Course: Cryptography and Network Security
Code: CS-34310
Branch: M.C.A - 4th Semester

Lecture – 9 : Log and Expo
ASYMMETRIC-KEY CRYPTOGRAPHY

Faculty & Coordinator : Dr. J Sathish Kumar (JSK)

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad,
Prayagraj-211004

EXPONENTIATION AND LOGARITHM

Exponentiation: $y = a^x$ $\rightarrow$ **Logarithm:** $x = \log_a y$

Exponentiation

- Fast Exponentiation
  - The idea behind the square-and-multiply method

$$x = x_{n_b-1} \times 2^{k-1} + x_{n_b-2} \times 2^{k-2} + \cdots + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

$$\boxed{y = a^{\boxed{x_{n_b-1} \times 2^{n_b-1} + x_{n_b-2} \times 2^{n_b-2} + \cdots + x_1 \times 2^1 + x_0 \times 2^0}}}$$

in which x_i is 0 or 1

$$\downarrow$$

$$\boxed{y = \boxed{\boxed{a^{2^{n_b-1}}} \text{ or } \boxed{1}} \times \boxed{\boxed{a^{2^{n_b-2}}} \text{ or } \boxed{1}} \times \cdots \times \boxed{\boxed{a^{2}} \text{ or } \boxed{1}} \times \boxed{\boxed{a} \text{ or } \boxed{1}}}$$

Example:

$$\boxed{y = a^9 = a^{1001_2} = a^8 \times 1 \times 1 \times a}$$

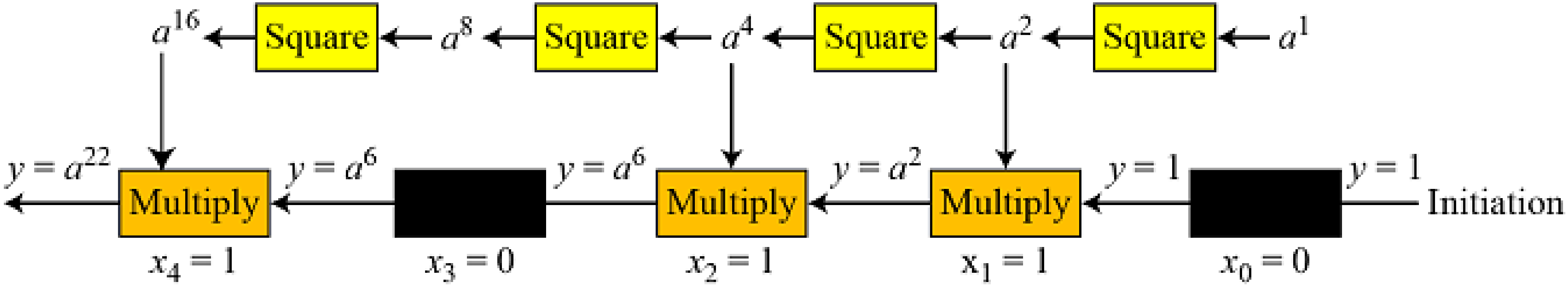
Exponentiation

```
Square_and_Multiply (a, x, n)
{
    y ← 1
    for (i ← 0 to n_b − 1)                  // n_b is the number of bits in x
    {
        if (x_i = 1)    y ← a × y  mod n     // multiply only if the bit is 1

        a ← a^2 mod n                        // squaring is not needed in the last iteration
    }
    return y
}
```

Exponentiation

- The process for calculating $y = a^x$
- In this case, $x = 22 = (10110)_2$ in binary.



Exponentiation

Calculation of 17^{22} mod 21

i	x_i	*Multiplication* *(Initialization: $y = 1$)*		*Squaring* *(Initialization: $a = 17$)*
0	0		$\rightarrow$	$a = 17^2 \bmod 21 = 16$
1	1	$y = 1 \times 16 \bmod 21 = 16$	$\rightarrow$	$a = 16^2 \bmod 21 = 4$
2	1	$y = 16 \times 4 \bmod 21 = 1$	$\rightarrow$	$a = 4^2 \bmod 21 = 16$
3	0		$\rightarrow$	$a = 16^2 \bmod 21 = 4$
4	1	$y = 1 \times 4 \bmod 21 = 4$	$\rightarrow$	

Logarithm

- In cryptography we need to discuss modular logarithm

Exhaustive search for modular logarithm

```
Modular_Logarithm (a, y, n)
{
    for (x = 1 to n −1)                         // k is the number of bits in x
    {
        if (y ≡ a^x mod n) return x
    }
    return failure
}
```

Logarithm

- Order of the Group.
- Example:
  - What is the order of group $G = \langle Z_{21}*, \times \rangle$?
    - $|G| = \phi(21) = \phi(3) \times \phi(7) = 2 \times 6 = 12$. There are 12 elements in this group: 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, and 20. All are relatively prime with 21.

Logarithm

- The order of an element, a, is the smallest integer i such that $a^i \equiv e \pmod{n}$.

  - Example:
    - Find the order of all elements in G = <Z_{10}*, ×>.
    - This group has only $\phi(10) = 4$ elements: 1, 3, 7, 9.

The order of an element divides the order of the group (Lagrange theorem).

a. $1^1 \equiv 1 \bmod (10) \rightarrow \mathrm{ord}(1) = 1.$

b. $3^1 \equiv 3 \bmod (10);\ 3^2 \equiv 9 \bmod (10);\ 3^4 \equiv 1 \bmod (10)\ \rightarrow \mathrm{ord}(3) = 4.$

c. $7^1 \equiv 7 \bmod (10);\ 7^2 \equiv 9 \bmod (10);\ 7^4 \equiv 1 \bmod (10)\ \rightarrow \mathrm{ord}(7) = 4.$

d. $9^1 \equiv 9 \bmod (10);\ 9^2 \equiv 1 \bmod (10)\ \rightarrow \mathrm{ord}(9) = 2.$

Logarithm

- Primitive roots
  - In the group G = $<Z_n*, \times>$, when the order of an element is the same as ϕ(n), that element is called the primitive root of the group.
  - Example
    - There are no primitive roots in G = $<Z_8*, \times>$ because no element has the order equal to $\phi(8) = 4$.

Logarithm

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
$a = 1$	**x: 1**	x: 1	x: 1	x: 1	x: 1	x: 1	x: 1
$a = 3$	x: 3	**x: 1**	x: 3	x: 1	x: 3	x: 1	x: 3
$a = 5$	x: 5	**x: 1**	x: 5	x: 1	x: 5	x: 1	x: 5
$a = 7$	x: 7	**x: 1**	x: 7	x: 1	x: 7	x: 1	x: 7

The first time when x is 1, the value of i gives us the order of the element (double-sided boxes). The orders of elements are ord(1) = 1, ord(3) = 2, ord(5) = 2, and ord(7) = 2.

Logarithm

- Example
  - the result of $a^i \equiv x \pmod 7$ for the group $G = \langle Z_7*, \times \rangle$. In this group, $\phi(7) = 6$.

		$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
	$a = 1$	x: 1	x: 1	x: 1	x: 1	x: 1	x: 1
	$a = 2$	x: 2	x: 4	x: 1	x: 2	x: 4	x: 1
Primitive root →	$a = 3$	x: 3	x: 2	x: 6	x: 4	x: 5	x: 1
	$a = 4$	x: 4	x: 2	x: 1	x: 4	x: 2	x: 1
Primitive root →	$a = 5$	x: 5	x: 4	x: 6	x: 2	x: 3	x: 1
	$a = 6$	x: 6	x: 1	x: 6	x: 1	x: 6	x: 1

The orders of elements are ord(1) = 1, ord(2) = 3, ord(3) = 6, ord(4) = 3, ord(5) = 6, and ord(6) = 2.

Therefore, this group has only two primitive roots: 3 and 5.

Logarithm

The group $G = <Z_n^*, \times>$ has primitive roots only if n is 2, 4, p^t, or $2p^t$.

If the group $G = <Z_n^*, \times>$ has any primitive root, the number of primitive roots is $\phi(\phi(n))$.

The group $G = <Z_n^*, \times>$ is a cyclic group if it has primitive roots.
The group $G = <Z_p^*, \times>$ is always cyclic.

The idea of Discrete Logarithm

Properties of $G = \langle Z_p^*, \times \rangle$:

1. Its elements include all integers from 1 to $p - 1$.
2. It always has primitive roots.
3. It is cyclic. The elements can be created using g^x where x is an integer from 1 to $\phi(n) = p - 1$.

4. The primitive roots can be thought as the base of logarithm. If the group has k primitive roots, calculations can be done in k different bases. Given $x = \log_g y$ for any element y in the set, there is another element x that is the log of y in base g. This type of logarithm is called **discrete logarithm**. A discrete logarithm is designated by several different symbols in the literature, but we will use the notation L_g to show that the base is g (the modulus is understood).

Solution to Modular Logarithm Using Discrete Logs

Tabulation of Discrete Logarithms

Discrete logarithm for $\mathbf{G} = \langle \mathbf{Z}_7^*, \times \rangle$

y	*1*	*2*	*3*	*4*	*5*	*6*
$x = L_3\, y$	6	2	1	4	5	3
$x = L_5\, y$	6	4	5	2	1	3

- Find x in each of the following cases:
  - a. $4 \equiv 3^x \pmod 7$
  - b. $6 \equiv 5^x \pmod 7$
- Solution
  - – Use the tabulation of the discrete logarithm
    - a. $4 \equiv 3^x \bmod 7 \rightarrow x = L_3 4 \bmod 7 = 4 \bmod 7$
    - b. $6 \equiv 5^x \bmod 7 \rightarrow x = L_5 6 \bmod 7 = 3 \bmod 7$

Logarithm

Using Properties of Discrete Logarithms

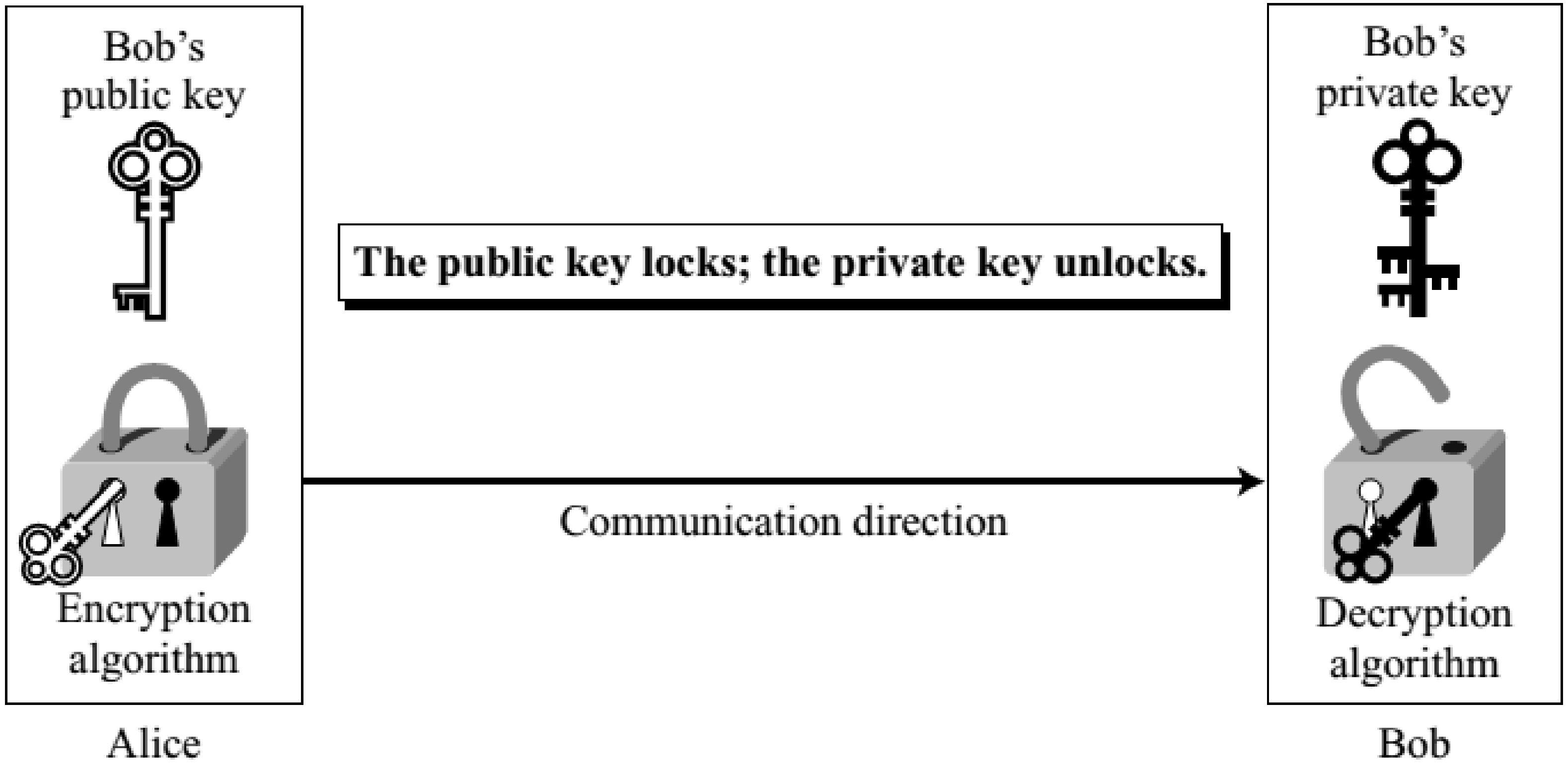
Traditional Logarithm	*Discrete Logarithms*
$\log_a 1 = 0$	$L_g 1 \equiv 0 \pmod{\phi(n)}$
$\log_a (x \times y) = \log_a x + \log_a y$	$L_g(x \times y) \equiv (L_g x + L_g y) \pmod{\phi(n)}$
$\log_a x^k = k \times \log_a x$	$L_g x^k \equiv k \times L_g x \pmod{\phi(n)}$

The discrete logarithm problem has the same complexity as the factorization problem.

Asymmetric-Key Cryptography

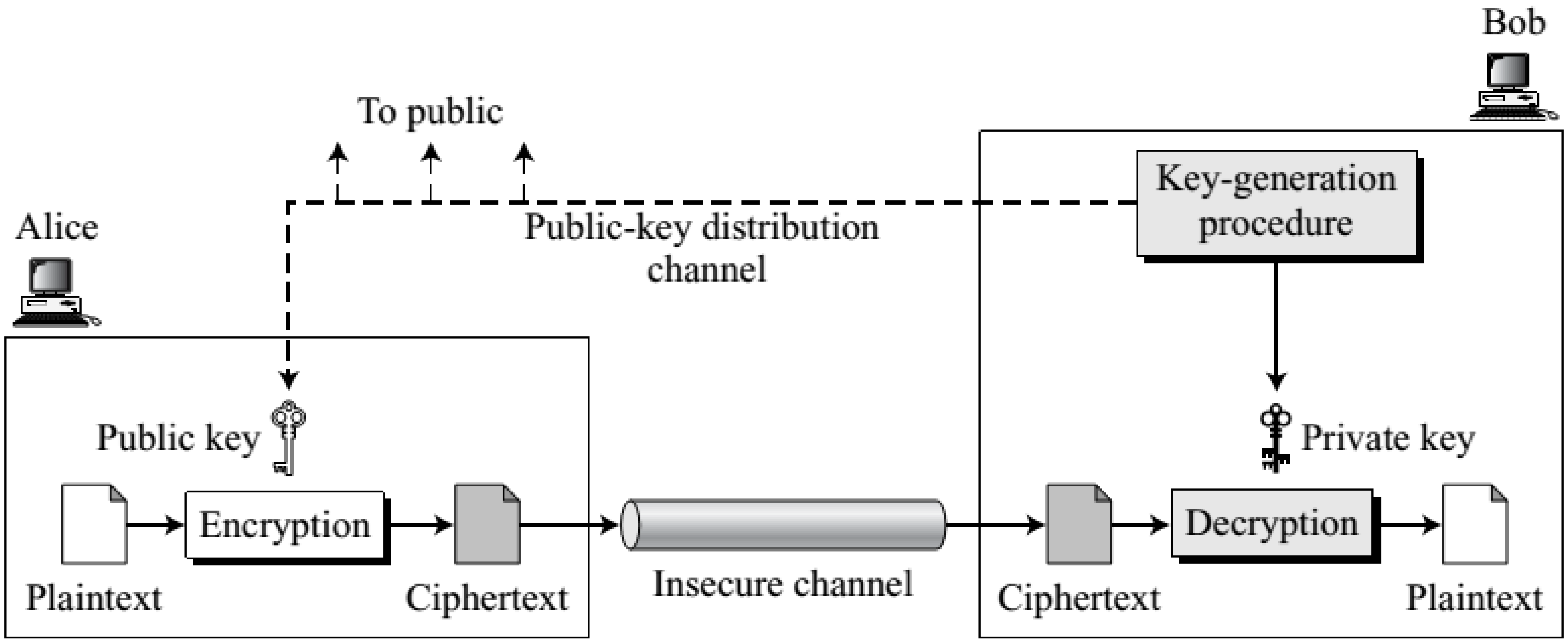
Asymmetric-Key Cryptography

- Locking and unlocking in asymmetric-key cryptosystem



Asymmetric-Key Cryptography

- General idea of asymmetric-key cryptosystem



Asymmetric-Key Cryptography

- Encryption/Decryption
  - The ciphertext can be thought of as $C = f(K_{public}, P)$;
  - The plaintext can be thought of as $P = g(K_{private}, C)$.
  - The function f is used only for encryption;
  - The function g is used only for decryption.
- Need for Both
  - Asymmetric-key cryptography is much slower than symmetric-key cryptography
  - Asymmetric-key cryptography is still needed for authentication, digital signatures, and secret-key exchanges.

Trapdoor One-Way Function

- The main idea behind asymmetric-key cryptography is the concept of the trapdoor oneway function.
- ***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:

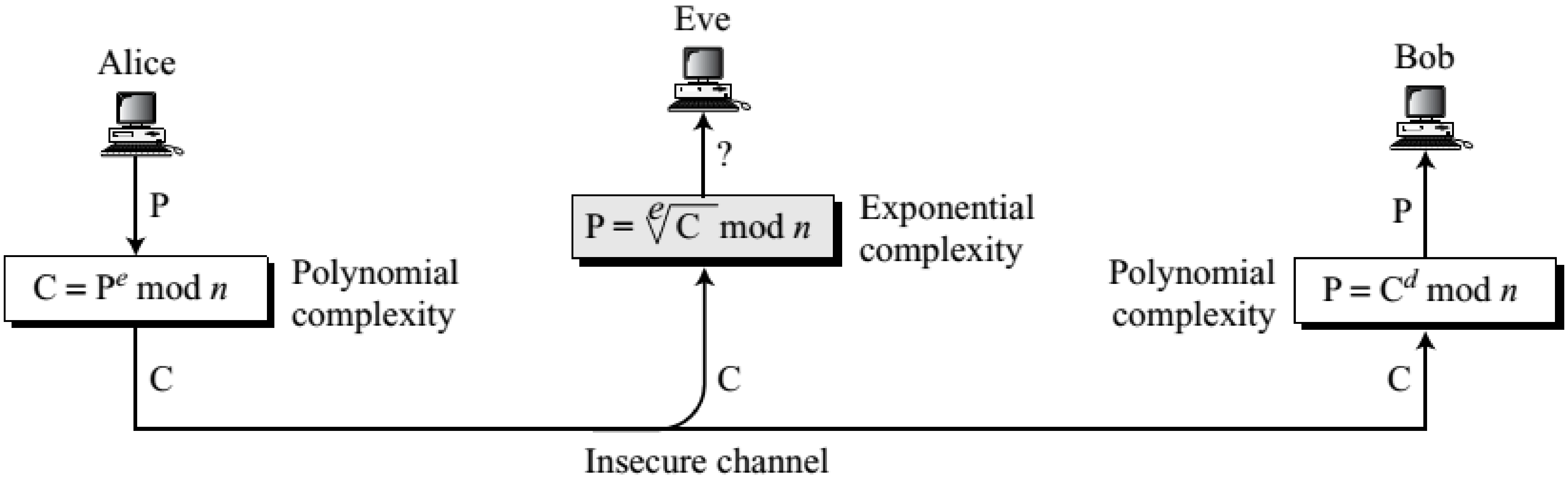
  3. Given y and a **trapdoor** (secret), x can be computed easily.

Trapdoor One-Way Function

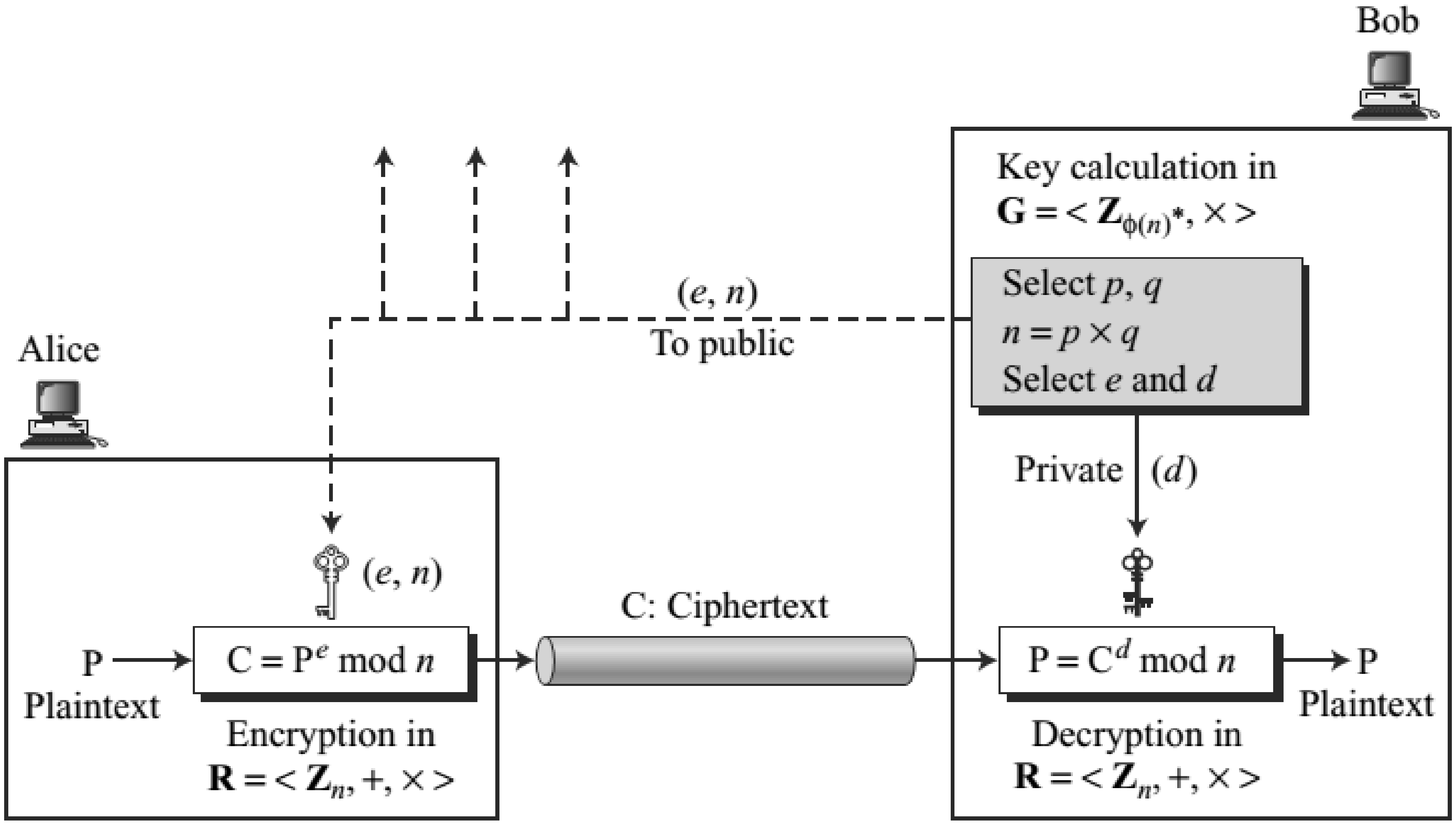
- When n is large, $n = p \times q$ is a one-way function.
- In this function x is a tuple (p, q) of two primes and y is n.
- Given p and q, it is always easy to calculate n; given n, it is very difficult to compute p and q.
- This is the factorization problem.
- There is not a polynomial time solution to the f^{-1} function in this case.
- 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
- Given y, k, and n, it is very difficult to calculate x.
- However, if we know the trapdoor, k' such that $k \times k' = 1 \bmod \varphi(n)$, we can use $x = y^{k'} \bmod n$ to find x.

RSA CRYPTOSYSTEM

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



Encryption, decryption, and key generation in RSA



RSA Key Generation

RSA_Key_Generation

{

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

$n \leftarrow p \times q$

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

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

}

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

Encryption and Decryption

```
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
}

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
}
```

Some Trivial Examples

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.

Some Trivial Examples

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

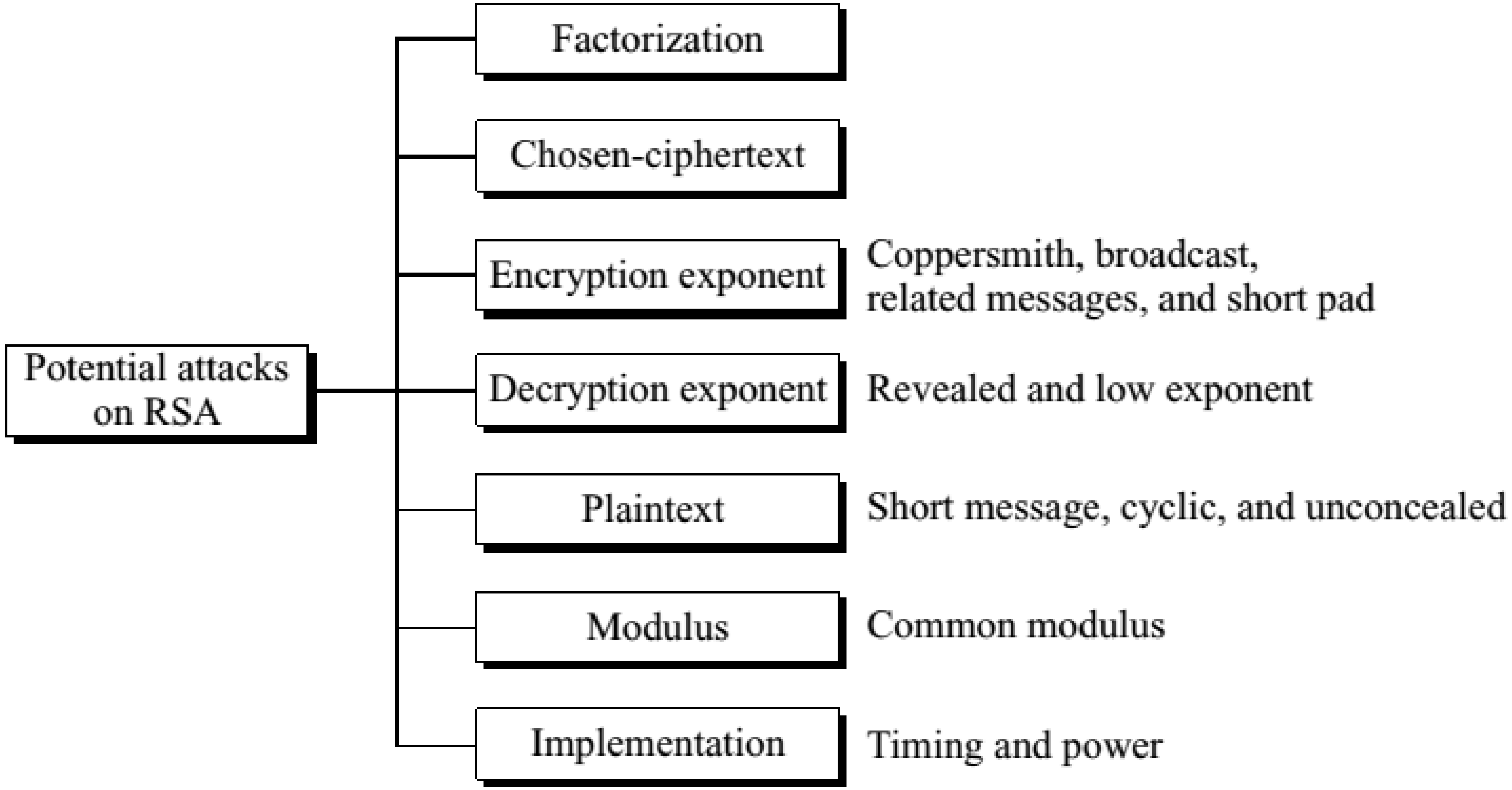
RSA

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

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).

Attacks on RSA

- No devastating attacks on RSA have been yet discovered.



Tutorial

Recommendations

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 (Tutorial)

Why modulus n must not be shared?

- 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 ϕ(n), and create a pair of exponents (ei, di) for each entity.
- Now assume Alice needs to send a message to Bob. The ciphertext to Bob is $C = P^{eB}$ mod n. Bob uses his private exponent, dB, to decrypt his message, $P = C^{dB}$ mod 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 dB.
- To thwart this type of attack, the modulus must not be shared.
- Each entity needs to calculate her or his own modulus.