# Asymmetric-Key Cryptography

Dr. B C Dhara

Department of Information Technology

Jadavpur University

# Objectives

- Distinguish between two cryptosystems: symmetric-key and asymmetric-key
- Introduce trapdoor one-way functions and their use in asymmetric-key cryptosystems
- Introduce the knapsack cryptosystem as one of the first ideas in asymmetric-key cryptography
- Discuss on
  - RSA cryptosystem, Rabin cryptosystem, ElGamal cryptosystem and Elliptic curve cryptosystem

# Introduction

- *Symmetric and asymmetric-key cryptography will exist in parallel and continue to serve the community*
- *People 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.

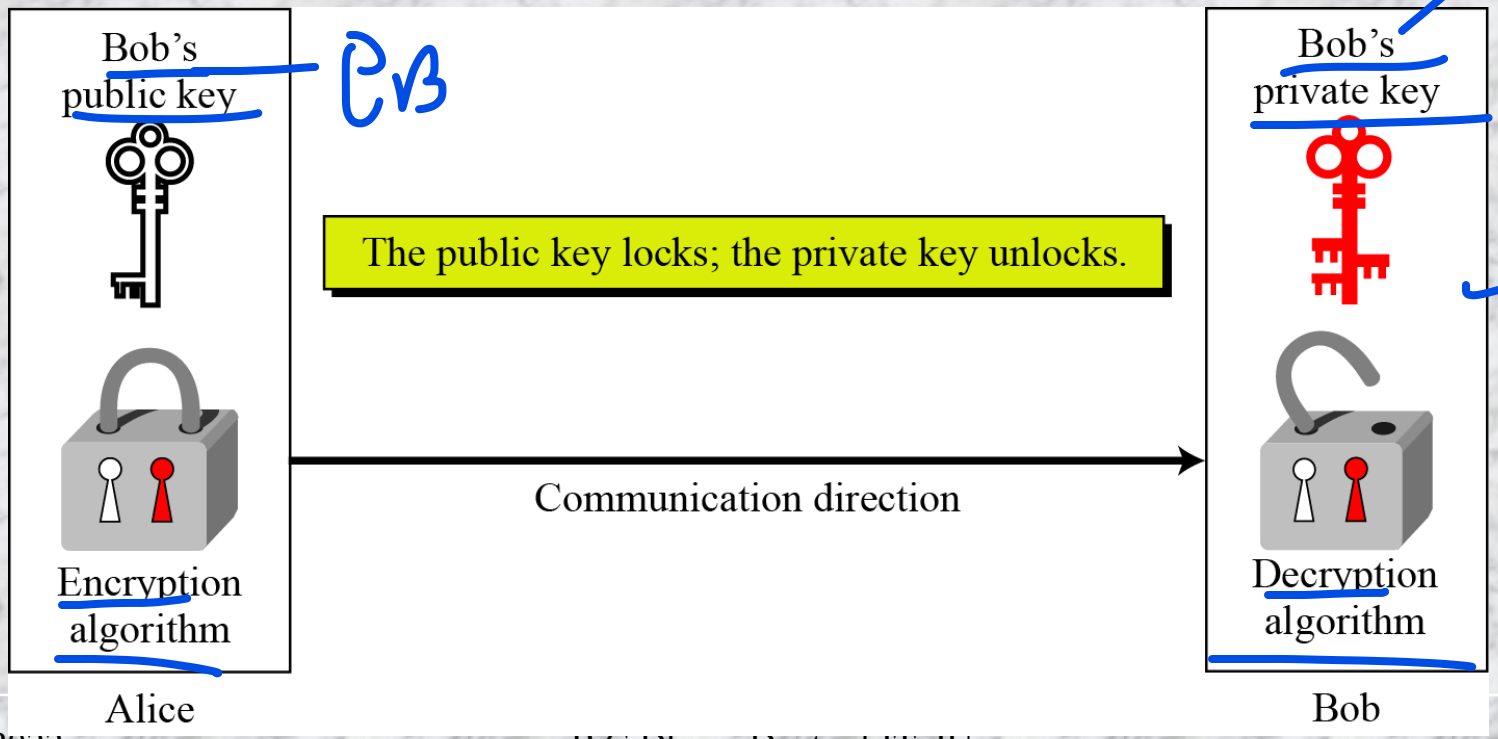
In a community of n people

symmetric-key cryptography needed n(n-1)/2 secrets

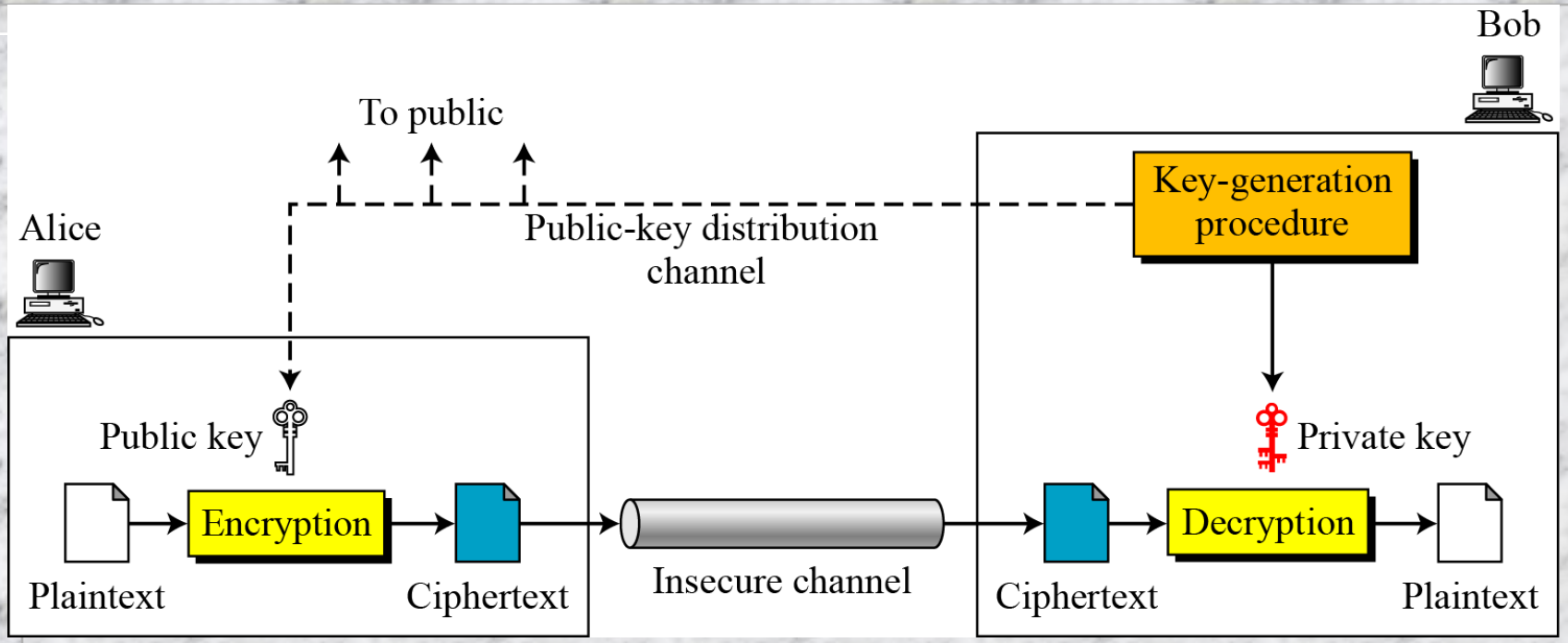
asymmetric-key cryptography needed n secrets

# Keys

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



# General idea of asymmetric-key cryptosystem




Sender and receiver cannot use same set of keys for two-way communication

# *General idea of asymmetric-key cryptosystem (contd…)*

- In symmetric-key cryptography plaintext and ciphertext are symbols which are either permuted or substituted
- In asymmetric- key cryptography plaintext and ciphertext are numbers, which are manipulated by mathematical functions

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

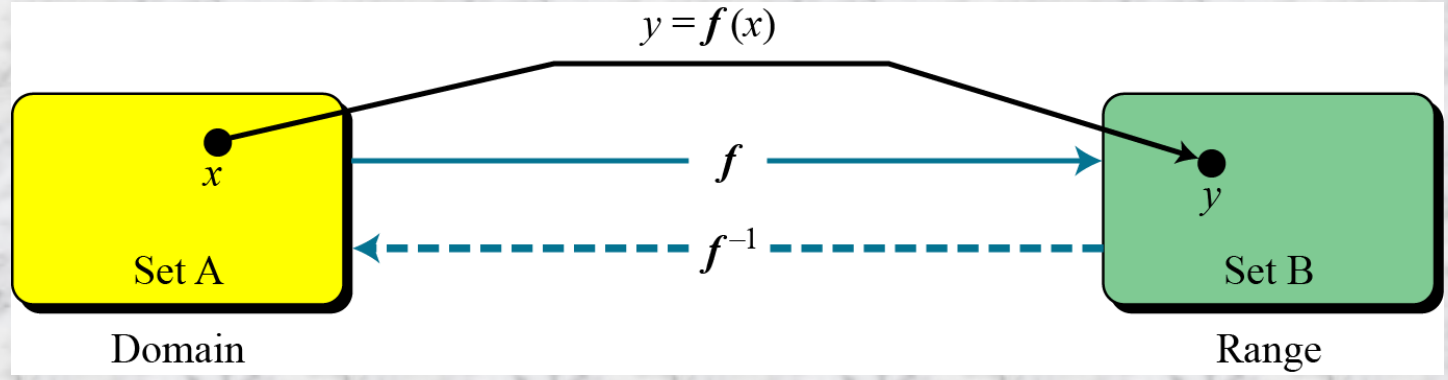
# *Need for Both cryptography*

- 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
- Asymmetric-key cryptography uses mathematical functions for both encryption and decryption
  - much slower than symmetric-key cryptography
  - for large message, symmetric-key cryptography is needed
  - Asymmetric-key cryptography is needed for authentication, digital signature, and secret-key exchange
- People believe that they are complements of each other

# *Trapdoor One-Way Function*

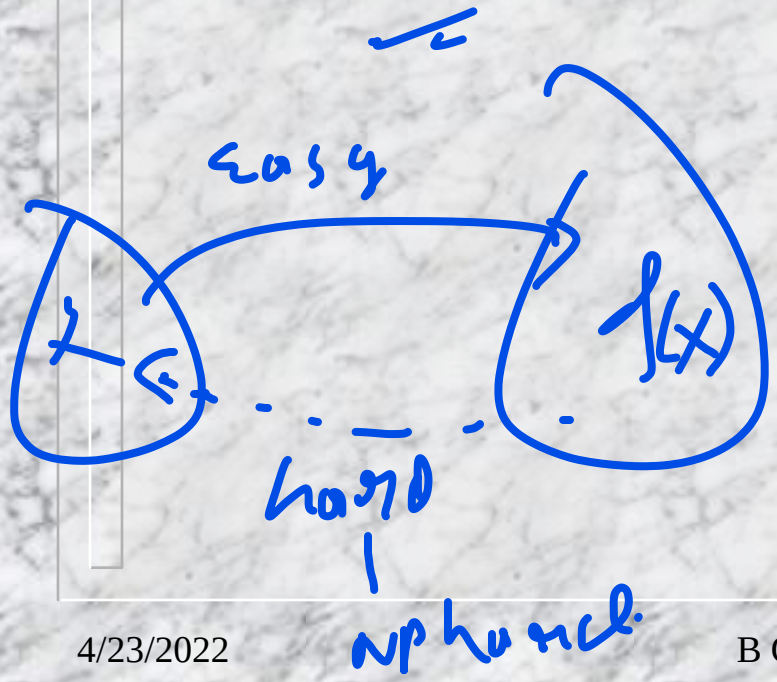
- *The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function*

- *Function:*



# Trapdoor One-Way Function

- *One-Way Function (OWF)*
  - *f is easy to compute*
  - *$f^{-1}$ is difficult to compute*



- *Trapdoor One-Way Function (TOWF)*
  - *f is easy to compute*
  - *$f^{-1}$ is difficult to compute*
  - *Given y and a **trapdoor**, x can be computed easily*

# Example

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

When $n$ is large, $y = x^k \bmod n$ 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

if we know the trapdoor, k′ such that $k \times k' = 1 \bmod n$, we can use x = $y^{k'} \bmod n$ to find x

# *Knapsack Cryptosystem*

- This Cryptosystem is insecure with today's standard
  - The main idea behind this cryptosystem gives an insight into recent public-key cryptosystems

*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.*

# *Knapsack Cryptosystem (contd…)*

- Finding solution of knapsack is easy if ‘a’ is superincreasing ,i.e., $a_i \geq a_1 + a_2 + \ldots + a_{i-1}$

**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]
}
```

# *Knapsack Cryptosystem (contd…)*

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:**

1. Consider superincreasing k-tuple $b=[b_1, b_2,\ldots, b_k]$
2. Select n , $n> b_1+ b_2+\ldots+b_k$
3. Select a random integer r, relatively prime to n and $1 \leq r <n$
4. Compute k-tuple $t= [t_1, t_2,\ldots, t_k]$, $b_i = r \times b_i,\ldots$mod n
5. *a*=permute(t)
6. Public-key: *a* ; private-key: n, r, b, permute

**Encryption:**

1 Convert message to k-tuple $x=[x_1, x_2,\ldots, x_k]$, $x_i \in \{0, 1\}$

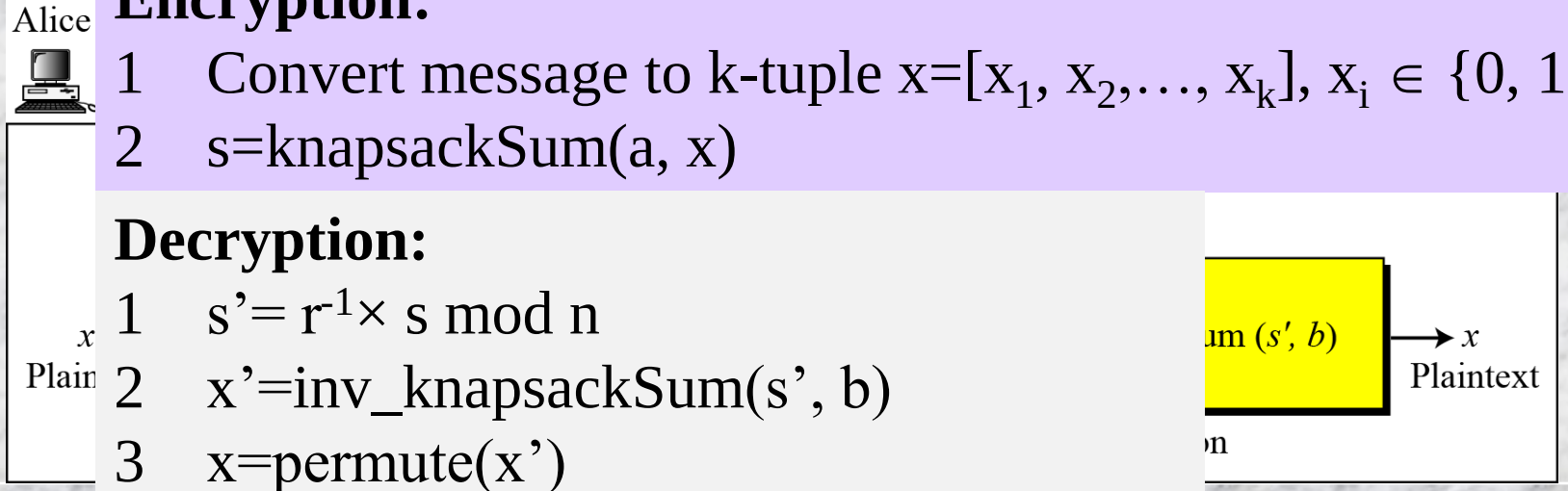
2 s=knapsackSum(a, x)

**Decryption:**

1 $s'= r^{-1}\times s$ mod n

2 x'=inv_knapsackSum(s', b)

3 x=permute(x')



# Example

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".

**Ke**

1.

2.

**E Decryption:**

1 1 s'= $r^{-1}$× s mod n

2 2 x'=inv_knapsackSum(s', b)

6 3 x=permute(x')

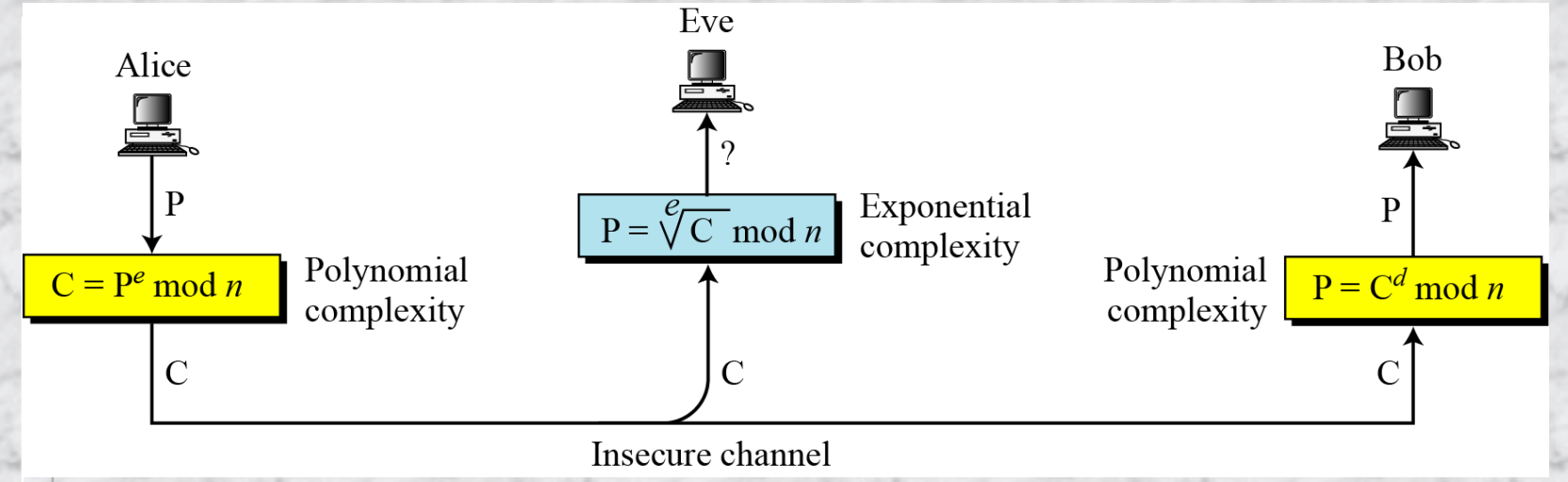
$_k$], $x_i \in \{0, 1\}$

# RSA Cryptosystem

- *The most common public-key algorithm is the RSA cryptosystem, named for its inventors (Rivest, Shamir, and Adleman)*
- *RSA uses two exponent: e and d*
  - *e is public and d is private*
  - *Encryption: $C = P^e \bmod n$*
  - *Decryption: $P = C^d \bmod n$*
  - *n is very large, created during key generation*

Both encryption and decryption uses modular exponent

# RSA Cryptosystem (contd…)

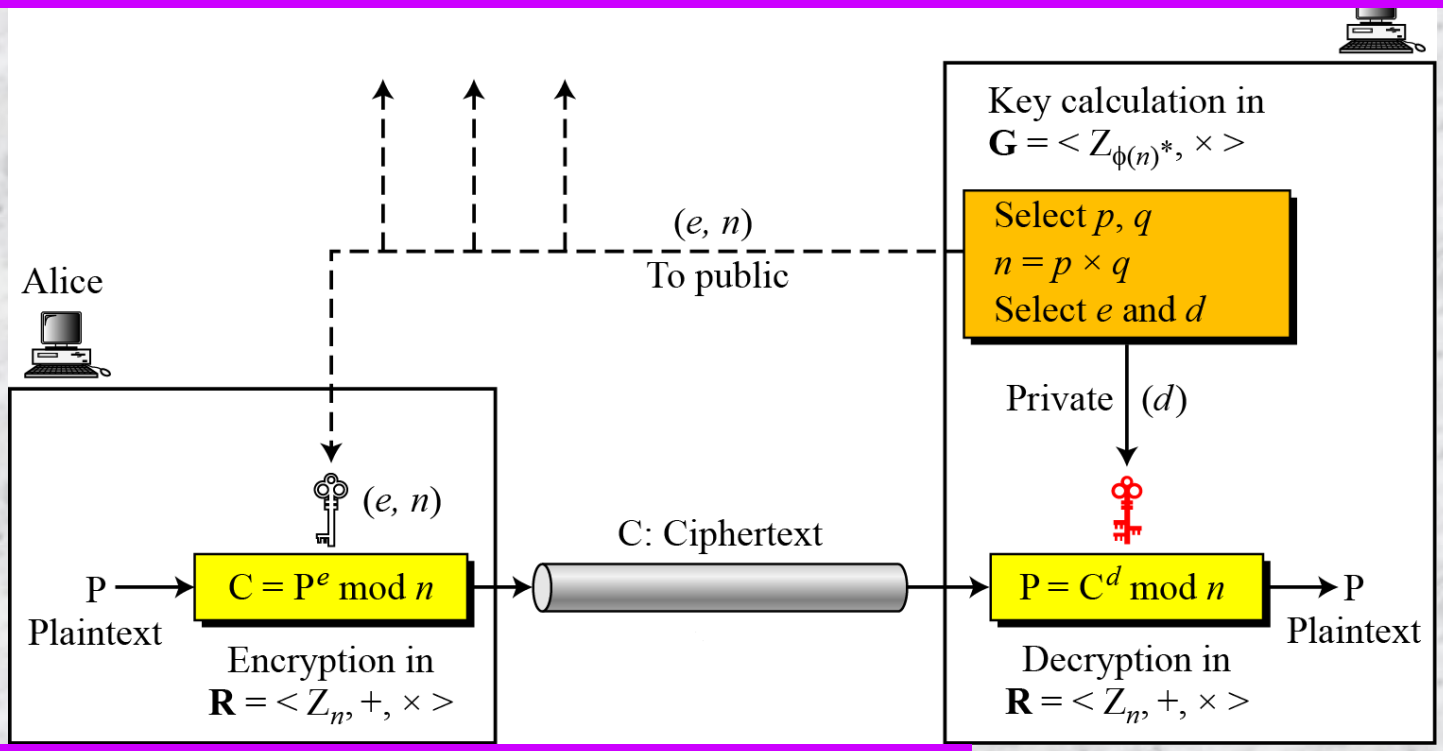



**RSA uses modular exponentiation for encryption/decryption; To attack it, Eve needs to calculate $\sqrt[e]{C} \bmod n$.**

Modular exponent is polynomial time
Modular logarithm is as hard as modulus factoring

# Procedure of RSA

Key is generated in multiplicative group $G = <Z_{\phi(n)}*, \times>$ *(private group)*




Encryption/decryption are done in commutative ring $R = <Z_n*, +, \times>$ *(public)*

(e, n) is public and d is private

# Key generation

**Algorithm 10.2** *RSA Key Generation*

**RSA_Key_Generation**

{

Select two large primes $p$ and $q$ such that $p \neq q$.

$n \leftarrow p \times q$

$\phi(n) \leftarrow (p-1) \times (q-1)$ // Discard p and q and $\phi(n)$

Select $e$ such that $1 < e < \phi(n)$ and $e$ is coprime to $\phi(n)$

$d \leftarrow e^{-1} \bmod \phi(n)$ // $d$ is inverse of $e$ modulo $\phi(n)$

Public_key $\leftarrow$ (e, $n$) // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

multiplicative inverse

# RSA encryption/decryption

**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.**

**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
}
```

# RSA encryption/decryption (contd…)

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)

# RSA cryptosystem: example1

- Bob chooses 7 and 11 as *p* and *q* and calculates *n* = 77. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60
- Select *e* and *d*, from $Z_{60}*$, $e \times d \bmod 60 = 1$ (e=13 and d=37 are inverses of each)
- Plaintext is 5, e=13 then

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

- The ciphertext is 26 and the private key 37 to decipher the ciphertext as:

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

# RSA cryptosystem: example2

- For another plaintext 63 with the same key e=13, the ciphertext will be

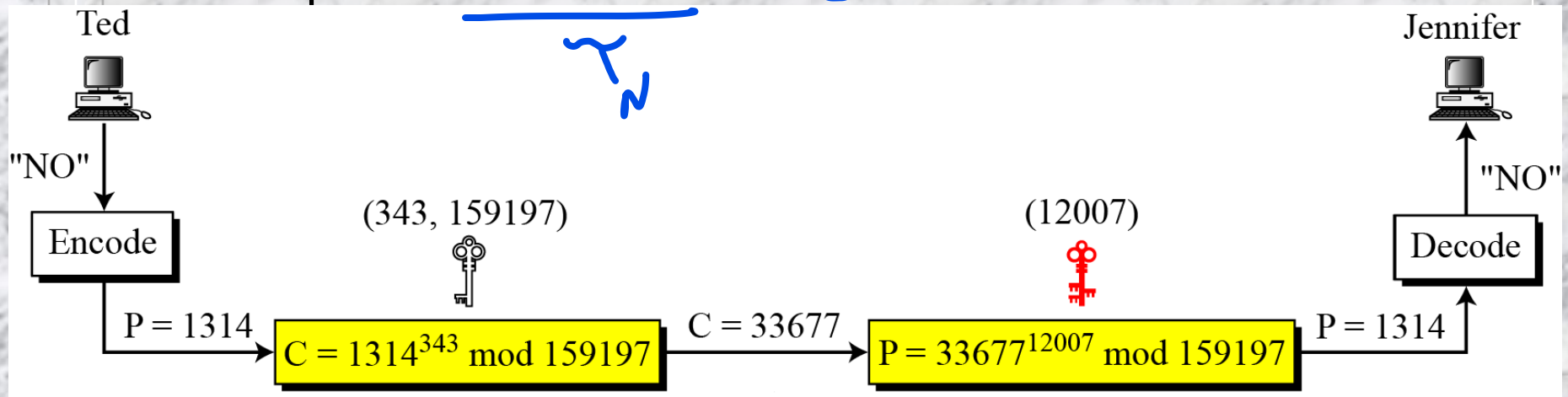
Plaintext: 63 $C = 63^{13} = 28 \bmod 77$ Ciphertext: 28

- For ciphertext 28 with private key 37, the plaintext is

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

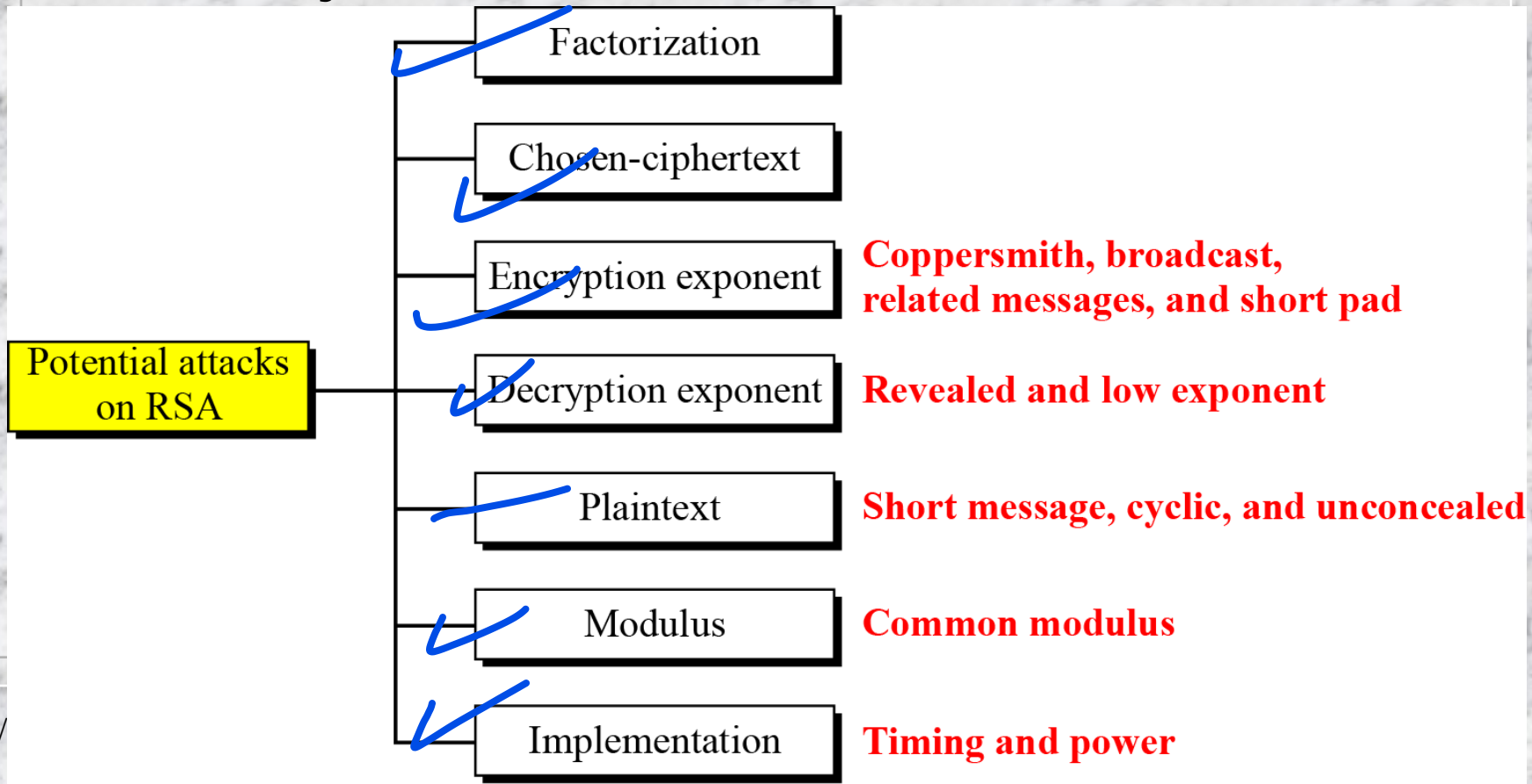
# RSA cryptosystem: example3

- For $p$ = 397 and $q$ = 401, $n$ = 159197
- $\phi(n)$ = 158400; say, e = 343 then d = 12007
- Suppose, message is “NO” to transmit
- Change each character to a number (from 00 to 25), with each character coded as two digits
- Then concatenates the two coded characters and gets a four-digit number
- The plaintext is 1314



# Attacks on RSA

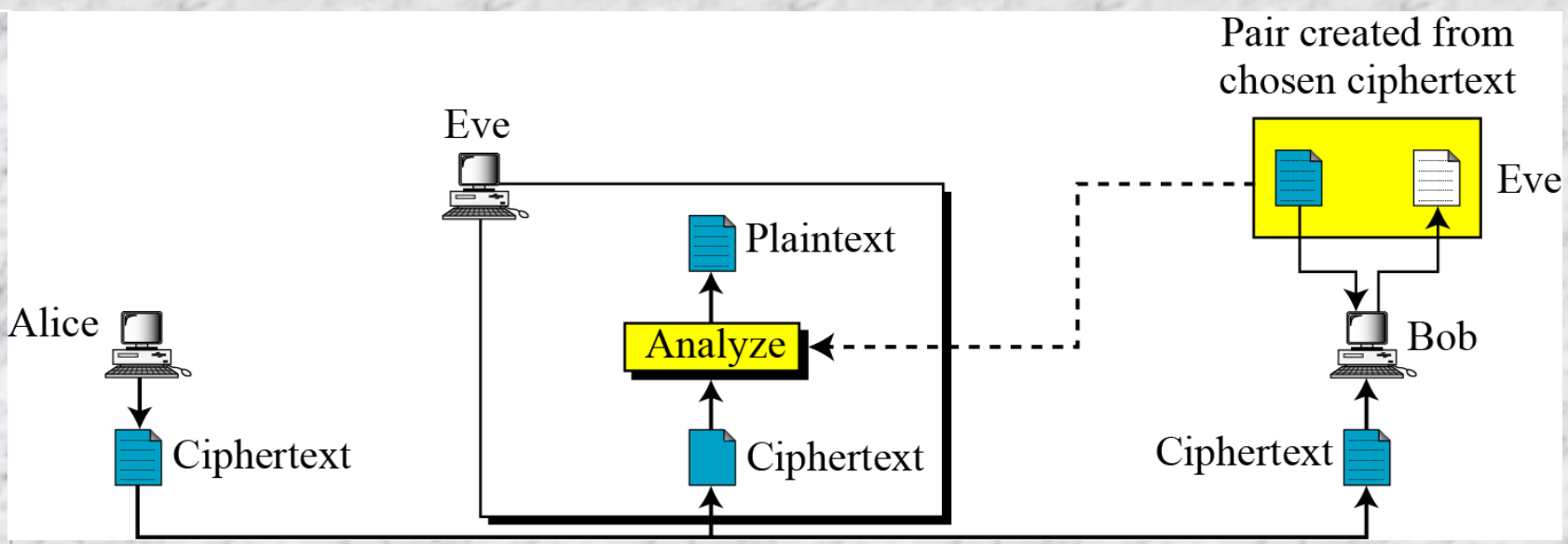
- No devastating attacks on RSA have been discovered yet



# Factorization attack on RSA

- Factoring n (n=p*q) is infeasible in reasonable time
- If n can be factored, then $\phi(n) = (p-1)*(q-1)$ can be computed, $d = e^{-1} \bmod \phi(n)$ can be calculated
- None of the factorization algorithm can factored a large number in polynomial time
- If n is number having more than 300 digits (i.e., more than 1024 bits)
  - It takes infeasible long time
- RSA is secure as long as no efficient factorization algorithm is designed

# Chosen-ciphertext attack



# Chosen-ciphertext attack

1. Sender creates $C = P^e \bmod n$ and send to receiver
2. Attacker:
   1. Choose a random number $X \in Z_n^*$
   2. Compute $Y = C \times X^e \bmod n$
   3. Y sends to receiver (i.e., attacker also have access machine of the receiver), i.e., $Z=Y^d \bmod n$
   4. Attacker find P

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

$$P = Z \times X^{-1} \bmod n$$

# Attacks on the encryption exponent

- Small exponent value e is vulnerable
  - Normally used value 3 (second prime)
  - Recommended value of e to use is $2^{16}+1$ =65537 (or a prime number close to this)
- Broadcast attacks:
  - Same message is sends to a group of recipients with same low exponent
  - $C_1 = P^3 \bmod n_1$ $C_2 = P^3 \bmod n_2$ and $C_3 = P^3 \bmod n_3$
  - Chinese remainder theorem: $C' = P^3 \bmod n_1 n_2 n_3$ and $P^3 < n_1 n_2 n_3$
  - $P = C'^{1/3}$

# Short pad attack

- $C_1$= Enc(M | r1, e, n) not reached to recipient
- $C_2$= Enc(M | r2, e, n)
- Attacker intercepts both C1 and C2 and drop
  - Since, both C1 and C2 are from same message and r1 and r2 are short, then original message can be easily recovered by the attacker

# Attacks on decryption exponent

- If d is compromised, then p, q, n, e and d must be regenerated
- The recommendation is to have $d \geq 1/3\ n^{¼}$ to prevent low decryption exponent attack

# Plaintext attacks

- Plaintext and ciphertext in RSA are permutation of each other
  - This allow some attacks on the plaintext
  - Short message attack, cycling attack and unconcealed attack
- Short message attack
  - Set of possible plaintexts in known to attacker
    - Additional information, ciphertext is permutation of plaintext
    - Exhaustive search
    - Message be padded with random bits (at front end and the end) before encryption using a method called OAEP

# Cycling attack

- Ciphertext is permutation of plaintext
  - Continuous encryption of the ciphertext will result the plaintext

$$C_1 = C^e \bmod n$$

$$C_2 = C_1^e \bmod n$$

....

$$C_k = C_{k-1}^e \bmod n$$

  - If $C_k = C$, then $P=C_{k-1}$

# Unconcealed message attack

- If C== P
  - In exponent, for P=0 or P=1, C=P
  - We have more for some particular exponent
  - The encryption program, checks the ciphertext and if its same as plaintext
    - Reject that plaintext

# Attacks on implementatio

- So far, we consider attacks on underlying structure of RSA
- Now, attacks on the implementation of RSA:
  - Timing attack, and
  - Power attack

# Timing attack

- In RSA, exponent is used
  - Exponent (definitely odd number) can be computed by square and multiplication method
    - If the corresponding bit value of exponent is 0 ➔ square
    - If the corresponding bit value of exponent is 1 ➔ both square and multiplication
      - Bit 1 ➔ more execution time
      - Bit 0 ➔ less execution time

# Power attack

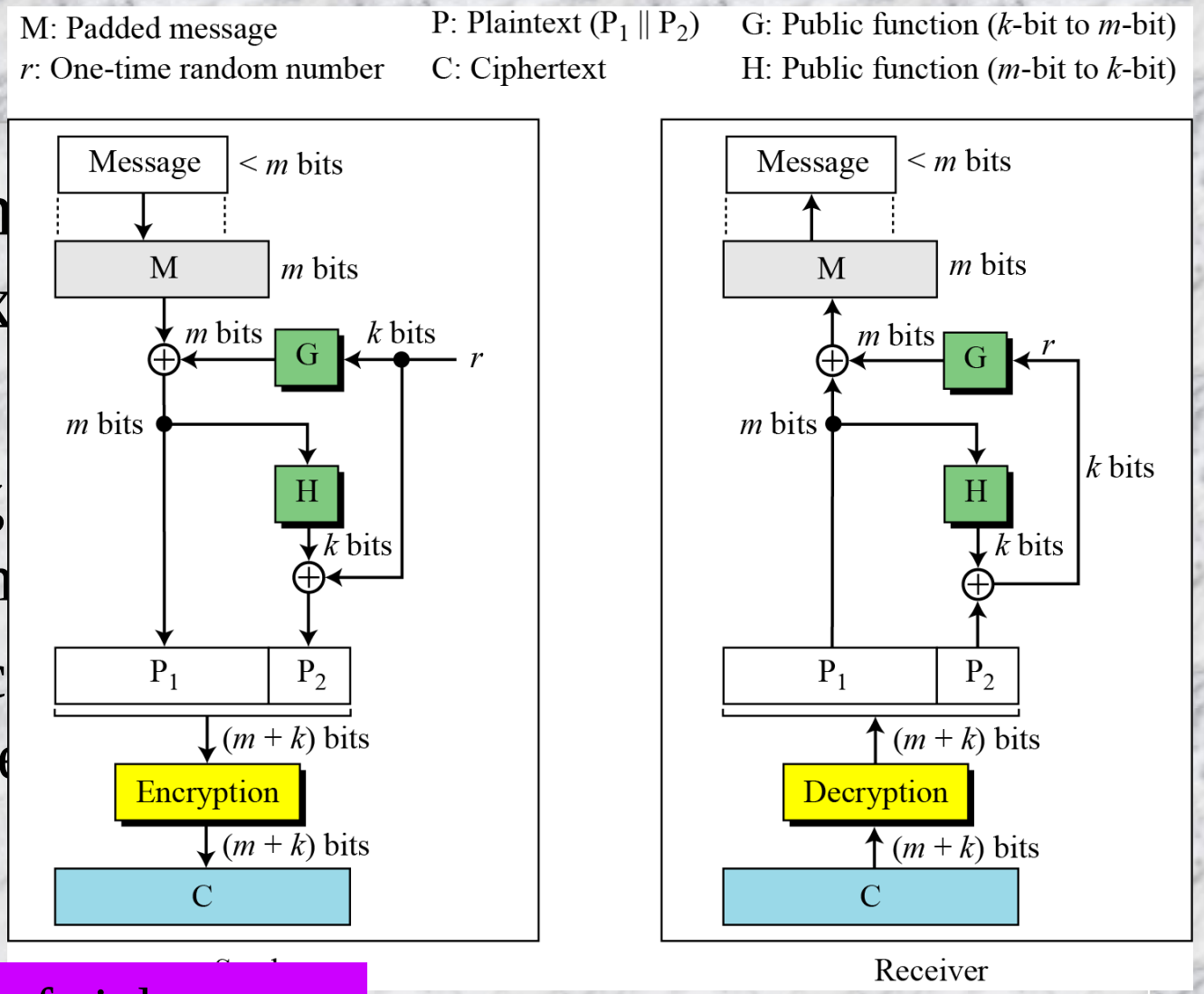
- Similar to timing attack
- Measures the power consumption

# Recommendations for RSA

- Number of bits for n should be1024, i.e., 309 decimal digits
- Two prime numbers should be at least 512 bits, i.e., 154 decimal digits
- Values of p and q should not be very close to each other
- p and q should have least one large prime factor
- The value of e should be close to $2^{16}+1$
- Message must be padded using OAEP

# OAEP (optimal asymmetric encryption padding)

- A short n
  ciphertex
  attack
  - Adding
    attack h
  - The rec
    asymme




Error in transmission of ciphertext, recovery of plaintext is not possible

# RABIN CRYPTOSYSTEM

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

# RABIN CRYPTOSYSTEM

**Algorithm 10.8** *Decryption in Rabin cryptosystem*

**Rabin_Decryption** ($p$, $q$, C) // C is the ciphertext; $p$ and $q$ are private keys

{

$a_1 \leftarrow +(C^{(p+1)/4}) \bmod p$

$a_2 \leftarrow -(C^{(p+1)/4}) \bmod p$

$b_1 \leftarrow +(C^{(q+1)/4}) \bmod q$

$b_2 \leftarrow -(C^{(q+1)/4}) \bmod q$

// The algorithm for the Chinese remainde

$P_1 \leftarrow$ Chinese_Remainder ($a_1$, $b_1$, $p$, $q$)

$P_2 \leftarrow$ Chinese_Remainder ($a_1$, $b_2$, $p$, $q$)

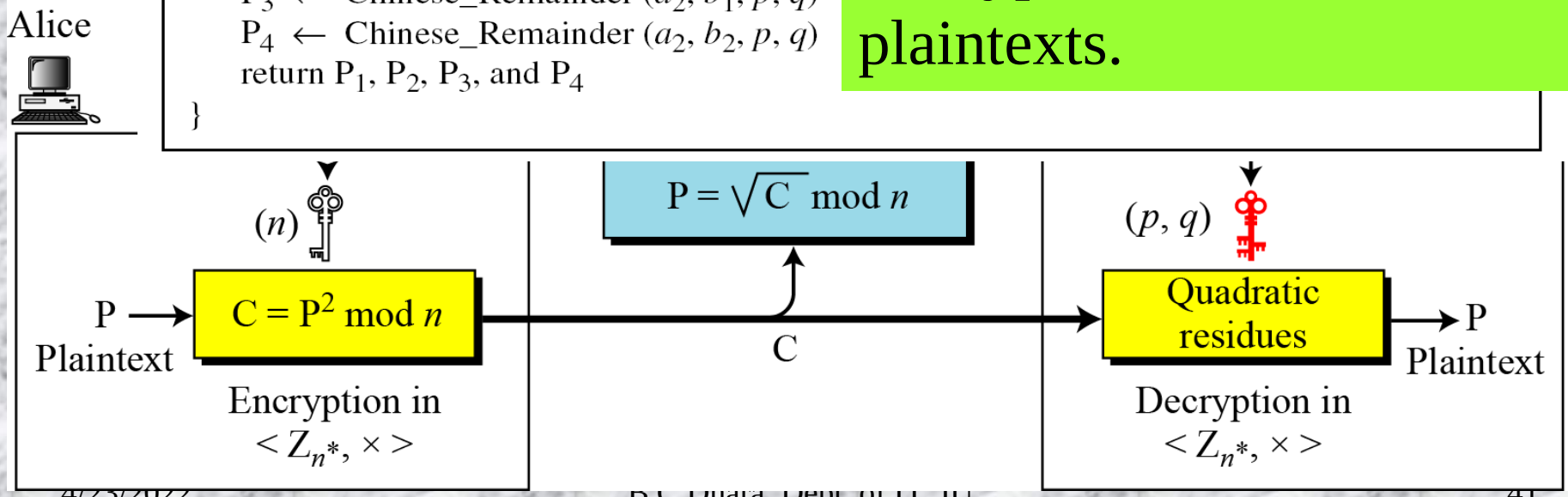
$P_3 \leftarrow$ Chinese_Remainder ($a_2$, $b_1$, $p$, $q$)

$P_4 \leftarrow$ Chinese_Remainder ($a_2$, $b_2$, $p$, $q$)

return $P_1$, $P_2$, $P_3$, and $P_4$

}

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



# Example: Rabin crypto system

1. Bob selects $p = 23$ and $q = 7$, both are congruent to 3 mod 4.

5. Bob receives 93 and calculates four values:

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

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

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

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

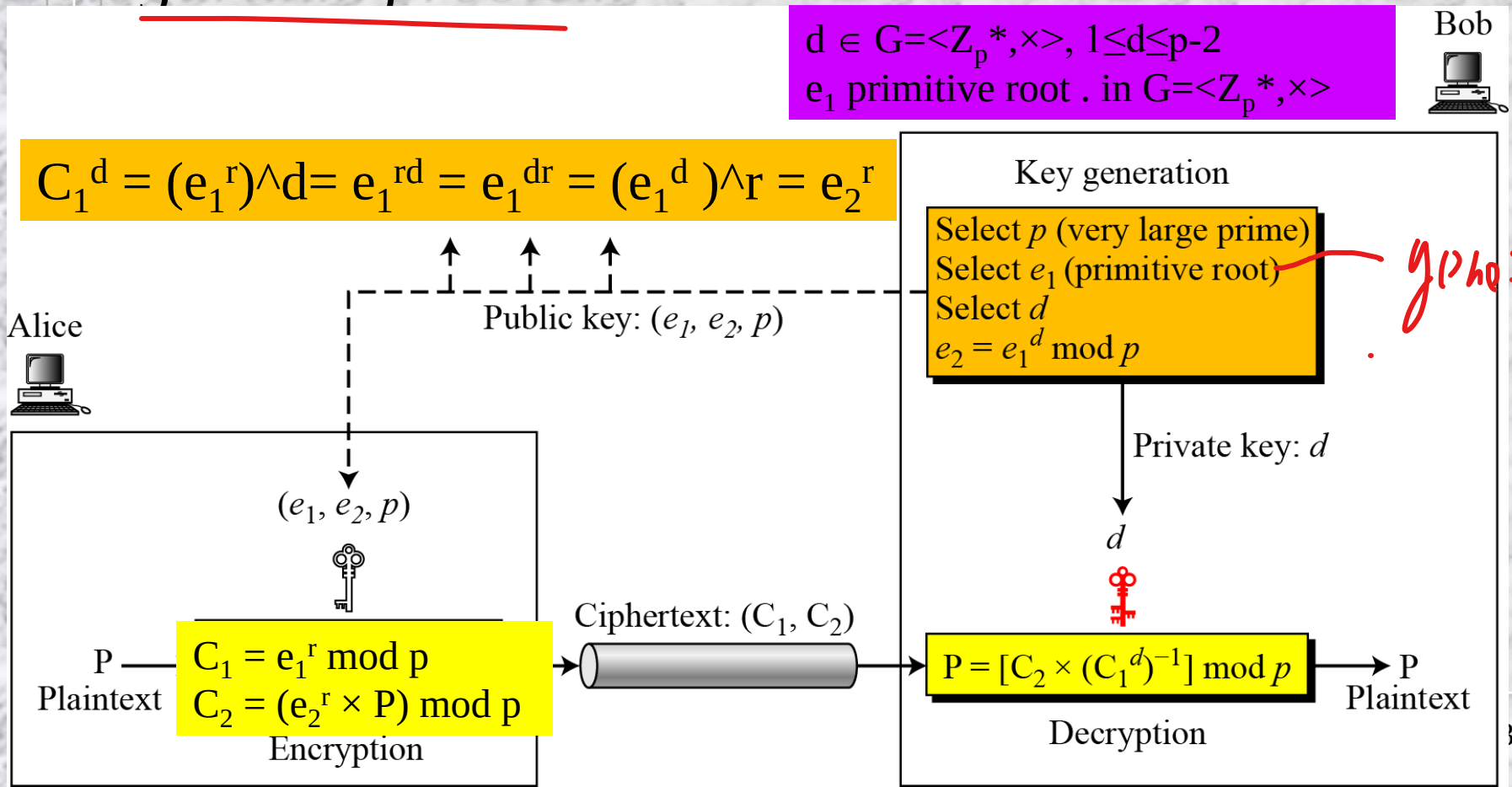
6. Bob takes four possible answers, $(a_1, b_1)$, $(a_1, b_2)$, $(a_2, b_1)$, and $(a_2, b_2)$, and uses the Chinese remainder theorem to find four possible plaintexts: 116, 24, 137, and 45.

7. $116^2 \equiv 93 \bmod 161$, $24^2 \equiv 93 \bmod 161$, $137^2 \equiv 93 \bmod 161$, $45^2 \equiv 93 \bmod 161$

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

# ELGAMAL CRYPTOSYSTEM

- *Besides RSA and Rabin, another public-key cryptosystem is ElGamal, based on the discrete logarithm problem*



# Example: ElGamal cryptosystem

*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.*

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

*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**

# Security of ElGamal

- Low modulus attack
  - If p is nor large enough, some efficient algorithm to find d and r
  - Recommended, p must be at least 300 digits
- Known-plaintext attack
  - If same r is used to encrypt two plaintexts P and P'
  - When P is known then P' can be discovered

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

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

# 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)*

# *Elliptic Curves over Real Numbers*

- *The general equation for an elliptic curve is*

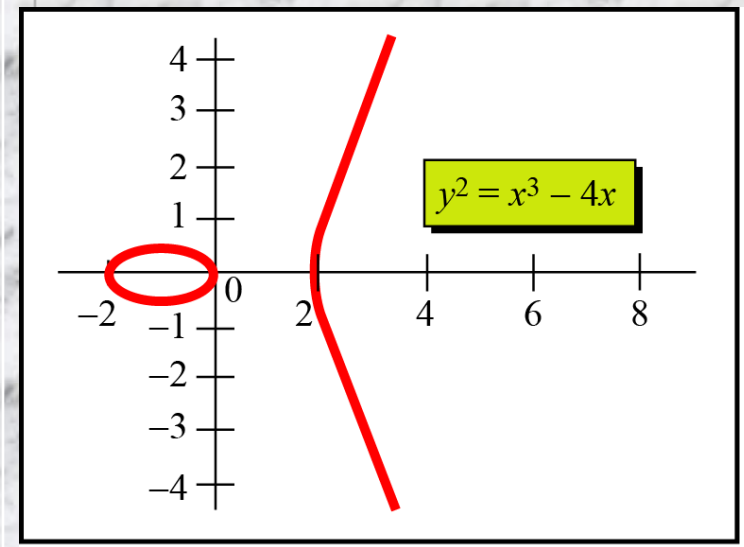
$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

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

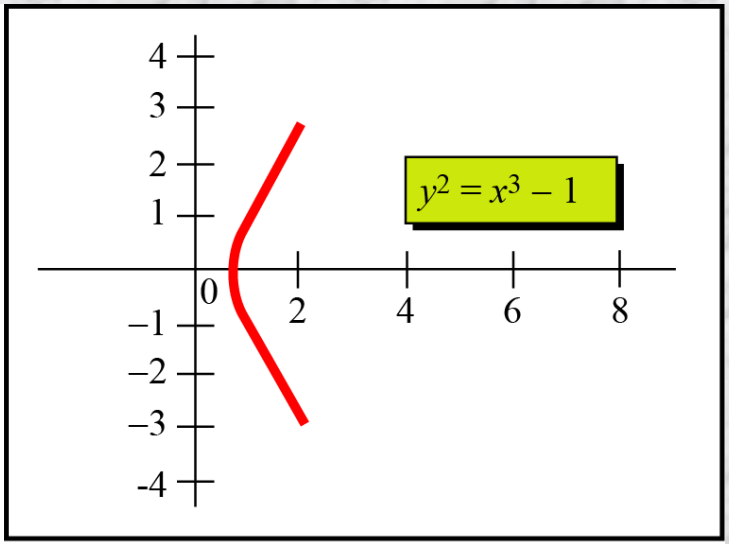
$$y^2 = x^3 + ax + b$$

# *Elliptic Curves over Real Numbers (contd…)*

- If $4a^3+27b^2 \neq 0$, the equation $x^3+ax+b=0$ has three distinct root



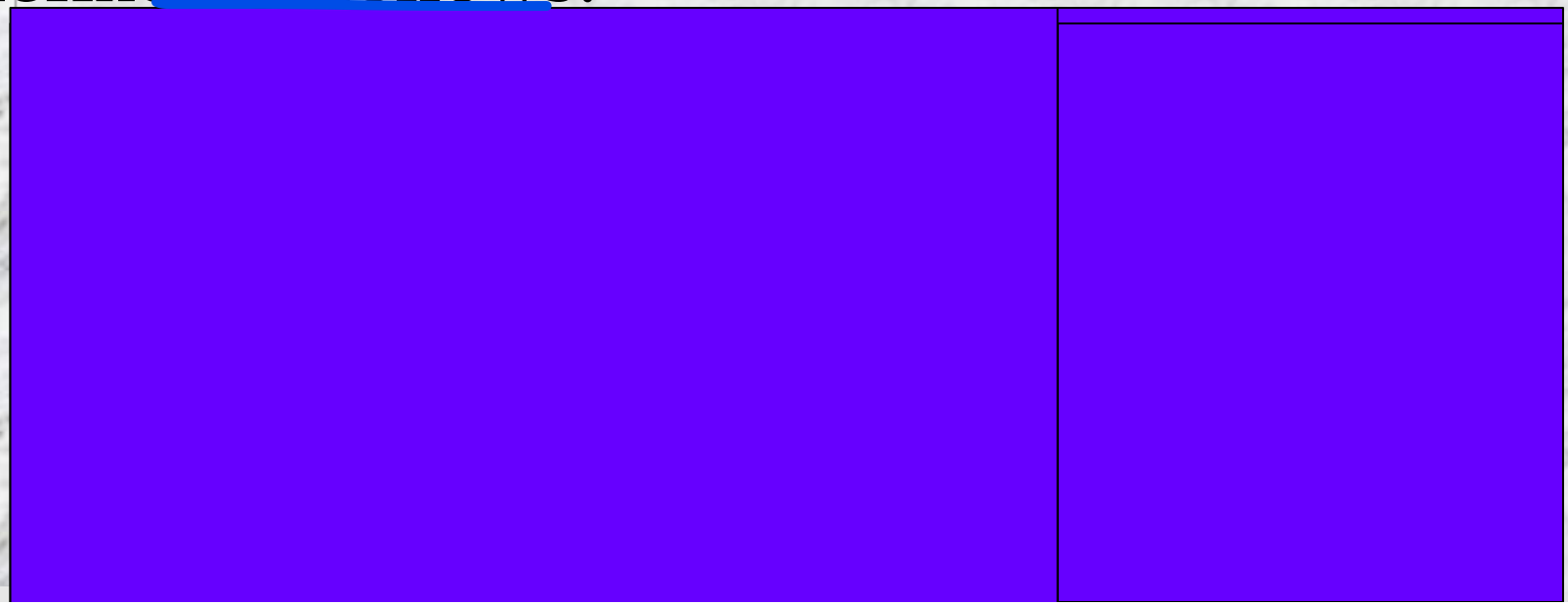

a. Three real roots




b. One real and two imaginary roots

# Points on Elliptic curve

- Points on Elliptic curve over real numbers forms an Abelian group under addition, where addition is defined as follows:



*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.*

# *Finding an Inverse in GF(p)*

- *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 Ep(a, b) in GF(p)*

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

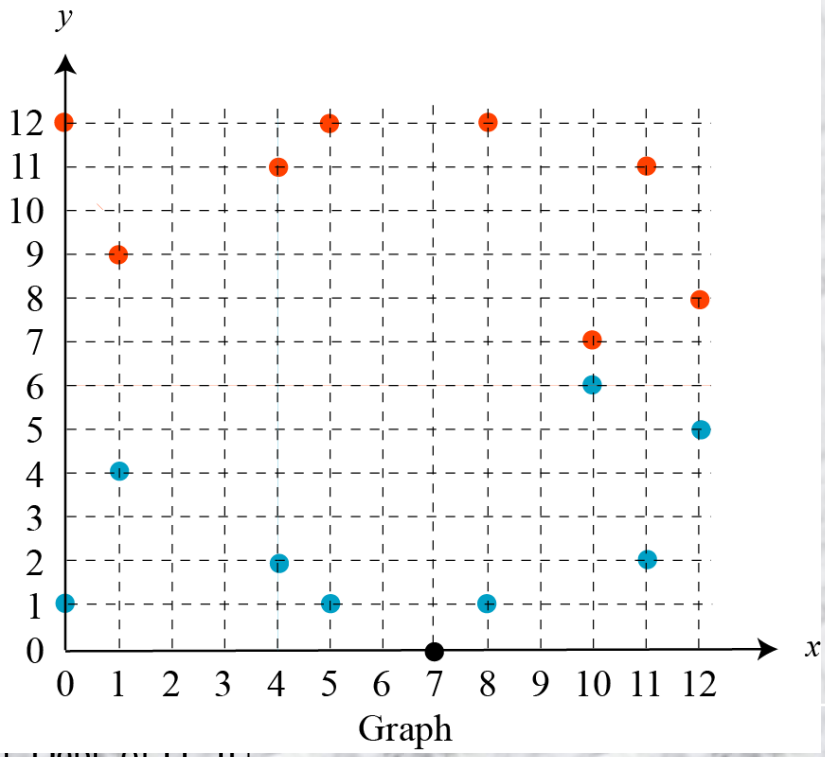
```
ellipticCurve_points (p, a, b)                    // p is the modulus
{
    x ← 0
    while (x < p)
    {
        w ← (x^3 + ax + b) mod p                  // w is y^2
        if (w is a perfect square in Z_p) output (x, √w) (x, − √w)
        x ← x + 1
    {
}
```

# Finding Points on the Curve $Ep(a, b)$ in $GF(p)$

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

| | P | –P |
|---|---|---|
| | (0, 1) | (0, 12) |
| | (1, 4) | (1, 9) |
| | (4, 2) | (4, 11) |
| | (5, 1) | (5, 12) |
| P = –P → | (7, 0) | (7, 0) |
| | (8, 1) | (8, 12) |
| | (10, 6) | (10, 7) |
| | (11, 2) | (11, 11) |
| | (12, 5) | (12, 18) |

Points



# Adding two points in GF(p)

*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.*

*How about $E_{23}(1,1)$, let $P=(3, 10)$ and $Q=(9,7)$: $P + Q$?; $2P$?*

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

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

# *Elliptic Curves over GF($2^n$)*

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

*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..*

# Points on EC in $GF(2^n)$

*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 |

# Points on EC in $GF(2^n)$ (contd…)

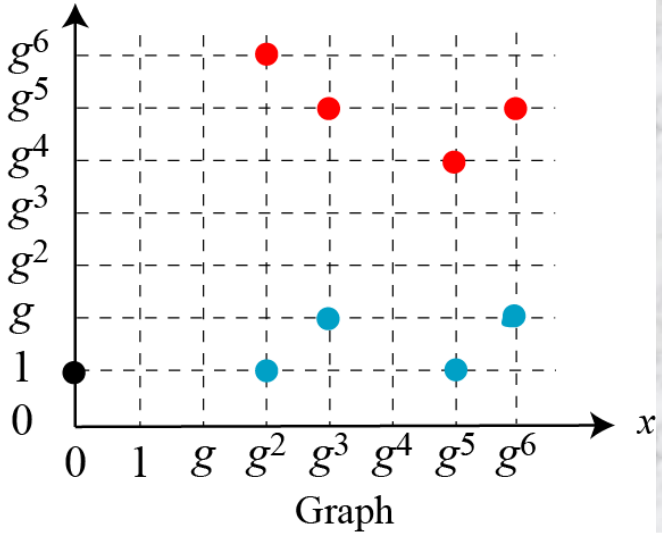
*Using the elliptic curve $y^2 + xy = x^3 + g^3x^2 + 1$, with $a = g^3$ and $b = 1$, we can find the points on this curve, as shown in Figure 10.15..*

| | | | |
|---|---|---|---|
| 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 |

$P = -P \longrightarrow$

| 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

# Adding two points on EC in $GF(2^n)$

*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*

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

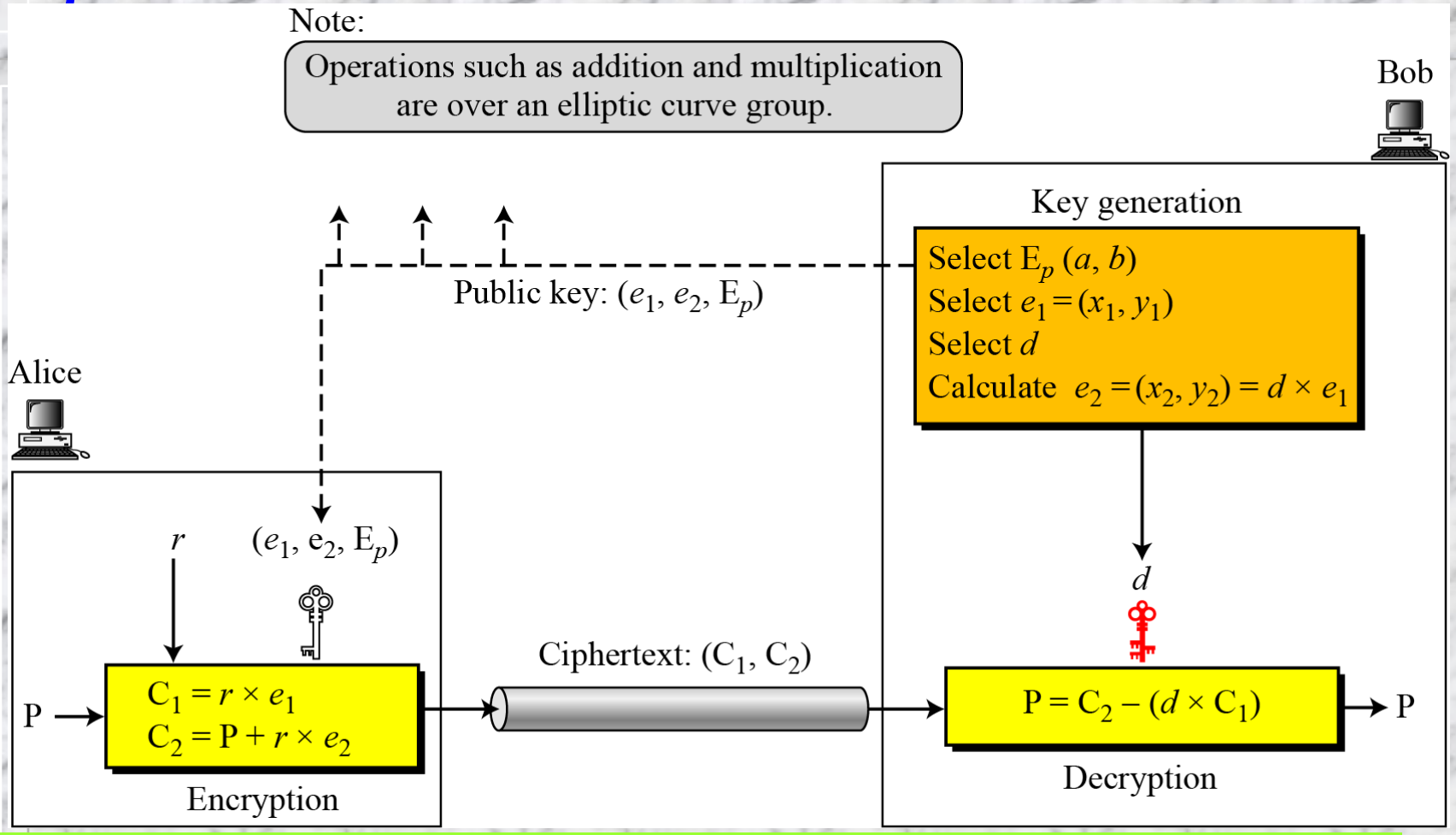
$$x_3 = \lambda^2 + \lambda + a \qquad y_3 = x_1^2 + (\lambda + 1) x_3$$

# Adding two points on EC in $GF(2^n)$: example

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

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

# ElGamal cryptosystem using the elliptic curve




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

# Example: ECC

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 sends the plaintext $P = (24, 26)$ to Bob. She selects $r = 2$.*
6. *Alice finds the point $C_1=(35, 1)$, $C_2=(21, 44)$.*
7. *Bob receives $C_1$, $C_2$. He uses $4xC_1(35,1)$ to get (23, 25), inverts the points (23, 25) to get the points (23, 42).*
8. *Bob adds (23, 42) with $C_2=(21, 44)$ to get the original one P=(24, 26).*

# *Comparable Key Sizes for Equivalent Security*

| Symmetric scheme (key size in bits) | ECC-based scheme (size of *n* in bits) | RSA/DSA (modulus size in bits) |
|---|---|---|
| 56 | 112 | 512 |
| 80 | 160 | 1024 |
| 112 | 224 | 2048 |
| 128 | 256 | 3072 |
| 192 | 384 | 7680 |
| 256 | 512 | 15360 |