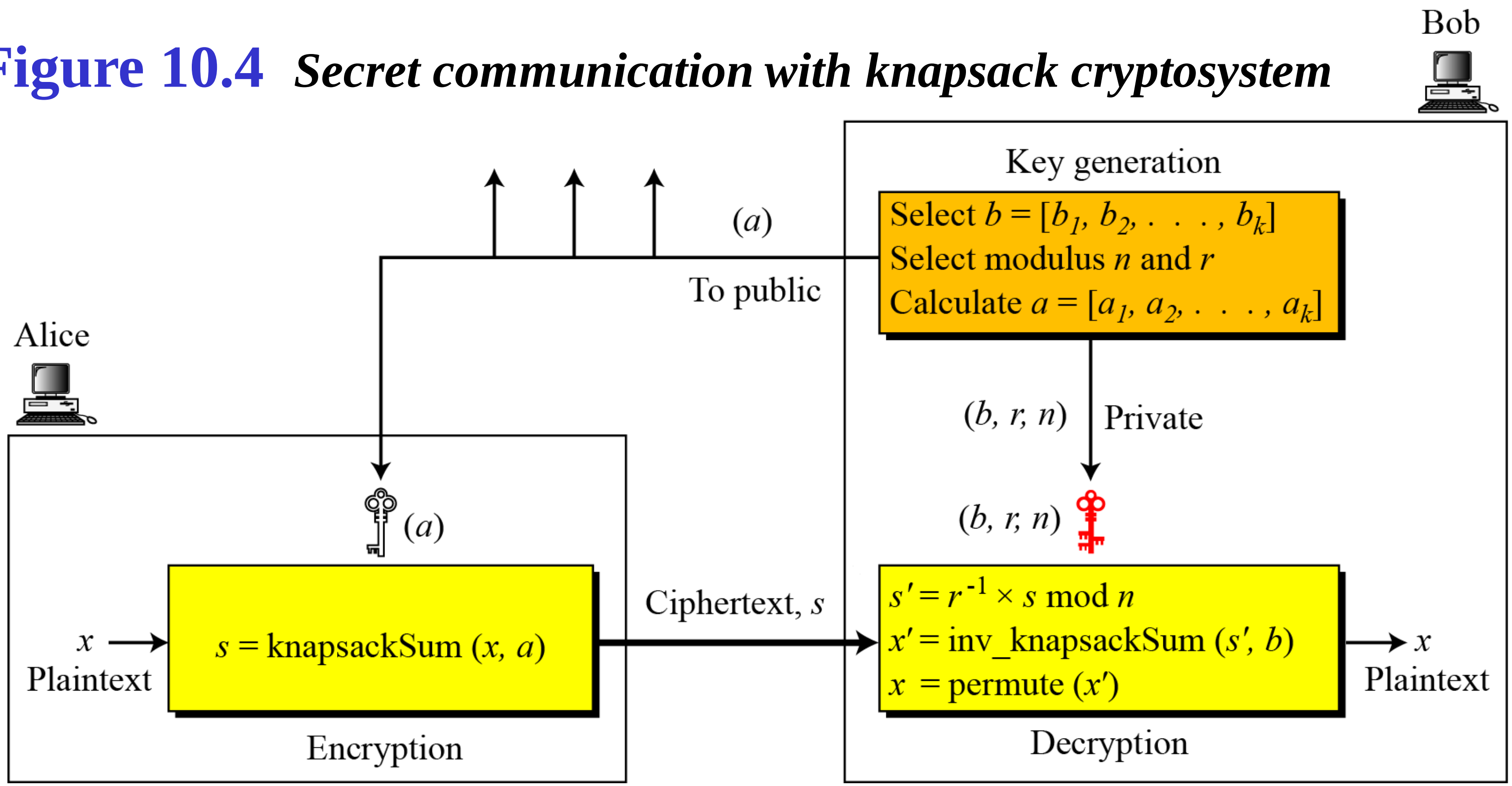
Table 10.1 *Values of i, a_i, s, and x_i in Example 10.3*

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

10.1.5 Continued

Secret Communication with Knapsacks.

Figure 10.4 *Secret communication with knapsack cryptosystem*



10.1.5 Continued

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".

10-2 RSA CRYPTOSYSTEM

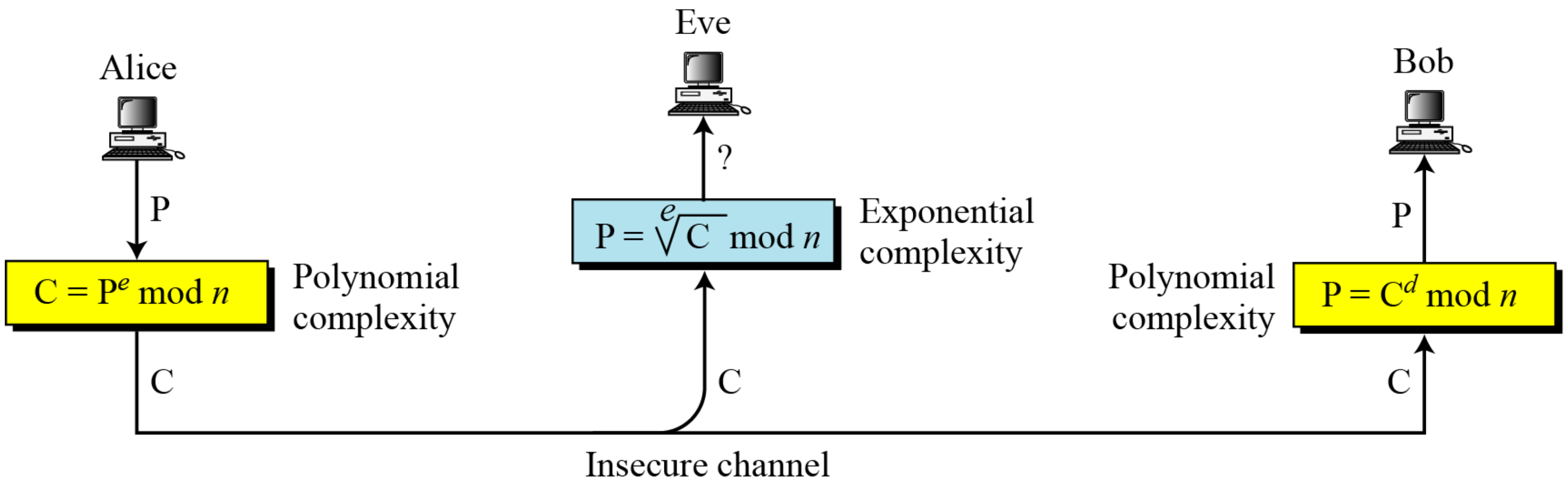
The most common public-key algorithm is the RSA cryptosystem, named for its inventors (Rivest, Shamir, and Adleman).

Topics discussed in this section:

10.2.1 Introduction
10.2.2 Procedure
10.2.3 Some Trivial Examples
10.2.4 Attacks on RSA
10.2.5 Recommendations
10.2.6 Optimal Asymmetric Encryption Padding (OAEP)
10.2.7 Applications

10.2.1 Introduction

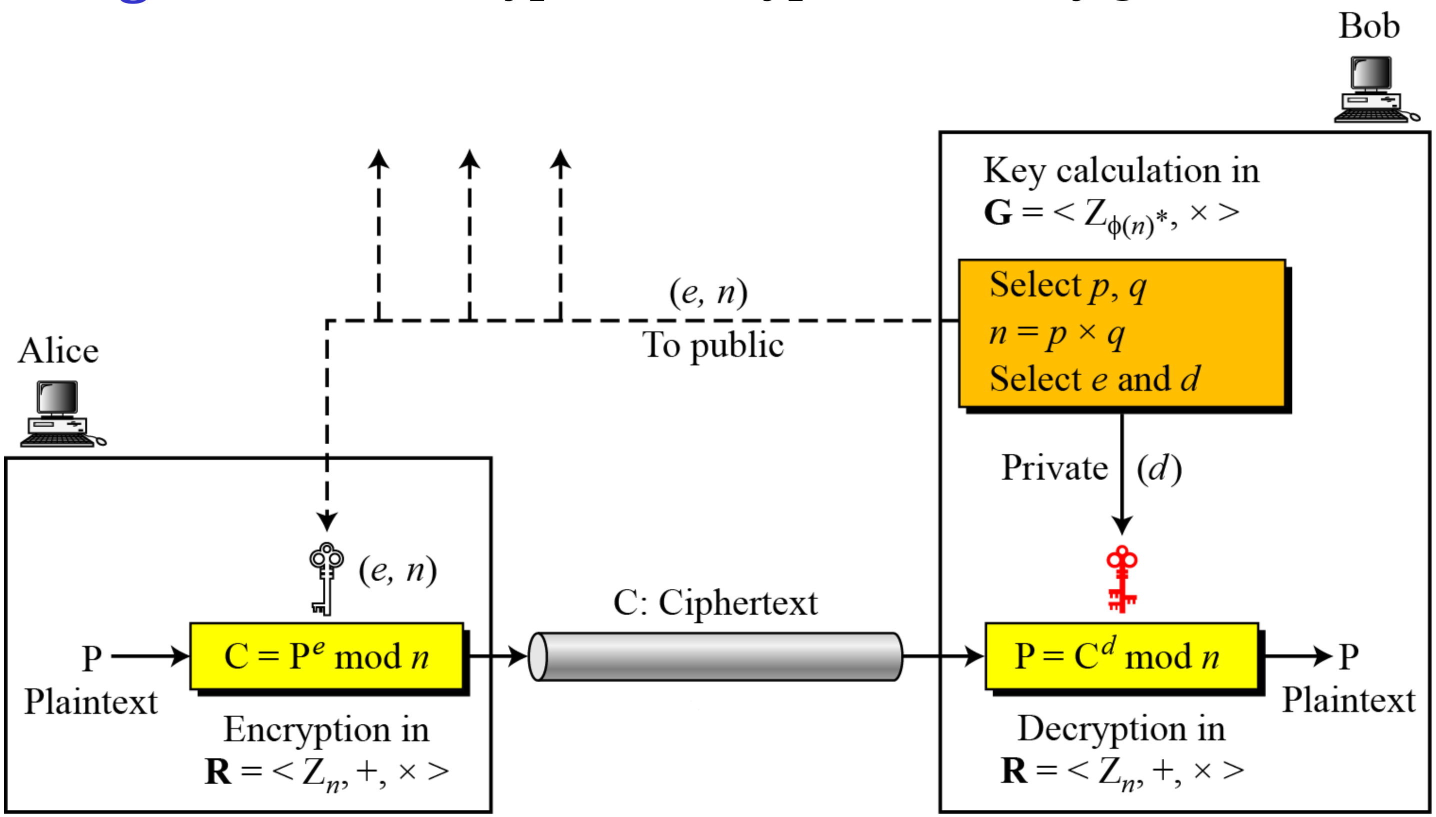
Figure 10.5 ***Complexity of operations in RSA***



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

10.2.2 Procedure

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



10.2.2 Continued

Two Algebraic Structures

Encryption/Decryption Ring: $R = <Z_n, +, \times>$

Key-Generation Group: $G = <Z_{\phi(n)}*, \times>$

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

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

10.2.2 Continued

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
}
```

10.2.2 Continued

Encryption

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
}
```

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

10.2.2 Continued

Decryption

Algorithm 10.4 *RSA decryption*

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

Proof of RSA

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

$P_1 = C^d \bmod n = (P^e \bmod n)^d \bmod n = P^{ed} \bmod n$

$ed = k\phi(n) + 1$ // 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)

10.2.3 Some Trivial Examples

Example 10. 5

Bob chooses 7 and 11 as *p* and *q* and calculates *n* = 77. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60. Now he chooses two exponents, *e* and *d*, from $Z_{60}*$. If he chooses *e* to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$ (they are inverses of each 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

10.2.3 Some Trivial Examples

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

10.2.3 Some Trivial Examples

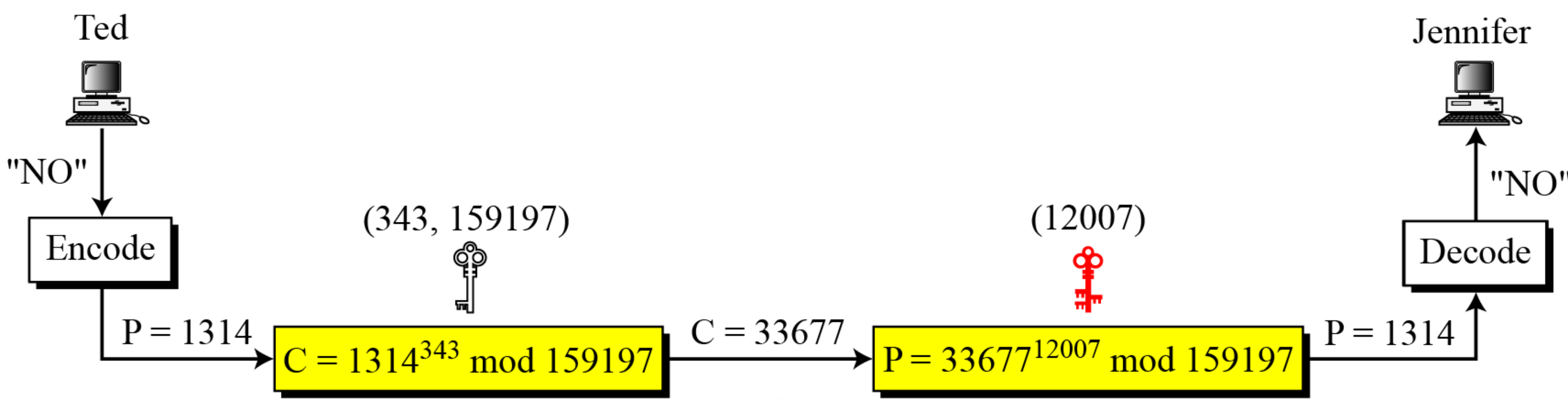
Example 10. 7

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

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. Figure 10.7 shows the process.

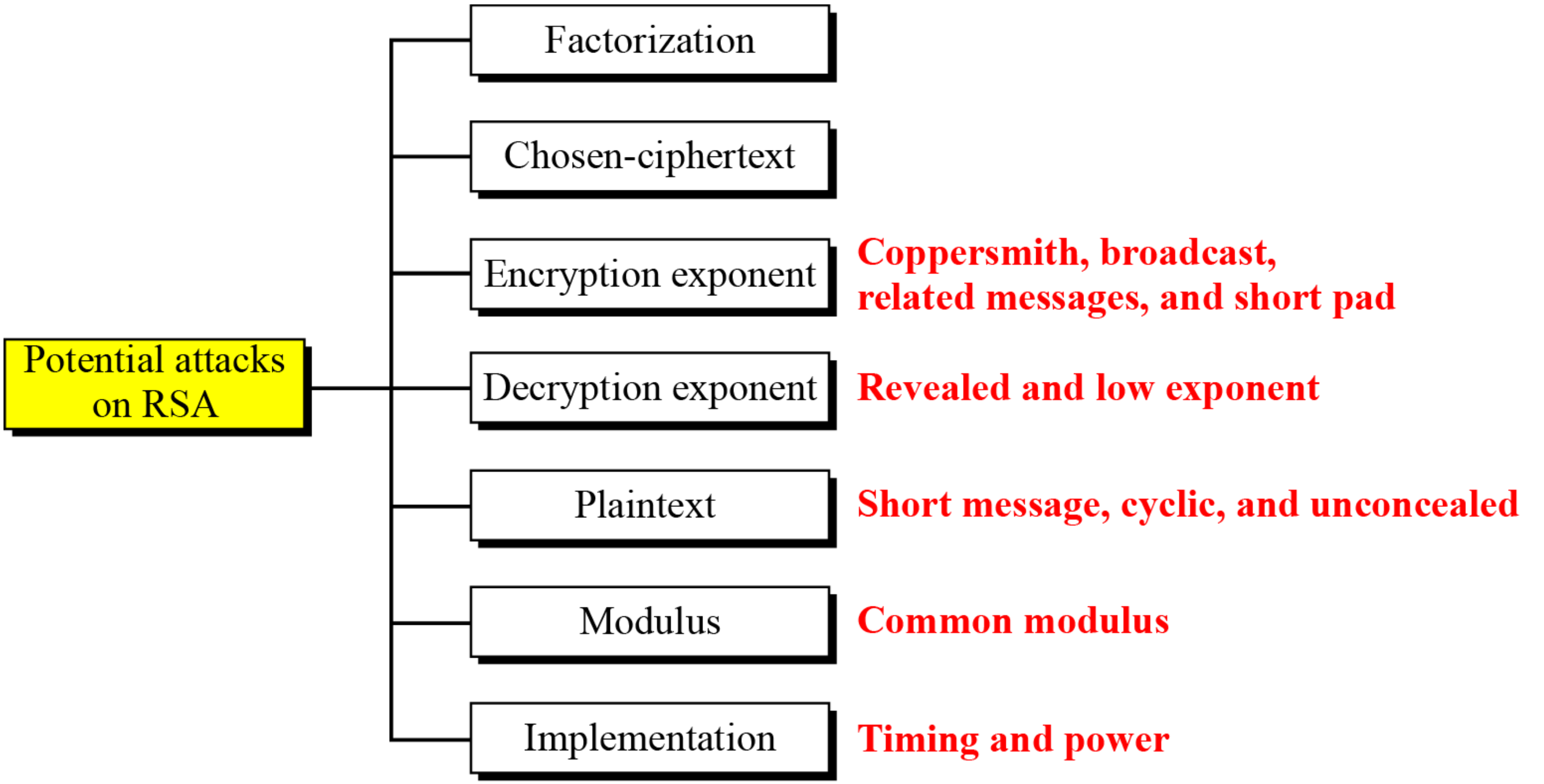
10.2.3 Continued

Figure 10.7 ***Encryption and decryption in Example 10.7***



10.2.4 Attacks on RSA

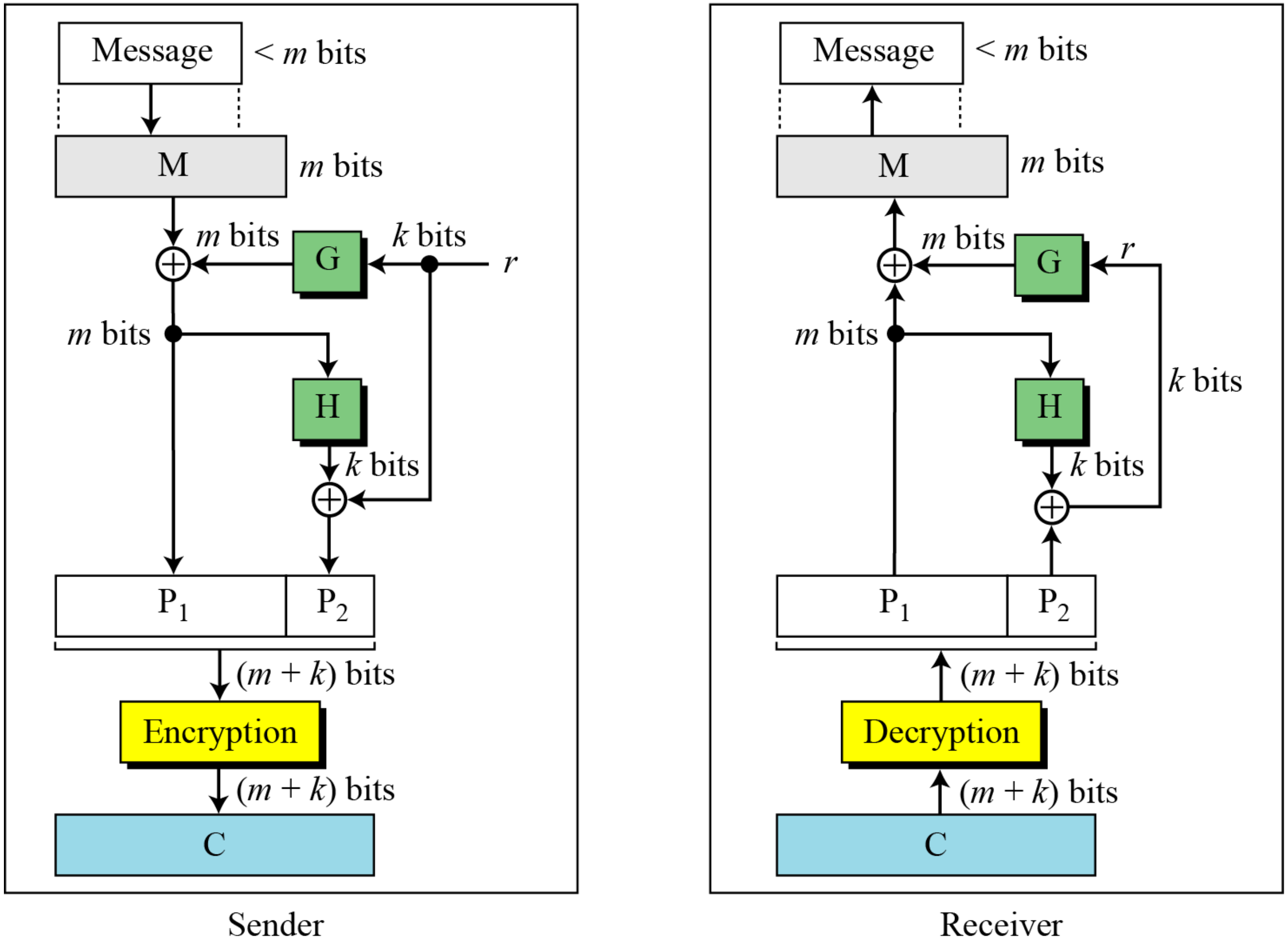
Figure 10.8 ***Taxonomy of potential attacks on RSA***



10.2.6 OAEP

Figure 10.9 ***Optimal asymmetric encryption padding (OAEP)***

M: Padded message
r: One-time random number
P: Plaintext ($P_1 \| P_2$)
C: Ciphertext
G: Public function (k-bit to m-bit)
H: Public function (m-bit to k-bit)



10.2.6 Continued

Example 10. 8

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

$p =$	96130345313583504574191581280615427909309845594996215822583150879647940455056470638491257160180347503120986666064924201918087806674210960633542199266612 09

$q =$	12060191957231446918276794204450896001555925054637033936061798321731482148483764659215389453209175225273226830107120695604602513887145524969000359660045617

10.2.6 Continued

Example 10. 8 *Continued*

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

Example 10. 8 *Continued*

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

e =	35535
d =	580083028600377639360936612896779175946690620896509621804228661113 805938528223587317062869100300217108590443384021707298690876006115 306202524959884448047568240966247081485817130463240644077704833134 010850947385295645071936774061197326557424237217617674620776371642 0760033708533328853214470885955136670294831

Example 10. 8 *Continued*

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 =	1907081826081826002619041819

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

C =	4753091236462268272063655506105451809423717960704917165232392430544529606131993285666178434183591141511974112520056829797945717360361012782188478927415660904800235071907152771859149751884658886321011483541033616578984679683867637337657774656250792805211481418440481418443081277305900469287424855916646210865 6

10.2.6 *Continued*

Example 10. 8 *Continued*

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

P =	1907081826081826002619041819

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

10-3 RABIN CRYPTOSYSTEM

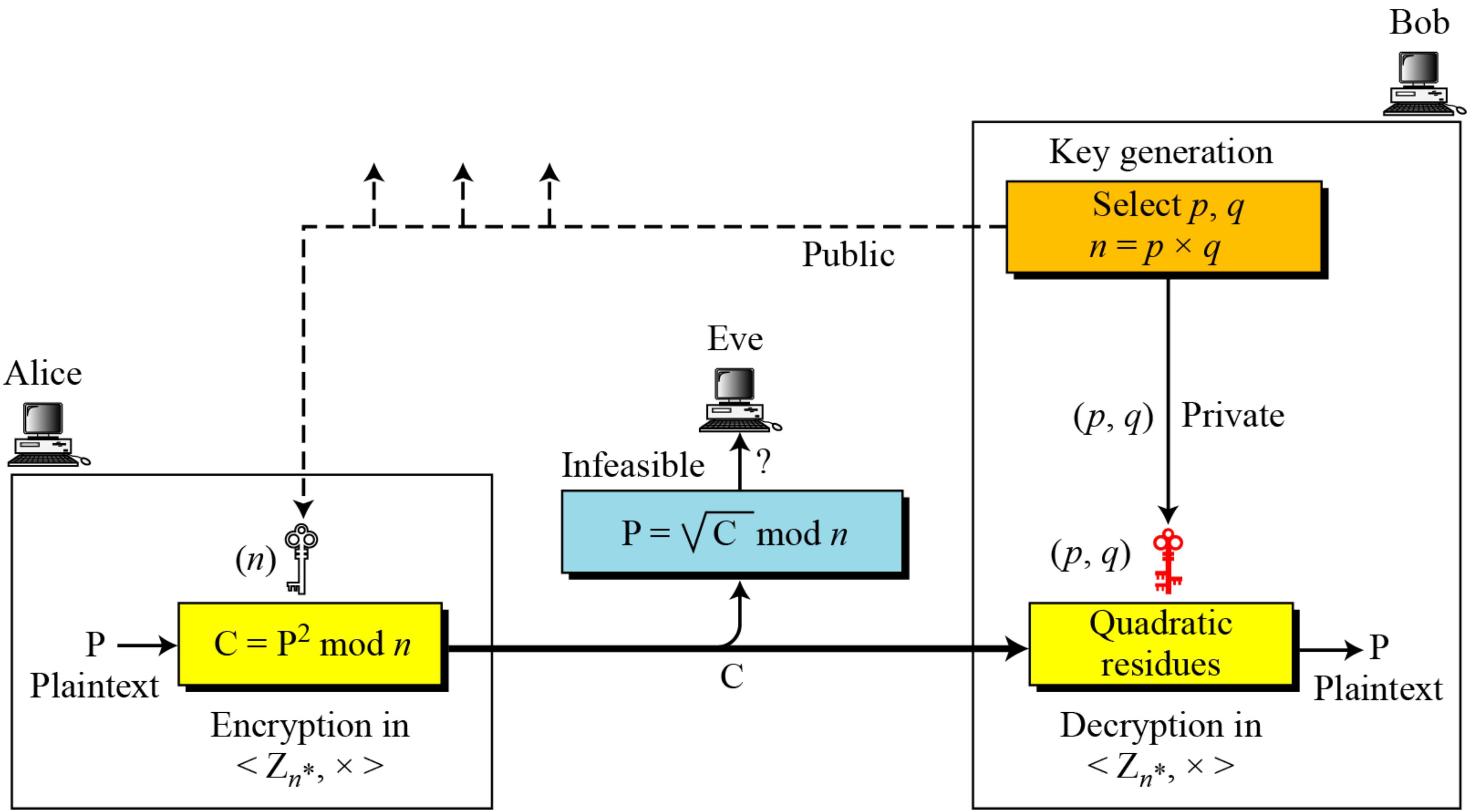
The Rabin cryptosystem can be thought of as an RSA cryptosystem in which the value of e and d are fixed. The encryption is $C \equiv P^2 \pmod{n}$ and the decryption is $P \equiv C^{1/2} \pmod{n}$.

Topics discussed in this section:

10.3.1 Procedure

10.3.2 Security of the Rabin System

Figure 10.10 ***Rabin cryptosystem***



10.3.1 Procedure

Key Generation

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 ← (q, n)                    // To be kept secret
    return Public_key and Private_key
}
```

10.3.1 Continued

Encryption

Algorithm 10.7 *Encryption in Rabin cryptosystem*

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

10.3.1 Continued

Decryption

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 algorithm 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
}
```

Note

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

10.3.1 *Continued*

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 $Z_{161}*$. She calculates $C = 24^2 = 93 \bmod 161$, and sends the ciphertext 93 to Bob.

10.3.1 Continued

Example 10. 9

5. Bob receives 93 and calculates four values:

   $a_1 = +(93^{(23+1)/4}) \bmod 23 = 1 \bmod 23$

   $a_2 = -(93^{(23+1)/4}) \bmod 23 = 22 \bmod 23$

   $b_1 = +(93^{(7+1)/4}) \bmod 7 = 4 \bmod 7$

   $b_2 = -(93^{(7+1)/4}) \bmod 7 = 3 \bmod 7$

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. Note that only the second answer is Alice's plaintext.

10-4 ELGAMAL CRYPTOSYSTEM

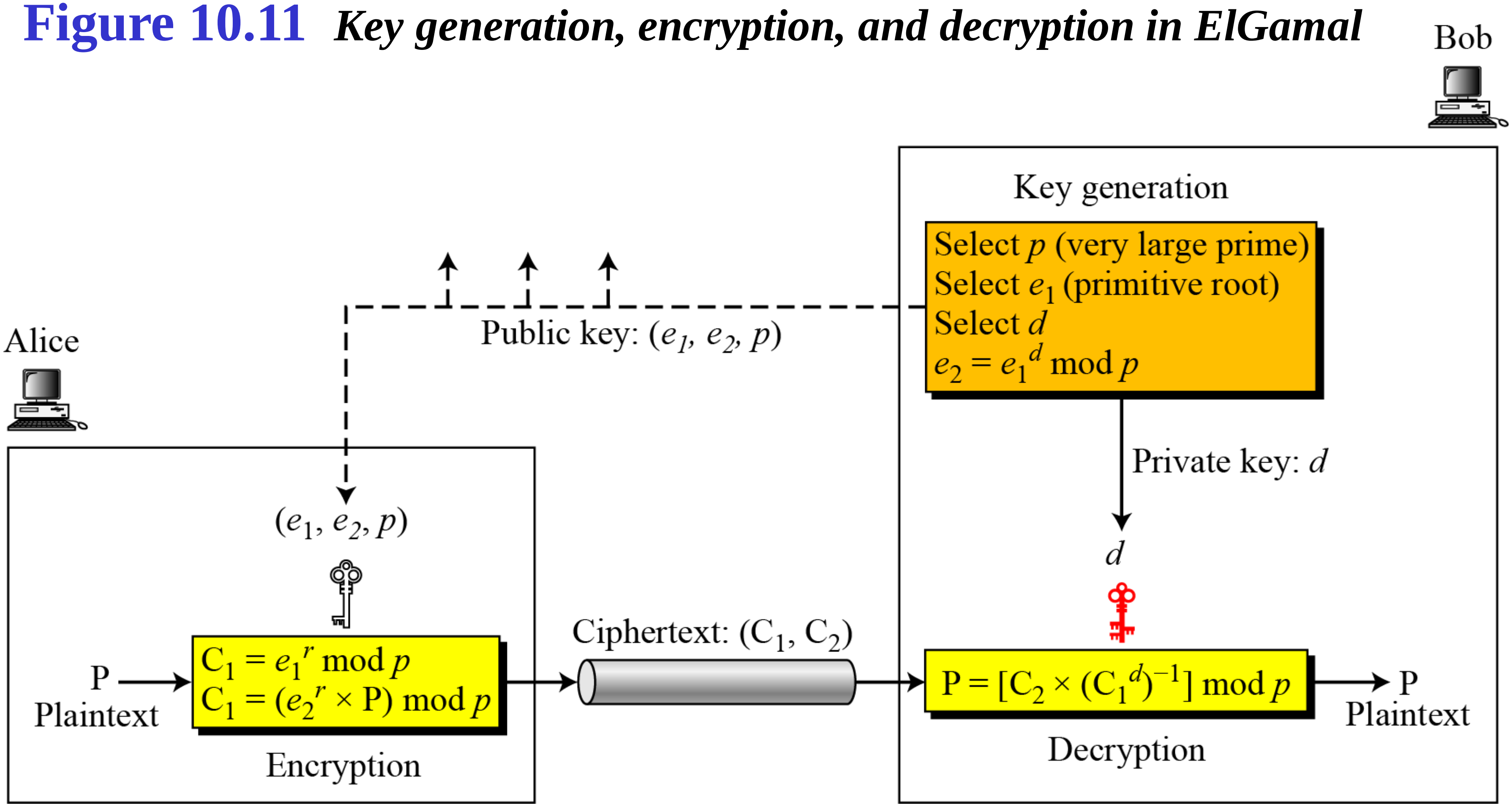
Besides RSA and Rabin, another public-key cryptosystem is ElGamal. ElGamal is based on the discrete logarithm problem discussed in Chapter 9.

Topics discussed in this section:

10.4.1 ElGamal Cryptosystem
10.4.2 Procedure
10.4.3 Proof
10.4.4 Analysis
10.4.5 Security of ElGamal
10.4.6 Application

10.4.2 Procedure

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



Key Generation

Algorithm 10.9 *ElGamal key generation*

```
ElGamal_Key_Generation
{
    Select a large prime p
    Select d to be a member of the group G = <Z_p*, ×> such that 1 ≤ d ≤ p − 2
    Select e_1 to be a primitive root in the group G = <Z_p*, ×>
    e_2 ← e_1^d mod p
    Public_key ← (e_1, e_2, p)                  // To be announced publicly
    Private_key ← d                             // To be kept secret
    return Public_key and Private_key
}
```

10.4.2 Continued

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
}
```

Where: **ElGamal_Encryption** (e_1, e_2, p, P); select a random integer r in the group $\mathbf{G} = \langle \mathbf{Z}_p^*, \times \rangle$; $\text{C}_1 \leftarrow e_1^r \bmod p$; $\text{C}_2 \leftarrow (\text{P} \times e_2^r) \bmod p$; return C_1 and C_2.

10.4.2 Continued

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
}
```

Note

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

10.4.3 Continued

Example 10. 10

Here is a trivial example. Bob chooses p = 11 and e_1 = 2. and d = 3 $e_2 = e_1^d$ = 8. So the public keys are (2, 8, 11) and the private key is 3. Alice chooses r = 4 and calculates C1 and C2 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

10.4.3 Continued

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 $P = [6 \times 5^{11-1-3}] \bmod 11 = 7 \bmod 11$.

Note

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

10.4.3 Continued

Example 10. 12

Bob uses a random integer of 512 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:

p =	115348992725616762449253137170143317404900945326098349598143469219 056898698622645932129754737871895144368891765264730936159299937280 61165964347353440008577
e_1 =	2

d =	1007
e_2 =	978864130430091895087668569380977390438800628873376876100220622332 554507074156189212318317704610141673360150884132940857248537703158 2066010072558707455

10.4.3 Continued

Example 10. 10

Alice has the plaintext P = 3200 to send to Bob. She chooses r = 545131, calculates C1 and C2, and sends them to Bob.

P =	3200
r =	545131
C_1 =	887297069383528471022570471492275663120260067256562125018188351429 417223599712681114105363661705173051581533189165400973736355080295 736788569060619152881
C_2 =	708454333048929944577016012380794999567436021836192446961774506921 244696155165800779455593080345889614402408599525919579209721628879 681350582779566430295 0

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

P =	3200

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).

Topics discussed in this section:

10.5.1 Elliptic Curves over Real Numbers
10.5.2 Elliptic Curves over GF(*p*)
10.5.3 Elliptic Curves over GF(2^n)
10.5.4 Elliptic Curve Cryptography Simulating ElGamal

10.5.1 Elliptic Curves over Real Numbers

The general equation for an elliptic curve is

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

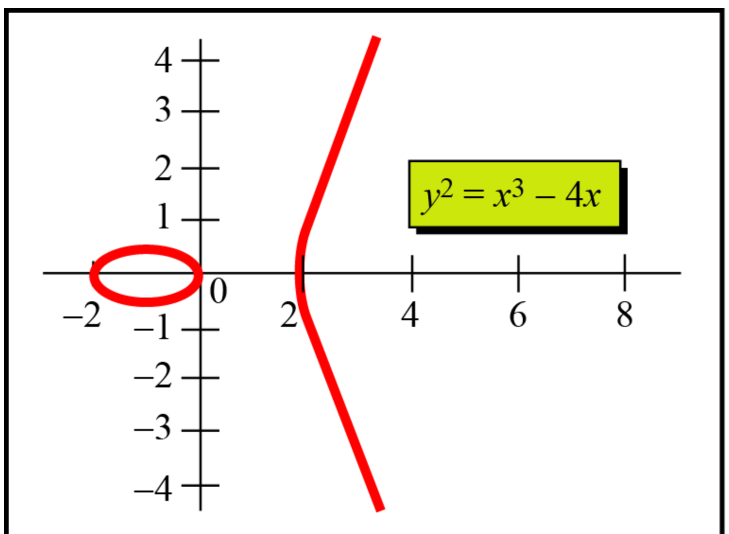
Elliptic curves over real numbers use a special class of elliptic curves of the form

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

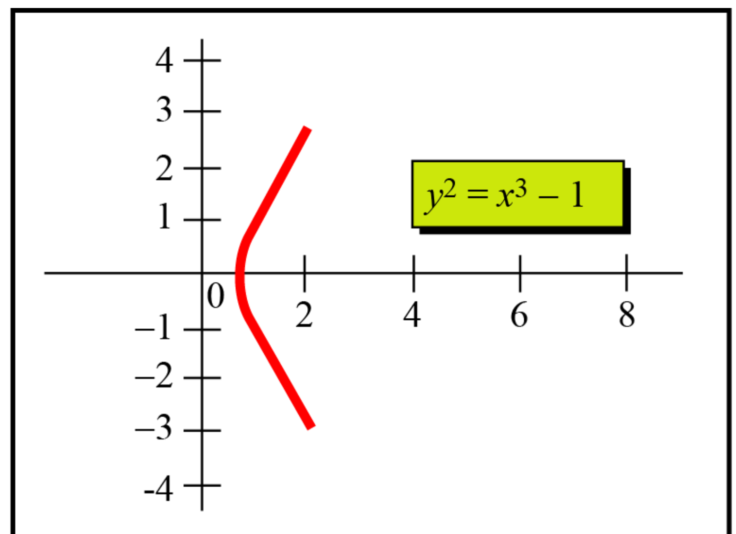
Example 10. 13

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*



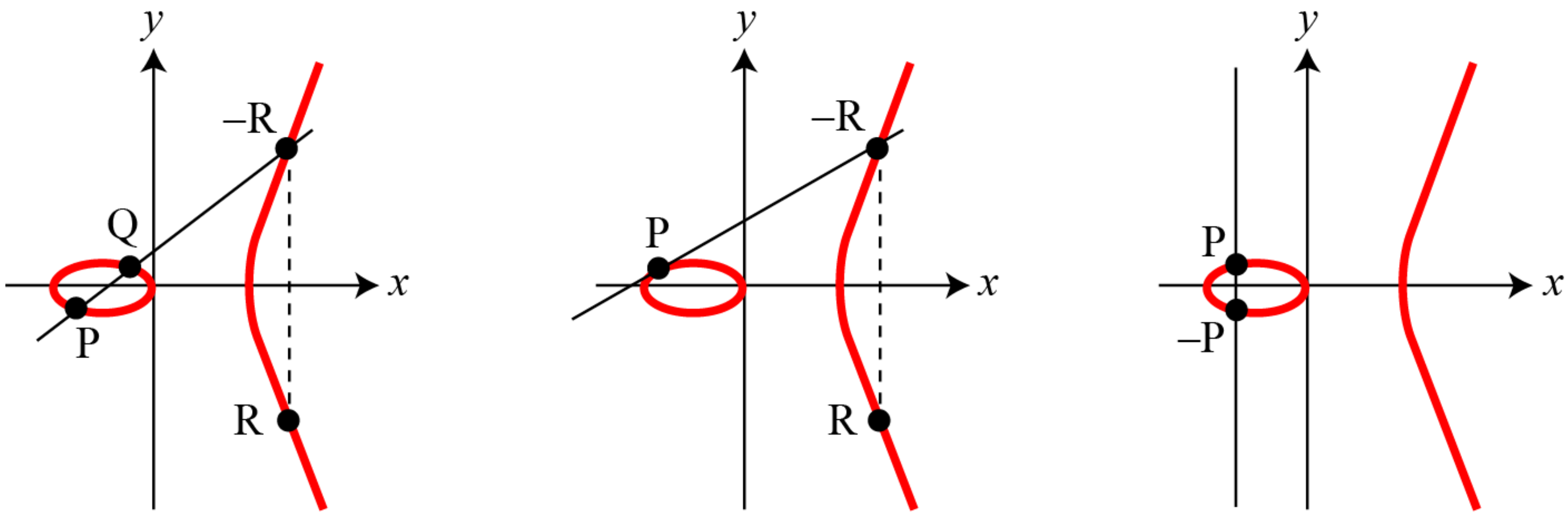
a. Three real roots



b. One real and two imaginary roots

10.5.1 Continued

Figure 10.13 ***Three adding cases in an elliptic curve***



a. (R = P + Q)   b. (R = P + P)   c. (O = P + (– P))

10.5.1 Continued

1.

$$\lambda = (y_2 - y_1) / (x_2 - x_1)$$

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

2.

$$\lambda = (3x_1{}^2 + a)/(2y_1)$$

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

3. The intercepting point is at infinity; a point O as the point at infinity or zero point, which is the additive identity of the group.

10.5.2 Elliptic Curves over GF(p)

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 Ep(a, b).

10.5.2 Continued

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

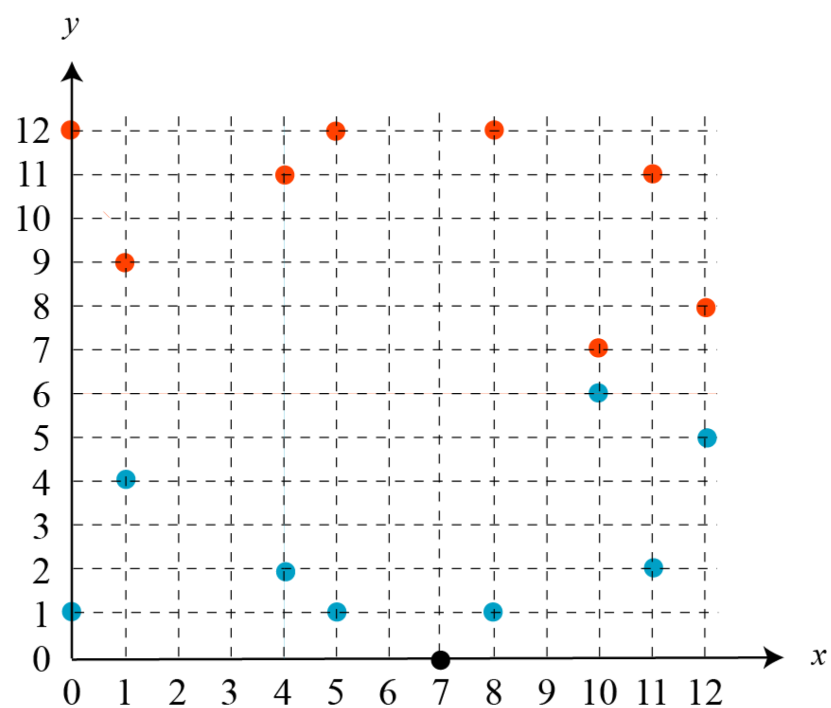
The equation is $y^2 = x^3 + x + 1$ and the calculation is done modulo 13.

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

P	–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, 18)

P = –P → (7, 0)

Points



Graph

10.5.2 Continued

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.

10.5.3 Elliptic Curves over GF(2^n)

To define an elliptic curve over GF(2^n), one needs to change the cubic equation. The common equation is

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

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 7..

10.5.3 Continued

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 7. This algorithm is left as an exercise. Following is a very trivial example.

10.5.3 Continued

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

10.5.3 Continued

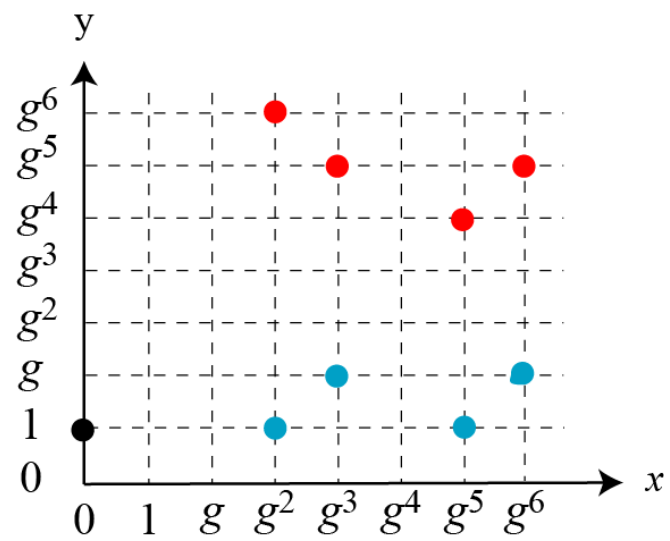
Example 10. 16 *Continued*

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(2n)***

	P	–P
P = –P →	(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

10.5.3 Continued

Adding Two Points

1. If $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $Q \neq -P$, and $Q \neq P$, then $R = (x_3, y_3) = P + Q$ can be found as

$$\lambda = (y_2 + y_1) / (x_2 + x_1)$$

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

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

$$\lambda = x_1 + y_1 / x_1$$

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

10.5.3 Continued

Example 10. 17

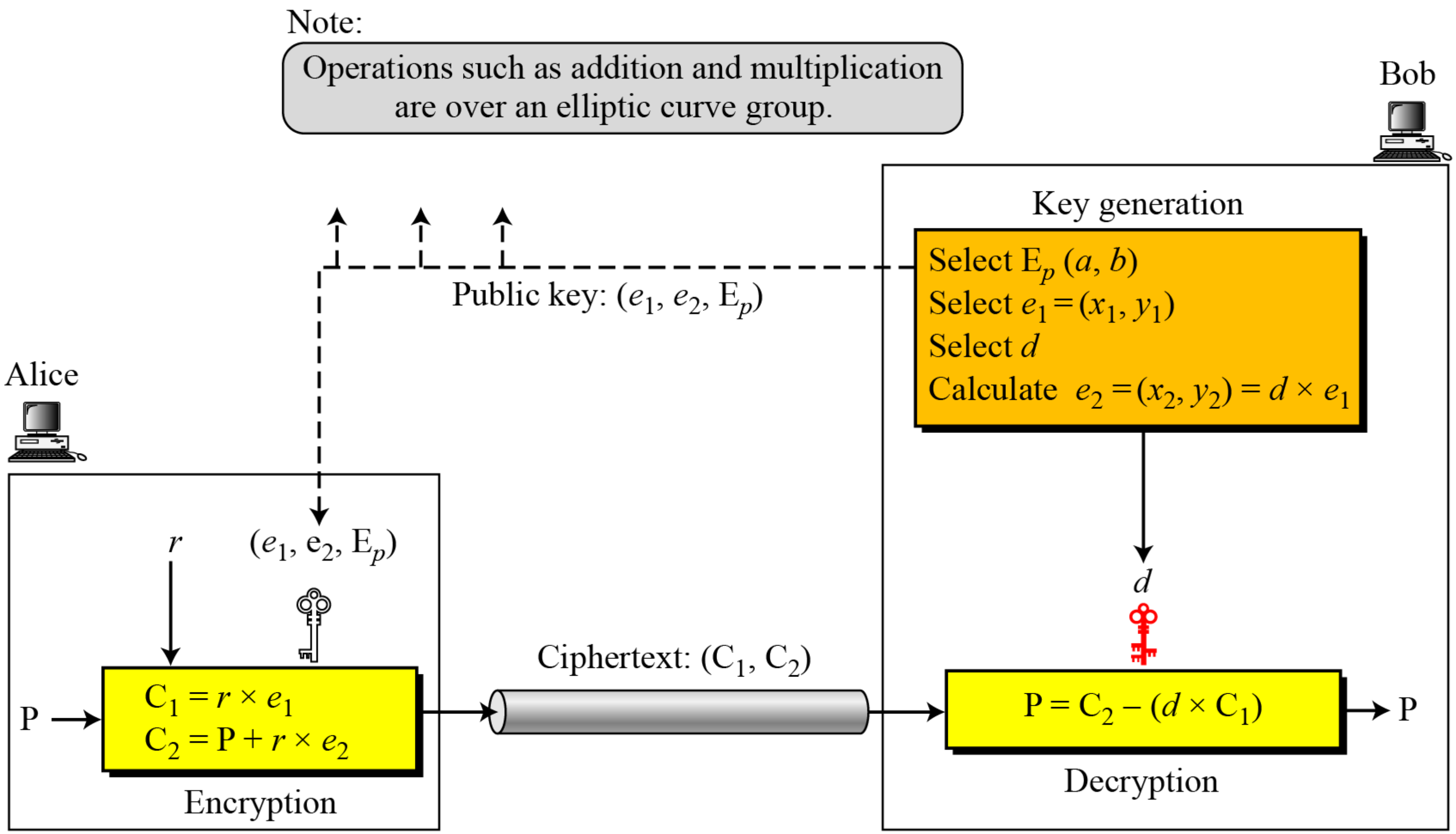
Let us find $R = P + Q$, where $P = (0, 1)$ and $Q = (g^2, 1)$. We have $\lambda = 0$ and $R = (g^5, g^4)$.

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 $R = (g^6, g^5)$.

10.5.4 ECC Simulating ElGamal

Figure 10.16 ***ElGamal cryptosystem using the elliptic curve***



10.5.4 Continued

Generating Public and Private Keys

$E(a, b)$ $e_1(x_1, y_1)$ d $e_2(x_2, y_2) = d \times e_1(x_1, y_1)$

Encryption

$C_1 = r \times e_1$ $C_2 = P + r \times e_2$

Decryption

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

Note

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

10.5.4 Continued

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 $r = 2$.***