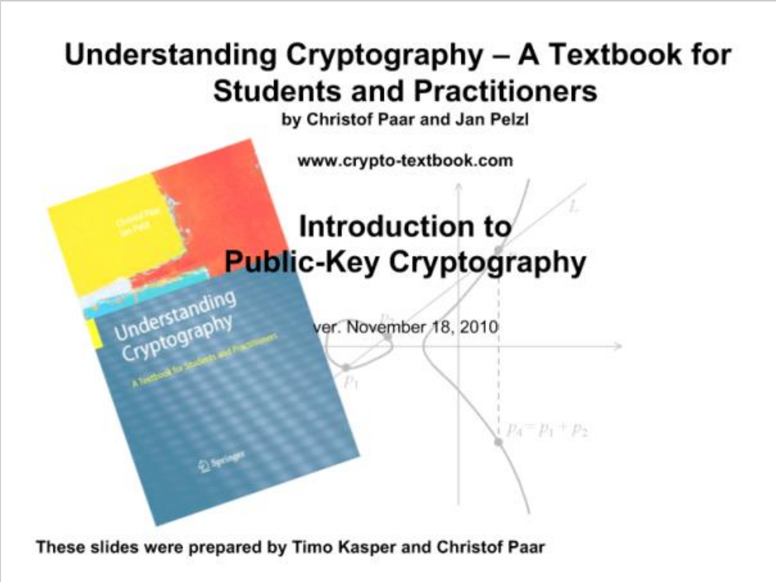
10 - Intro to public key crypto

2024年9月30日 星期一 11:10

10 - Intro to public key...



Some legal stuff (sorry): Terms of Use

Content of this Chapter

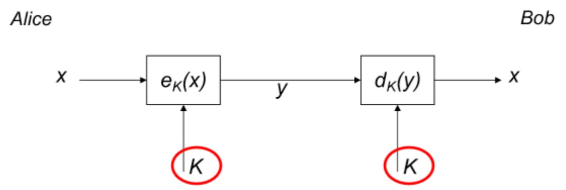
- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Content of this Chapter

- **Symmetric Cryptography Revisited**
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Symmetric Cryptography revisited

对称加密,



Two properties of symmetric (secret-key) crypto-systems:

- The **same secret key** ***K*** is used for encryption and decryption
- Encryption and Decryption are very similar (or even identical) functions

Symmetric Cryptography: Analogy

Safe with a strong lock, only Alice and Bob have a copy of the key

- Alice encrypts → locks message in the safe with her key
- Bob decrypts → uses his copy of the key to open the safe

优点

- **速度快**：对称加密算法通常比非对称加密算法（如 RSA）更快，因此在需要处理大量数据时更高效。
- **资源消耗低**：对称加密对计算资源的消耗较少，非常适合硬件和嵌入式系统。

缺点

- **密钥分发问题**：双方必须事先共享密钥，密钥的安全分发是一个挑战。
- **不支持不可否认性**：因为加密和解密使用相同的密钥，无法证明是谁发送了消息（无法防止发送方否认曾发送过消息）。

Symmetric Cryptography: Shortcomings

- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread **but**:

- Key distribution problem: The secret key must be **transported securely**
- Number of keys: In a network, each pair of users requires an individual key

→ *n* users in the network require $n \cdot (n-1)$ keys, each user stores *(n-1)* keys

防止发送方否认曾发送过消息）。

Symmetric Cryptography: Shortcomings

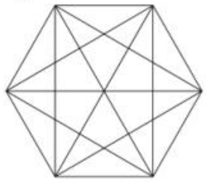
- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread **but**:

- Key distribution problem: The secret key must be **transported securely**
- Number of keys: In a network, each pair of users requires an individual key

→ *n* users in the network require $\frac{n \cdot (n-1)}{2}$ keys, each user stores *(n-1)* keys

Example:
6 users (nodes)

$\frac{6 \cdot 5}{2} = 15$ keys (edges)



hard to have million key
if there's 1000 users

- No **non-repudiation**: Alice or Bob can **cheat each other**, because they have identical keys.

Alice 和 Bob 都可以伪造消息并声称是对方发送的

Content of this Chapter

- Symmetric Cryptography Revisited
- **Principles of Asymmetric Cryptography**
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Idea behind Asymmetric Cryptography

New Idea:

Use the „good old mailbox" principle:

Everyone can drop a letter

But: Only the owner has the correct key to open the box

1976: first publication of such an algorithm by Whitfield Diffie and Martin Hellman,and also by Ralph Merkle.

Asymmetric (Public-Key) Cryptography

Principle: “Split up” the key

K

Public Key (K_{pub})
(Encrypt)

Secret Key (K_{pr})
(Decrypt)

→ During the key generation, a key pair K_{pub} and K_{pr} is computed

Asymmetric Cryptography: Analogy

Safe with public lock and private lock:

- Alice deposits (encrypts) a message with the - *not secret* - public key K_{pub}
- Only Bob has the - *secret* - private key K_{pr} to retrieve (decrypt) the message

Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- **Practical Aspects of Public-Key Cryptography**
- Important Public-Key Algorithms
- Essential Number Theory for Public-Key Algorithms

Basic Protocol for Public-Key Encryption

Alice	Bob
	$(K_{pubB}, K_{prB}) = K$
$\xleftarrow{K_{pubB}}$
x	
$y = e_{K_{pubB}}(x)$	$\xrightarrow{y}$
	$x = d_{K_{prB}}(y)$

→ Key Distribution Problem solved *

*) at least for now; public keys need to be authenticated, cf.Chptr. 13 *of Understanding Cryptogr.*

Security Mechanisms of Public-Key Cryptography

Here are main mechanisms that can be realized with asymmetric cryptography:

- **Key Distribution** (e.g., Diffie-Hellman key exchange, RSA) without a pre-shared secret (key)
- **Nonrepudiation and Digital Signatures** (e.g., RSA, DSA or ECDSA) to provide message integrity
- **Identification**, using challenge-response protocols with digital signatures
- **Encryption** (e.g., RSA / Elgamal)
  Disadvantage: Computationally very intensive
  (1000 times slower than symmetric Algorithms!)

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

Basic Key Transport Protocol 1/2

In practice: **Hybrid systems,** incorporating asymmetric and symmetric algorithms

1. **Key exchange** (for symmetric schemes) and **digital signatures** are performed with (slow) **asymmetric algorithms**
2. **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

Basic Key Transport Protocol 2/2

Example: Hybrid protocol with AES as the symmetric cipher

Alice	Bob
$(K_{pubB}, K_{prB}) = K$
$\xleftarrow{K_{pubB}}$

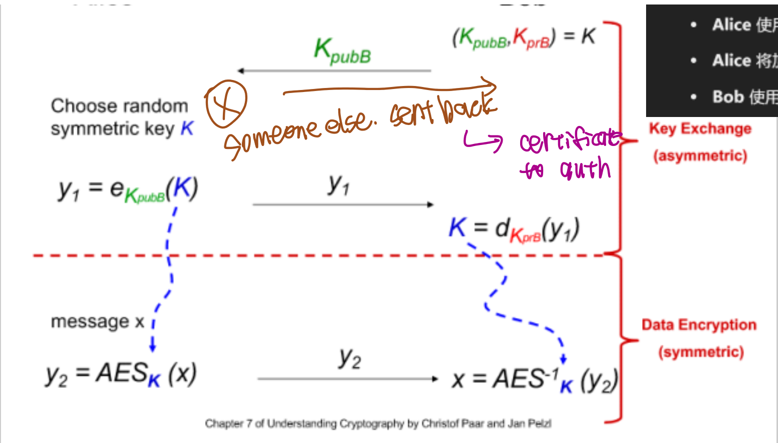
- **流程解析：**

  步骤1：密钥交换（非对称加密部分）

  - Alice 从随机数生成器中选择一个随机对称密钥 K。
  - Alice 使用 Bob 的公钥 K_{pubB} 加密该对称密钥，得到 $y_1 = e_{K_{pubB}}(K)$。
  - Alice 将加密后的密钥 y_1 发送给 Bob。
  - Bob 使用他的私钥 K_{prB} 解密 y_1，得到对称密钥 $K = d_{K_{prB}}(y_1)$。

  步骤2：数据加密（对称加密部分）

  - Alice 使用协商好的对称密钥 K 加密消息 x，得到密文 $y_2 = AES_K(x)$。
  - Alice 将加密后的数据 y_2 发送给 Bob。
  - Bob 使用相同的对称密钥 K 解密密文，得到原始消息 $x = AES_K^{-1}(y_2)$。



- **Alice** 使用协商好的对称密钥 K 加密消息 x，得到密文 $y_2 = AES_K(x)$。
- **Alice** 将加密后的数据 y_2 发送给 **Bob**。
- **Bob** 使用相同的对称密钥 K 解密密文，得到原始消息 $x = AES_K^{-1}(y_2)$。

Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- **Important Public-Key Algorithms**
- Essential Number Theory for Public-Key Algorithms

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

How to build Public-Key Algorithms

Asymmetric schemes are based on a **„one-way function“ *f()***:

- Computing $y = f(x)$ is computationally easy
- Computing $x = f^{-1}(y)$ is computationally infeasible

[hash]
no key.

One way functions are based on **mathematically hard problems**.

Three main families:

- **Factoring integers** (RSA, ...):
  数学难题支撑公钥算法的安全性，主要有以下三类
  Given a composite integer n, find its prime factors
  (Multiply two primes: easy)

- **Discrete Logarithm** (Diffie-Hellman, Elgamal, DSA, …):
  Given a, y and m, find x such that $a^x = y \bmod m$
  (Exponentiation a^x: easy)

- **Elliptic Curves (EC)** (ECDH, ECDSA): Generalization of discrete logarithm

Note: The problems are considered mathematically hard, but no proof exists (so far).

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

Key Lengths and Security Levels

x2

Symmetric	*ECC*	*RSA, DL*	*Remark*
64 Bit	128 Bit key	≈ 700 Bit	Only short term security (a few hours or days)
80 Bit	160 Bit	≈ 1024 Bit	Medium security (except attacks from big governmental institutions etc.)
128 Bit	256 Bit	≈ 3072 Bit	Long term security (without quantum computers)

- The exact complexity of RSA (factoring) and DL (Index-Calculus) is difficult to estimate
- The existence of quantum computers would probably be the end for ECC, RSA & DL.

Content of this Chapter

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- **Essential Number Theory for Public-Key Algorithms**

Euclidean Algorithm 1/3

最大公约數

- Compute the **greatest common divisor *gcd* (r_0, r_1)** of two integers r_0 and r_1

- gcd is **easy for small numbers**:
  1. factor r_0 and r_1
  2. gcd = highest common factor

- Example:

  $r_0 = 84 = 2 \cdot 2 \cdot 3 \cdot 7$

  $r_1 = 30 = 2 \cdot 3 \cdot 5$

  big num. x (factor)

  → The gcd is the product of all common prime factors:
  $2 \cdot 3 = 6 = gcd\ (30,84)$

- **But:** Factoring is complicated (and often infeasible) for large numbers

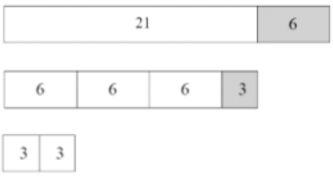
Euclidean Algorithm 2/3

- Observation: $gcd(r_0, r_1) = gcd(r_0 - r_1, r_1) = gcd(r_0 \ mod \ r_1, r_1)$
- Core idea:
  - Reduce the problem of finding the gcd of two given numbers to that of the **gcd of two smaller numbers**
  - Repeat process recursively
  - The final $gcd(r_i, 0) = r_i$ is the answer to the original problem !

直到其中一个数变为0，此时另一个数就是gcd.

Euclidean Algorithm 3/3

- **Example:** $\gcd(r_0, r_1)$ for $r_0 = 27$ and $r_1 = 21$



$\gcd(27, 21) = \gcd(1 \cdot 21 + 6, 21) = \gcd(21, 6)$

3·6
21=18+3

$\gcd(21, 6) = \gcd(3 \cdot 6 + 3, 6) = \gcd(6, 3)$

$\gcd(6, 3) = \gcd(2 \cdot 3 + 0, 3) = \gcd(3, 0) = 3$

- Note: very efficient method even for long numbers:
  The complexity grows **linearly** with the number of bits

For the full Euclidean Algorithm see Chapter 6 in *Understanding Cryptography*.

Extended Euclidean Algorithm

$r_1^{-1} \bmod r_0$

- Extend the Euclidean algorithm to **find multiplicative inverse** of r_1 mod r_0
- EEA computes *s*,*t*, and the gcd : $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$.
- To compute the inverse $r_1^{-1} \ mod \ r_0$, apply EEA to find s, t such that:

$$\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1 = 1$$

- Since $s \cdot r_0 + t \cdot r_1 = 1$, by the definition of equivalence modulo r_0:

$$t \cdot r_1 \equiv 1 \ mod \ r_0$$

- That is: t is the multiplicative inverse of r_1 modulo r_0.
- Note that if $\gcd(r_0, r_1) \neq 1$ then the inverse does not exist.
- Section 6.3.2 in *Understanding Cryptography* for more details.

Euler's Phi Function 1/2

- *New problem, important for public-key systems, e.g., RSA:*
  Given the set of the *m* integers {0, 1, 2, ..., *m* -1},
  How many numbers in the set are **relatively prime to *m*** ? 有多少个数与m互质?

互质 → gcd(x, y)=1

如: 1 2 3 4 (for 5)

- Answer: **Euler's Phi function *Φ(m)***

- **Example** for the sets {0,1,2,3,4,5} (*m*=6), and {0,1,2,3,4} (*m*=5)

$$\begin{aligned} \gcd(0,6) &= 6 \\ \gcd(1,6) &= 1 \leftarrow \\ \gcd(2,6) &= 2 \\ \gcd(3,6) &= 3 \\ \gcd(4,6) &= 2 \\ \gcd(5,6) &= 1 \leftarrow \end{aligned}$$

$$\begin{aligned} \gcd(0,5) &= 5 \\ \gcd(1,5) &= 1 \leftarrow \\ \gcd(2,5) &= 1 \leftarrow \\ \gcd(3,5) &= 1 \leftarrow \\ \gcd(4,5) &= 1 \leftarrow \end{aligned}$$

→ 1 and 5 relatively prime to *m*=6, hence **Φ(6) = 2**

→ **Φ(5) = 4**

- Testing one gcd per number in the set is **extremely slow for large *m*.**

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

Euler's Phi Function 2/2

分解成因数.

- **If** canonical factorization of *m* known: (where p_i primes and e_i positive integers)

$$m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$$

- **then** calculate Phi according to the relation

$$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$$

- Phi especially easy for e_i = 1, e.g., *m* = *p* · *q* → *Φ(m)* = (*p*-1) · (*q*-1)

- **Example** *m* = 899 = 29 · 31:
  Φ(899) = (29-1) · (31-1) = 28 · 30 = **840**

- **Note:** Finding *Φ(m)* is computationally easy **if factorization of *m* is known**
  (otherwise the calculation of *Φ(m)* becomes computationally infeasible for large numbers)

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

Fermat's Little Theorem

- Given a **prime *p*** and an **integer *a***: $\boxed{a^p \equiv a \pmod{p}}$
- Can be rewritten as $a^{p-1} \equiv 1 \pmod{p}$

- **Use: Find modular inverse, if p is prime.** Rewrite to $a \cdot a^{p-2} \equiv 1 \pmod{p}$
- Comparing with definition of the modular inverse $a \cdot a^{-1} \equiv 1 \bmod m$
  → $a^{-1} \equiv a^{p-2} \pmod{p}$ is the modular inverse modulo a prime *p*

Example: *a* = 2, *p* = 7

$$a^{p-2} = 2^5 = 32 \equiv 4 \bmod 7$$

$$\text{verify: } 2 \cdot 4 \equiv 1 \bmod 7 \checkmark$$

- Fermat's Little Theorem works only **modulo a prime *p***

Chapter 7 of Understanding Cryptography by Christof Paar and Jan Pelzl

RSA（Rivest-Shamir-Adleman）是一种基于 **非对称加密** 的密码算法

RSA算法步骤

1. 生成密钥对

- 选择两个 **大质数** p 和 q，确保它们足够大且随机。
- 计算 $n = p \times q$，这是模数。
- 计算欧拉函数 $\Phi(n) = (p-1) \times (q-1)$。
- 选择一个整数 e（通常为 65537，因为它是一个常用的值，计算效率高），使得 $1 < e < \Phi(n)$，且 e 与 $\Phi(n)$ 互质。
- 计算 d，即 **私钥**，使得：

$$d \times e \equiv 1 \pmod{\Phi(n)}$$

这可以通过 **扩展欧几里得算法** 来求解。

- **公钥**：(e, n)
- **私钥**：(d, n)

2. 加密

- 给定消息 M，首先将其转化为整数 m（通常通过字符编码），且 $0 < m < n$。
- 使用公钥 (e, n) 进行加密：

$$c = m^e \mod n$$

得到密文 c。

3. 解密

- 使用私钥 (d, n) 进行解密：

$$m = c^d \mod n$$

恢复出原始消息 m。

Euler's Theorem

- Generalization of Fermat's little theorem to **any integer modulus**
- Given two **relatively prime integers *a* and *m*** : $a^{\Phi(m)} \equiv 1 \pmod{m}$
- **Example**: m=12, a=5
  1. Calculate Euler's Phi Function

     $\Phi(12) = \Phi(2^2 \cdot 3) = (2^2 - 2^1)(3^1 - 3^0) = (4-2)(3-1) = 4$
  2. Verify Euler's Theorem

     $5^{\Phi(12)} = 5^4 = 25^2 = 625 \equiv 1 \bmod 12$
- Fermat's little theorem = special case of Euler's Theorem
- for a prime **p**: $\Phi(p) = (p^1 - p^0) = p - 1$

  → Fermat: $a^{\Phi(p)} = a^{p-1} \equiv 1 \pmod{p}$

Lessons Learned

- Public-key algorithms have **capabilities that symmetric ciphers don't have**, in particular digital signature and key establishment functions.

公钥算法计算开销大（也可以理解为计算速度慢）不适合大量数据加密更多用于密钥交换或认证等任务，而不是直接加密大数据。

- Public-key algorithms are **computationally intensive** (a nice way of saying that they are *slow*), and hence are poorly suited for bulk data encryption.

2probs dis alg. discript.

- Only **three families of public-key schemes** are widely used. This is considerably fewer than in the case of symmetric algorithms.
- The **extended Euclidean algorithm** allows us to compute **modular inverses** quickly, which is important for almost all public-key schemes.
- **Euler's phi function** gives us the number of elements smaller than an integer n that are relatively prime to n. This is important for the RSA crypto scheme.

The RSA Crytptosystem

The RSA Cryptosystem

- Martin Hellman and Whitfield Diffie published their landmark public-key paper in 1976
- Ronald Rivest, Adi Shamir and Leonard Adleman proposed the asymmetric RSA cryptosystem in1977
- Until now, RSA is the most widely use asymmetric cryptosystem although elliptic curve cryptography (ECC) becomes increasingly popular
- RSA is mainly used for two applications
  - Transport of (i.e., symmetric) keys (cf. Chptr 13 of *Understanding Cryptography*)
  - Digital signatures (cf. Chptr 10 of *Understanding Cryptography*)

Encryption and Decryption

1. 公钥 (Public Key):
   - 公钥由一对数 (n, e) 组成。
   - 其中 $n = p \times q$，e 是加密指数。
   - 公钥 $k_{\text{pub}} = (n, e)$。
2. 私钥 (Private Key):
   - 私钥为一个数 d，即解密指数。
   - 私钥 $k_{\text{pr}} = d$。

整数环 Z_n

- RSA operations are done over the integer ring Z_n (i.e., arithmetic modulo n), where $n = p * q$, with p, q being large primes
- Encryption and decryption are simply exponentiations in the ring

Definition

Given the public key $(n,e) = k_{pub}$ and the private key $d = k_{pr}$ we write

$$y = e_{k_{pub}}(x) \equiv x^e \bmod n$$

$$x = d_{k_{pr}}(y) \equiv y^d \bmod n$$

x 明文 y 密文.

where $x, y \in Z_n$.

We call $e_{k_{pub}}()$ the encryption and $d_{k_{pr}}()$ the decryption operation.

- In practice x, y, n and d are very long integer numbers (≥ 1024 bits)
- The security of the scheme relies on the fact that it is hard to derive the "private exponent" d given the public-key (n, e)

从 public-key (n,e) 中推导出 d 的困难性

私钥

一、RSA 密钥生成流程

RSA 加密算法包括公钥和私钥的生成，具体步骤如下：

1. **选择两个大质数 p 和 q：**
   - 这两个质数应足够大且不同，以保证安全性。
2. **计算 $n = p \times q$：**
   - n 将用于公钥和私钥。
3. **计算欧拉函数 $\Phi(n) = (p-1) \times (q-1)$：**
   - 欧拉函数用于确定加密指数 e 和解密指数 d。
4. **选择加密指数 e：**
   - e 满足 $1 < e < \Phi(n)$，并且 $\gcd(e, \Phi(n)) = 1$，即 e 与 $\Phi(n)$ 互质。
   - 确保 e 有乘法逆元 d。
5. **计算私钥指数 d：**
   - d 是 e 的逆元，使得 $d \times e \equiv 1 \pmod{\Phi(n)}$。
   - 这意味着 $d \times e$ 除以 $\Phi(n)$ 的余数为 1。
6. **返回公钥和私钥：**
   - 公钥 $k_{\text{pub}} = (n, e)$
   - 私钥 $k_{\text{pr}} = d$

Key Generation

- Like all asymmetric schemes, RSA has set-up phase during which the private and public keys are computed

Algorithm: RSA Key Generation

Output: public key: $k_{pub} = (n, e)$ and private key $k_{pr} = d$

1. Choose two large primes p, q
2. Compute $n = p * q$
3. Compute $\Phi(n) = (p-1) * (q-1)$
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n)-1\}$ such that $\gcd(e, \Phi(n)) = 1$ e 满足与 φ(n) 互素.
5. Compute the private key d such that $d * e \equiv 1 \bmod \Phi(n)$ ⇒ d 是 e 的模 φ(n) 的乘法逆元
6. **RETURN** $k_{pub} = (n, e)$, $k_{pr} = d$

Remarks:

- Choosing two large, distinct primes p, q (in Step 1) is non-trivial
- $\gcd(e, \Phi(n)) = 1$ ensures that e has an inverse and, thus, that there is always a private key d

Example: RSA with small numbers

ALICE

Message $\boldsymbol{x = 4}$

BOB

1. Choose $p = 3$ and $q = 11$
2. Compute $n = p * q = 33$
3. $\Phi(n) = (3-1) * (11-1) = 20$
4. Choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$

$\xleftarrow{K_{pub} = (33,3)}$

$y = x^e \equiv 4^3 \equiv 31 \bmod 33$

64

Alice 加密消息

$\xrightarrow{y = 31}$

$y^d = 31^7 \equiv \boldsymbol{4 = x} \bmod 33$

Implementation aspects

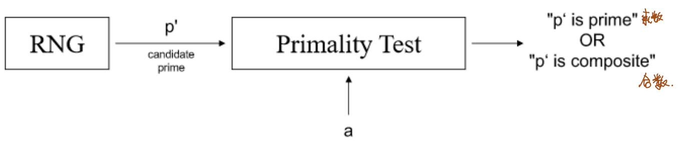
- The RSA cryptosystem uses only one arithmetic operation (modular exponentiation) which makes it conceptually a simple asymmetric scheme
- Even though conceptually simple, due to the use of very long numbers, RSA is orders of magnitude slower than symmetric schemes, e.g., DES, AES
- When implementing RSA (esp. on a constrained device such as smartcards or cell phones) close attention has to be paid to the correct choice of arithmetic algorithms
- The square-and-multiply algorithm allows fast exponentiation, even with very long numbers.

实现方面（Implementation aspects）

1. **RSA 密码系统的操作**：RSA 密码系统只使用一种算术运算，即模幂运算（modular exponentiation），这使其成为一个概念上简单的非对称加密方案。
2. **性能问题**：尽管 RSA 的概念比较简单，由于它使用了非常长的数字，RSA 的速度远比对称加密方案（如 DES、AES）慢得多。
3. **资源受限设备中的 RSA 实现**：在实现 RSA（尤其是在诸如智能卡或手机等受限设备上）时，需要特别注意选择正确的算术算法，以确保其高效执行。
4. **平方-乘法算法（Square-and-Multiply）**：为了加速模幂运算，平方-乘法算法允许快速计算，即使是非常长的数字。

Finding Large Primes

- Generating keys for RSA requires finding two large primes p and q such that $n = p * q$ is sufficiently large 如果 n 是 2048 位，则 p 和 q 应该各为 1024 位
- The size of p and q is typically half the size of the desired size of n
- To find primes, random integers are generated and tested for primality:



- The random number generator (RNG) should be non-predictable otherwise an attacker could guess the factorization of n

n的因数

Primality Tests

- Factoring *p* and *q* to test for primality is typically not feasible
- However, we are not interested in the factorization, we only want to know whether *p* and *q* are composite
- Typical primality tests are probabilistic, i.e., they are not 100% accurate but their output is correct with very high probability
- A probabilistic test on a number p has two outputs:
  - “p is composite” – always true
  - “p is a prime” – only true with a certain probability
- Among the well-known primality tests are the following
  - Fermat Primality-Test
  - Miller-Rabin Primality-Test

- **因数分解测试素性不现实**：对 p 和 q 进行因数分解以测试其是否为素数在大多数情况下是不可行的，尤其是在很大数的情况下。
- **不关心因数分解，只关心合数性**：在实际应用中，我们并不需要知道 p 和 q 的完整因数分解，只需要知道它们是否为合数（也就是说，它们是否可以被其他数整除）。
- **典型的素性测试是概率性的**：大多数素性测试是概率性的，这意味着它们并不能百分之百准确，但其结果在极高的概率下是正确的。
- **概率素性测试的两个输出**：
  - **“p 是合数”**：这个结果是确定的，即如果输出表明 p 是合数，那么 p 绝对是合数。
  - **“p 是素数”**：这个结果仅在一定概率下是正确的，意味着测试可能会错误地认为一个合数是素数。
- **著名的素性测试**：
  - **费马素性测试（Fermat Primality-Test）**：这是基于费马小定理的概率性素性测试。
  - **米勒-拉宾素性测试（Miller-Rabin Primality-Test）**：这是一种更为广泛使用且更可靠的素性测试，特别是在大数的情况下。

Attacks and Countermeasures 1/3

- There are two distinct types of attacks on cryptosystems
  - Analytical attacks try to break the mathematical structure of the underlying problem of RSA 即通过因式分解 nnn 来获取 Φ (n) 或私钥 d
  - Implementation attacks try to attack a real-world implementation by exploiting inherent weaknesses in the way RSA is realized in software or hardware (eg, 缓存时间功耗等物理特性来推断)

Attacks and Countermeasures 2/3

RSA is typically exposed to these analytical attack vectors

- **Mathematical attacks**
  - The best known attack is factoring of *n* in order to obtain $\Phi(n)$
  - Can be prevented using a sufficiently large modulus *n*
  - The current factoring record is 664 bits. Thus, it is recommended that *n* should have a bit length between 1024 and 3072 bits
- **Protocol attacks** 协议缺陷.
  - Exploit the malleability of RSA, i.e., the property that a ciphertext can be transformed into another ciphertext which decrypts to a related plaintext – without knowing the private key
  - Can be prevented by proper padding

- 该攻击利用了 RSA 的**可塑性特性**，即攻击者可以在不知私钥的情况下，通过修改密文使其解密为相关的明文。
- **防御措施**：
  - 采用**合适的填充（Padding）机制**，如 OAEP（Optimal Asymmetric Encryption Padding），可以防止这种攻击。

Attacks and Countermeasures 3/3

- Implementation attacks can be one of the following
  - **Side-channel analysis**
    - Exploit physical leakage of RSA implementation (e.g., power consumption, EM emanation, etc.)

电磁辐射.

  - **Fault-injection attacks**

中国剩余定理.

    - Inducing faults in the device while CRT is executed can lead to a complete leakage of the private key

More on all attacks can be found in Section 7.8 of *Understanding Cryptography*

Lessons Learned

- RSA is the most widely used public-key cryptosystem
- RSA is mainly used for key transport and digital signatures
- The public key *e* can be a short integer, the private key *d* needs to have the full length of the modulus *n*
- RSA relies on the fact that it is hard to factorize *n*
- Currently 1024-bit cannot be factored, but progress in factorization could bring this into reach within 10-15 years. Hence, RSA with a 2048 or 3076 bit modulus should be used for long-term security
- A naïve implementation of RSA allows several attacks, and in practice RSA should be used together with padding

- **RSA是使用最广泛的公钥加密系统**：
  - RSA广泛用于密钥传输和数字签名。
- **公钥和私钥的不同长度**：
  - 公钥e可以是一个短整数，而私钥d需要有与模数n一样的全长。
- **RSA依赖于大数分解的困难**：
  - RSA的安全性基于将大数n分解为两个素数的难度。
- **未来可能面临的分解风险**：
  - 虽然当前1024位的模数尚无法被分解，但随着分解算法的进展，10到15年内可能会变得可行。因此，为了长期安全，建议使用2048或3076位的模数。
- **RSA的基本实现容易遭受攻击**：
  - 简单实现的RSA存在多种攻击风险，因此在实际应用中，RSA应与填充方案一起使用，以增加安全性。