Cryptography and Network Security
RSA

M2L5

Slide 1 to 19 are prerequisite

Public Key Cryptography and RSA

Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.

—*The Golden Bough,* Sir James George Frazer

Private-Key Cryptography

- traditional **private/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
- hence does not protect sender from receiver forging a message & claiming is sent by sender

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

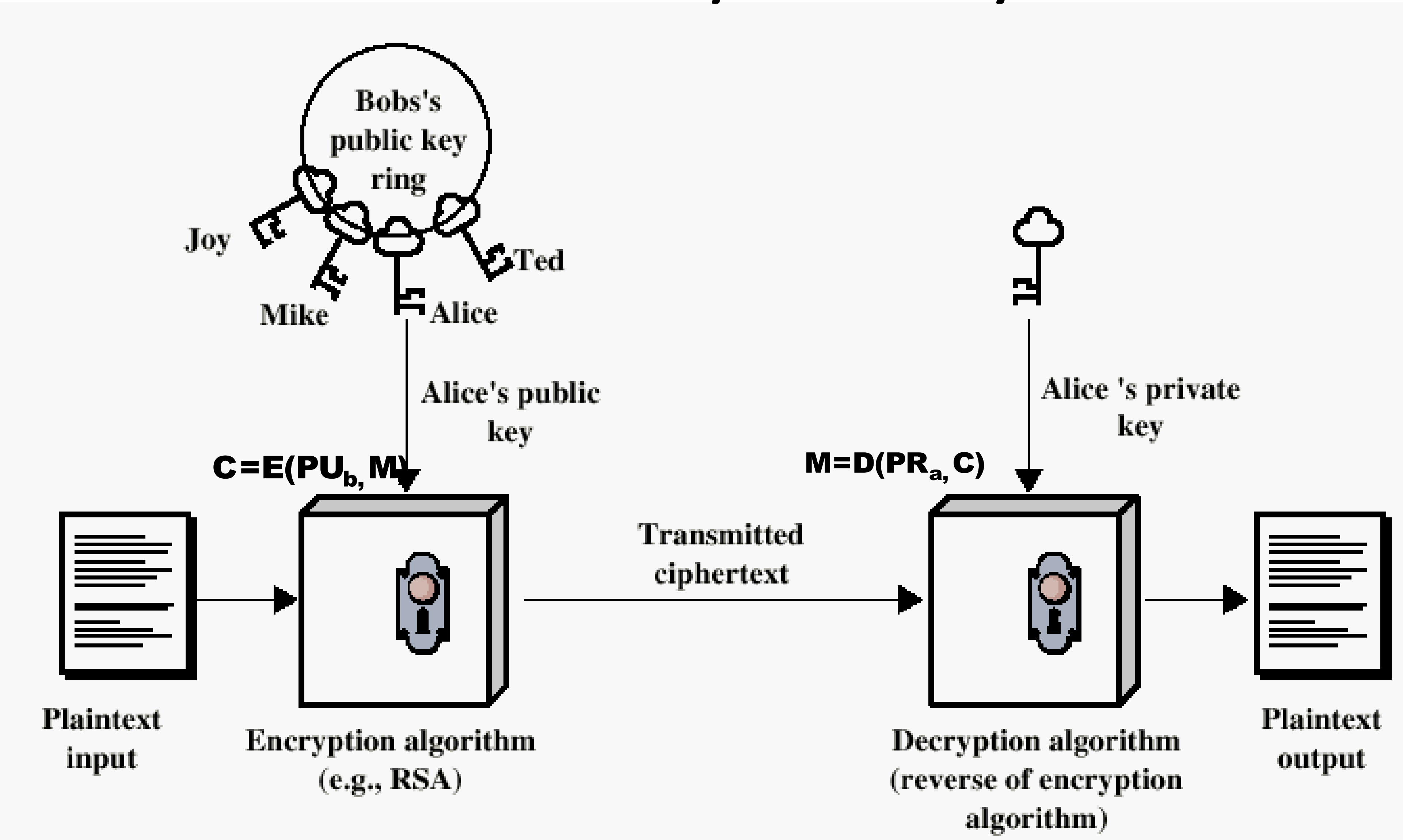
Why Public-Key Cryptography?

- 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
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

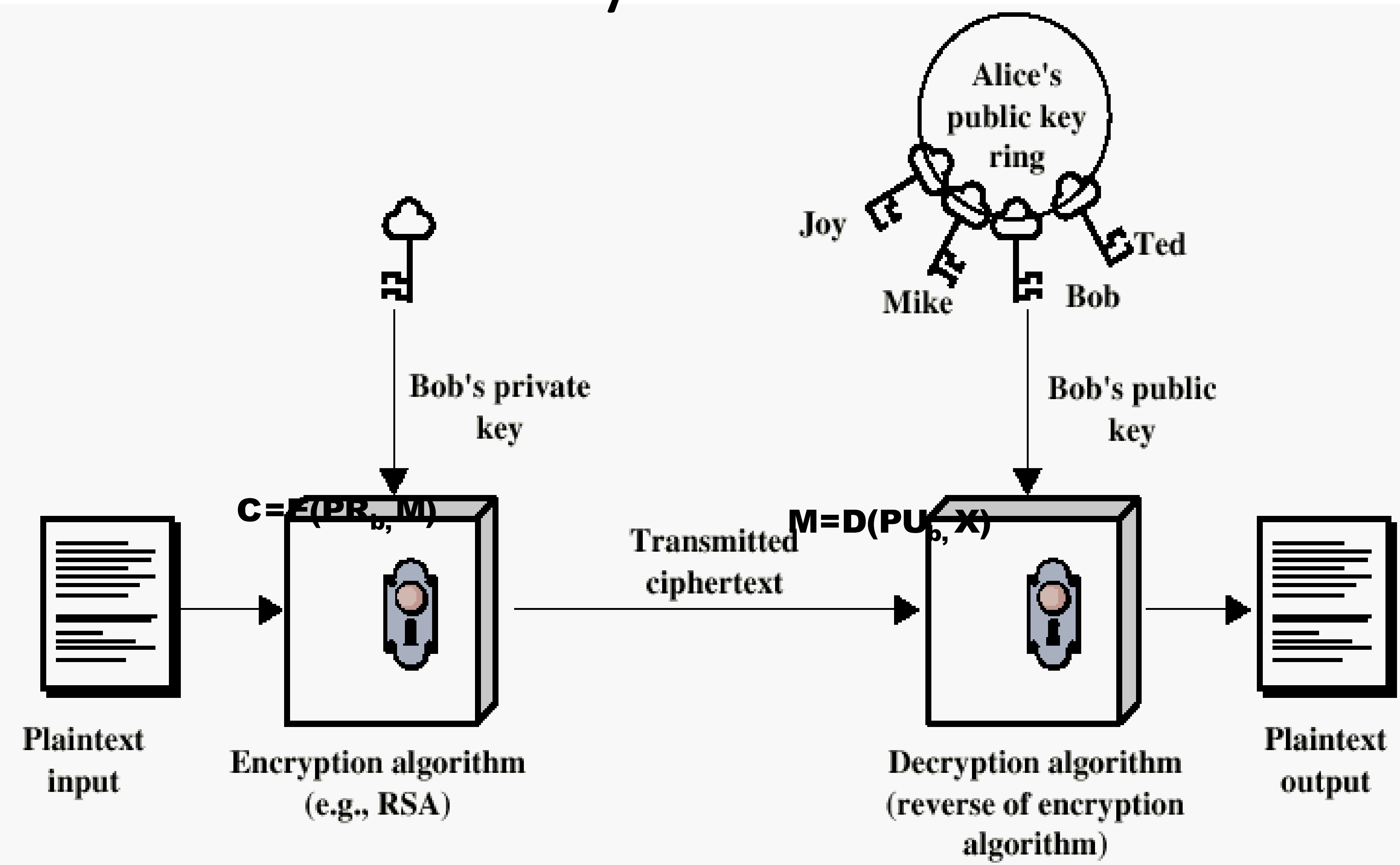
Public-Key Cryptography

- **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

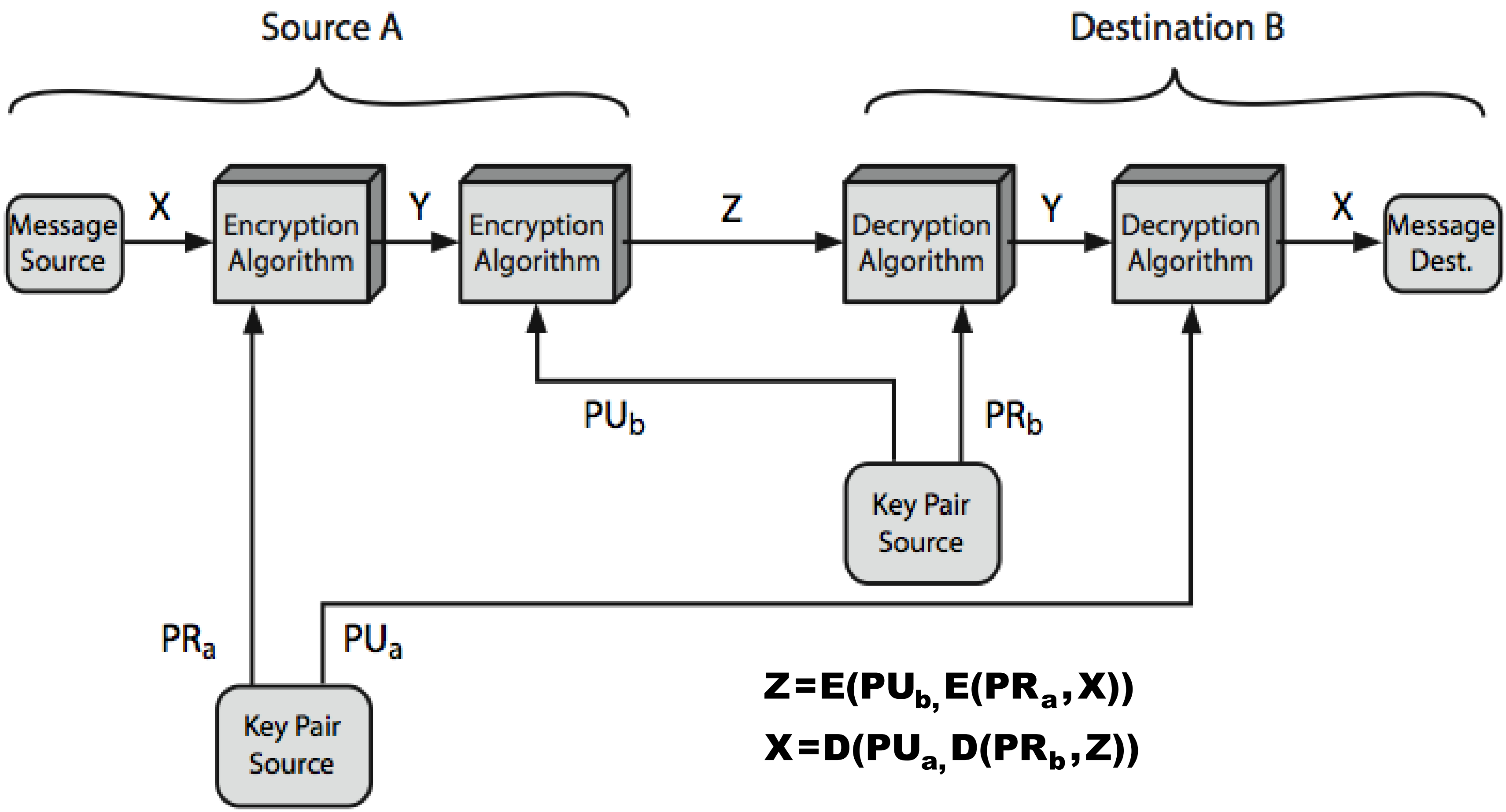
Public-Key Secrecy



Public-Key Authentication



Public-Key Authentication & Secrecy



$$Z=E(PU_b, E(PR_a, X))$$
$$X=D(PU_a, D(PR_b, Z))$$

Prime Factorisation

- to **factor** a number `n` is to write it as a product of other numbers: `n=a x b x c`
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorisation** of a number `n` is when its written as a product of primes
  - eg. $91=7\times13$; $3600=2^4\times3^2\times5^2$

$$a = \prod_{p \in \mathrm{P}} p^{a_p}$$

Relatively Prime Numbers & GCD

- two numbers `a, b` are **relatively prime** if have **no common divisors** apart from 1
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor
- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
  - eg. $300=2^1 \times 3^1 \times 5^2$ $18=2^1 \times 3^2$ hence $GCD(18,300)=2^1 \times 3^1 \times 5^0=6$

Fermat's Theorem

- $\mathbf{a^{p-1} = 1 \pmod{p}}$
  - where p is prime and $\mathbf{gcd(a,p)=1}$
- also known as Fermat's Little Theorem
- also $\mathbf{a^p = p \pmod{p}}$
- useful in public key and primality testing

Euler Totient Function $\varnothing(\mathtt{n})$

- when computing arithmetic modulo n
- **complete set of residues** is: **`0..n-1`**
- **reduced set of residues** is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}
- number of elements in reduced set of residues is called the **Euler Totient Function ø(n)**

Euler Totient Function $\varnothing(n)$

- to compute ø(n) we need to count number of residues to be excluded
- in general we need prime factorization, but
  - for p (p prime) $\varnothing(p) = p-1$
  - for p.q (p,q prime) $\varnothing(pq) = (p-1)\times(q-1)$
- eg.

  ø(37) = 36

  ø(21) = (3–1)x(7–1) = 2x6 = 12

Euler's Theorem

- a generalisation of Fermat's Theorem
- $a^{\varnothing(n)} = 1 \pmod{n}$
  - for any **a,n** where **gcd(a,n)=1**
- eg.

  $a=3; n=10; \varnothing(10)=4;$

  hence $3^4 = 81 = 1 \bmod 10$

  $a=2; n=11; \varnothing(11)=10;$

  hence $2^{10} = 1024 = 1 \bmod 11$

Chinese Remainder Theorem

- used to speed up modulo computations
- if working modulo a product of numbers
  - eg. $\mathbf{mod\ M = m_1m_2..m_k}$
- Chinese Remainder theorem lets us work in each moduli $\mathbf{m_i}$ separately
- since computational cost is proportional to size, this is faster than working in the full modulus **M**

Chinese Remainder Theorem

- We can implement CRT in several ways
- to compute **`A(mod M)`**
  - first compute all $\mathbf{a_i = A \bmod m_i}$ separately
  - determine constants $\mathbf{c_i}$ below, where $\mathbf{M_i = M/m_i}$
  - then combine results to get answer using:

$$A \equiv \left(\sum_{i=1}^{k} a_i c_i \right) (\bmod M)$$

$$c_i = M_i \times (M_i^{-1} \bmod m_i) \quad \text{for } 1 \le i \le k$$

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 private key schemes

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Algorithm

- 1) Key generation; PU={e,n} and PR={d,n}
- 2) Encryption $C = M^e \bmod n$
- 3) Decryption $M = C^d \bmod n = (M^e) \bmod n = M^{ed} \bmod n$
- Both sender and receiver have ***n***. The sender has ***e*** and only the receiver has ***d***.

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 encryption key e
  - where $1<e<ø(n)$, $\gcd(e,ø(n))=1$
- solve following equation to find decryption key d
  - $e.d=1 \bmod ø(n)$ and $0\leq d\leq n$
- publish their public encryption key: **PU={e,n}**
- keep secret private decryption key: **PR={d,n}**

The RSA Algorithm – Key Generation

1.	Select p,q	p and q both prime, $p<>q$
2.	Calculate	$n = p \times q$
3.	Calculate	$\phi(n) = (p-1)(q-1)$
4.	Select integer e	$gcd(\phi(n),e)=1; 1<e< \varnothing(n)$
5.	Calculate d	$d=e\ INV\ (mod\ \phi\ (n))$
6.	Public Key	$PU = \{e,n\}$
7.	Private key	$PR = \{d,n\}$

The RSA Algorithm - Encryption

- Plaintext: $M<n$
- Ciphertext: $C = M^e \pmod n$

The RSA Algorithm - Decryption

- Ciphertext: C
- Plaintext: $M = C^d \ (mod\ n)$

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient **PU={e,n}**
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- 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:
  - $a^{ø(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $ø(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses mod $ø(n)$
  - hence $e.d=1+k.ø(n)$ for some k
- hence :

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

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

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:

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

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

Example of RSA Algorithm

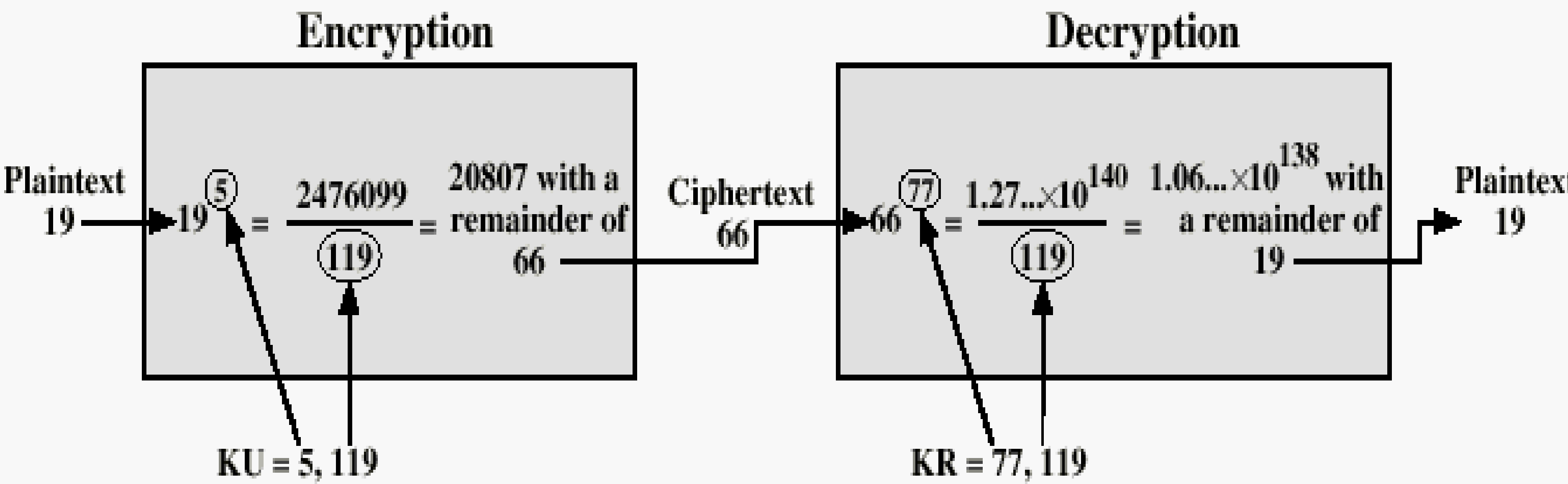


Figure 3.9 Example of RSA Algorithm

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. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation algo for Computing $a^b mod\ n$

```
c = 0; f = 1
for i = k downto 0
     do c = 2 x c
        f = (f x f) mod n
     if b_i == 1 then
        c = c + 1
        f = (f x a) mod n
return f
```

Exponentiation in Modular Arithmetic

$$[(a \bmod n) * (b \bmod n)] \bmod n = (a * b) \bmod n$$

Find a^b (a and b positive)

Expres b as a binary number $b = \sum_{b_i != 0} 2^i$

Therefore

$$a^b = a^{\left(\sum_{b_i != 0} 2^i\right)} = \prod_{b_i != 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i != 0} a^{(2^i)}\right] \bmod n = \left(\prod_{b_i != 0} \left[a^{(2^i)} \bmod n\right]\right) \bmod n$$

Efficient Encryption

- 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
  - using Chinese remainder theorem & 3 messages with different modulii
- if e fixed must ensure **`gcd(e,ø(n))=1`**
  - ie reject any p or q not relatively prime to e

Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

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 (all possible private keys)
  - mathematical attacks (based on difficulty of computing ø(n), by factoring modulus n, product)
  - timing attacks (on running of decryption algo)
  - chosen ciphertext attacks (exploit given properties of RSA)
  - Hardware fault-based attacks