Information Security

Ch9: Public-key cryptography and RSA

Howon Kim

2022.4

- 정보보호 및 사물지능(AIoT) 연구실 http://infosec.pusan.ac.kr
- 블록체인 플랫폼 연구센터 http://itrc.pusan.ac.kr
- 지능형 융합보안대학원 http://aisec.pusan.ac.kr

정보보호 및 사물지능 연구실
Information Security & AIoT (AI of Things)

블록체인 플랫폼 연구센터
BLOCKCHAIN PLATFORM RESEARCH CENTER

부산대학교 융합보안대학원
PNU GRADUATE SCHOOL OF CONVERGENCE SECURITY

Agenda

- Principles of public key cryptosystems
- The RSA algorithm
- Exponentiation on RSA
- Attacks on RSA

Private-Key Cryptography

- Traditional cryptography uses a single key.
  - The key is shared by both the sender and the receiver.
    - Symmetric Key Cryptography
    - Conventional Cryptography
  - If this key is disclosed, communications are compromised.
    - Secret Key Cryptography
- Key distribution problem
  - How to share the secret key?
  - The number of keys is n(n-1)/2 for n parties.

Public-Key Cryptography

- PKC is probably most significant advance in the 3,000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- PKC complements **rather than** replaces private key crypto algorithm
- **Properties**
  - PKC is based on hard problems.
  - algebraic operations on large numbers
  - poor performance compared with the symmetric key system

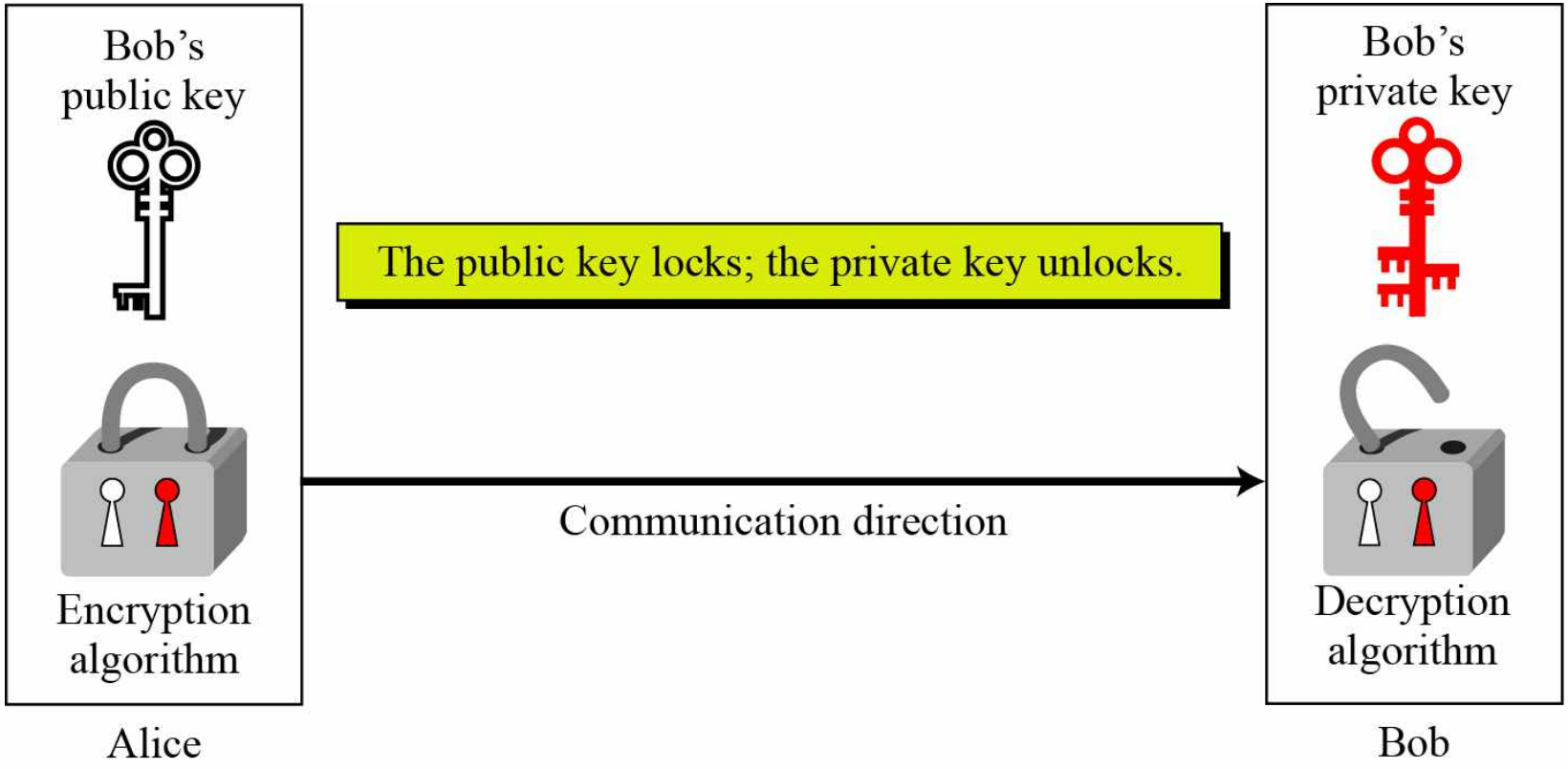
Public-Key Cryptography

Who uses this key for this purpose ? (owner?/others?)

- **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

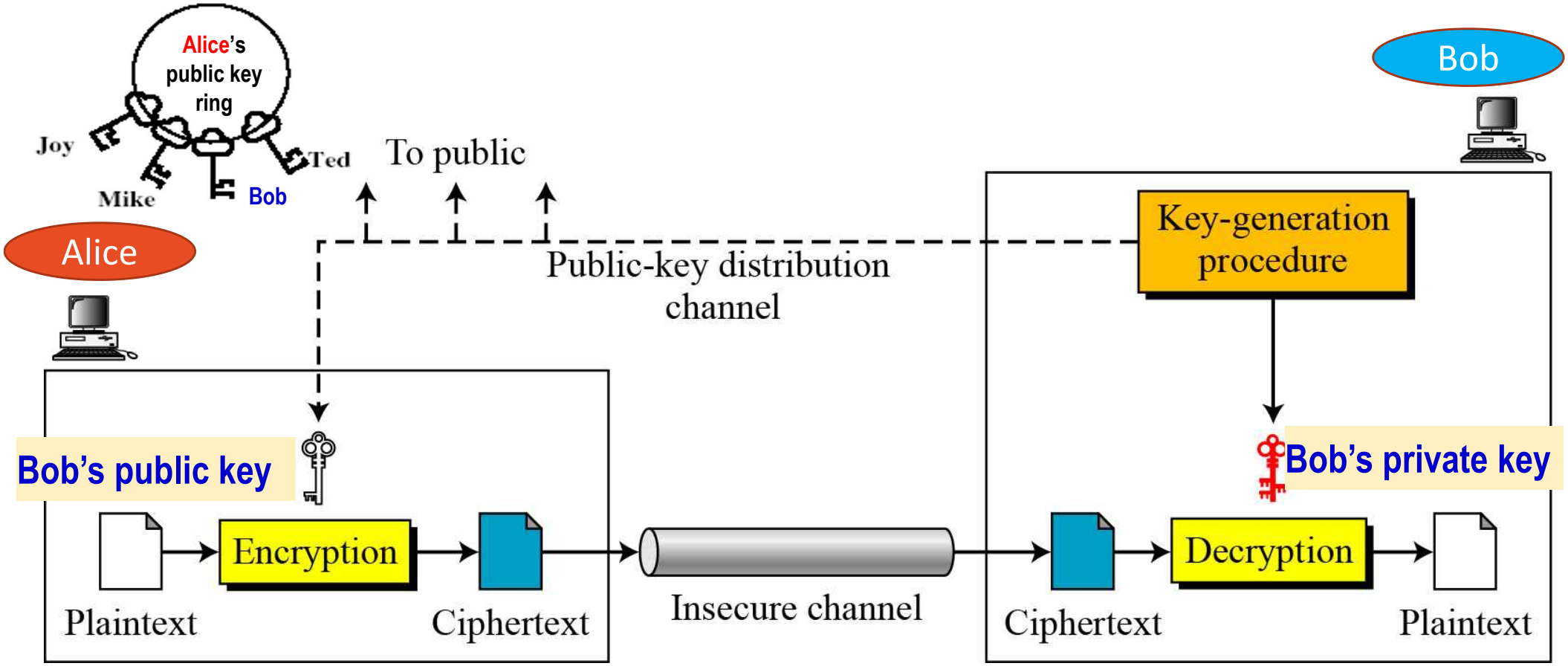
Public-Key Cryptography

- Asymmetric key cryptography uses two separate keys: one private and one public.



Public-Key Cryptography

- General idea of asymmetric-key cryptosystem



Public-Key Cryptography

- Plaintext/Ciphertext
  - Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers in asymmetric-key cryptography.

- Encryption/Decryption

$$C = f(K_{public}, P) \qquad P = g(K_{private}, C)$$

There is a very important fact that is sometimes misunderstood: The advent of asymmetric-key cryptography does not eliminate the need for symmetric-key cryptography.

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 (not altered or damaged) from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Univ. in 1976
  - known earlier in classified community

PKC: 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

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

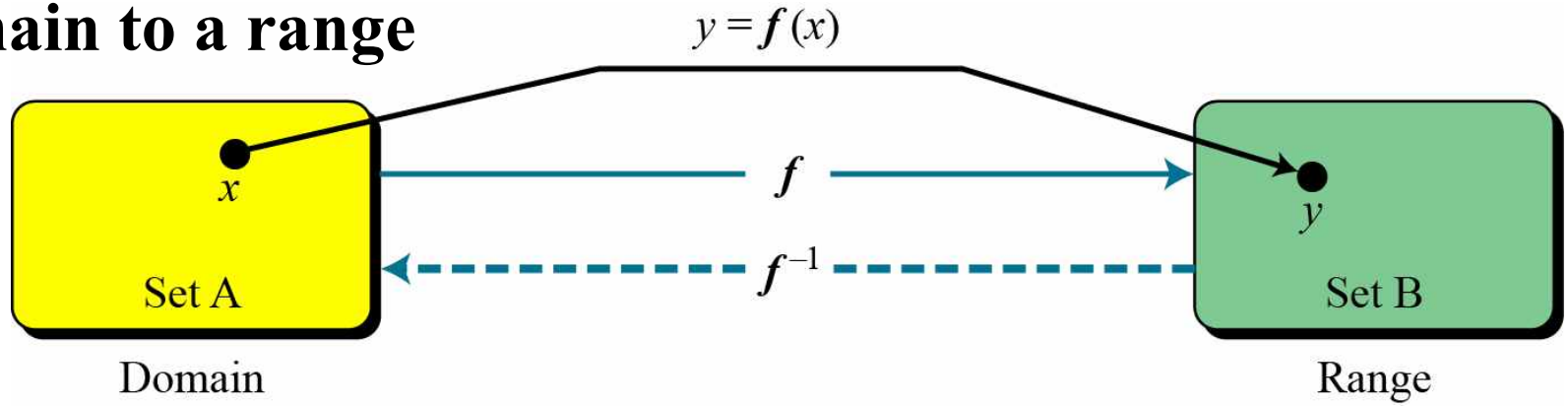
PKC: requirements

- 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)
- these are formidable requirements which only a few algorithms have satisfied

PKC: requirements – trapdoor one-way function

The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function.

Trapdoor one-way function: A function as rule mapping a domain to a range



One-Way Function (OWF)

1. f is easy to compute.
2. f^{-1} is difficult to compute.

Trapdoor One-Way Function (TOWF)

3. Given y and a trapdoor, x can be computed easily.

PKC: requirements – trapdoor one-way function

Example 1

- When n is large, $n = p \times q$ is a one-way function.
- Given p and q , it is always easy to calculate n ; given n, it is very difficult to compute p and q.
- **This is the factorization problem.**

Example 2

- When n is large, the function $y = x^k \bmod n$ is a trapdoor one-way function.
- Given x, k, and n, it is easy to calculate y. Given y, k, and n, it is very difficult to calculate x.
- **This is the discrete logarithm problem**.
- However, if we know the trapdoor, k′ such that $k \times k' = 1 \bmod \phi(n)$, we can use $x = y^{k'} \bmod n$ to find x.

$$y^{k'} \rightarrow x^{k k'} \rightarrow x$$

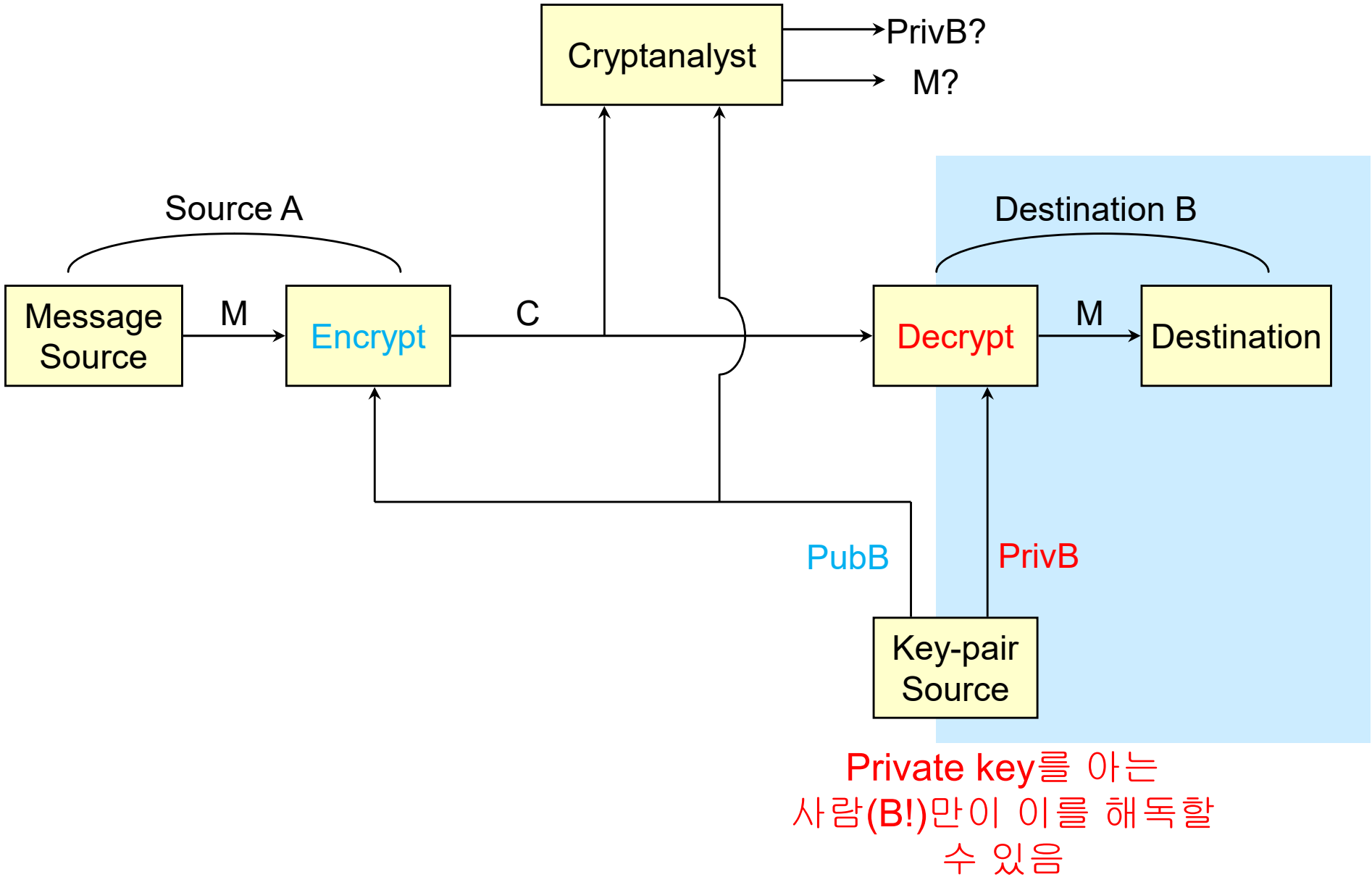
PKC: requirements

- need a trapdoor one-way function
- one-way function has
  - Y = f(X) easy
  - X = $f^{-1}(Y)$ infeasible
- a trap-door one-way function has
  - Y = $f_k(X)$ easy, if k and X are known
  - X = $f_k^{-1}(Y)$ easy, if k and Y are known
  - X = $f_k^{-1}(Y)$ infeasible, if Y known but k not known
- a practical public-key scheme depends on a suitable trap-door one-way function

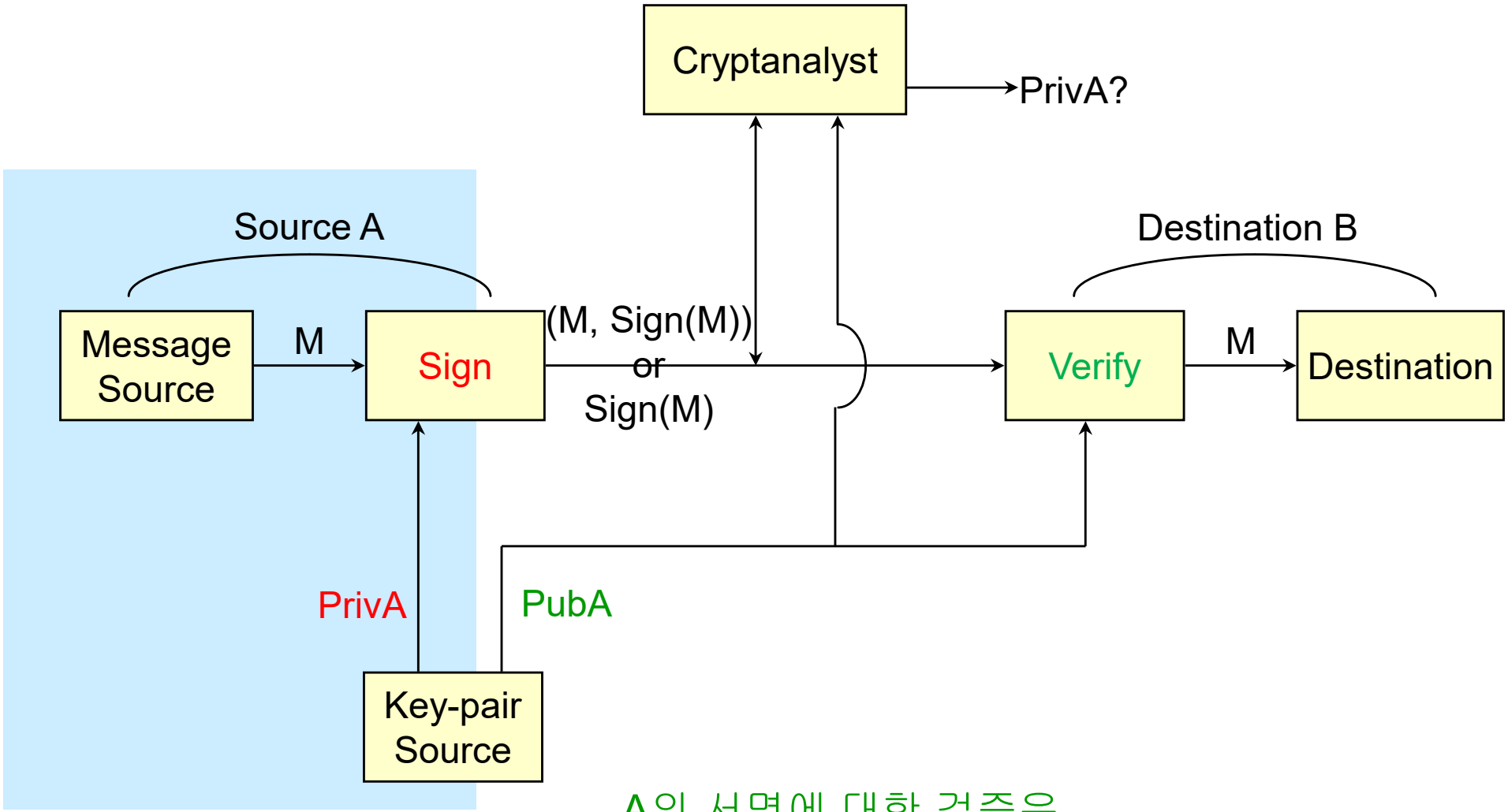
Security of Public Key Schemes

- 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

PKC: Encryption / Decryption



PKC: Digital Signature



A의 서명에 대한 검증은
A의 공개키로 이뤄짐

Terminology related to Asymmetric Encryption

Asymmetric Keys

Two related keys, a public key and a private key that are used to perform complementary operations, such as encryption and decryption or signature generation and signature verification.

Public Key Certificate

A digital document issued and digitally signed by the private key of a Certification Authority that binds the name of a subscriber to a public key. The certificate indicates that the subscriber identified in the certificate has sole control and access to the corresponding private key.

Public Key (Asymmetric) Cryptographic Algorithm

A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that deriving the private key from the public key is computationally infeasible.

Public Key Infrastructure (PKI)

A set of policies, processes, server platforms, software and workstations used for the purpose of administering certificates and public-private key pairs, including the ability to issue, maintain, and revoke public key certificates.

Symmetric vs. Public Key

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

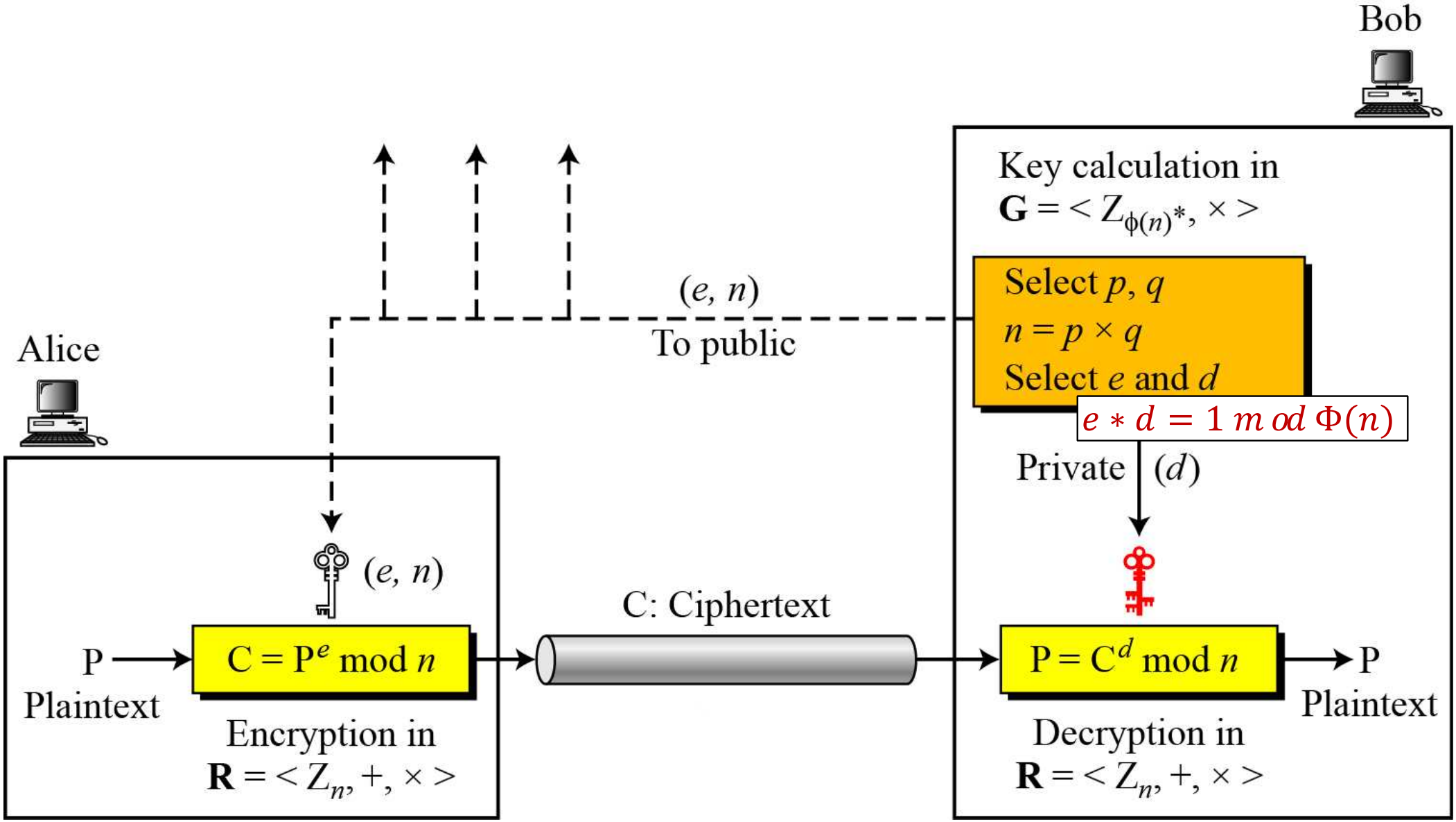
Agenda

- Principles of public key cryptosystems
- The RSA algorithm
- Exponentiation on RSA
- Attacks on RSA

RSA

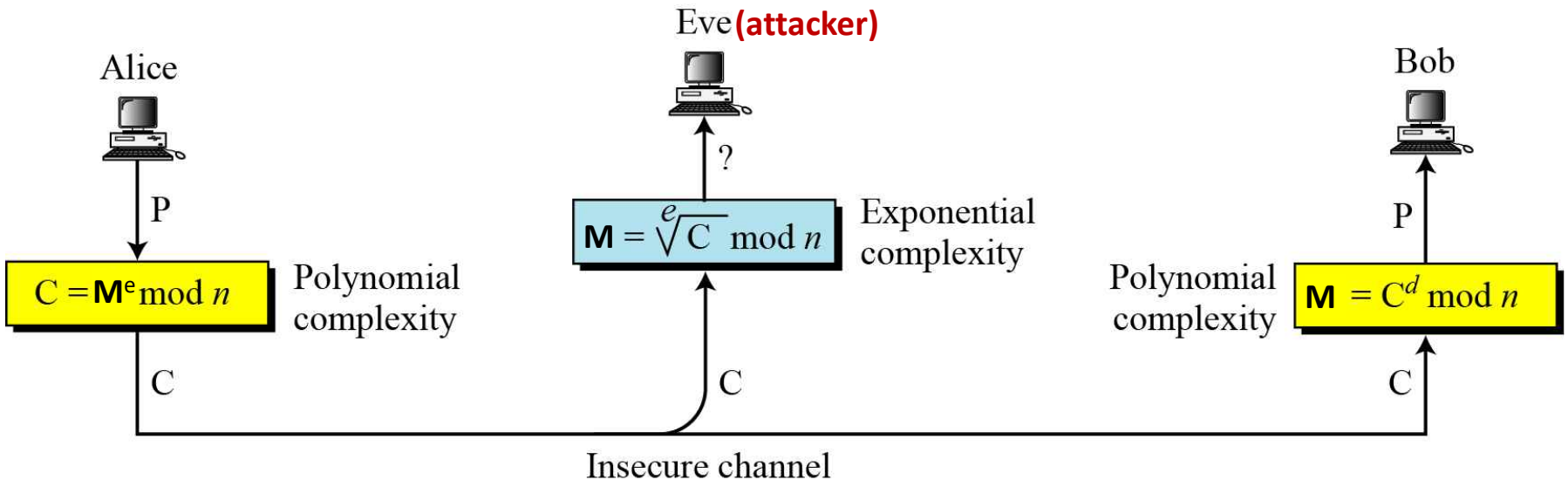
- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring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

RSA


Bob
Key calculation in
$\mathbf{G} = < Z_{\phi(n)}^*, \times >$
Select p, q
$n = p \times q$
Select e and d
$e * d = 1 \bmod \Phi(n)$
Private (d)
(e, n)
To public
Alice
(e, n)
C: Ciphertext
P
Plaintext
$C = P^e \bmod n$
Encryption in
$\mathbf{R} = < Z_n, +, \times >$
$P = C^d \bmod n$
Decryption in
$\mathbf{R} = < Z_n, +, \times >$
P
Plaintext

RSA

- **Complexity of operations in RSA**



RSA uses modular exponentiation for encryption/decryption; To attack it, Eve needs to calculate $\sqrt[e]{C} \bmod n$.

RSA

- **Two Algebraic Structures**

Encryption/Decryption Ring: $R = <Z_n, +, \times>$

Key-Generation Group: $G = <Z^*_{\varphi(n)}, \times>$

RSA uses two algebraic structures:
a public ring $R = <Z_n, +, \times>$ and a private group $G = <Z_{\phi(n)}^*, \times>$.

In RSA, the tuple (e, n) is the public key; the integer d is the private key.

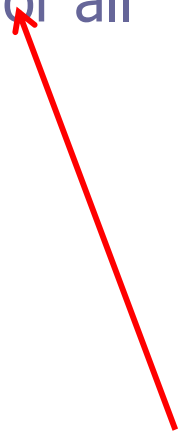
RSA

- Plaintext is encrypted in blocks, with each block having a binary value less than some number n
- That is, the block size is less than or equal to $\log_2(n)$
- The bock size is i bits, where $2^i < n <= 2^{i+1}$
- Encryption is done as follows: (sender)
  - computes: $C=M^e \bmod n$
- Decryption is done as follows: (receiver)
  - computes: $M=C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$
- Both sender & receiver must know the value of n
- Thus, this is the PKC with public key of $PU=\{e\}$ and a private key of $PR=\{d\}$

Actually, PR=(d,p,q}

RSA

- For this algorithm to be satisfactory for PKC, the following requirements must be met:
  1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$.
  2. It is relatively easy to calculate mod $M^e \bmod n$ and C^d for all values of $M < n$.
  3. It is infeasible to determine d given e and n.



$ed = \phi(n) + 1 \rightarrow ed \bmod \phi(n) = 1$
즉, $M^{\phi(n)+1} = M$ 이므로

RSA

- It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$.
  - From the euler function, $a^{\varnothing(n)+1} = a \pmod n$, where $\gcd(a,n)=1$, $ed=\varnothing(n)+1$ → $ed \bmod \varnothing(n) = 1$
  - This is equivalent to saying. $ed = 1 \bmod \varnothing(n)$. $d=e^{-1} \bmod \varnothing(n)$
  - That is, e and d are multiplicative inverses $\bmod \varnothing(n)$.
  - Note that, according to the rules of modular arithmetic, e is relatively prime to $\varnothing(n)$. that is, $\gcd(\varnothing(n),e)=1$. Also, equivalently, $\gcd(\varnothing(n),d)=1$.

아니면 e는 $\varnothing(n)$과 약수를 가져서 $ed \bmod \varnothing(n) = 0$ 이 됨

RSA

- Now, we ready to state the RSA scheme
  - **p, q,** two prime numbers → private, chosen
  - n = pq → public, calculated
  - e, with gcd(ø(n), e)=1; 1<e<ø(n) → public, chosen
  - **d**=e^{-1}(mod ø(n)) → private, calculated

※ Private key PR= **{d, p, q}**, public key PU= {e, n}

RSA Key Setup

RSA_Key_Generation

{

Select two large primes p and q such that $p \neq q$.

$n \leftarrow p \times q$

$\phi(n) \leftarrow (p-1) \times (q-1)$

Select e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$

$d \leftarrow e^{-1} \bmod \phi(n)$ // d is inverse of e modulo $\phi(n)$

Public_key $\leftarrow$ (e, n) // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

RSA Key Setup - Example

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

RSA En/Decryption

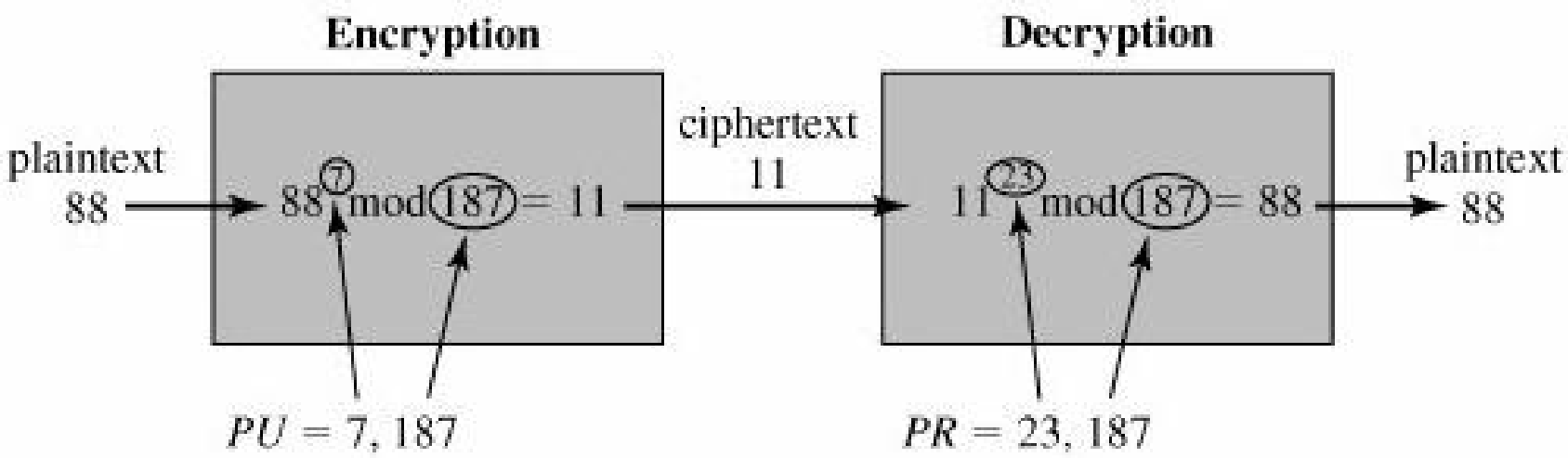
- Suppose that A has published its public key, $PU = \{e, n\}$ and user B wishes to send the message M to A.
- Then B calculates $C = M^e \bmod n$ and transmits C. Here, $M < n$
- On receipt of this ciphertext, user A decrypts by calculating $M = C^d \bmod n$.

RSA En/Decryption - Example

- Example of RSA encryption/decryption.
- Key : PU={7,187}, PR={23,17,11}
- Given message M = 88 (note, 88<187)
- Encryption:

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

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



Proof of RSA

If $n = p \times q$, $a < n$, and k is an integer, then $a^{k \times \phi(n)+1} \equiv a \pmod{n}$.

$P_1 = C^d \bmod n = (P^e \bmod n)^d \bmod n = P^{ed} \bmod n$

$ed = k\phi(n) + 1$ // d and e are inverses modulo $\phi(n)$

$P_1 = P^{ed} \bmod n \rightarrow P_1 = P^{k\phi(n)+1} \bmod n$

$P_1 = P^{k\phi(n)+1} \bmod n = P \bmod n$ // Euler's theorem (second version)

Agenda

- Principles of public key cryptosystems
- The RSA algorithm
- Exponentiation on RSA
- Attacks on RSA

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

참고 : SPA attack on RSA Square & Multiply Algorithm
http://joye.site88.net/papers/JO05encyclo.pdf

- Algorithm for computing $a^b \bmod n$, b is expressed as a binary number $b_k b_{k-1} \ldots b_0$

$c \leftarrow 0; f \leftarrow 1;$
for $i \leftarrow k$ downto 0
  do $c \leftarrow 2 \times c$
    $f \leftarrow (f \times f) \bmod n$
  if $b_i = 1$, then $c \leftarrow c + 1, f \leftarrow (f \times a) \bmod n$
return f

- $a^b \bmod n$, where $a=7, b=560=1000110000, n=561$

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

Exponentiation

- Computing $(x)^c$ mod n
  - Example: suppose that c=53=110101
  - $x^{53}=((x^{13})^2)^2 \cdot x=(((x^3)^2)^2 \cdot x)^2)^2 \cdot x = (((x^2 \cdot x)^2)^2 \cdot x)^2)^2 \cdot x$ mod n

Alg: Square-and-multiply (x, n, c = $c_{k-1}\ c_{k-2} \ldots c_1\ c_0$)

```
z=1
for i ← k-1 downto 0 {
        z ← z² mod n
        if c_i = 1 then z ← (z × x) mod n
}
return z
```

이전 page의 algorith과 본 algorithm의 차이점은 초기setting 과정 부분

- Computes x^{26} without modular reduction
  - Binary representation of exponent:
    26=(1,1,0,1,0)$_2$=(h_4,h_3,h_2,h_1,h_0)$_2$

Step		Binary exponent	Op	Comment
1	$x = x^1$	$(1)_2$		Initial setting, h_4 processed
1a	$(x^1)^2 = x^2$	$(10)_2$	SQ	Processing h_3
1b	$x^2 * x = x^3$	$(11)_2$	MUL	$h_3 = 1$
2a	$(x^3)^2 = x^6$	$(110)_2$	SQ	Processing h_2
2b	-	$(110)_2$	-	$h_0 = 0$
3a	$(x^6)^2 = x^{12}$	$(1100)_2$	SQ	Processing h_1
3b	$x^{12} * x = x^{13}$	$(1101)_2$	MUL	h_1=1
4a	$(x^{13})^2 = x^{26}$	$(11010)_2$	SQ	Processing h_0
4b	-	$(11010)_2$	-	$h_0 = 0$

(1,1,0,1,0)$_2$

(1,1,0,1,0)$_2$

(1,1,0,1,0)$_2$

(1,1,0,1,0)$_2$

- Observe how the exponent evolves into $x^{26} = x^{11010}$

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 $\gcd(e,\phi(n))=1$
  - 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

Exponentiation: Speed-up techniques

- Modular exponentiation is computationally intensive
- Even with the square-and-multiply algorithm, RSA can be quite slow
- on constrained devices such as smart cards
- Some important tricks:
  - **Short public exponent *e***
  - **Chinese Remainder Theorem (CRT)**
  - Exponentiation with pre-computation *(not covered here)*

ref: Understanding Cryptography by C. Paar

Exponentiation: **Short public exponent**

- Fast encryption with small public exponent
  - Choosing a small public exponent *e* does not weaken the security of RSA
  - A small public exponent improves the speed of the RSA encryption significantly

Public Key	e as binary string	#MUL + #SQ
$2^1+1 = 3$	$(11)_2$	1 + 1 = 2
$2^4+1 = 17$	$(1\ 0001)_2$	4 + 1 = 5
$2^{16} + 1$	$(1\ 0000\ 0000\ 0000\ 0001)_2$	16 + 1 = 17

  - This is a commonly used trick (e.g., SSL/TLS, etc.) and makes RSA the fastest asymmetric scheme with regard to encryption!

ref: Understanding Cryptography by C. Paar

Exponentiation: **Chinese Remainder Thm**

- Choosing a small private key *d* results in security weaknesses!
  - In fact, d must have at least *0.3t* bits, where *t* is the bit length of the modulus *n*
- However, the Chinese Remainder Theorem (CRT) can be used to (somewhat) accelerate exponentiation with the private key *d*
- Based on the CRT we can replace the computation of

$$x^{d \bmod \Phi(n)} \bmod n$$

by two computations

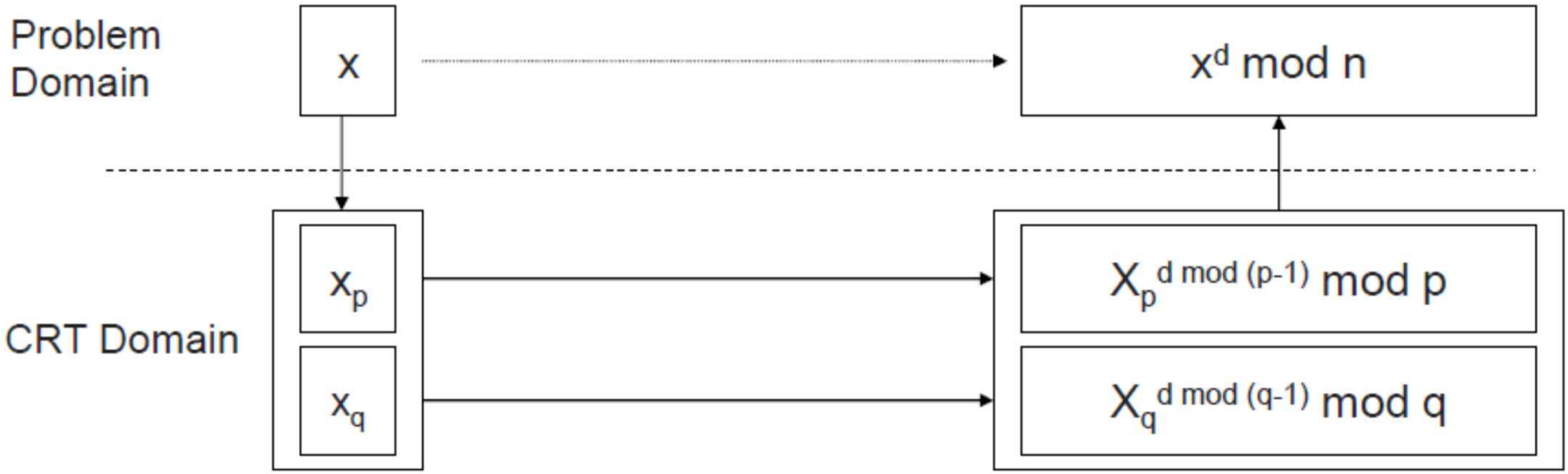
$$x^{d \bmod (p-1)} \bmod p \quad \text{and} \quad x^{d \bmod (q-1)} \bmod q$$

where *q* and *p* are „small“ compared to *n*

ref: Understanding Cryptography by C. Paar

Exponentiation: **Chinese Remainder Thm**

- Basic principle of CRT-based exponentiation



- CRT involves three distinct steps

  (1) Transformation of operand into the CRT domain

  (2) Modular exponentiation in the CRT domain

  (3) Inverse transformation into the problem domain

- These steps are equivalent to one modular exponentiation in the problem domain

ref: Understanding Cryptography by C. Paar

Exponentiation: **Chinese Remainder Thm**

CRT: Step 1 – Transformation

- Transformation into the CRT domain requires the knowledge of p and q
- p and q are only known to the owner of the private key, hence CRT cannot be applied to speed up encryption
- The transformation computes (x_p, x_q) which is the representation of x in the CRT domain. They can be found easily by computing

$$x_p \equiv x \bmod p \quad \text{and} \quad x_q \equiv x \bmod q$$

ref: Understanding Cryptography by C. Paar

Exponentiation: **Chinese Remainder Thm**

- CRT: Step 2 – Exponentiation

- Given d_p and d_q such that

$$d_p \equiv d \bmod (p-1) \quad \text{and} \quad d_q \equiv d \bmod (q-1)$$

one exponentiation in the problem domain requires two exponentiations in the CRT domain

$$y_p \equiv x_p^{dp} \bmod p \quad \text{and} \quad y_q \equiv x_q^{dq} \bmod q$$

- In practice, p and q are chosen to have half the bit length of n, i.e., $|p| \approx |q| \approx |n|/2$

ref: Understanding Cryptography by C. Paar

Exponentiation: **Chinese Remainder Thm**

CRT: Step 3 – Inverse Transformation

- Inverse transformation requires modular inversion twice, which is computationally expensive

$$c_p \equiv q^{-1} \bmod p \quad \text{and} \quad c_q \equiv p^{-1} \bmod q$$

- Inverse transformation assembles y_p, y_q to the final result $y \bmod n$ in the problem domain

$$y \equiv [q * c_p] * y_p + [p * c_q] * y_q \bmod n$$

- The primes p and q typically change infrequently, therefore the cost of inversion can be neglected because the two expresssions

$$[q * c_p] \text{ and } [p * c_q]$$

can be precomputed and stored

ref: Understanding Cryptography by C. Paar

Exponentiation: **Chinese Remainder Thm**

Complexity of CRT

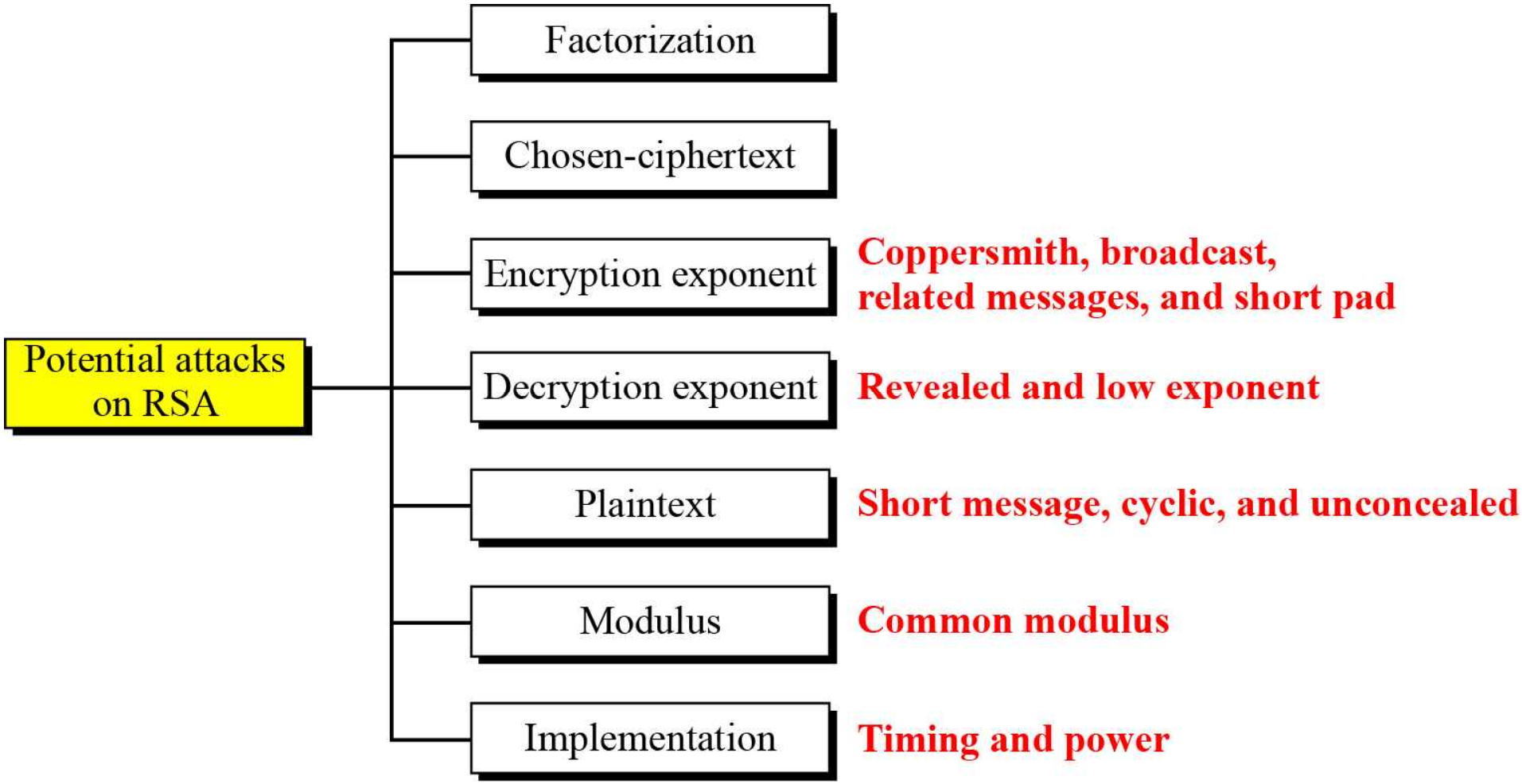
- We ignore the transformation and inverse transformation steps since their costs can be neglected under reasonable assumptions
- Assuming that n has $t+1$ bits, both p and q are about $t/2$ bits long
- The complexity is determined by the two exponentiations in the CRT domain. The operands are only $t/2$ bits long. For the exponentiations we use the square-and-multiply algorithm:
  - # squarings (one exp.): $\#SQ = 0.5\ t$
  - # aver. multiplications (one exp.): #MUL = 0.25t
  - Total complexity: $2 * (\#MUL + \#SQ) = 1.5t$
- This looks the same as regular exponentations, but since the operands have half the bit length compared to regular exponent., each operation (i.e., multipl. and squaring) is 4 times faster!
- Hence CRT is **4 times** faster than straightforward exponentiation

Agenda

- Principles of public key cryptosystems
- The RSA algorithm
- Exponentiation on RSA
- Attacks on RSA

Attacks on RSA

- **Taxonomy of potential attacks on RSA**



Factoring Problem

- mathematical approach takes 3 forms:
  - factor $N=p.q$, hence find $\varnothing(N)$ and then d
  - determine $\varnothing(N)$ directly and find d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of Aug-99 best is 130 decimal digits (512) bit with GNFS
  - biggest improvement comes from improved algorithm
    - cf "Quadratic Sieve" to "Generalized Number Field Sieve"
  - barring dramatic breakthrough 1024+ bit RSA secure
    - ensure p, q of similar size and matching other constraints

Progress in Factoring

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

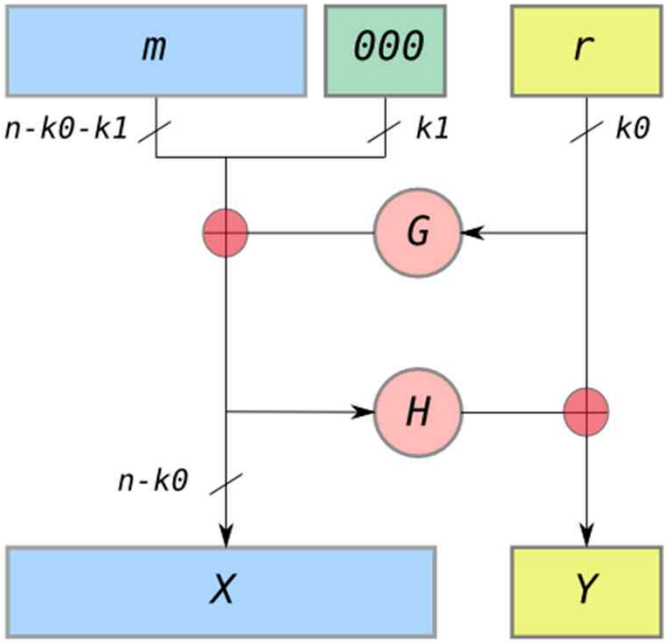
Timing Attacks

- developed in mid-1990's
- exploit timing variations in operations
  - eg. multiplying by small vs large number
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations
- Side channel attack
  - Reference) www.cryptography.com Kocher

Chosen Ciphertext Attacks against RSA

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- attackers chooses ciphertexts & gets decrypted plaintext back
- choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- can counter with random pad of plaintext
- or use Optimal Asymmetric Encryption Padding (OAEP)
  - OAEP is a padding scheme often used together with RSA (by Bellare and Rogaway)
  - OAEP algorithm is a form of Feistel network which uses a pair of random oracles G and H to process the plaintext prior to asymmetric encryption
  - When combined with any secure trapdoor one-way permutation f, this processing is proved in the random oracle model to result in a combined scheme which is semantically secure under chosen plaintext attack.

Chosen Ciphertext Attacks against RSA



OAEP Diagram

In the diagram,

- n is the number of bits in the RSA modulus.
- k_0 and k_1 are integers fixed by the protocol.
- m is the plaintext message, an $(n - k_0 - k_1)$-bit string
- G and H are typically some cryptographic hash functions fixed by the protocol.
- $\oplus$ is an xor operation.

To encode,

1. messages are padded with k_1 zeros to be $n - k_0$ bits in length.
2. r is a randomly generated k_0-bit string
3. G expands the k_0 bits of r to $n - k_0$ bits.
4. $X = m00..0 \oplus G(r)$
5. H reduces the $n - k_0$ bits of X to k_0 bits.
6. $Y = r \oplus H(X)$
7. The output is $X \| Y$ where X is shown in the diagram as the leftmost block and Y as the rightmost block.

To decode,

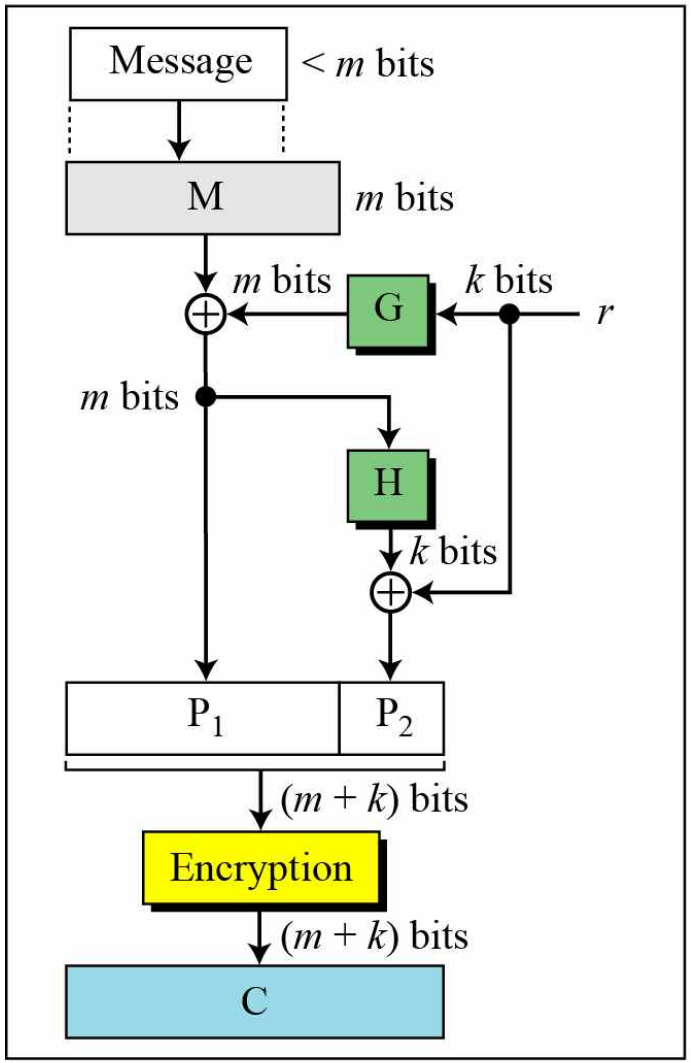
1. recover the random string as $r = Y \oplus H(X)$
2. recover the message as $m00..0 = X \oplus G(r)$

The "all-or-nothing" security is from the fact that to recover m, you must recover the entire X and the entire Y; X is required to recover r from Y, and r is required to recover m from X. Since any changed bit of a cryptographic hash completely changes the result, the entire X, and the entire Y must both be completely recovered.

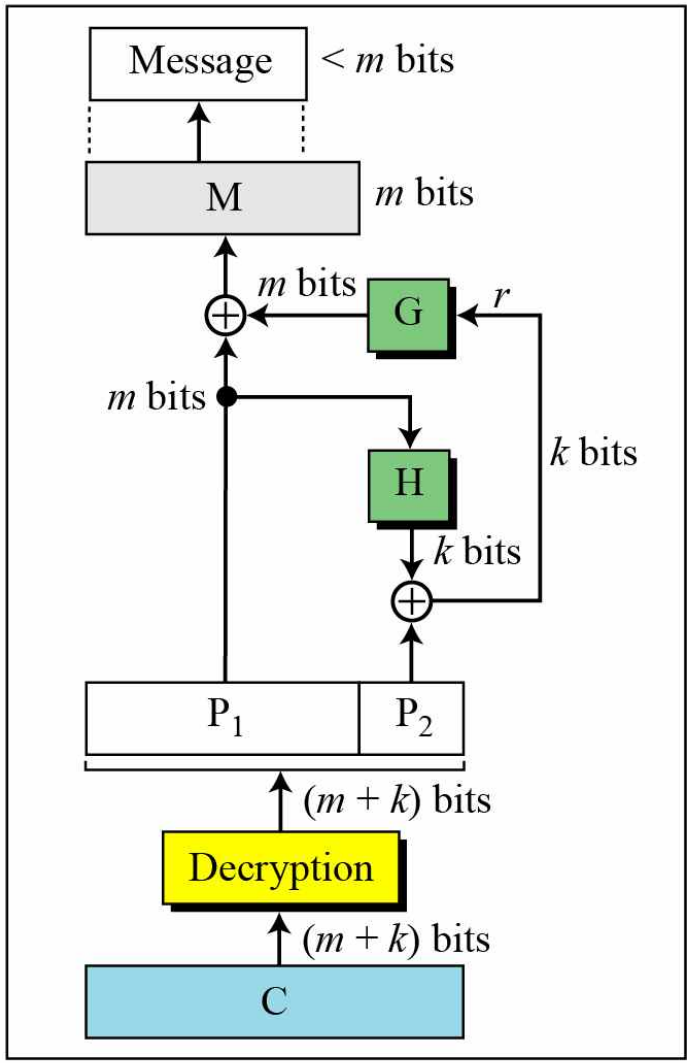
OAEP

- Optimal asymmetric encryption padding (OAEP)

M: Padded message
r: One-time random number
P: Plaintext ($P_1 \| P_2$)
C: Ciphertext
G: Public function (*k*-bit to *m*-bit)
H: Public function (*m*-bit to *k*-bit)



Sender



Receiver

Summary

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

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

Next…

- We will study on key management and other public-key cryptosystems such as ECC….


Q&A