Cryptography and Network Security: Principles and Practice (5e)

by William Stallings

Chapter 9

Public-Key Cryptography and RSA

Secret-Key (Symmetric) Cryptography

- both encryption & decryption use same key
- The key is secretly shared by sender and receiver (unknown to third party)
  - Hence, **secret-key** crypto (a.k.a. private-key)
  - If the key is disclosed, the communications are compromised
- not protect sender from "receiver" forging a message & claiming it sent by sender

Public-Key (Asymmetric) Cryptography

- encryption and decryption use two different keys
  - One key known to everybody, called **public** key, for encryption
  - The other kept secretly, called **private** key, for decryption

- most significant advance in the 3000 year history of cryptography
  - Probably the only true revolution

- complements **rather than** replaces secret-key
  - Both have issues with key distribution

Chapter 9. Public-Key Cryptography and RSA

9.1 Principles of Public-Key Cryptosystems

- Public-Key Cryptosystems
- Applications for Public-Key Cryptosystems
- Requirements for Public-Key Cryptography
- Public-Key Cryptanalysis

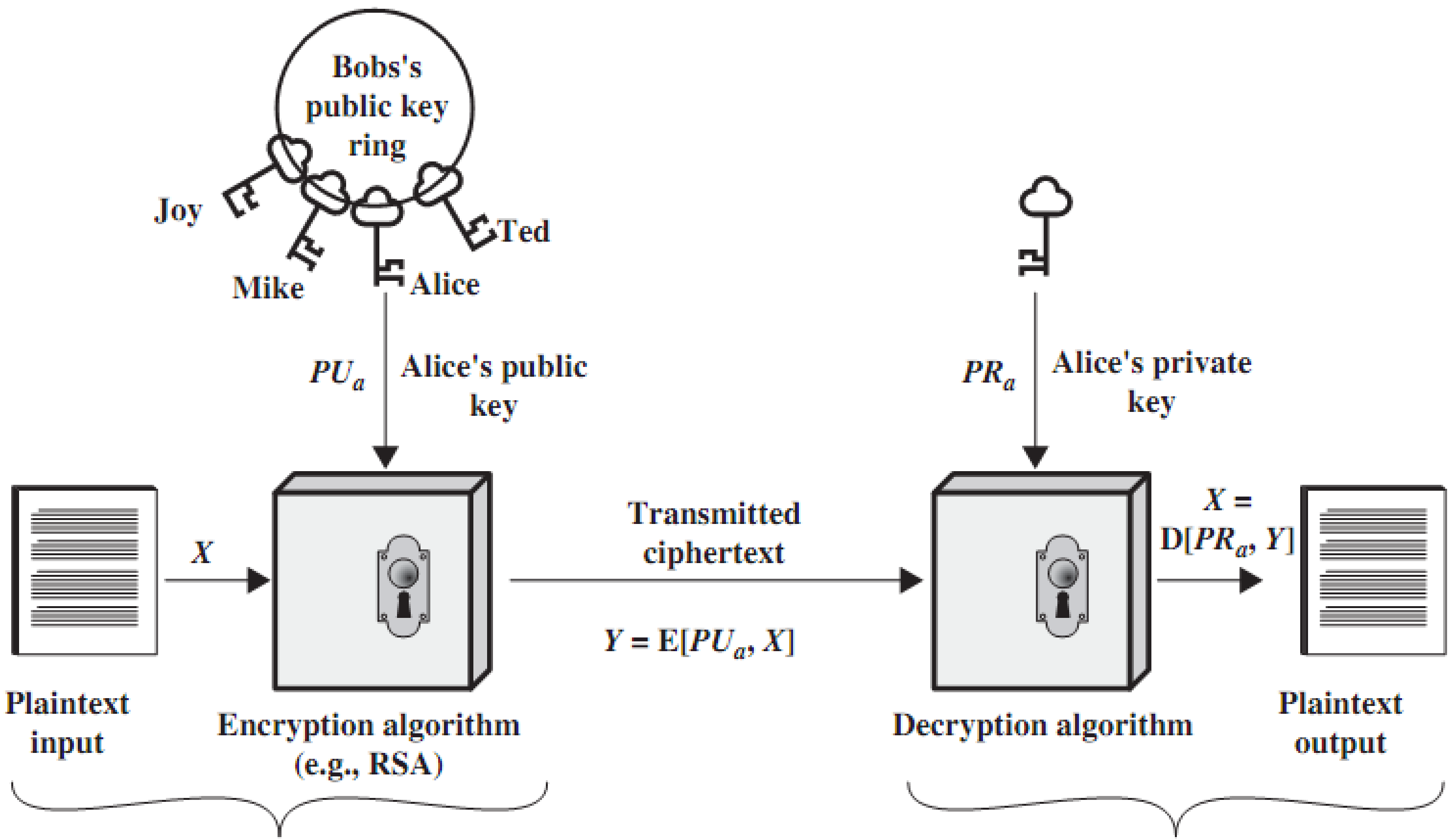
9.2 The RSA Algorithm

- Description of the Algorithm
- Computational Aspects
- The Security of RSA

Why Public-Key Cryptography?

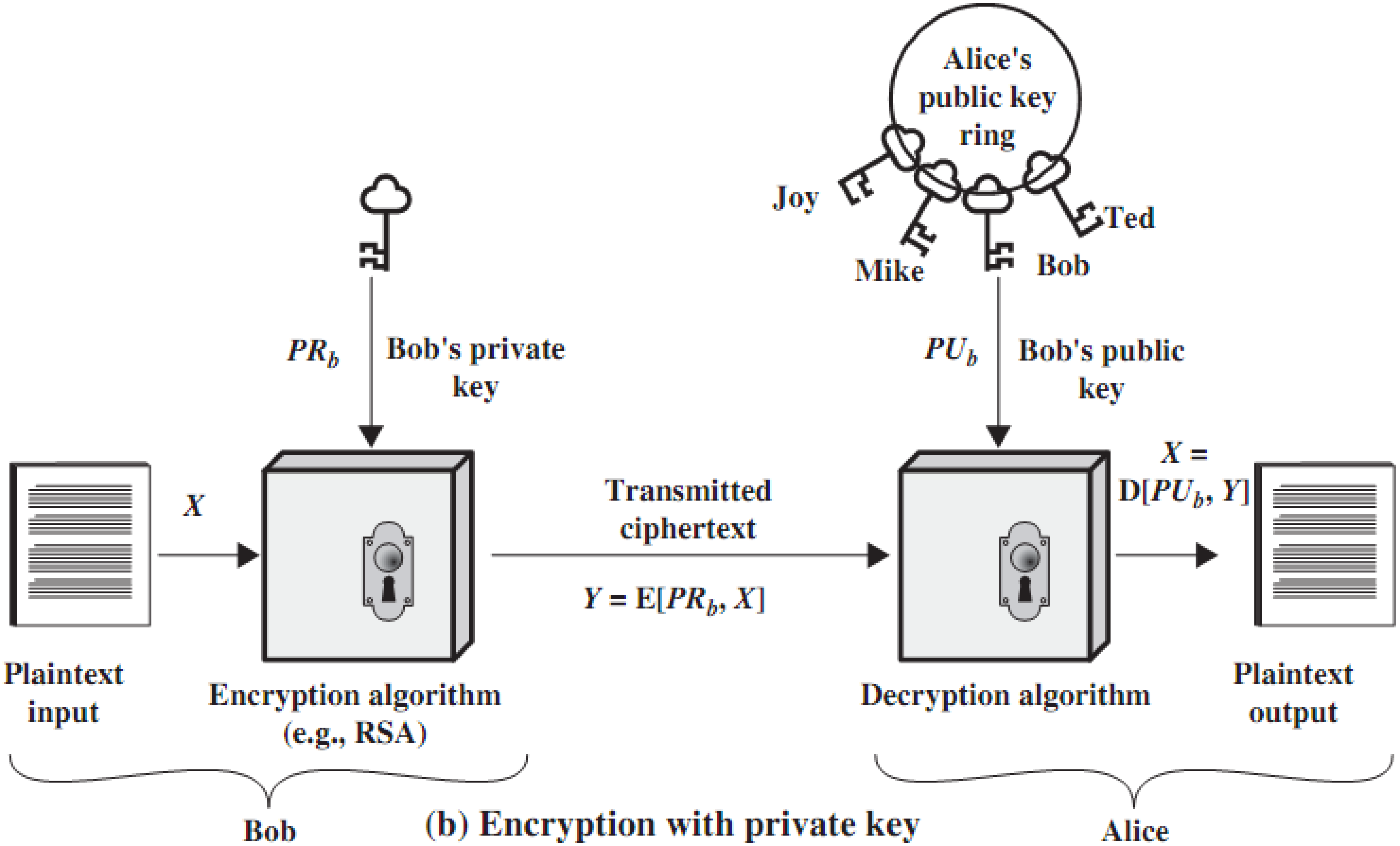
- developed to address two major issues:
  - **key distribution**
    how to set up secure communications
    w/o having to trust a Key-Distrib.-Centre w your key
    - the first scheme by D-H was only for that purpose
  - **digital signatures**
    how to verify a message comes intact from the claimed sender
- Whitfield Diffie & Martin Hellman (Stanford 1976)
  - interesting to note that RSA first, then Diffie-Hellman

Public-Key for Encryption
Private-Key for Decryption



(a) Encryption with public key

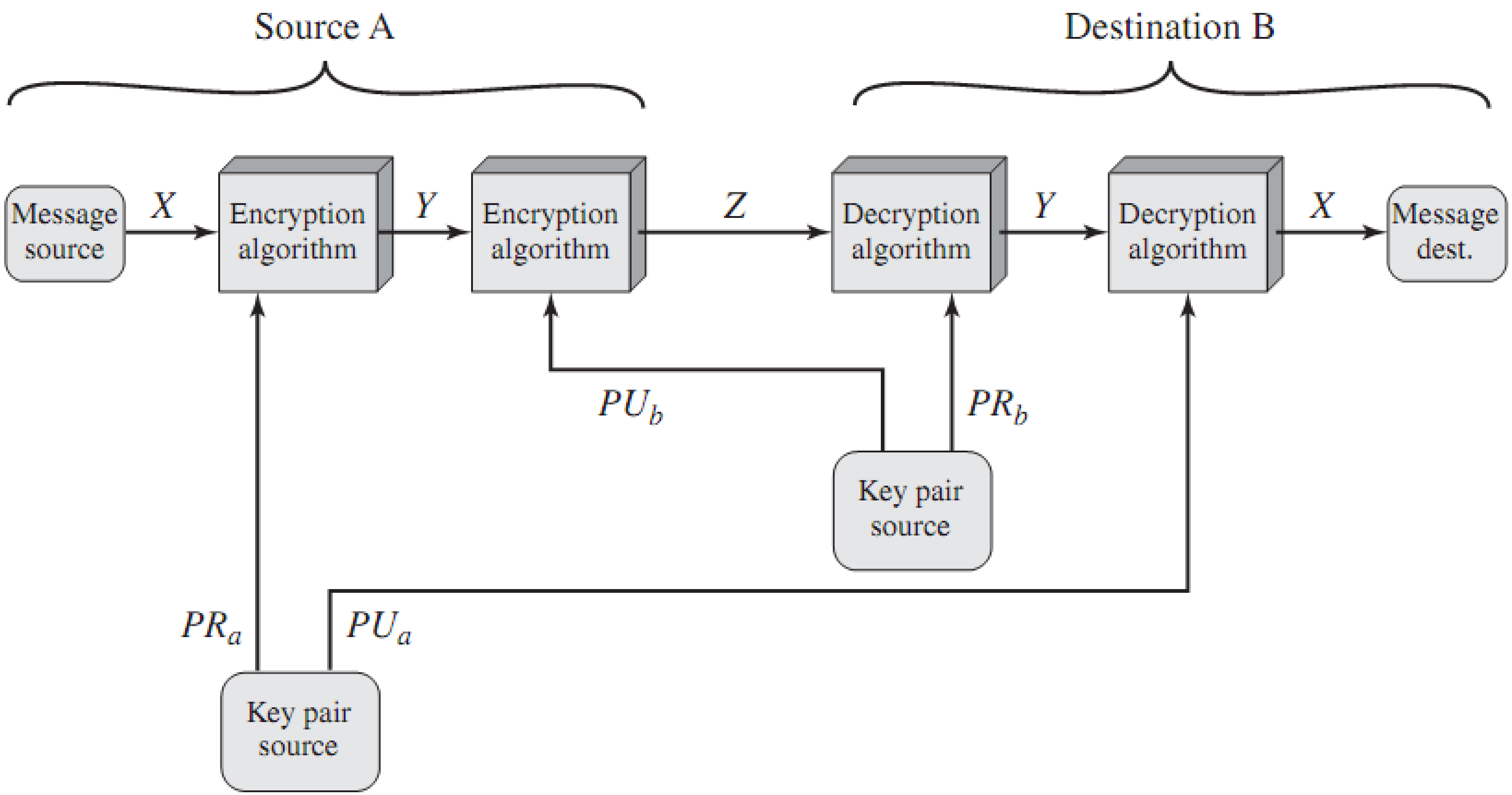
Private-Key for Encryption
Public-Key for Decryption



(b) Encryption with private key

→ authentication → digital signature

Both Authentication & Secrecy



Public-Key Cryptography

Emphasize once again

- **public-key / two-key / asymmetric** crypto:
  - 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**
- **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

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
  - any of the two related keys can be used for encryption, for some algorithms

Public-Key Applications

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

Security of Public Key Schemes

- Brute-force attack is theoretically possible
  – Keys too large (>512bits) to be computational feasible

- security relies on a large difference in difficulty btwn **easy** problem(encrypt/decrypt) and **hard** problem (to derive private key from public key)
  – **trapdoor** functions whose inverses are hard to get
  – e.g., multiplication is easy while its inverse, prime factorization, is very hard
  – e.g.,exponentiation easy while discrete logarithm hard

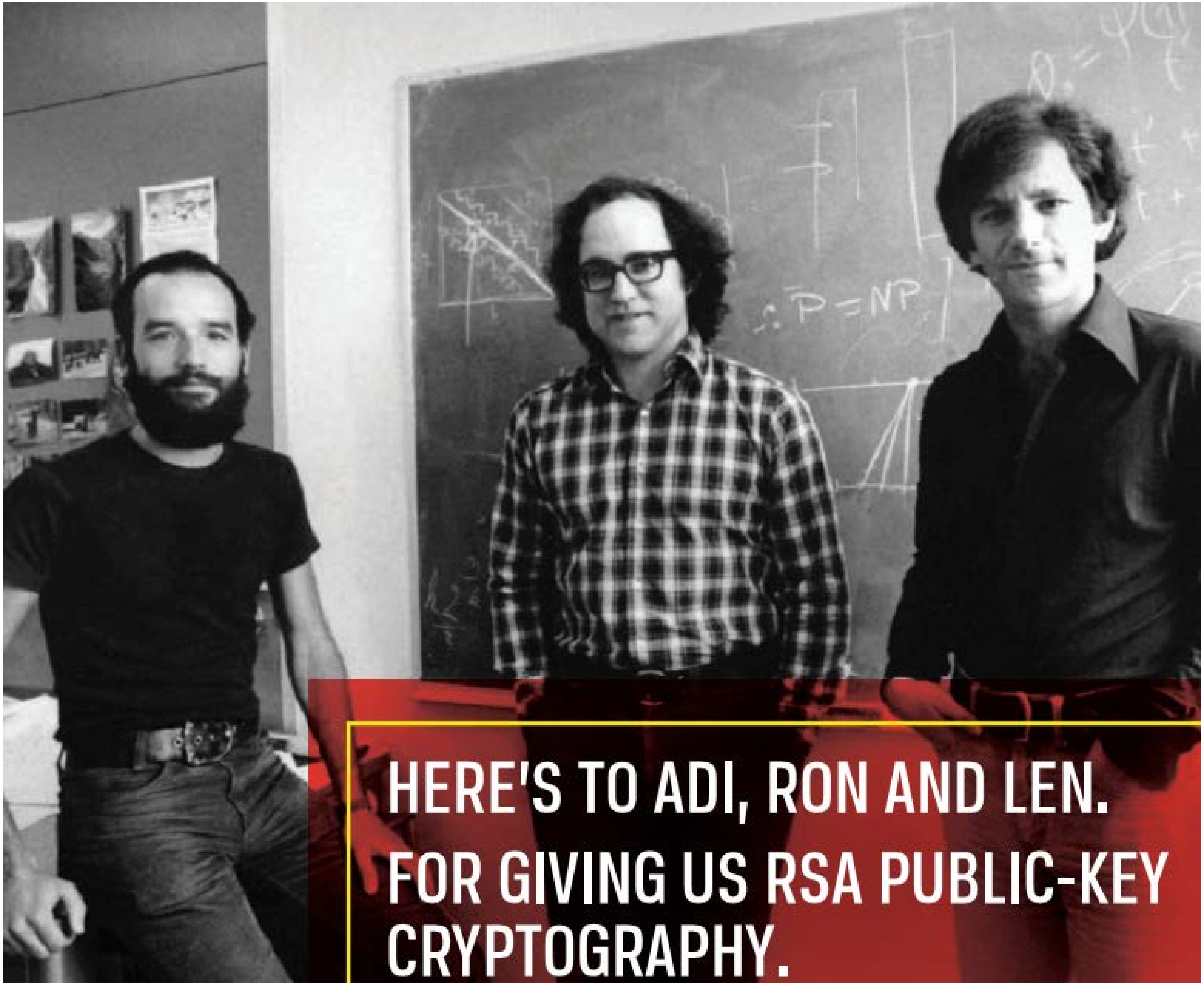
Chapter 9. Public-Key Cryptography and RSA

9.1 Principles of Public-Key Cryptosystems

- Public-Key Cryptosystems
- Applications for Public-Key Cryptosystems
- Requirements for Public-Key Cryptography
- Public-Key Cryptanalysis

9.2 The RSA Algorithm

- Description of the Algorithm
- Computational Aspects
- The Security of RSA


HERE'S TO ADI, RON AND LEN.
FOR GIVING US RSA PUBLIC-KEY CRYPTOGRAPHY.

RSA: Public Key Cryptographic Algorithm

- by Rivest, Shamir & Adleman (MIT in 1977)
- best known & widely used public-key scheme
- based on exponentiation in a finite field (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 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**
  - compute **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**
  - where **e.d = 1 mod ø(n)**
- public key: PU={e,n} , private key: PR={d,n}

RSA Encryption / Decryption

- 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 recipient :
  - uses own private key **PR = {d, n}**
  - computes: $M = C^d \bmod n$
- note that the message **M** must be smaller than the modulus **n**

RSA Example - Key Setup

1. Select primes: $p = 17$ **and** $q = 11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $ø(n)=(p–1)(q-1)= 16 \times 10=160$
4. Select : $\gcd(e,160)=1$; e.g., $e = 7$
5. Determine $de = 1 \pmod{160}$ & $d < 160$
   get $d = 23$ as $23 \times 7 = 161 = 10 \times 160+1$
6. Publish public key $PU=\{e,n\}=\{7,187\}$
7. Keep secret private key $PR=\{d,n\}=\{23,187\}$

RSA Example - En/Decryption

- **given message $M = 88$ (nb. $88<187$)**

- **encryption:**

  $C = 88^7 \bmod 187 = 11$

- **decryption:**

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

The Correctness of RSA

- by Fermat (Little) Theorem

  Chinese Remainder Theorem

- RSA:

  $n = p \cdot q$

  $\varnothing(n)=(p-1)(q-1)$

  choose e **&** d **such that** $e \cdot d = 1 \pmod{\varnothing(n)}$

  so $e \cdot d = 1 + k \cdot \varnothing(n)$ **for some** k

- hence :

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

Proof of $M^{1+k\cdot\phi(n)} \bmod n = M$

- First, consider $M^{1+k\cdot\phi(n)} \bmod p$
- There are two cases:

Case 1: $\gcd(M,p)\neq 1$ meaning $M = 0 \pmod p$

then $M \bmod p = M^{1+k\cdot\phi(n)} \bmod p$

Case 2: $\gcd(M,p)=1$, then **Fermat** applies:

$M^{1+k\cdot\phi(n)} \bmod p$

$= M^1\cdot(M^{p-1} \bmod p)^{k(q-1)} \bmod p$

$= M^1\cdot(1)^{k(q-1)} \bmod p$ by Fermat's

$= M \bmod p$

Correctness (cont')

- Therefore, we always have

  $M^{1+k\cdot ø(n)} \bmod p = M \bmod p$

- Similarly,

  $M^{1+k\cdot ø(n)} \bmod q = M \bmod q$

- By Chinese Remainder Theorem:

  $M^{1+k\cdot ø(n)} \bmod p\cdot q = M \bmod p\cdot q$

- Finally, notice that $0 \le M < n$,

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

How to generate Key efficiently

- determine two primes **p, q** at random

  Miller-Rabin probabilistic primality test

  p,q must not be easily derived from **n = p.q**

- Select **e** such that gcd(**e**,ø(**n**))=1

  probability that two random numbers relatively prime 0.6

- compute **d** such that **e.d** = 1 (mod ø(**n**))

  Extended Euclidean Algorithm finds inverses

How: Modular Exponentiation

- the Square and Multiply Algorithm for efficient exponentiation $\mathbf{a}^{\mathbf{b}}$
- takes $O(\log_2 \mathbf{b})$ multiplications for $\mathbf{a}^{\mathbf{b}}$, simple eg:
  - $7^5 = 7^4 . 7^1 = 3.7 = 10 \pmod{11}$
  - $3^{129} = 3^{128} . 3^1 = 5.3 = 4 \pmod{11}$
- Algorithm with exponent **b** in binary
  - repeatedly squaring the base **a**
  - multiplying the base **a** for the ones in exponent **b**

Modular Exponentiation

- Let **$\langle b_k, \ldots, b_0 \rangle$** binary representation of **b**

$$b = \sum_{b_i \neq 0} 2^i$$

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

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

The Algorithm

```
MODULAR-EXPONENTIATION(a,b,n)
c = 0; d = 1;
Let <b_k,…,b_0> binary representation of b
for i = k downto 0
do  c = 2 · c;
    d = (d · d) mod n;
    if b_i == 1
    then    c = c + 1;
            d = (d · a) mod n;
return d;   /*d = a^c for understanding*/
```

Efficient Encryption

- efficient modular exponentiation to power e
  - if e is small, it will be even faster
    often choose e = 65537 (2^{16}-1)
  - but, not too small otherwise it is vulnerable to attack

Efficient Decryption

- efficient modular exponentiation to power d
  - d is likely large, insecure if not
- One who does decryption usually is the owner of private key (knowing p & q) can
  - compute (mod p) and (mod q) separately
  - then combine to get desired answer
  - approx 4 times faster than doing directly

Efficient Version of Miller-Rabin PrimalityTest

- Fermat's theorem
- Square root of 1
- Trying **ln(n)** times could find a prime near **n**
- **Error rate at most $(\frac{1}{2})^s$**

```
MILLER-RABIN(n, s)
   for j = 1 to s do
         a = RANDOM(1, n-1)
         if WITNESS(a, n)
         then return COMPOSITE
   return PRIME
```

- **Where WITNESS tells if n is composite**

```
WITNESS(a,n)
   d = 1
   Let <b_k,…,b_0> binary representation of n-1
   for i = k downto 0 do
      x = d
      d = (d · d) mod n
      if d==1 and x≠1 and x≠n-1
      then return TRUE              // sqrt(1)
      if b_i == 1
      then d = (d · a) mod n

   if d≠1 then return TRUE // Fermat
   return FALSE
```

- Implementation based on Modular Exponentiation

Probabilistic Considerations

- if WITNESS returns true, the number is definitely not prime
- otherwise is a prime or a pseudo-prime
- chance it detects a pseudo-prime is $<\frac{1}{4}$ (err)
- If repeat test with different random a then chance n is prime after s tests is
  - Pr(n prime after s tests) = $1 - (\frac{1}{4})^s$
  - eg. for s=10 this probability is > 0.99999

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)