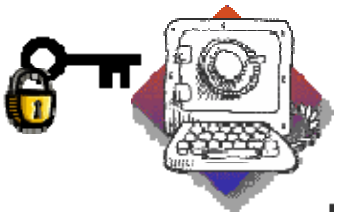


Chapter 4
Introduction to Public-Key Cryptography

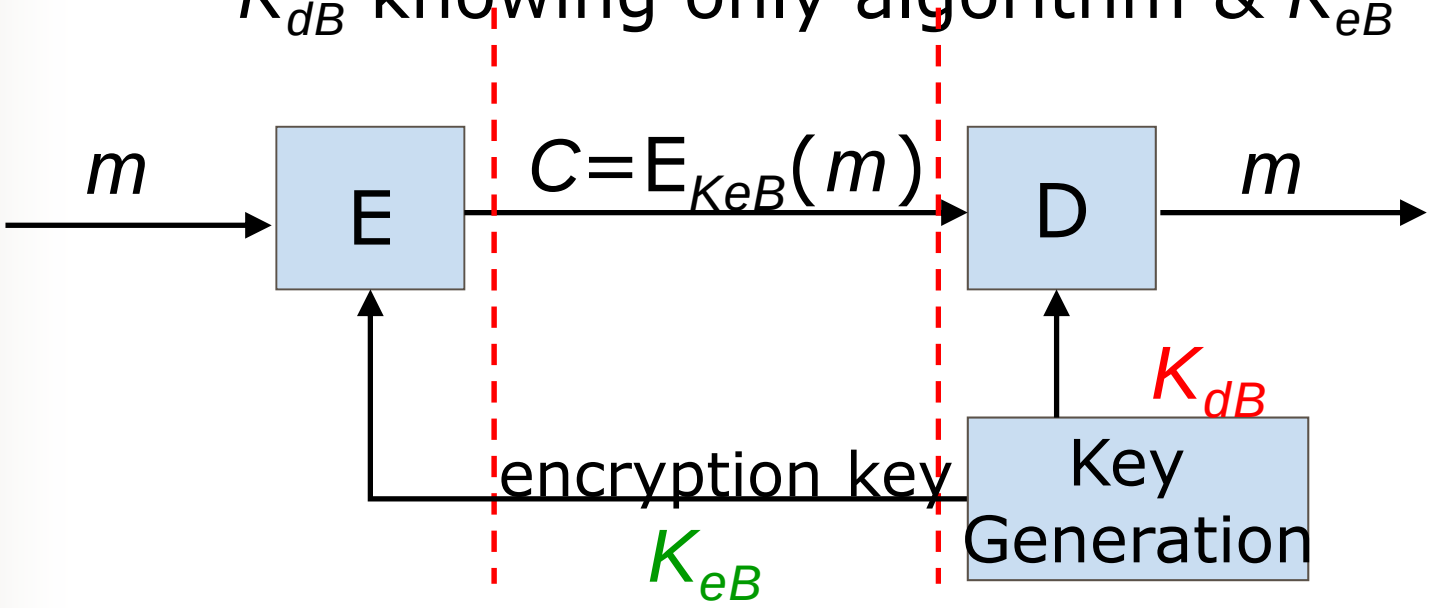
顏嵩銘 (Sung-Ming Yen)

中央大學 資訊工程系所
密碼與資訊安全實驗室
Laboratory of Cryptography and Information Security
http://www.csie.ncu.edu.tw/~yensm/lcis.html

Tel： (03) 4227151 Ext- 35316
Fax： (03) 4222681
E-Mail：yensm@csie.ncu.edu.tw

The Model of PKC

- The model
  - m: plaintext C: ciphertext
  - $E_{KeB}()$: encryption using B's encryption key
  - $D_{KdB}()$: decryption using B's decryption key
    - computationally "infeasible" to find K_{dB} knowing only algorithm & K_{eB}



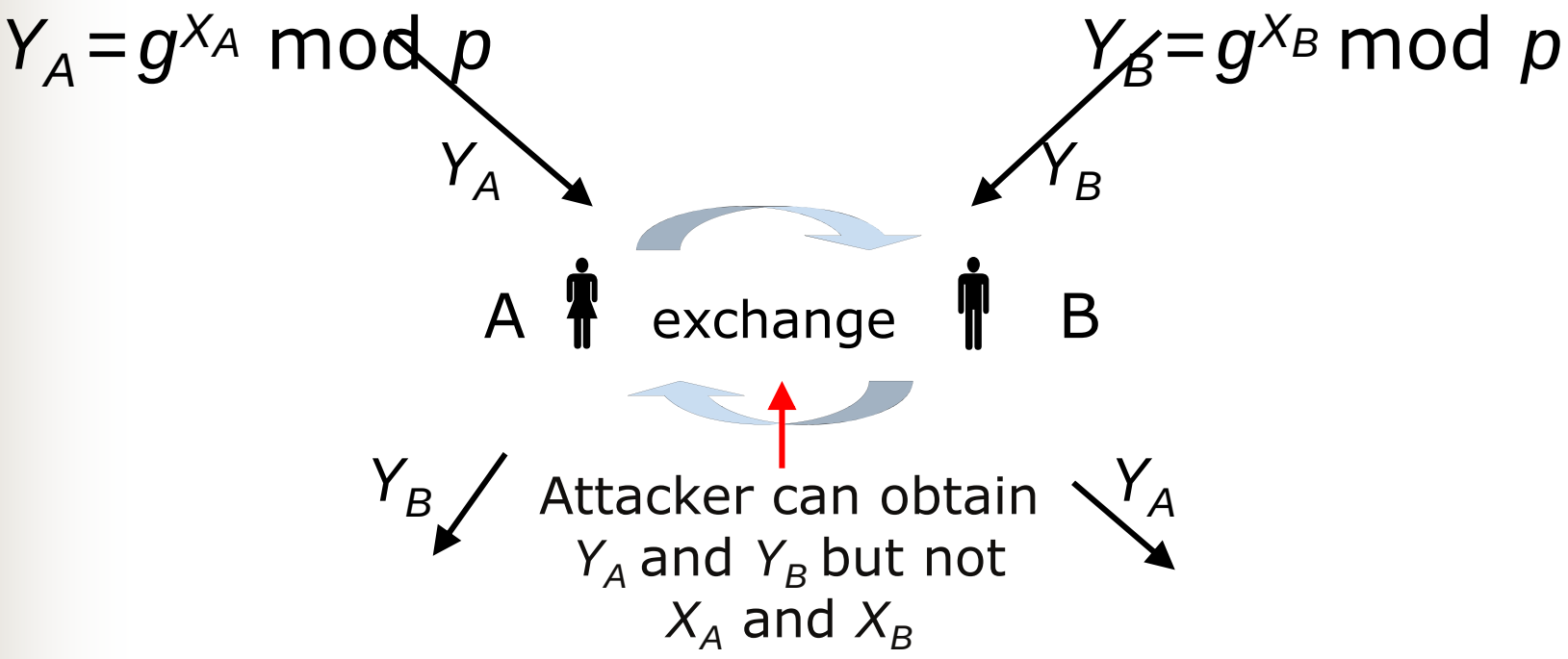
- Information can be enciphered by encoding it through a very hard problem such that breaking the cipher would require solving the hard problem in a usual way.
  - with the deciphering key, however, a short cut solution would be possible.
- Example: Knapsack problem
  - given $(r_t, \ldots, r_1, r_0)$ & $(m_t, \ldots, m_1, m_0)$ where r_i are random integers & $m_i \in \{0, 1\}$
  - it is **easy** to compute $C = \sum_{i=0}^{t} r_i \times m_i$ but it is **hard** to recover all m_i from C.
  - if $r_i = 2^i * \boldsymbol{K} \bmod p$, then knowing K it becomes very easy to recover all m_i from $C * K^{-1} \bmod p$ where $p \geq 2^{t+1}$.

- Classification of security
  - *Computational* security: *exhaustive* search attack exploiting some math. properties of the cryptosystem is *theoretically possible* however it is *computationally infeasible*
    - with bounded storage & computing power
  - *Information Theoretical* security
- *Post-quantum* cryptography
  - Some hard problems (e.g., factorization) become much easier under a quantum computer

- Why Public-Key Cryptography?
  - developed to address two issues:
    - **key distribution** – how to have secure communication?
      - set up secure communication between **any two** parties
      - without having to **trust** a KDC
    - **digital signature** – how to verify?
      - message comes from claimed sender
      - signature generated by claimed sender
- The first PKC is due to Diffie & Hellman at Stanford University in 1976
  - known earlier in classified community (I will show you the documents later)

Diffie-Hellman Public Key-Exchange

Alice "Diffie-Hellman" Bob



$Z = Y_B^{X_A} \bmod p$

$= g^{X_B * X_A}$

$Z = Y_A^{X_B} \bmod p$

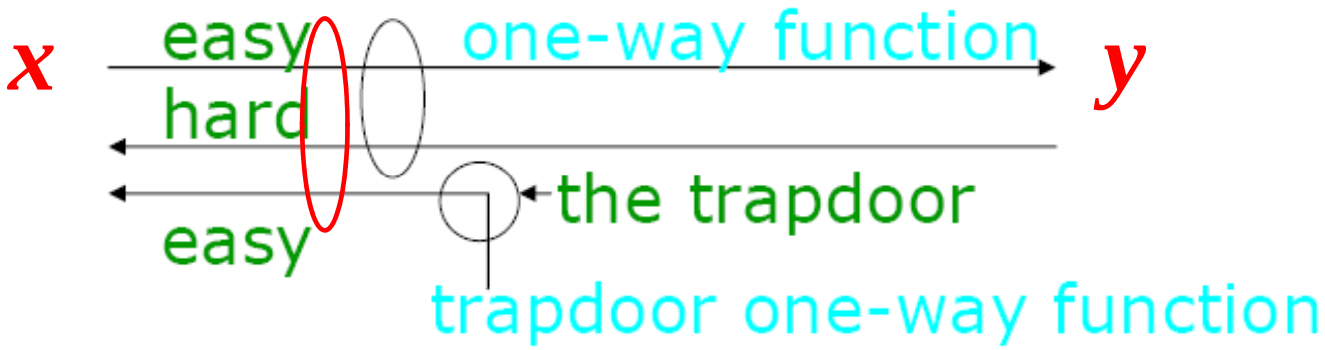
$= g^{X_A * X_B}$

- Security of Diffie-Hellman scheme
  - **passive** adversary is impossible
    - given g, g^a, and g^b, no efficient algorithm exists to compute g^{ab}
  - however, no protection against **active** attack
    - authentication on g^a and g^b is necessary to avoid/detect active attack
      - man-in-the-middle attack
      - to impersonate a user

Definition of One-way Function

- It is easy to compute $y = f(x)$ for $\forall\, x$
- For almost all y, it is computationally infeasible to compute $x = f^{-1}(y)$ even if f is known. For example,
  - computing discrete logarithm

$$x = \log_g y \mod P \quad (\text{ or } g^x \equiv y \pmod P)$$

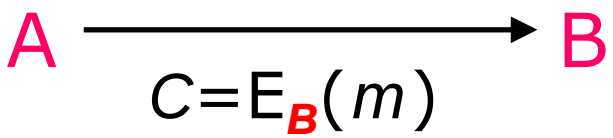


Definition of Trapdoor One-way Function

- Same as one-way function, but it is easy to compute $f^{-1}(y)$ given some additional information, e.g.,
  - knowing how to factorize an integer $n=p*q$ then we know how to compute $y^{1/3} \bmod n$ (e.g., RSA) or $y^{1/2} \bmod n$ (e.g., Rabin)
  - knowing how to transform Knapsack problem to an **easy** Knapsack problem

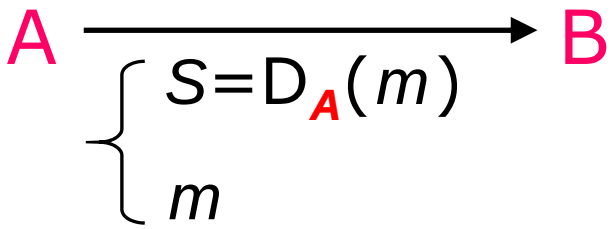
Operation Models of PKC

- Secrecy in PKC



$$D_B(C)=D_B(E_B(m))=m$$

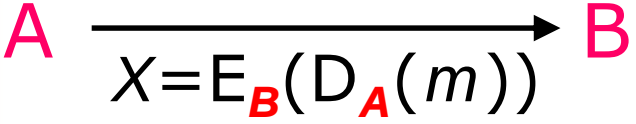
- Authenticity in PKC



$$E_A(S)=E_A(D_A(m))=m'$$

Checks $m' \stackrel{?}{=} m$

- Both secrecy and authenticity in PKC



$$E_A(D_B(X))$$

$$=E_A(D_B(E_B(D_A(m)))) =m$$

- RSA can perform both secrecy and authenticity
- Signcryption (by Yuliang Zheng)

Issues of RSA Cryptosystem

- Basics of RSA
- Security of RSA
- Large parameters generation (prime testing)
- Implementation of RSA
  - Efficient implementation of RSA
  - Secure implementation against physical attack

The RSA Cryptosystem

- System parameters: each user computes

  $n=p*q$ (two large primes)

  e: ***public*** key (encryption key)

  d: ***private*** key ∋ $e*d \equiv 1 \bmod \varphi(n)$

  where $\varphi(n)=(p-1)*(q-1)$

  and $\gcd(e,\varphi(n))=1$ for d to exist

RSA En/Decryption

- Encryption:

$$C = m^e \bmod n$$

- Decryption:

$$m \equiv C^d \equiv m^{ed} \pmod{n}$$
$$\equiv m^{1+k\varphi(n)} \pmod{n}$$
$$\equiv m$$

where $e*d \equiv 1 \pmod{\varphi(n)}$

$e*d = 1 + k\varphi(n)$ for some integer k

and

$m^{\varphi(n)} \equiv 1 \pmod{n}$

Why not just let $ed = 1 + \varphi(n)$:

$C = m^e \bmod n$

$C^d \equiv m^{ed} \pmod{n}$

$\equiv m^{1+\varphi(n)} \pmod{n}$

$\equiv m$

- Two cases of m: $(1 \leqq m \leqq n\text{-}1)$

$m=0$ is a trivial case

(1) For all $m \in \mathbf{Z}_n^*$ (i.e., $\gcd(m, n)=1$)

$m^{\Phi(n)} \equiv 1 \pmod{n}$ → Euler's generalization

Therefore, $m^{1+k\Phi(n)} \bmod n$

$= m^1 * (m^{\Phi(n)})^k \bmod n$

$= m * 1 = m$

(2) For all $m \in \mathbf{Z}_n \setminus \{\mathbf{Z}_n^*, 0\}$

Probability of such $m = \frac{n\text{-}1\text{-}\varphi(n)}{n\text{-}1} = \frac{pq\text{-}1\text{-}(p\text{-}1)(q\text{-}1)}{pq\text{-}1}$

$= \frac{p+q\text{-}2}{pq\text{-}1}$ (large or small?)

Let $|p| \approx |q| = 512$ bit,

then $\frac{p+q\text{-}2}{pq\text{-}1} \approx \frac{1}{2^{511}}$

- If this indeed occurs, can RSA decrypt correctly?

  $\gcd(m,n)=p$ or q The user's RSA key has been broken

  Let $m=ap$ where $1 \leq a \leq q-1$ → $\gcd(a,q)=1$

(i) $m \xrightarrow{\text{CRT}} (m \bmod p,\ m \bmod q)$

$=(0,\ ap \bmod q)$

(ii) $C=m^e \bmod n$

$C \xrightarrow{\text{CRT}} (0^e \bmod p\ ,\ (ap)^e \bmod q)$

when decryption:

$C^d \bmod n \xrightarrow{\text{CRT}} (0^d \bmod p\ ,\ (ap)^{ed} \bmod q)$

$=(0\ ,\ (ap)^{1+k(p-1)(q-1)} \bmod q)$

$=(0\ ,\ (ap)*[(ap)^{q-1}]^{k(p-1)} \bmod q)$

$=(0\ ,\ ap \bmod q)$

this is m

$\because \gcd(ap, q)=1$

$\therefore (ap)^{q-1} \equiv 1 \bmod q$

from Fermat's theorem

RSA Signature

- RSA used as digital signature

signer A $\rightarrow\rightarrow$ receiver B (Verifier)

$\{m, S\}$ $m' = S^{e_A} \pmod{n_A}$

$S = m^{d_A} \pmod{n_A}$ checks if $m' \stackrel{?}{=} m$

Reblocking: when used for both secrecy & authenticity:

(i) When $n_A < n_B$: $X = (m^{d_A} \bmod n_A)^{e_B} \bmod n_B$ $(m^{e_B} \bmod n_B)$

$\rightarrow$ $m = (X^{d_B} \bmod n_B)^{e_A} \bmod n_A$

Probability of reblocking problem $= (n_A - n_B)/n_A$ if $n_A > n_B$

Encryption first then signature, is used when $n_A > n_B$

(but preferred order of operations is to sign first!)

(ii) Two moduli per entity: smaller one for signature & larger one for encryption; always sign then encrypt.

(iii) Prescribing the form of modulus n & always sign then encrypt: refer to Handbook of Applied Cryptography, pp.435-436.

- Usually, cryptographic *one-way* ***hash*** $h()$ is required to compute the *digest* of m

  $S = h(m)^{d_A} \bmod n_A$

  - given $h(m)$ it's **hard** (?) to find m
  - for 3 purposes:
    - to improve *performance*
    - to improve *data integrity* of large message
    - to avoid signature *forgery* (to discuss later)

- Known RSA & D-H in classified community (Government Communications Headquarters in UK)
  - before Diffie & Hellman (in 1976)

An Early RSA in 1973

20 November 1973

A NOTE ON 'NON-SECRET ENCRYPTION'

by C C Cocks

A possible implementation is suggested of J H Ellis's proposed method of encryption involving no sharing of secret information (key lists, machine set-ups, pluggings etc) between sender and receiver.

- Public key: $n=p*q$ where $\gcd(p, q-1)=1$ & $\gcd(q, p-1)=1$
- Encryption: $C = m^n \bmod n$

  or represented by CRT: $C \equiv m^q \pmod p$ & $C \equiv m^p \pmod q$
- Private key: $q' * q \equiv 1 \pmod{\varphi(n)}$ & $p' * p \equiv 1 \pmod{\varphi(n)}$

  or just $q' * q \equiv 1 \pmod{p-1}$ & $p' * p \equiv 1 \pmod{q-1}$
- Decryption: $m = C^{p'*q'} = (m^{p*q})^{p'*q'} \bmod n$

  Cocks cleverly proposed to speed up by using CRT:

  $m \equiv C^{q'} \pmod p$ & $m \equiv C^{p'} \pmod q$, CRT equation is??
- Signature scheme?

Encryption by Public Discussion (like D-H) in 1974

NON-SECRET ENCRYPTION USING A FINITE FIELD

by M J Williamson, 21 January 1974

A possible implementation is suggested of J H Ellis's proposed method ofencryption involving no sharing of secret information (key lists, machine set-ups, pluggings etc) between sender and receiver.

- Public parameter: a prime p
- Sender A encrypts m to Receiver B: $C_1 = m^x \bmod p$
  where x is A's private parameter
- Receiver B replies: $C_2 = C_1^y = m^{x*y} \bmod p$
  where y is B's private parameter
- Sender A computes and returns: $C_3 = C_2^{X'} = m^y \bmod p$
  where $X' * x \equiv 1 \pmod{p-1}$
- Receiver B decrypts m: $m = C_3^{Y'} \bmod p$
  where $Y' * y \equiv 1 \pmod{p-1}$

Security Consideration of RSA

- Three important/fundamental issues of system parameters

(1) How many primes in [2, x]?
- Is it easy to get the same p or q in RSA?
- Problem of "primes distribution" (next page)

(2) Is it easy to factorize $p*q$?

(3) Some precautions for RSA usage

Distribution of Primes

- The number of primes $\leqq x$: denoted as $\pi(x)$
  - Asymptotically, $\pi(x)=x/\ln x$

x	$x/\ln x$
2^{256}	10^{74}
2^{512}	10^{151}

- Density of prime: $1/\ln x$
  - consider only odd integer, density $\rightarrow 2/\ln x$
  - Ex: for x near 2^{512}

    $2/\ln x = 2/(512*\ln 2) = 1/177$

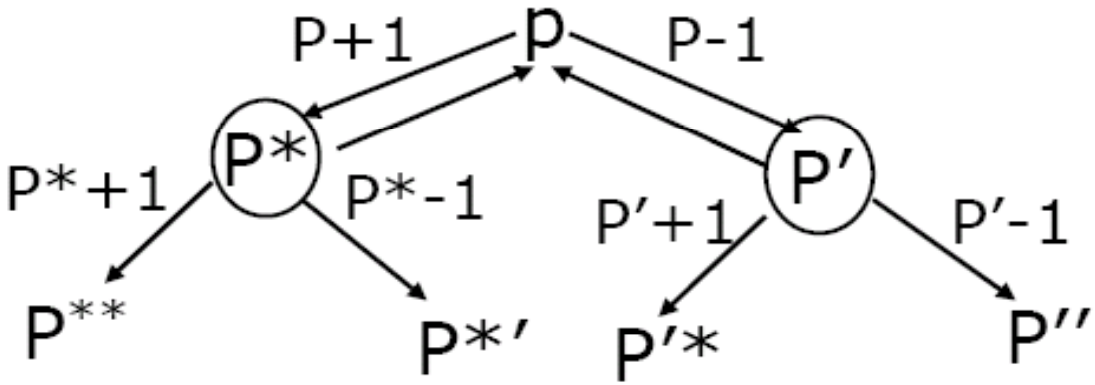
Factorization Problem

- The ways to attack RSA (given e and n)
  - factorize $n = p * q$ and hence obtain $\varphi(n)$ and d
  - determine $\varphi(n)$ directly and obtain d
    - this also reveals the factorization of n
  - find d directly
    - this usually also reveals the factorization
  - special cryptanalytic hardware
    - TWIRL (Shamir & Tromer): to cryptanalyze RSA-1024 in 6 weeks on a US$10M budget
- Factorization is still a hard problem
  - there is still small progress
    - quadratic sieve (old technique)
    - generalized number field (GNF) sieve

Security Precautions for RSA

- Some precautions of system parameters:

(1) strong prime: optional



(2) p & q had better differ in length by a few digits

(p & q should not be too close to each other)

$\because n=pq,\ (\frac{p+q}{2})^2 - pq = (\frac{p-q}{2})^2$

- Ex: Let p=13 and q=11, n=11*13=143
  $\sqrt{n} = \sqrt{143} = 11.958^{+} \approx \mathbf{12}$ (an easy problem!)
  $(X/2)^2 - 143 = (Y/2)^2$
  Suppose X=2*12=24

$$\therefore \left(\frac{24}{2}\right)^2 - 143 = 144 - 143 = 1^2 = \left(\frac{2}{2}\right)^2$$

$$\begin{cases} p + q = 24 \\ p - q = 2 \end{cases} \longrightarrow p=13 \text{ and } q=11$$

- Ex: Let p=3 and q=29, n=3*29=87
  Since $\sqrt{87}=9.33$, then we have to try $(p+q)/2$ from **10** until **16** (=(3+29)/2) to factorize n.

$$\left(\frac{32}{2}\right)^2 - 87 = 256 - 87 = 169 = 13^2 = \left(\frac{29-3}{2}\right)^2$$

(3) Of course, p and q should be large enough against factoring

(4) d can not be too small

(***why*** small? For performance)

Some attacks exist for $d < n^{1/4}$

- Some precautions of using RSA:

(1) Guessable message attack when encryption:

If a ***small set*** of possible message will be encrypted and sent to the receiver, the attacker cannot decrypt the ciphertext but he can just encrypt the possible message and compare it with the received ciphertext.

Note: The countermeasure

Deterministic encryption→ probabilistic encryption

(to ***concatenate*** the message with a large ***random*** number (e.g. 64 bits long))

(2) Encrypting message that is of much less the bit length of n where $e=3$

Ex: Let $|n|=512$ and $|m|=160$ then $|m^3|=480$

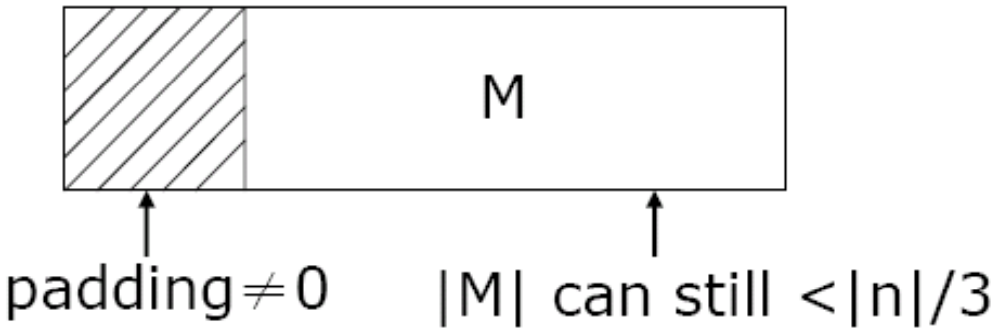
$\therefore C = m^3 \bmod n = m^3 \rightarrow \sqrt[3]{C} = m$ (an easy problem!)

Note: The countermeasure: random padding

To force $|m'|>|n|/3$

where

m' :



Note: given gcd(x,y)=1, x^{-1} (mod y) can be found

(3) Send the same message m to more than 3 recipients and all with e=3

Ex: m → $C_1 = m^3 \bmod n_1$, $C_2 = m^3 \bmod n_2$, $C_3 = m^3 \bmod n_3$ vs. $\boldsymbol{C} = m^3 \bmod (n_1 \times n_2 \times n_3)$

where n_1, n_2, and n_3 are pairwise relatively prime.

(i) Based on CRT, a number $\boldsymbol{C}$ in [0, $n_1 \times n_2 \times n_3$-1] can be found ∋ $C \bmod n_1 = C_1$; $C \bmod n_2 = C_2$; $C \bmod n_3 = C_3$

In fact, $\boldsymbol{C}$ is $\boldsymbol{m^3}$. Note that $m < n_1$, $m < n_2$, and $m < n_3$, therefore $m^3 < n_1 \times n_2 \times n_3$.

(ii) From m^3, $\sqrt[3]{m^3} = m$ (it is an easy problem).

(iii) Countermeasure: random padding

(4) Common-modulus attack (Why common?)

- Alice and Bob are assigned the same modulus n, but assigned different e_i & d_i. Suppose d_i is stored in a **tamper proof** device provided by the system manager.
  - (e_a, n): Alice's public key
  - (e_b, n): Bob's public key
  - suppose $\gcd(e_a, e_b)=1$
- If Sunny sends ciphers $c_i = m^{e_i} \bmod n$ to Alice and Bob
  - r & s can be easily found ∋ $re_a + se_b = 1$ by using extended Euclid's algorithm
  - then $(c_a)^r \times (c_b)^s \equiv m^{re_a+se_b} \equiv m \pmod n$ 1
- Countermeasure: random padding

(5) Blind-*signing* attack (→ **blind decryption**)

- Bob's signing/decryption key: d
  encryption key: (e, n)
- Alice sends $c = m^e \bmod n$ to Bob
- How to *decrypt* c by the assistance of Bob
  - attacker sends $x = r^e c \bmod n$ to Bob
    (r: selected by attacker, so he knows r^{-1})
  - gets signature: $(r^e * c)^d = r * c^d = r * m$
  - decrypts m by: $m = (r * m) * r^{-1}$
    $r^{-1} \pmod n$: by extended Euclid's algorithm
  - do NOT sign blindly if possible
- RSA blind signature as anonymous payment

(6) RSA signature forgery

- (a) **Existential** signature forgery:
  - any X is a signature on message m

    $m = X^e \bmod n$

    $\because \quad m^d \equiv (X^e)^d \equiv X \pmod{n}$
  - however, $m = X^e \bmod n$ will **not** be a meaningful message (or with extremely low probability)
  - **countermeasure**: use of **one-way** *hash* will improve security since it is hard (?) to find pre-image of m
    - say, given m to find $M \ni m = h(M)$
    - to sign by computing $S = (h(m))^d \bmod n$

- (6) RSA signature forgery (cont.)
  - (b) Smooth number and **multiplicative** attack
    - smooth number: product of reasonably small primes, Ex:81345=3×5×11×17×29
    - multiplicative attack:

given $\{m_1;\ s_1=m_1^d \bmod n\}$ $\{m_2;\ s_2=m_2^d \bmod n\}$

$\rightarrow \{(m_1 \times m_2 \bmod n);\ (s_1 \times s_2 \bmod n)\}$

$\rightarrow \{m_1^{-1} \bmod n;\ s_1^{-1} \bmod n\}$

$*m_1^{-1}$ & s_1^{-1} mod n can be computed easily

$\rightarrow \{m_1^j \bmod n;\ s_1^j \bmod n\}$

$\rightarrow \{(m_1^j \times m_2^k \bmod n);\ (s_1^j \times s_2^k \bmod n)\}$

- **countermeasure**: to sign by computing $S=(h(m))^d \bmod n$ since $h(m_1) \times h(m_2)=h(?)$

Rabin's Encryption & Signature Schemes

- Rabin's **encryption** scheme
  - The *ciphertext* C for message m is

$$C = m^2 \bmod n \quad \text{where } n = p * q,\ p \,\&\, q \equiv 3 \pmod 4$$

the *plaintext* is one of the **4** (or **2** if $\gcd(m,n) \neq 1$) solutions of $\sqrt{C} \pmod n$

    - format or redundancy is used to find m
    - A **try** but **fails (why?)**

      2-bit header is sent with C: (a,b)

      a: Jacobian symbol $J(m,n)$

      b: sign bit of m, $b=0$ if $0 \le m < n/2$, $b=1$ otherwise.

- Rabin's encryption scheme (cont.)
  - If u=CRT(x_1,x_2), -u=CRT(-x_1,-x_2), v=CRT(x_1,-x_2), and -v=CRT(-x_1,x_2) are the 4 square roots of C, then ***a* = J(*m*,*n*)** is used to distinguish ***u*** & ***v***. (or -***u*** & -***v***)
    - J(u,n)=J(u,p)*J(u,q)
    - J(u,q)=-J(v,q) if q≡**3** (mod 4) [same for p]
    - Theorem: Let q be an odd prime

      $L(-1,q)=(-1)^{(q-1)/2}$ = +1 if q≡1 (mod 4)

      = -1 if q≡3 (mod 4)
  - sign bit ***b*** is used to distinguish ***u*** and ***-u***.
  - Exactly **one** of u, v, -u, -v is in $\mathbf{QR}_n$ (to discuss later, see **Blum integer**)

Rabin's encryption scheme (cont.)

- A **valid** method to select m from 4 square roots of C is to use **replication** (part) of m itself or with a fixed format.
  - $C=(m||[\text{last 64 bits of } \boldsymbol{m}])^2 \bmod n$
  - $C=(m||[\text{64 bits of "0"}])^2 \bmod n$
  - $C=(m||[\text{64 bits of "1"}])^2 \bmod n$

    any better solutions?

  - $C=([\text{64 bits of "}\mathbf{1}\text{"}]||m)^2 \bmod n$

    Why padding on **MSB** & with leading "**1**"?

    Similar case as RSA with small e (page 27).
  - $C=([\text{32 bits }\mathbf{1}]||[\text{32-bit random}]||m)^2 \bmod n$

    Why random? For probabilistic encryption.

    Why not just 64-bit random with leading "1"?

Rabin's encryption scheme (cont.)

- Security of Rabin's encryption scheme (i)
  - It's a "provably secure" cryptosystem
    - "breaking Rabin's scheme" & "factoring *n*" are **computationally equivalent**
  - "factoring → breaking Rabin's" is trivial
  - If an **attacker** can break Rabin's scheme (i.e., he can compute $SQRT_n(C)$), then **we** can use him to factorize *n*.
    - We randomly compute $C=m_1^2 \bmod n$, then send *C* to the attacker and we receive one $m_2= SQRT_n(C)$. Then, $\gcd(m_1 \pm m_2, n)$ gives *p* or *q* with probability of 0.5.
  - Since factoring *n* is infeasible now, then Rabin's scheme is secure (unbreakable).

- Rabin's encryption scheme (cont.)
  - Security of Rabin's encryption scheme (ii)
    - $C=m^2 \bmod n$ is secure against passive (ciphertext-only) attack.
    - But, simple version $C=m^2 \bmod n$ might be vulnerable to "chosen-ciphertext" attack.
      - Attacker selects x, computes $C=x^2 \bmod n$, then asks "decryption oracle" the plaintext y. Attacker computes $\gcd(x \pm y, n)$ which gives p or q with probability of 0.5.
    - $C=(m||[\text{last 64 bits of } m])^2 \bmod n$ is secure against the above attack. **Why?**
      - How about other padding schemes?

- Rabin's **signature** scheme
  - The *signature* for message m is (S, x)

$$S = \sqrt{h(m,x)} \mod n$$

$$\text{where } n = p * q, \;\; p \;\&\; q \equiv 3 \pmod 4$$

    - x is a random integer selected to make $h(m,x)$ be in QR_n.
  - Security of Rabin's signature scheme
    - Existential forgery (as RSA): randomly select s, related message is $m = s^2 \bmod n$. If with $h()$?
    - Never sign on received random value because of the factorization attack if without hash.
      - **Blind signing** for RSA will **NOT** lead to factorization attack even without hash.

Rabin's schemes: Why p & $q \equiv 3 \pmod 4$?

- Let $a \in QR_p$ and $p \equiv 3 \pmod 4$, then square roots of a are

$$\pm\, a^{(p+1)/4} \bmod p$$

  - don't need algorithm in Handbook (p.100)

Proof: Since $a \in QR_p$, then $a^{(p-1)/2} \equiv 1 \bmod p$ and $a * a^{(p-1)/2} \equiv a^{(p+1)/2} \equiv a \pmod p$.

$p \equiv 3 \pmod 4$, so $p+1 \equiv 0 \pmod 4$ and $(p+1)/4$ is an integer. We can have $a^{(p+1)/4} \bmod p$ as a square root.

- After finding square roots mod p & mod q, CRT is used to compute message or signature.

Blum Integer & Its Application

- Blum integer: $n=p*q$ where $p\&q\equiv 3 \pmod 4$
  - If n is a Blum integer, then

SQ_n: $y=x^2 \bmod n$ ($x\in QR_n$; **QR_n**-->QR_n; permutation)

SQ_n^{-1} is a *trapdoor* permutation (**hard**)

only select $\sqrt{y} \in QR_n$ (principal square root)

- Why SQ_n is a bijective function?

If $x_1\neq x_2$, then $x_1^2\neq x_2^2 \pmod n$ for $p\&q\equiv 3 \pmod 4$

Proof by contradiction $x_1^2 - x_2^2\equiv 0 \pmod n$

(1) $x_1=x_2$ (contradiction!)

(2) $x_1=-x_2$: if $x_2\in QR_n$, then $-x_2$ is NOT! (p.34)

- Theorem: Let p & q be distinct odd primes and $n=p*q$. Then, $|QR_n|=(p-1)*(q-1)/4$.

  - Application for one-time password (How?)

- Given the trapdoor p & q: SQ_n^{-1} is **easy**

  $SQ_n^{-1}(y) = y^{[(p-1)(q-1)+4]/8} \bmod n$

  - Can $SQ_n^{-1}(y)$ be used in Rabin's **signature**?
    - Do we need it? (suppose $y \in QR_n$)
      - Answer:
        YES. We can **define** the signature to be a QR_n and use $SQ_n^{-1}(y)$ to sign.
        However, we can just use
        $\pm y^{(p+1)/4} \bmod p$ (& mod q) & CRT to have more efficient solution (**2** times efficient; need to check $y^{(p+1)/4} \in QR_p$ or NQR_p).
        But, if we apply CRT on $SQ_n^{-1}(y)$ directly then **4** times of speedup is possible. (p.55, 59)
  - Can $SQ_n^{-1}(y)$ be used in Rabin's **decryption**?
    - Answer: **NO**. Because $SQ_n^{-1}(y)$ cannot always reach the solution with format (m||[last 64 bits of m])

Primality Test – Algorithm & Skill

Types of Primality Test

- Traditional trial division approach
  - divided by all primes less than $n^{1/2}$
  - time complexity $\rightarrow O(n^{1/2})$
  - infeasible for very large n

- Probabilistic approach
  - based on number theoretic properties of
    - **prime** number
    - **composite** number (optional)
  - but pseudo-primes (composite number) sometimes pass the test

A First Try -- Fermat's Test

- Based on Fermat's theorem, if n is prime, then always $a^{n-1} \bmod n = 1 \quad (a<n)$
- On the other hand,
- if $\boldsymbol{a}^{n-1} \bmod n \neq 1$ $(1<a<n-1)$, n is composite
- but for some $\boldsymbol{a}$, $\boldsymbol{a}^{n-1} \bmod n = 1$, n might be prime or *pseudoprime*
  - if pseudoprime, the random number $\boldsymbol{a}$ is called a *liar*
- Why Fermat's test is not popular?
- **1st disadvantage**: at most **1/2** of a in $1<a<n-1$ are liars (cf. **1/4** of Miller-Rabin test on p.49)
- 2nd disadvantage: Carmichael numbers (most a are liars, but Carmichael number is of rare case)

- Carmichael numbers (smallest one is 561)
  http://en.wikipedia.org/wiki/Carmichael_number
  - **Definition 1**: composite integer n, but $a^{n-1} \bmod n = 1$ for all $\gcd(a, n)=1$
    - for n with most prime factors *large*, $\varphi(n)$ is a large value → most a are *liars*!
  - Definition 2: composite number n is a Carmichael number iff it is square-free and $p_i - 1 | n-1$ for all prime divisors p_i of n
- Relationship between Def. 1 & 2:
  - For any composite n & $\gcd(a, n)=1$, the order of a always divides lcm(all p_i -1);
    refer to next page (also notice: lcm vs. $\varphi(n)$)
  - For Carmichael number, lcm(all p_i -1)$|n-1$
    So $a^{n-1} \bmod n = 1$

$a^i \bmod 21$; $21=3*7$; **lcm**(ø(3),ø(7))=lcm(2,6)=6

has
4 liars

a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4
3	3	9	6	18	12	15	3	9	6	18	12	15	3	9	6	18	12	15	3	9
4	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16
5	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20	16	17	1	5	4
6	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1
9	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15	9	18
10	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13	4	19	1	10	16
11	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8	4	2	1	11	16
12	12	18	6	9	3	15	12	18	6	9	3	15	12	18	6	9	3	15	12	18
13	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1
14	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
16	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4
17	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20	4	5	1	17	16
18	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15	18	9
19	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13	16	10	1	19	4
20	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1

- Example of Carmichael numbers
  - **561**=3*11*17; 2|560, 10|560, 16|560
    1105=5*13*17; 4|1104, 12|1104, 16|1104
  - (6*k*+1)*(12*k*+1)*(18*k*+1) is a Carmichael number if its three factors are all prime

smallest Carmichael number

- Distribution of Carmichael numbers
  - Carmichael numbers are substantially rarer than prime numbers
  - *C*(*X*): number of Carmichael numbers ≤ *X*

n	3	4	5	6	7	8	9	10	11
C(10^n)	1	7	16	43	105	255	646	1547	3605
	12	13	14	15	16	17	18	19	20
	8241	19279	44706	105212	246683	585355	1401644	3381806	8220777

Miller-Rabin Test

- Basic property of prime number

  Let n be an odd **prime**, and $n-1=2^s r$ where r is odd. Let a be any integer $\gcd(a, n)=1$, then

  - $a^{(r*2^s)} \equiv \mathbf{1} \pmod{n}$, or $n \mid a^{(r*2^s)}-1$
    - $a^{(r*2^s)} \equiv \mathbf{1} \pmod{n}$ due to Fermat's Th.

  so $n \mid a^{(r*2^s)}-1$ $\qquad x^2-1=(x+1)*(x-1)$

  $n \mid (a^{(r*2^{s-1})}+1)*(a^{(r*2^{s-1})}-1)$

  $n \mid (a^{(r*2^{s-1})}+1)*(a^{(r*2^{s-2})}+1)*\ldots*(a^r+1)*(a^r-1)$

  Therefore,

  - $a^r \equiv \mathbf{1} \pmod{n}$ due to $n \mid (a^r-1)$ or
  - $a^{(r*2^j)} \equiv \mathbf{-1} \pmod{n}$ for some j, $\mathbf{0} \le j \le s-1$ due to $n \mid (a^{(r*2^j)}+1)$

- Probability of liar
  - if n is an odd **composite** integer, but the above two properties are satisfied

    $\rightarrow$ n is a *pseudoprime* and a is a *liar*
  - less than **1/4** of a in $1<a<n-1$ are liars for n

    $P_{pp} < 1/4$
    - more precisely: at most $\varphi(n)/4$ liars
  - Ex. of pseudoprime $n=21$
    - Miller-Rabin test on next page: with only 2 liars ($a=1, -1$)
    - cf. Fermat's test on p.46: with 4 liars ($a=1, 8, 13, -1$)

a^i mod 21; 21=3*7; ø(3*7)/4=3; $a^5=\pm1$ or $a^{10}=-1$

only 2 liars

a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4
3	3	9	6	18	12	15	3	9	6	18	12	15	3	9	6	18	12	15	3	9
4	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16
5	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20	16	17	1	5	4
6	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1
9	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15	9	18
10	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13	4	19	1	10	16
11	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8	4	2	1	11	16
12	12	18	6	9	3	15	12	18	6	9	3	15	12	18	6	9	3	15	12	18
13	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1
14	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
16	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4
17	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20	4	5	1	17	16
18	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15	18	9
19	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13	16	10	1	19	4
20	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1

- Basic property of composite number
  (optional: because of extremely low probability)

Let n be an odd integer, if

$$a \neq \pm 1 \pmod{n}$$

but $a^2 = 1 \pmod{n}$.

Then, n is an odd **composite** number.

- gcd($a \pm 1$, n) is a nontrivial factor of n.
  Ex: See last page, let n=3*7, $a^2 = 1 \pmod{n}$ has two nontrivial square roots 8 & 13. gcd(8-1, 21)=7 & gcd(8+1, 21)=3 & gcd(13-1, 21)=3 & gcd(13+1, 21)=7.

- The Miller-Rabin algorithm

Input: n & security parameter $t \geq 1$

let $n-1=2^s r$ where r is odd

For $i=1$ to t do

randomly select base a

$y=a^r \bmod n$

If ($y \neq 1$ and $y \neq -1$) then

$j=1$ ← it's not "0"

While ($j \leq s-1$ and $y \neq -1$) do

$y=y^2 \bmod n$

if ($y==1$) then return "n **IS** composite"

$j=j+1$

If ($y \neq -1$) then Return "n **IS** composite"

Return "**MAYBE** n is prime"

property of composite number

* NOTE: **most** implementation does not consider this.
* Let y' be previous y, then **gcd(y'-1,n)** is a nontrivial factor of n.

property of prime number

Probability Consideration

- In Miller-Rabin test (with t iterations)
  - if return "*n* **IS** composite"

    → *n* is 100% **not** *prime* by Fermat's Theorem or the property of composite number
  - if return "**MAYBE** *n* is prime"

    → *n* can be a *prime* or a *pseudoprime* by property of prime number & existence of liars
    - probability of *n* to be a prime
      $P_{pp}^{t} < (1/4)^{t}$ → $1\text{-}P_{pp}^{t} > 1 - (1/4)^{t}$

t =	probability of *n* to be a prime
10	$0.999999 = 1\text{-}10^{-6}$
20	$0.99999999999 = 1\text{-}10^{-12}$
30	$0.999999999999999 = 1\text{-}10^{-18}$
40	$0.999999999999999999999999 = 1\text{-}10^{-24}$

Trial Division Sieve

- Useful pre-sieve skill before Miller-Rabin test
- The probability $Q(x) = \prod_{p \in Px} (1 - \frac{1}{p})$
  where p_x is the set of all primes $\leq x$
  - $Q(x)$: the probability for n to be relatively prime with ***all*** primes in p_x when $n \gg x$
  - let T be product of all elements in p_x
    - T is about **9KB** for $x=10^3$
- Pre-sieve: test gcd(T, n) = 1?

x	Q(x)
10	0.229
10^2	0.120
10^3	0.081
10^4	0.061
10^5	0.049
10^6	0.041

$0.081^{-1}=12.35$

RSA Speedup with CRT

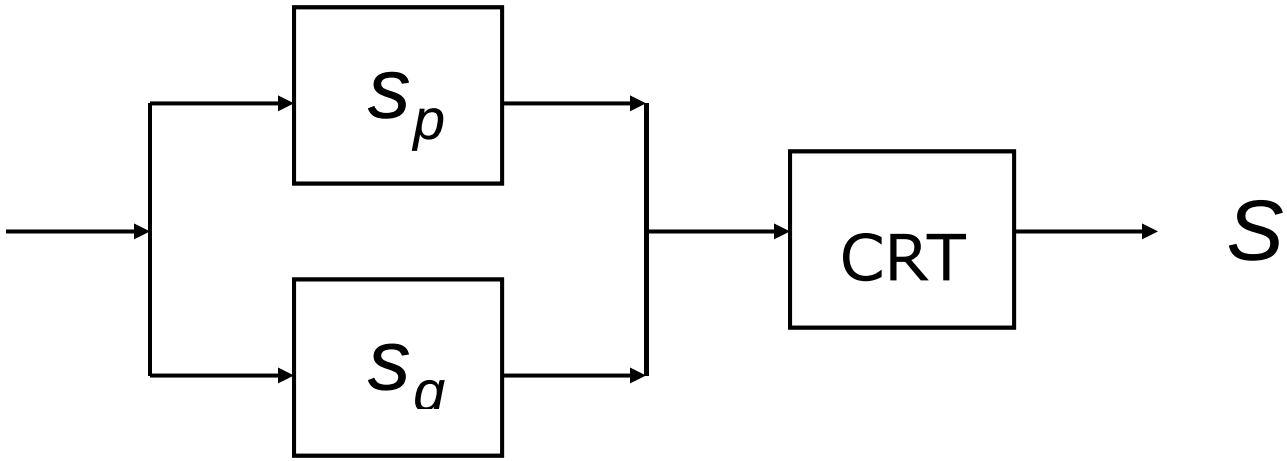
RSA speedup based on CRT:

- Given p, q, ($n=p*q$), d, and m,

  $S=m^d \bmod n$ can be sped up by

$$s_p=(m \bmod p)^{d \bmod (p-1)} \bmod p$$

$$s_q=(m \bmod q)^{d \bmod (q-1)} \bmod q$$



Why RSA with CRT?

- Let $|p|=|q|=512$ and $|n|=1024$
  Also, $|m| \approx 1024$ and $|d| \approx 1024$
  - Now, $|m_p| \approx |m_q| \approx 512$ and $|d_p| \approx |d_q| \approx 512$
  - **Half** the number of "multiplication" for $m_p^{d_p} \bmod p$
  - **Half** the number of "addition" for $m_p * m_p$
  - **Half** the "ripple carry" length of each ***addition***

- Totally, $(2^3)/2 =$ **4** times improvement can be achieved.

CRT recombination algorithms:

- Gauss's CRT recombination
  - a standard representation but it takes more memory space & time

$$S=\mathrm{CRT}(s_p, s_q)$$
$$=[s_p \times \underline{q \times (q^{-1} \bmod p)} + s_q \times \underline{p \times (p^{-1} \bmod q)}] \bmod n$$
$$= [s_p \times X_p + s_q \times X_q] \bmod n$$

- Garner's CRT recombination
  - widely used because it takes ***fewer memory space* (=1/4) & *time* (<1/2)**

$$S=\mathrm{CRT}(s_p, s_q)$$
$$=\{s_q + [(s_p - s_q) \times \underline{(q^{-1} \bmod p)}] \times q\} \bmod n$$
$$= s_q + [(s_p - s_q) \times \underline{(q^{-1} \bmod p)} \ \mathbf{mod\ p}] \times q$$

max=n-1, don't need mod n

Basic Exponentiation Algorithms

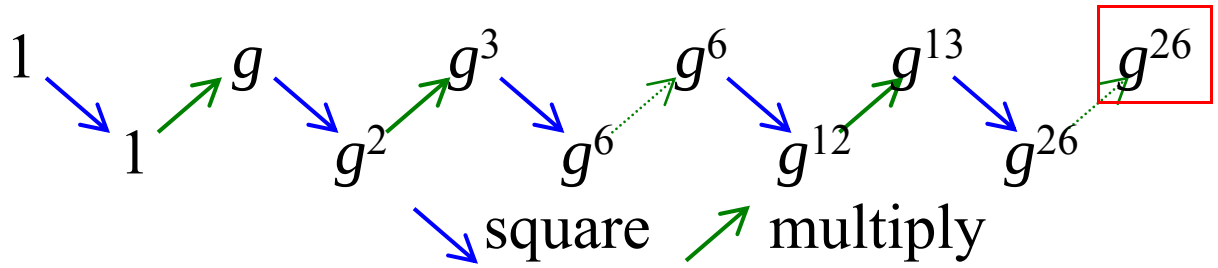
Exponentiation Algorithms

- Efficient method to compute g^d
- *Exponentiation* algorithms are basically classified into two categories:
  1. *Left-to-Right* (*MSB-to-LSB*) algorithms
  2. *Right-to-Left* (*LSB-to-MSB*) algorithms

L-to-R Binary Exponentiation

- Example: g^{26}

d=26: $(1\ 1\ 0\ 1\ 0)_2$



- Main idea -- Horner's rule: Given d

$$d = \sum_{i=0}^{k-1}(d_i \times 2^i) = (\ldots((d_{k-1} \times 2) + d_{k-2}) \times 2 + \ldots + d_1) \times 2 + d_0$$

g^d can be computed as

$$g^d = (\ldots((g^{d_{k-1}})^2 \times g^{d_{k-2}})^2 \times \ldots \times g^{d_1})^2 \times g^{d_0}$$

L-to-R Binary Exponentiation

- Algorithm:

```
01:  R = 1
02:  for i = (k−1) downto 0
03:      R = R^2
04:      if (d_i == 1) then R = R × g
05:  return R
```

  - Performance: k squarings + on average $(k/2)$ multiplications

R-to-L Binary Exponentiation

- Example: g^{26}

$d = 26$: (1 1 0 1 0)

g^{16} g^{8} g^{4} g^{2} g

$g^{26} = g^{16} \times g^{8} \times g^{2}$

- Main idea: Given exponent $d = \sum_{i=0}^{k-1}(d_i \times 2^i)$, g^d can be computed as

$$g^d = g^{\sum_{i=0}^{k-1}(d_i \times 2^i)} = \prod_{i=0}^{k-1}(g^{2^i})^{d_i}$$

R-to-L Binary Exponentiation

- Algorithm:

```
01:  R = 1, T = g
02:  for i = 0 to (k−1)
03:      if (d_i == 1) then R = R×T
04:      T = T^2
05:  return R
```

  - Performance: k squarings + on average $(k/2)$ multiplications

Back to RSA+CRT

L-to-R Windowing Method – Performance Enhancement

- 2^w-ary method: scan w bits in each iteration
  - to reduce the number of *multiplications*
- The algorithm: (w=2)

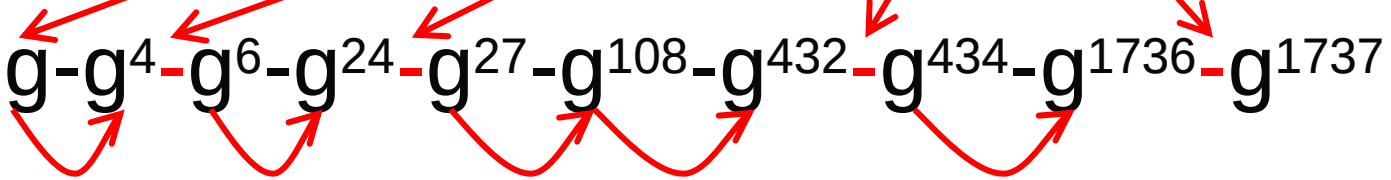
01: $R = 1$; precompute: g^2, g^3 (look-up table)
02: for $i = 2 \times \lfloor (k\text{-}1)/2 \rfloor$ downto 0 step -2
03: $R = R^4$
04: if $((d_{i+1}d_i)_2 \neq 0)$ then $R = R \times g^{(d_{i+1}d_i)_2}$
05: return R

- Performance: k squarings + on average (**3/4**)*(k/2) multiplications

- Example of 2^w-ary method (w=2), 4-ary:

$(1\ \ 2\ \ 3\ \ 0\ \ 2\ \ 1)_4$

d = 1737: $(\underline{1}\ \underline{10}\ \underline{11}\ \underline{00}\ \underline{10}\ \underline{01})_2$



R-to-L Windowing Method – Approach 1 (not good)

- Example (w=2)

$1\times4^5 + 2\times4^4 + 3\times4^3 \quad + 2\times4^1 + 1\times4^0$

$(1 \quad 2 \quad 3 \quad 0 \quad 2 \quad 1)_4$

$d = (\underline{1} \quad \underline{10} \quad \underline{11} \quad \underline{00} \quad \underline{10} \quad \underline{01})_2 = 1737$

g^{1024}	g^{256}	g^{64}	g^{16}	g^{4}	g	for $x^{(01)}$
g^{2048}	g^{512}	g^{128}	g^{32}	g^{8}	g^{2}	for $y^{(10)}$
g^{3072}	g^{768}	g^{192}	g^{48}	g^{12}	g^{3}	for $z^{(11)}$

$g^{1024} \times g^{512} \times g^{192} \quad \times g^{8} \times g = g^{1737}$

- Performance: extremely inefficient

R-to-L Windowing Method – Approach 2

- Example (w=2)

$$d = (\underbrace{1}_{g^{1024}}\ \underbrace{10}_{g^{256}}\ \underbrace{11}_{g^{64}}\ \underbrace{00}_{g^{16}}\ \underbrace{10}_{g^{4}}\ \underbrace{01})_2 = 1737 \quad \text{for } x^{(01)}$$

$$(g^{1024} \times g)^1 = g^{1025}$$

$$(g^{256} \times g^4)^2 = g^{520}$$

$$(g^{64})^3 = g^{192}$$

$$g^{1025} \times g^{520} \times g^{192} = g^{1737}$$

  - Performance:
    - much better than approach 1
    - asymptotically similar to L-to-R version
    - however, a little worse than L-to-R version

Sliding Window Method (L-to-R)

- Main ideas: (reduce both *space* & *time*)
  - Reduce size of look-up table (space)
    - Example (window size=2):
      only pre-compute g^3 (((1×4)+2)×4+3)×4 ....
      $d = (1\ \mathbf{1\ 0}\ 1\ 1\ 0\ 0\ \mathbf{1\ 0}\ 0\ 1)_2$ (fixed window)
      $d = (1\ 1\ \mathbf{0}\ 1\ 1\ 0\ 0\ 1\ \mathbf{0\ 0}\ 1)_2$ (sliding window)
      ((((3×2)×4+3)×2)×2)×2)+1 ....

Key point: mix of binary & 4-ary

  - Reduce # of multiplications (time)
    - Example (window size=2):
      bypassing unnecessary multiplication
      ~~ 0 1 **1 0** ~~ (fixed window)
      ~~ **0** 1 1 **0** ~~ (sliding window)
  - Performance (***w***-bit window): *k* squarings + on average ***k/(w+1)*** multiplications

Performance Comparison

K/(2×4/3)
= K/(2+2/3)

- Performance of exponentiation algorithms

Algorithm	Table Size	Squaring	Multiplication	
			Average	Worse
Right-to-Left	1	k	$k/2$	k
Left-to-Right	**1 (or No)**	k	$\boldsymbol{k/2}$	k
L-to-R (2-bit)	3 (or 2)	k	$\frac{3}{4} \times k/2 = 3k/8$	$k/2$
L-to-R (w-bit)	2^w-1 (or 2^w-2)	k	$(2^w-1)/2^w \times k/w$	k/w
Sliding (2-bit)	**2 (or 1)**	k or (k-2)	$\boldsymbol{k/(2+1)}$	$k/2$
Sliding (w-bit)	2^{w-1} or $(2^{w-1}-1)$	k or (k-w)	$\boldsymbol{k/(w+1)}$	k/w

Multiple Exponentiation

- How to compute $g^a h^b$ efficiently?
  - useful to implement many important cryptosystems: e.g., **DSA**, ElGamal, Schnorr
  - **not** to compute g^a and h^b **individually**
  - **square together** & **multiply together**; pre-compute $g \times h$ (if window size $w=1$)
  - key point: Horner's rule in vector form
- Example: (for $w=1$)

$a = (1\ 0\ 1\ 0\ 1)_2 = 21$

$b = (1\ 1\ 1\ 0\ 0)_2 = 28$

$gh \cdot g^2h^2 \cdot g^2h^3 \cdot g^4h^6 \cdot g^5h^7 \cdot g^{10}h^{14} \cdot g^{20}h^{28} \cdot g^{21}h^{28}$

Sliding Window Multiple Exponentiation

- How to compute $g^a h^b$ even **more** efficiently?
  - reduce size of look-up table (if window size $w>1$)
  - reduce number of multiplications
- Performance
  - k-bit exponents; ***w***-bit window size; t-exponentiations (e.g., $g^a h^b$ has $t=2$)
  - it takes: k squarings + on average $\boldsymbol{k/(w+1/(2^t-1))}$ multiplications [from Yen94]