信息安全（04）

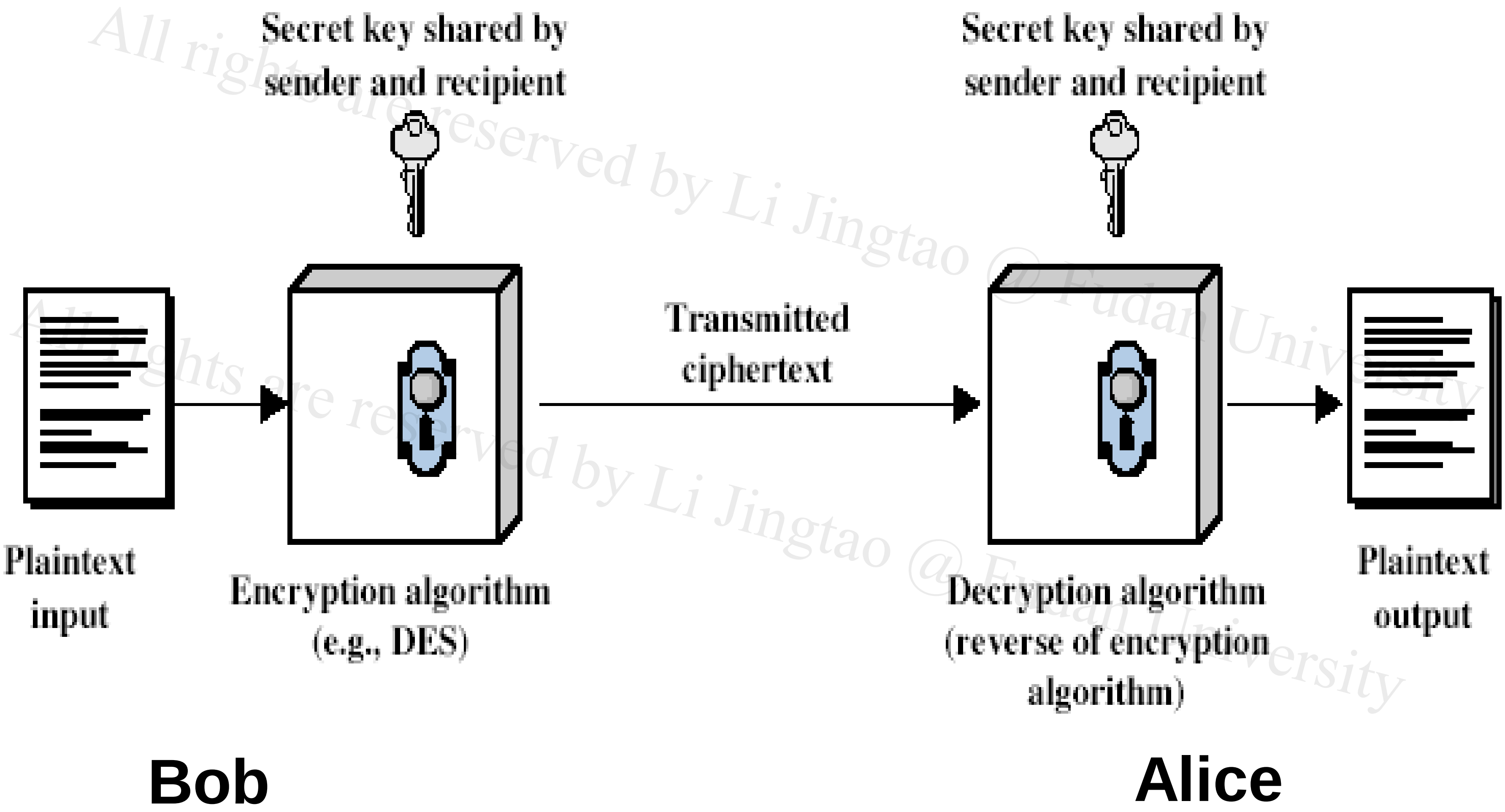
Introduction to Cryptography

- Public Key Cryptography, RSA

Public Key Cryptography

- Principles of Public-Key Cryptosystems
- The RSA Algorithm

Review: Symmetric Cipher Model

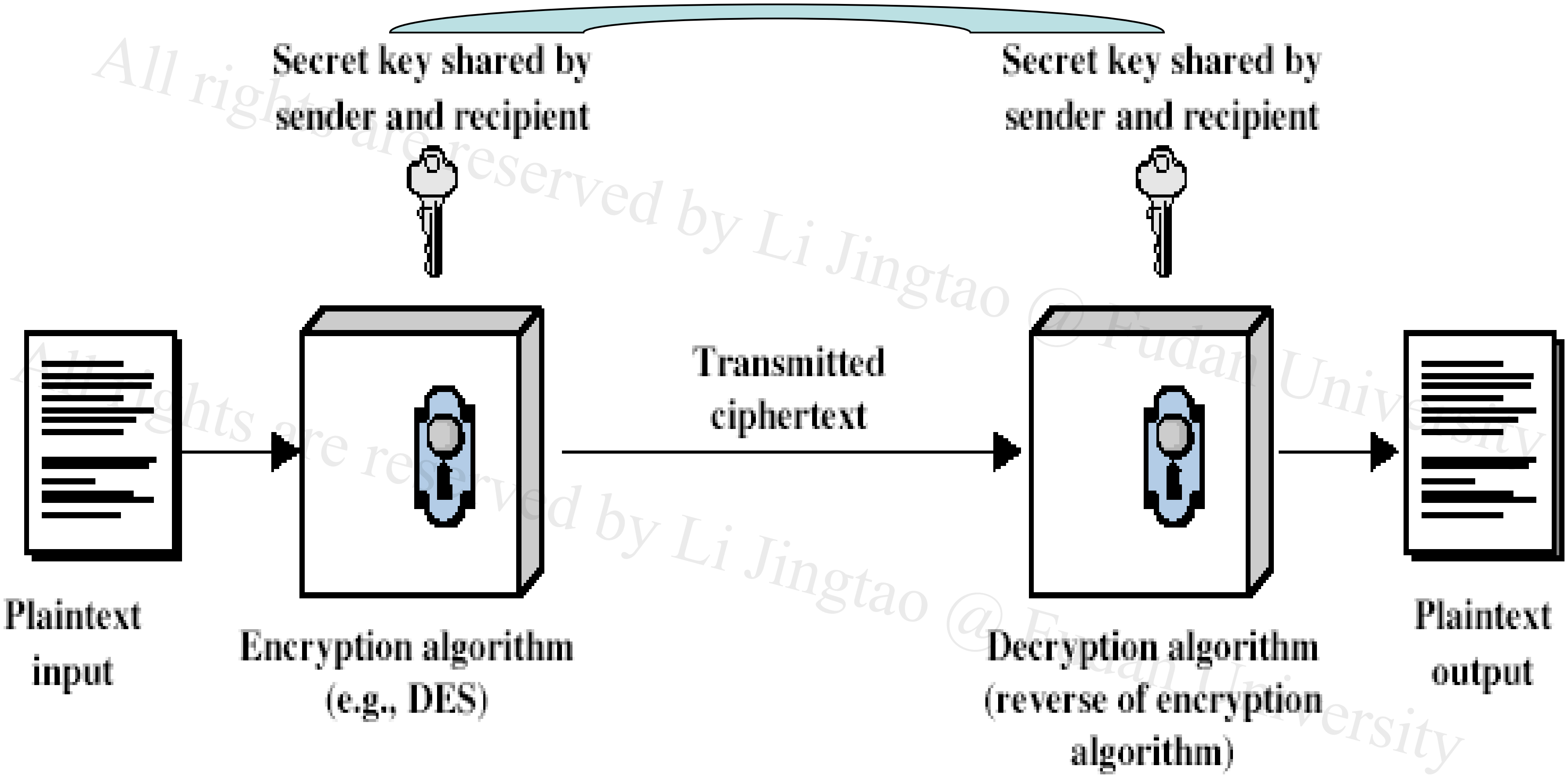


Symmetric Cryptography

- traditional **symmetric/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal

Review: Symmetric Cipher Model

Secure channel



Bob **Alice**

Asymmetric Cipher Model

- Every body have two keys
  - Public key —— 公开
  - Private key —— 保密

Bob   **Alice**

Asymmetric Cipher Model

公开

- Bob's Public key Alice's Public key

- Bob's Private key | Alice's Private key

Bob **Alice**

Asymmetric Cipher Model

公开

- Bob's Public key

Alice's Public key

保密

- Bob's Private key

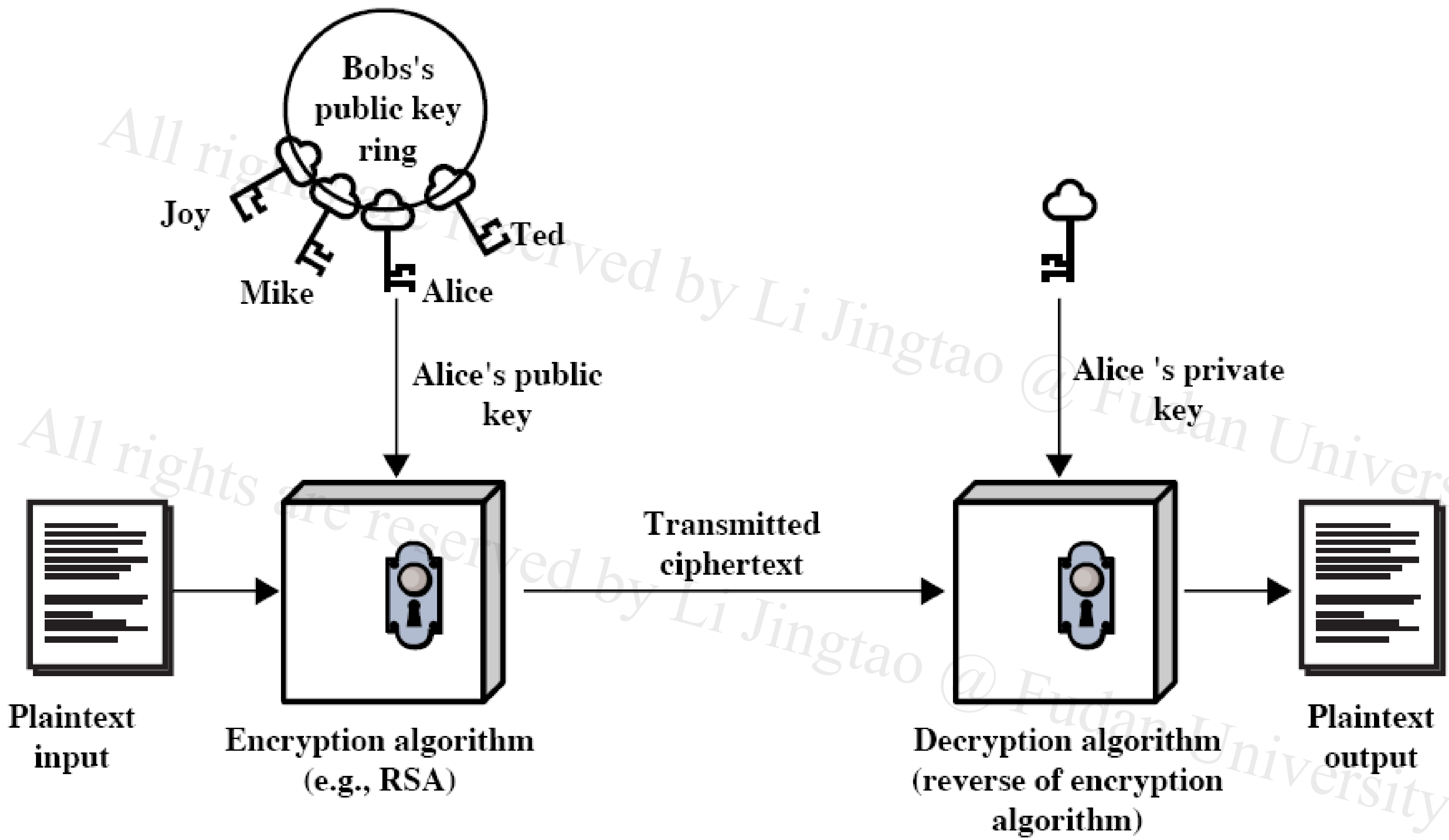
Bob

保密

Alice's Private key

Alice

Asymmetric Cipher Model



Bob **Alice**

Cryptography Catalog

- **The number of the keys used**
  - **Symmetric** , **single-key**, **secret-key**, **conventional encryption**: Both sender and receiver use the **same** key
  - **Asymmetric**, **two-key**, or **public-key encryption**: the sender and receive each uses a **different** key

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

History

- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  – known earlier in classified community.
  – note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- Diffie-Hellman Key Exchange

公开密钥加密系统

- 一个公开密钥系统由六要素组成：
  - 明文
  - 公开和私有密钥
  - 加密算法
  - 密文
  - 解密算法

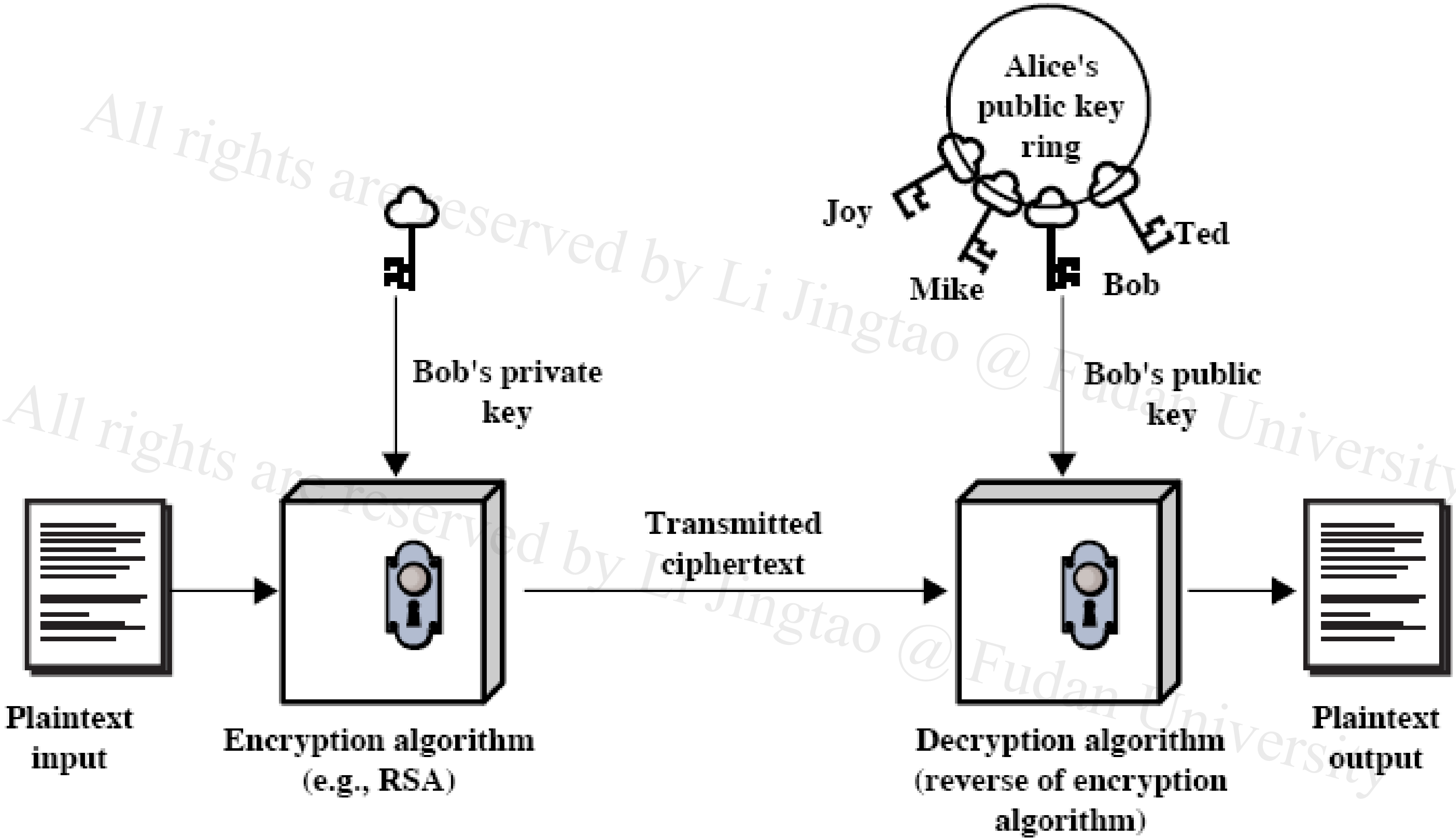
公开密钥加密

- 参与方B容易通过计算产生出一对密钥（公开密钥KU_b，私有密钥KR_b）
- 发送方A很容易计算产生密文 $C = E_{KUb}(M)$
- 接收方B通过计算解密密文 $M = D_{KRb}(C) = D_{KRb}[E_{KUb}(M)]$
- 敌对方即使知道公开密钥KU_b，要确定私有密钥KR_b在计算上是不可行的
- 敌对方即使知道公开密钥KU_b和密文C，要确定明文M在计算上是不可行的
- 密码对互相之间可以交换使用 $M = D_{KRb}[E_{KUb}(M)] = D_{KUb}[E_{KRb}(M)]$

Public-Key Cryptography: The progress

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender (Authentication)
- protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography: The progress



(b) Authentication

Public-Key Cryptography: The progress

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

RSA Algorithm

- 1977年由MIT的Rivest，Shamir和Adleman 三人提出
- 是一个分组加密方法
- 目前被最广泛地采用
- 采用的单向函数是大素数相乘，相乘很容易，但因子分解很困难
- 基于数论中的Fermat（小）定理实现

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random: p, q
- computing their system modulus n=p.q
  - note ø(n)=(p-1)(q-1)
- selecting at random the e

  where 1<e<ø(n), gcd(e,ø(n))=1
- solve following equation to find the d

  e.d=1 mod ø(n) and 0≤d≤n
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

当e＝1001， ø(n)＝3837时

方程为 x * 1001 = 1 (mod 3837)

求解过程:

3837 = 3 * 1001 + 834
1001 = 1 * 834 + 167
834 = 4 * 167 + 166
167 = 166 + 1

所以
1 = 167 - 166
= 167 - (834 - 4 * 167)
= 5 * 167 - 834
= 5 *(1001 - 834) - 834
= 5 * 1001 - 6 *834
= 5 * 1001 - 6 * (3837 -3 *1001)
= 23 * 1001 - 6 *3837

```
program InvEuclid;
const
``dim=100; {K深度，大约与数据大小成ln(x)的关系，懒得算了，胡乱写的}
var
``K:array[1..dim] of longint; {储存历次商}
``a,b:longint;
``n,i:integer;
procedure swap(var a:longint;var b:longint);
var
``tmp:longint;
begin
``tmp:=a;
``a:=b;
``b:=tmp;
end;
begin
``readln(a,b);
``if a<b then swap(a,b);
``{主算法开始}
``n:=0;
``repeat {Euclid辗转相除算法}
````inc(n);
````K[n]:=a div b;
````a:=a mod b;
````swap(a,b);
``until b<2;
``if b>0 then begin {b=0则无解}
````{逆推过程}
````a:=1; b:=K[n];
````for i:=n-1 downto 1 do begin
``````inc(a,b*k[i]);
``````swap(a,b);
````end;
````writeln(b);
``end else
````writeln('No Solution!');
``{主算法结束}
end.
```

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient PU={e,n}
  - computes: $C = M^e \bmod n$, where $0 \le M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key PR={d,n}
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

Why RSA Works

- because of Euler's Theorem:
  - $a^{\varnothing(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\varnothing(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses mod $\varnothing(n)$
  - hence $e.d=1+k.\varnothing(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.\varnothing(n)} = M^1.(M^{\varnothing(n)})^k$$

$$= M^1.(1)^k = M^1 = M \bmod n$$

RSA Example - Key Setup

1. Select primes: p=17 & q=11
2. Compute $n = pq$ =17 x 11=187
3. Compute ø(n)=(p–1)(q-1)=16 x 10=160
4. Select e: gcd(e,160)=1; choose e=7
5. Determine d: de=1 mod 160 and d < 160
   Value is d=23 since 23x7=161= 10x160+1
6. Publish public key PU={7,187}
7. Keep secret private key PR={23,187}

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message `M = 88` (nb. `88<187`)
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

  $M = 11^{23} \bmod 187 = 88$

Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Computational Aspects

- Encryption and Decryption
  - Both require *modular exponentiation*
  - Can use the following efficient algorithm to compute a^b mod n
  - Square and multiply

```
Modular-Exponentiation(a, b, n)
1.   d ← 1
2.   let b_k b_{k-1}…b_0 be the binary representation of b
3.   for i ← k downto 0 do
4.       d ← (d × d) mod n
5.       if b_i = 1 then d ← (d × a) mod n
6.   return d
```


- Key Generation
  - Determining two *prime* numbers, p and q (Miller-Rabin Test)
  - Selecting either e or d and calculating the other (Extended Euclid)

Efficient Encryption/Decryption

- encryption uses exponentiation to power e
- hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- but if e too small (eg e=3) can attack
- decryption uses exponentiation to power d
  - this is likely large, insecure if not

RSA Key Generation

- users of RSA must:
  - determine two primes at random: `p, q`
  - select either `e` or `d` and compute the other
- primes `p,q` must not be easily derived from modulus `n=p.q`
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents `e, d` are inverses, so use Inverse algorithm to compute the other

RSA Security

- possible approaches to attacking RSA are:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ø(n), by factoring modulus n)
  - timing attacks (on running of decryption)
  - chosen ciphertext attacks (given properties of RSA)

Factoring Problem

- mathematical approach takes 3 forms:
  - factor n=p.q, hence compute ø(n) and then d
  - determine ø(n) directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - cf QS to GHFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

Factoring

- For a large n with large prime factors, factoring is a hard problem - $O(e^{\sqrt{\ln(n)\ln(\ln(n))}})$
- RSA factoring challenge
  - **Sponsored by RSA Labs.**
  - **To encourage research into computational number theory and the practical difficulty factoring large integers**
  - **A cash prize is awarded to the first person to factor each challenge number**

Progress in Factorization

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

RSA Factoring Challenge

Numbers are designated "RSA-XXXX", where XXXX is the number's length in bits

Challenge Number	Prize ($US)	Status
RSA-576 (174 Digits)	$10,000	Factored (Dec 2003)
RSA-640 (193 Digits)	$20,000	Factored (Nov 2005)
RSA-704 (212 Digits)	$30,000	Not Factored
RSA-768 (232 Digits)	$50,000	Not Factored
RSA-896 (270 Digits)	$75,000	Not Factored
RSA-1024 (309 Digits)	$100,000	Not Factored
RSA-1536 (463 Digits)	$150,000	Not Factored
RSA-2048 (617 Digits)	$200,000	Not Factored

RSA-704

Decimal Digits: 212

74 03756 34795 61712 82804 67960 97429 57314 25931 88889 23128 90849 36232 63897 27650 34028 26627 68919 96419 62511 78439 95894 33050 21275 85370 11896 80982 86733 17327 31089 30900 55250 51168 77063 29907 23963 80786 71008 60969 62537 93465 05637 96359

RSA Factoring Challenge

- Latest result is RSA 200 (663 bits)
  - Reported May 2005
- Factored with Lattice Sieve
- 55 years on a single 2.2GHz Opteron CPU
  - Matrix step : 3 months on a cluster of 80 2.2GHz Opterons
  - Sieving began in late 2003 and matrix step was completed in May 2005

Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)

How to find Public-Key Algorithms

- One-way function
- Trap-door one-way function
  - Discrete Logarithm
  - Factoring
  - ……

Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one

Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to Symmetric schemes

Summary

- have considered:
  - – principles of public-key cryptography
  - – RSA algorithm, implementation, security

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root mod q
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- each user makes public that key y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is K_{AB}:

  $K_{AB} = a^{x_A . x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (which **B** can compute)

  $= y_B^{x_A} \bmod q$ (which **A** can compute)

- K_{AB} is used as session key in symmetric encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log