Asymmetric-Key Cryptography

Objectives

- To distinguish between two cryptosystems: symmetric-key and asymmetric-key
- 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 ElGamal cryptosystem

INTRODUCTION

- 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.

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

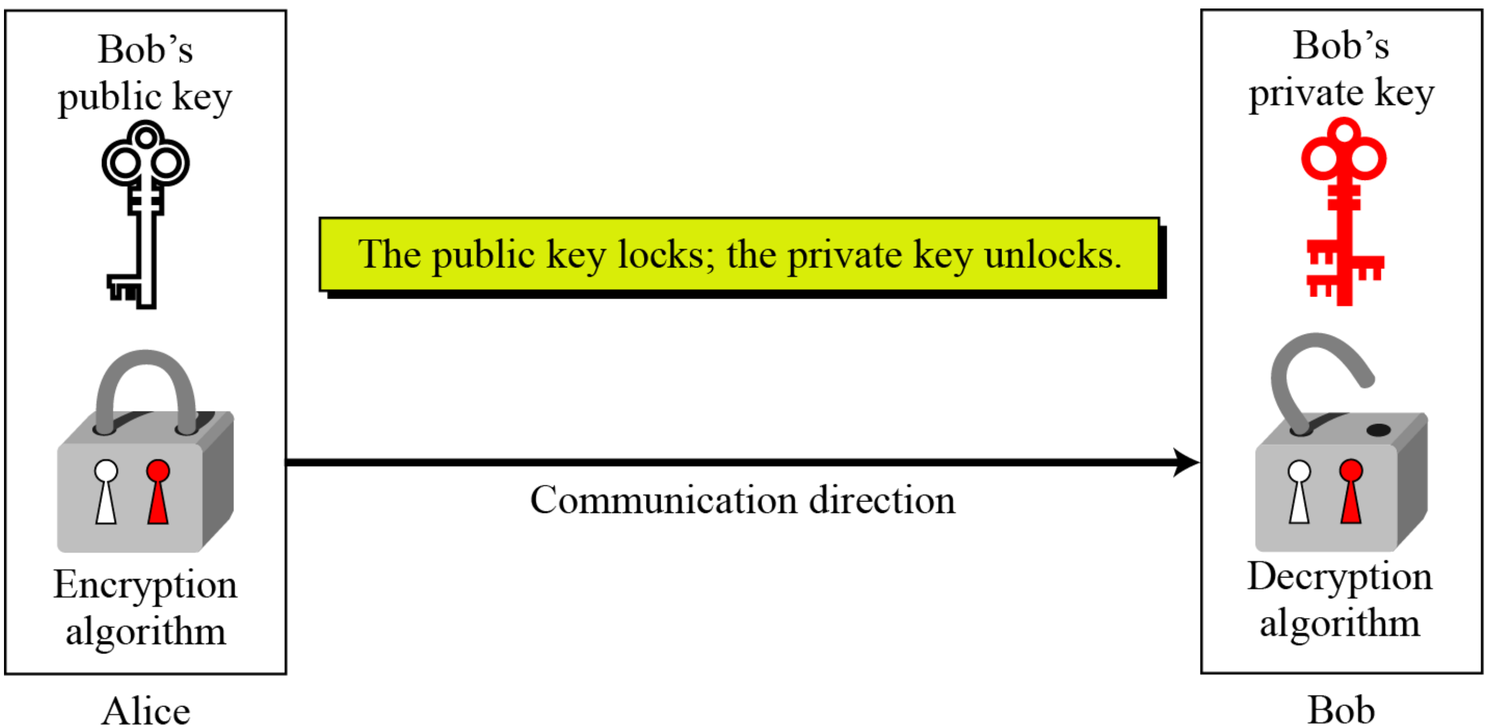
Difference Between Symmetric And Asymmetric Key Cryptography

- Symmetric is based on substitution and permutation of symbols whereas asymmetric is based on applying mathematical functions to numbers.
- In symmetric, plaintext and cipher text are thought of as a combination of symbols whereas in asymmetric plain text and cipher text are numbers.

Keys

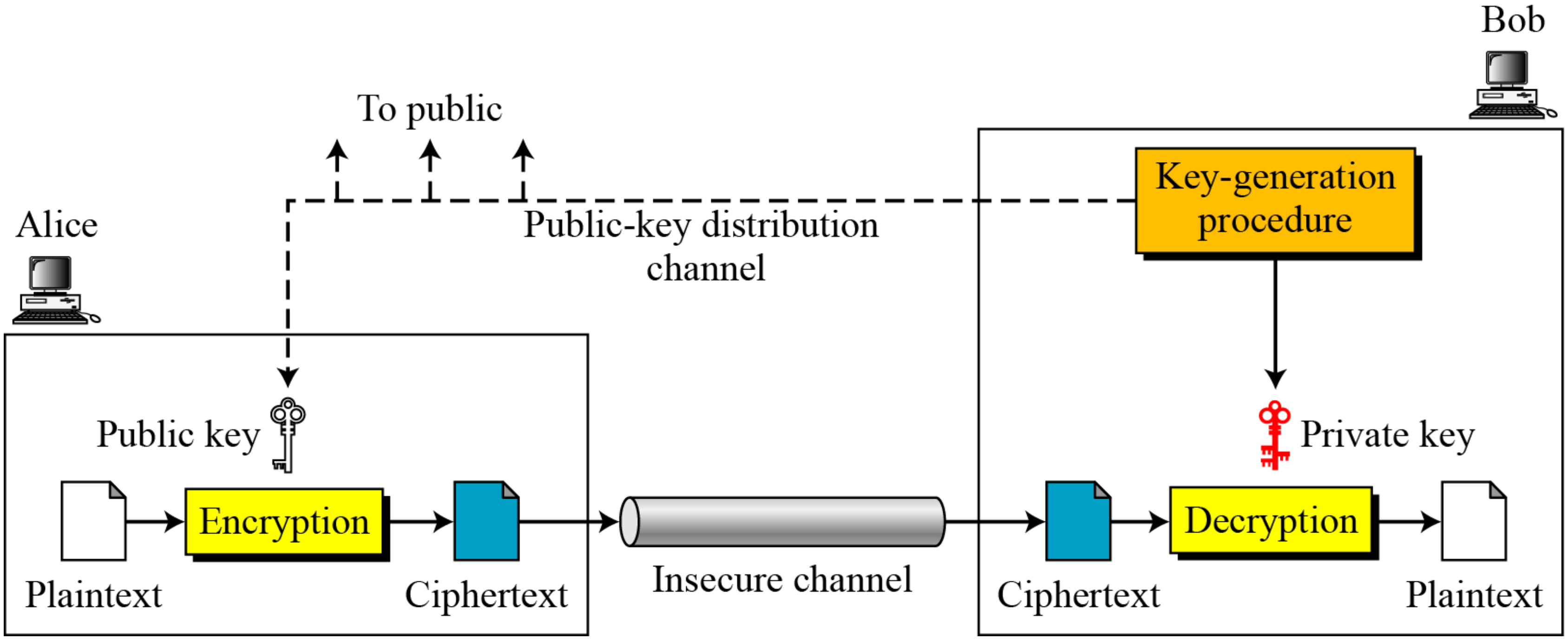
Asymmetric key cryptography uses two separate keys: one private and one public.

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



General Idea

Figure ***General idea of asymmetric-key cryptosystem***



Continued

Plaintext/Ciphertext

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

Encryption/Decryption

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

Need for Both

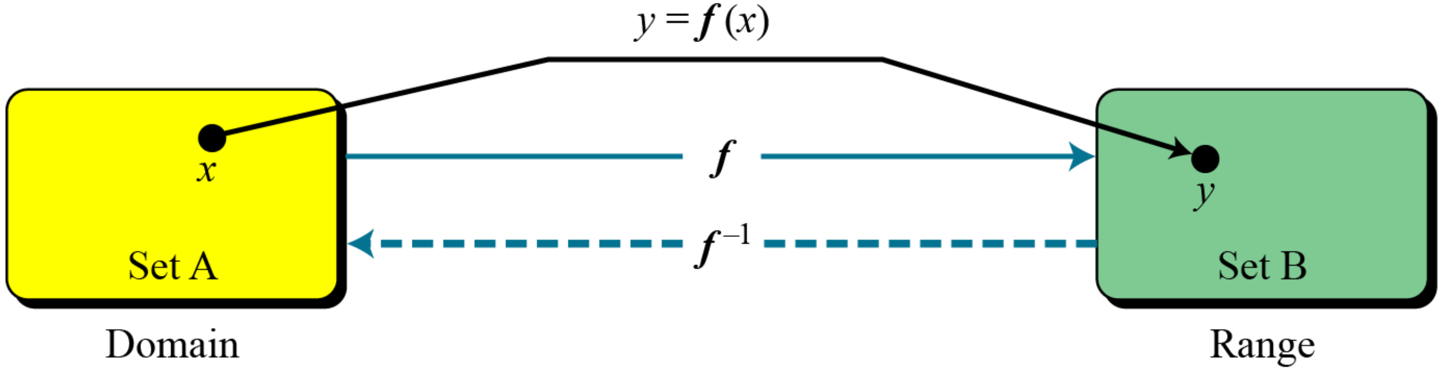
There is a very important fact that is sometimes misunderstood: The advent of asymmetric-key cryptography does not eliminate the need for symmetric-key cryptography.

Trapdoor One-Way Function

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

Functions

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



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.

Continued

Example

When n is large, $\boldsymbol{n = p \times q}$ 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

When n is large, the function $\boldsymbol{y = x^k \bmod n}$ 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 $\boldsymbol{k \times k' = 1 \bmod \phi(n)}$, we can use $\mathbf{x = y^{k'} \bmod n}$ to find x.

Knapsack Cryptosystem

Definition

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

$$\boxed{s = knapsackSum\,(a, x) = x_1 a_1 + x_2 a_2 + \cdots + x_k a_k}$$

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

Superincreasing Tuple

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

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] }

e.g. : Assume that a = [17, 25, 46, 94, 201,400] and s = 272 are given.

Continued

Example

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

Secret Communication with Knapsacks.

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 $1 \leq r \leq n - 1$.

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.

Continued

Secret Communication with Knapsacks.

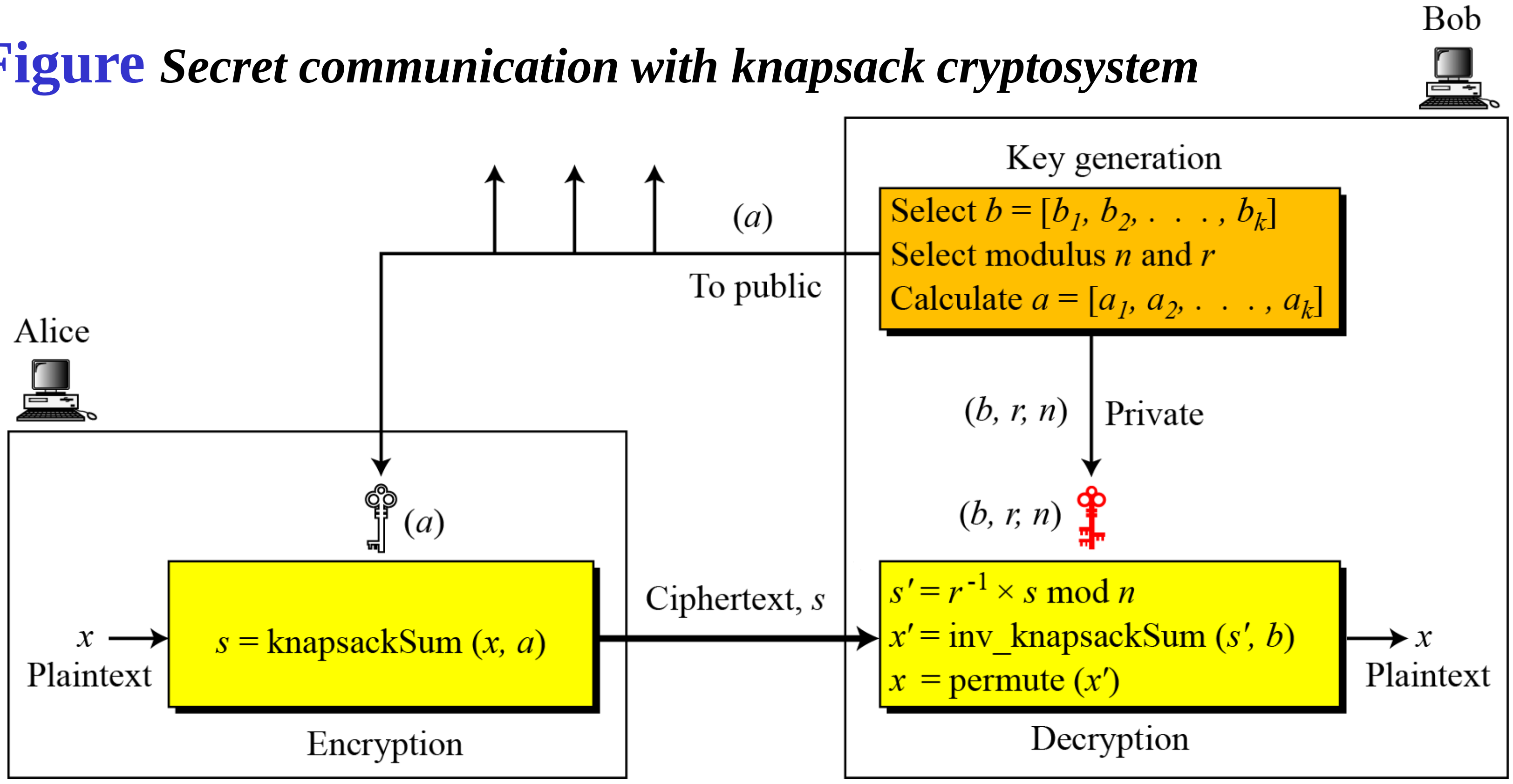


Figure ***Secret communication with knapsack cryptosystem***

Example 10.4

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

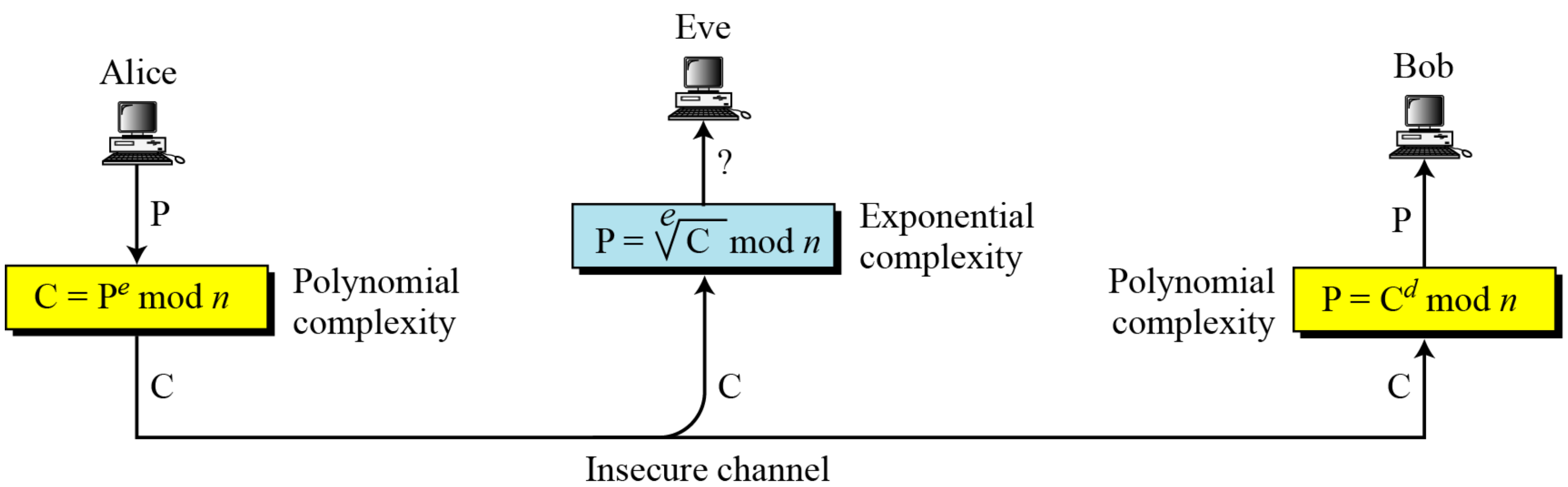
1. Key generation:
   a. Bob creates the superincreasing tuple $b = [7, 11, 19, 39, 79, 157, 313]$.
   b. Bob chooses the modulus $n = 900$ and $r = 37$, and [4 2 5 3 1 7 6] as permutation table.
   c. Bob now calculates the tuple $t = [259, 407, 703, 543, 223, 409, 781]$.
   d. Bob calculates the tuple 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".

RSA CRYPTOSYSTEM

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

Introduction

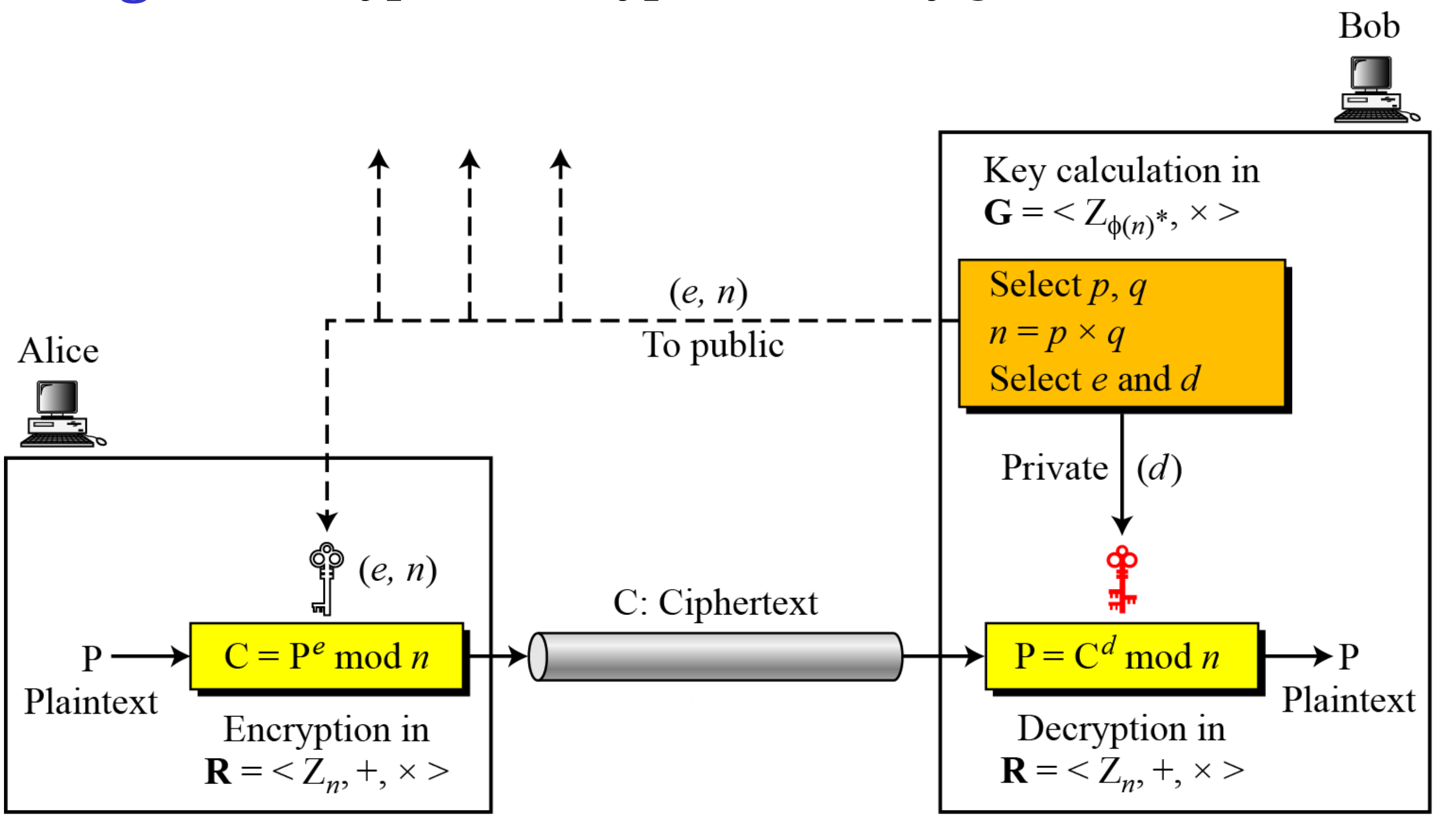
Figure ***Complexity of operations in RSA***



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

Procedure

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



Continued

Two Algebraic Structures

Encryption/Decryption Ring: $R = \langle Z_n , +, \times \rangle$

Key-Generation Group: $G = \langle Z_{\phi(n)}^*, \times \rangle$

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

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

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
}
```

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.

Continued

Decryption

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
}
```

Example : Encrypt P using RSA algorithm if p=7, q=11, e=13 and P=5.

Some Trivial Examples

Example

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

Some Trivial Examples

Example

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

Some Trivial Examples

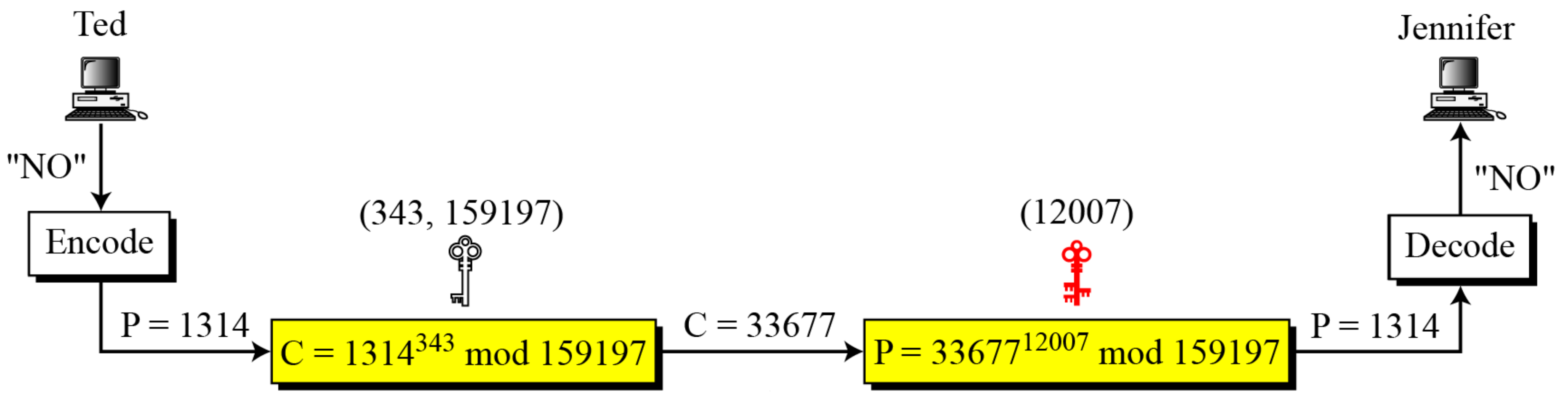
Example

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.

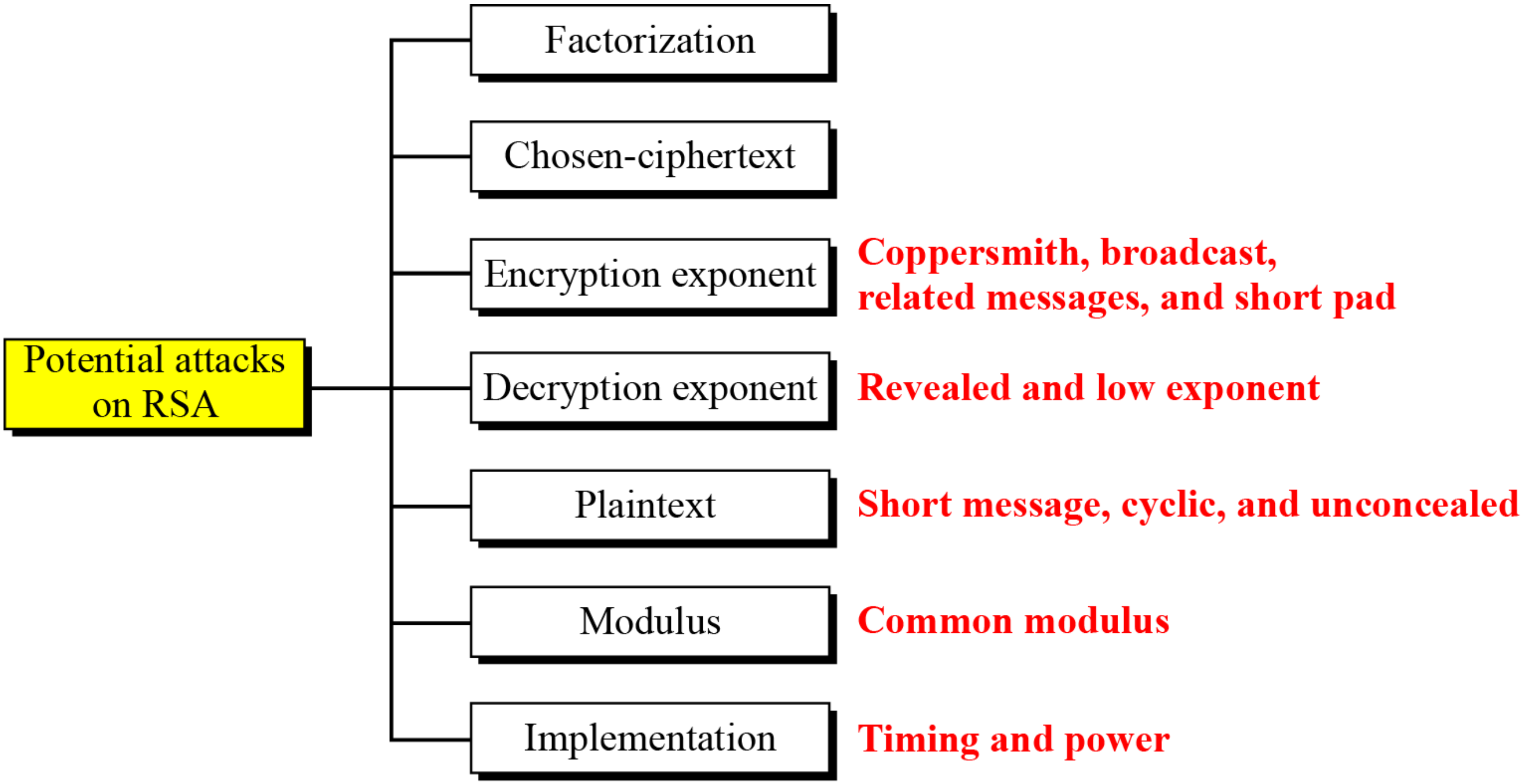
Continued

Figure ***Encryption and decryption***



Attacks on RSA

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



Factorization Attack

1. Eve can factor n and obtain p and q and once p and q is obtained then nothing left.
2. To be secure, RSA requires that n should be more than 300 decimal digits , which means that modulus must be at least 1024 bits.

Chosen cipher text Attack

Attacker intercepts C and uses 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

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

$$Z = (P \times X) \bmod n \quad \rightarrow \quad P = Z \times X^{-1} \bmod n$$

Attacks on Encryption Exponent (e)

Recommendation is to use $e = 2^{16} + 1$ i.e. 65537 (or a prime close to this value)

1. Coppersmith theorem attack
2. Broadcast attack:

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

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

3. Related Message Attack
4. Short Pad Attack

Attacks on Decryption Exponent (d)

1. Revealed Decryption exponent attack : If d is compromised, then p, q, n, e and d must be regenerated
2. Low decryption exponent attack: recommendation is to have d › =1/3 $n^{1/4}$

Plaintext attacks

1. Short message attack: Strongly recommended that messages be padded with random bits before encryption using OAEP.
2. Cycling Attack:

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 $P = C_{k-1}$

3. Unconcealed message attack

Attacks on Modulus (n)

1. Common modulus attack: to prevent this type of attack, the modulus must not be shared. Each entity needs to calculate her or his own modulus.

Attacks on Implementation

1. Timing Attack:

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 $P = P_1 \times r^{-1} \bmod n$.

2. Power Attack: Same techniques used to prevent timing attack can be used to prevent power attacks

ELGAMAL CRYPTOSYSTEM

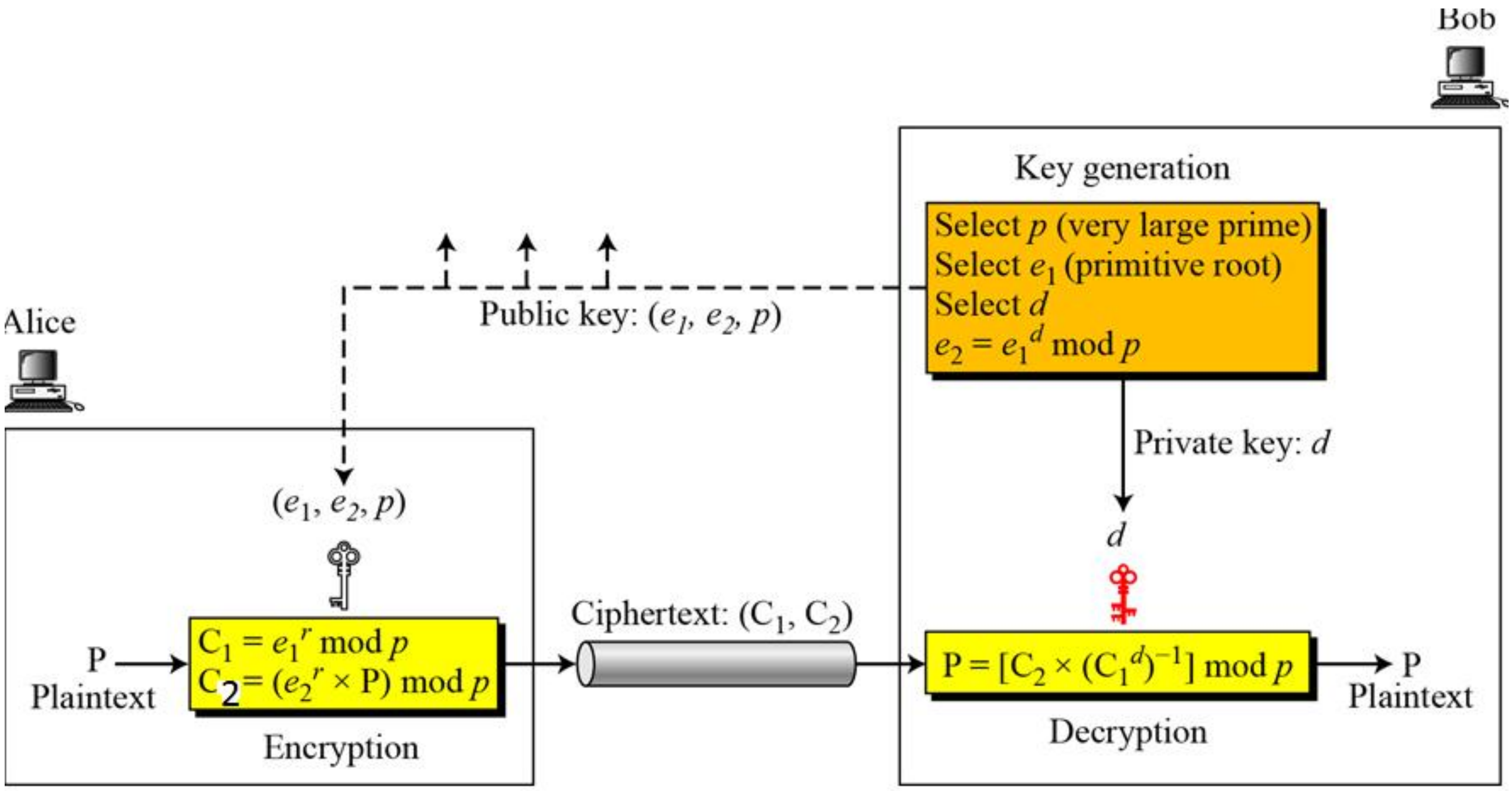
Besides Knapsack, RSA another public-key cryptosystem is ElGamal. ElGamal is based on the discrete logarithm problem.

ElGamal Cryptosystem

If **p** is a very large prime, **e1** is a primitive root in the group **G=‹Zp*,X›** and **r** is an integer, then **$e2=e1^r$ mod p** is easy to compute using Fast Exponential algorithm(square and multiply method)
But.... **Given e2, e1 and p**, it is infeasible to calculate **r i.e. r = loge1e2 mod p** (discrete logarithm problem)

Procedure

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



Continued

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 = <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
}
```

Continued

Algorithm 10.10 *ElGamal encryption*

ElGamal_Encryption (e_1, e_2, p, P) // P is the plaintext

{

Select a random integer r in the group $\mathbf{G} = < \mathbf{Z}_p{}^*, \times >$

$C_1 \leftarrow e_1{}^r \bmod p$

$C_2 \leftarrow (P \times e_2{}^r) \bmod p$ // C_1 and C_2 are the ciphertexts

return C_1 and C_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
}
```

$$P \leftarrow [C_2 (C_1^{d})^{-1}] \bmod p$$

Note

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:

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

Continued

Example

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

Continued

Example

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

Analysis of ElGamal

ElGamal cryptosystem is a puzzle. It can be solved as follows:

1. $C_1 = e_1^r \bmod p$
   $C_2 = (e_2^r \times P) \bmod p$

2. $P = [C_2 \times (C_1^d)^{-1}] \bmod p$

3. $e_2 = e_1^d \bmod p$

Security of ElGamal

Two attacks have been mentioned for this cryptosystem:

1. **Low-Modulus Attack**:

If p is not large enough, attacker can use efficient algorithms to solve discrete logarithm problem to find d or r.
Recommended that p be at least 1024 bits (300 decimal digits)

2. **Known-Plaintext Attack:**

It is recommended that sender use a fresh value of r to prevent this type of attack.

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

Application

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