제7장 공개키 암호

7.1 공개키 암호 시스템의 원리

7.2 RSA 알고리즘

7.1 공개키 암호 시스템의 원리

❑ 암호학 전체의 역사에서 최대 혁명적 발견

❖ 1976년 Diffie와 Hellman에 의해 제기

❑ 관용 암호 방식의 문제점

: 전치와 치환기법을 기본방법으로 사용

❖ 키 분배의 문제

❖ 디지털 서명이 불가능

❑ 공개키 암호방식

: 관용 암호방식과 근본적으로 다른 수학적 함수에 근거

❖ 키 분배문제 해결 : 비밀키 전송

❖ 디지털서명 : 문서 서명효과, 발신처 인증, 부인봉쇄

7.1 공개키 암호 시스템의 원리

❑ 공개키 암호방식에 대한 오해

- ❖ 관용 암호방식 보다 안전
  - ➢ 안전성은 키 길이와 계산시간의 관계
- ❖ 범용 기술
  - ➢ 키 관리와 서명에 효과적 ; 응용/관용방식 보다 계산 부하
- ❖ 키 분배 문제 해결
  - ➢ 간단하거나 효율적이라고 단정 불가하고 별도의 관리 프로토콜 필요

7.1 공개키 암호 시스템의 원리

❑ 공개키 알고리즘

: 두 개의 다른 키 사용

공개키 : 모든 사람이 접근 가능한 키 (공개)

개인키 : 각 사용자 자신만이 소유 (비밀)

(관용 암호에 사용되는 키는 비밀키라고 함)

❑ 공개키 알고리즘의 특징

❖ 암호 알고리즘과 암호키를 알아도 복호키 계산 불가능

❖ 두 개의 키 중 하나는 암호에 다른 하나는 복호에 사용

7.1 공개키 암호 시스템의 원리

- 공개키(public key)
  - 암호화 키는 일반에게 공개해도 무방
  - 수신자에게 메일로 전달해도 무방
  - 신문의 광고란에 실어도 무방
  - 간판으로 해서 길가에 세워도 무방
  - Web 페이지를 통하여 전 세계에서 읽을 수 있도록 해도 무방
  - 도청자에게 공개 키가 도청되는 것을 신경 쓸 필요가 없다

- 개인 키(private key)
  - 복호화 키는 미공개
  - 이 키는 본인만 사용
  - 개인 키는 다른 사람에게 보이거나, 건네주거나 해서는 안 됨
  - 개인 키는 자신의 통신 상대에게도 보여서는 안 됨

7.1 공개키 암호 시스템의 원리

- 키 쌍(key pair)
  - 공개 키와 개인 키는 둘이 한 쌍
  - 공개 키로 암호화한 암호문은 그 공개 키와 쌍이 되는 개인 키가 아니면 복호화할 수 없다
  - 수학적인 관계
    - 키 쌍을 이루고 있는 2개의 키는 서로 밀접한 관계
    - 공개 키와 개인 키 쌍은 별개로 만들 수 없음

7.1 공개키 암호 시스템의 원리

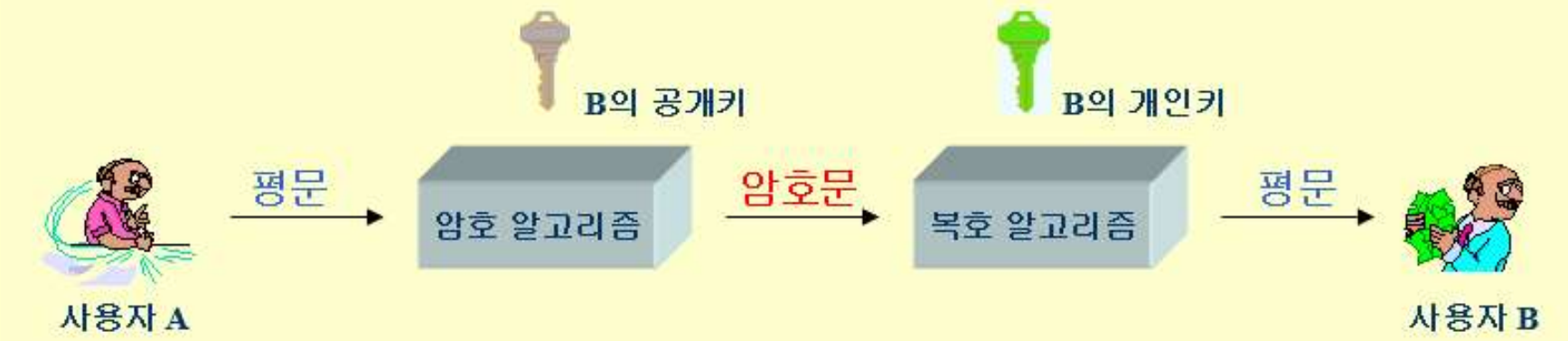
❑ 관용 암호와 공개키 암호 비교

관용 암호	공개키 암호
▪ 암호/복호에 동일한 키와 동일한 알고리즘 사용 ▪ 수신자와 송신자는 키를 교환해야 함 ▪ 공유한 키(비밀키)는 비밀로 유지 ▪ 키 분배의 어려움/디지털 서명 불가능 ▪ 속도가 빠름	▪ 암호/복호에 각각 서로 다른 키 와 동일한 알고리즘 사용 ▪ 수신자와 송신자는 연관된 키 쌍 중 하나를 알아야 함 ▪ 키 쌍중 하나(개인키)를 비밀로 유지 공개키를 공개 ▪ 디지털 서명 가능 ▪ 속도가 느림

7.1 공개키 암호 시스템의 원리

❑ 공개키 암호의 단순 모델

: A가 B에게 암호화 메시지를 보내는 경우

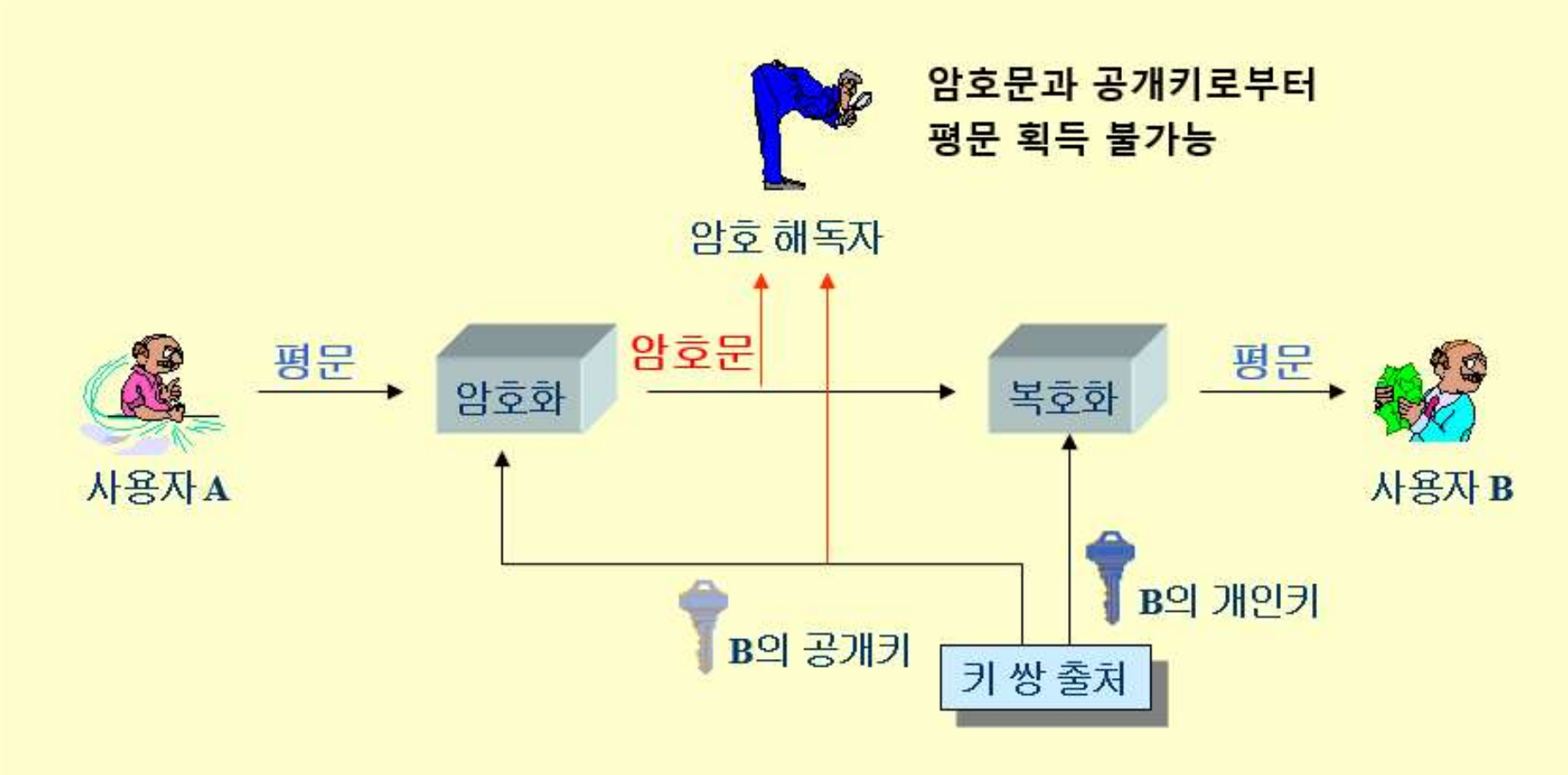


1. 공개키와 개인키 생성
2. 공개키는 공개하고 개인키는 개인이 소유
3. A는 B의 공개키로 메시지를 암호화
4. B는 자신의 개인키로 메시지 복호화
   (B의 개인키를 모르는 제 3자는 메시지 복호 불가능)

7.1 공개키 암호 시스템의 원리

❑ 공개키 암호 시스템 : 기밀성

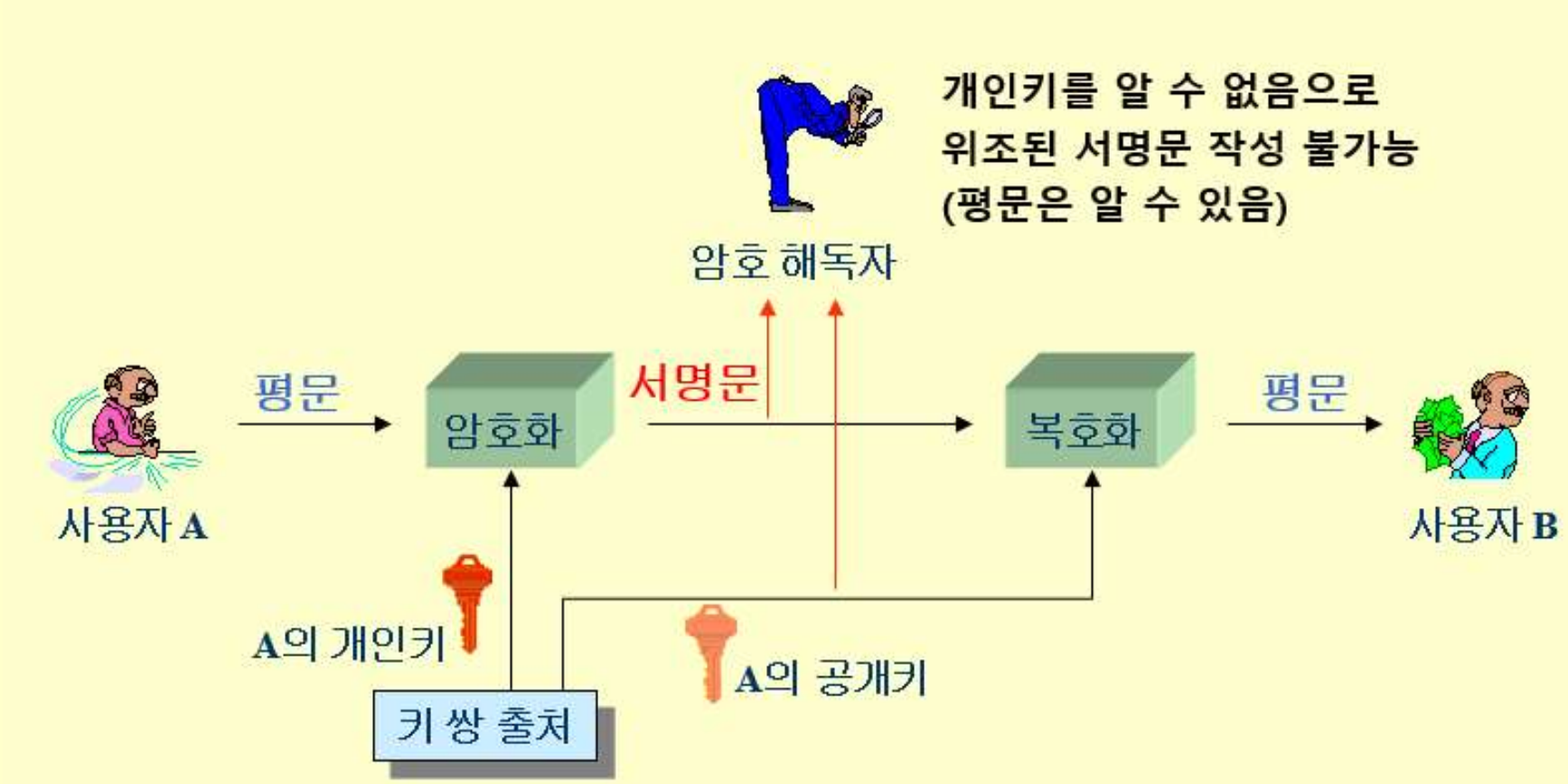
: 공개키로 암호화함으로써 메시지 기밀성 제공



7.1 공개키 암호 시스템의 원리

❑ 공개키 암호 시스템 : 인증

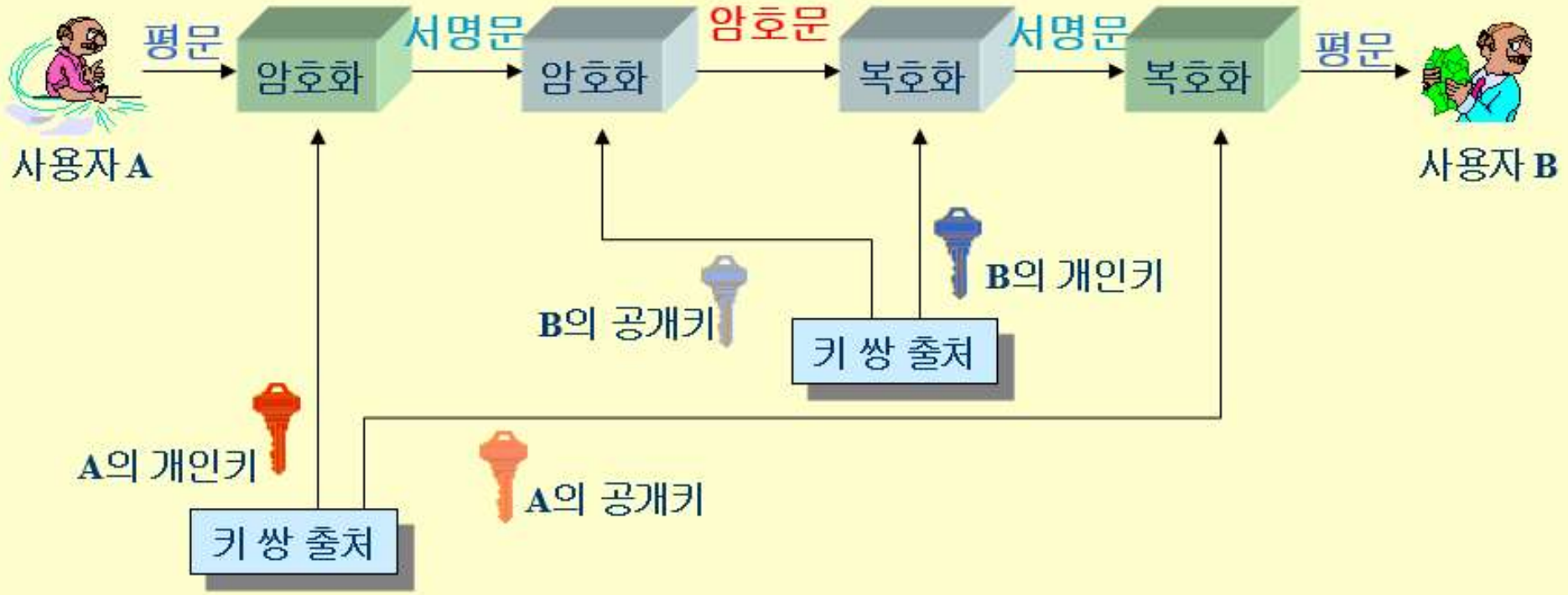
: 개인키로 서명함으로써 송신자 인증 제공



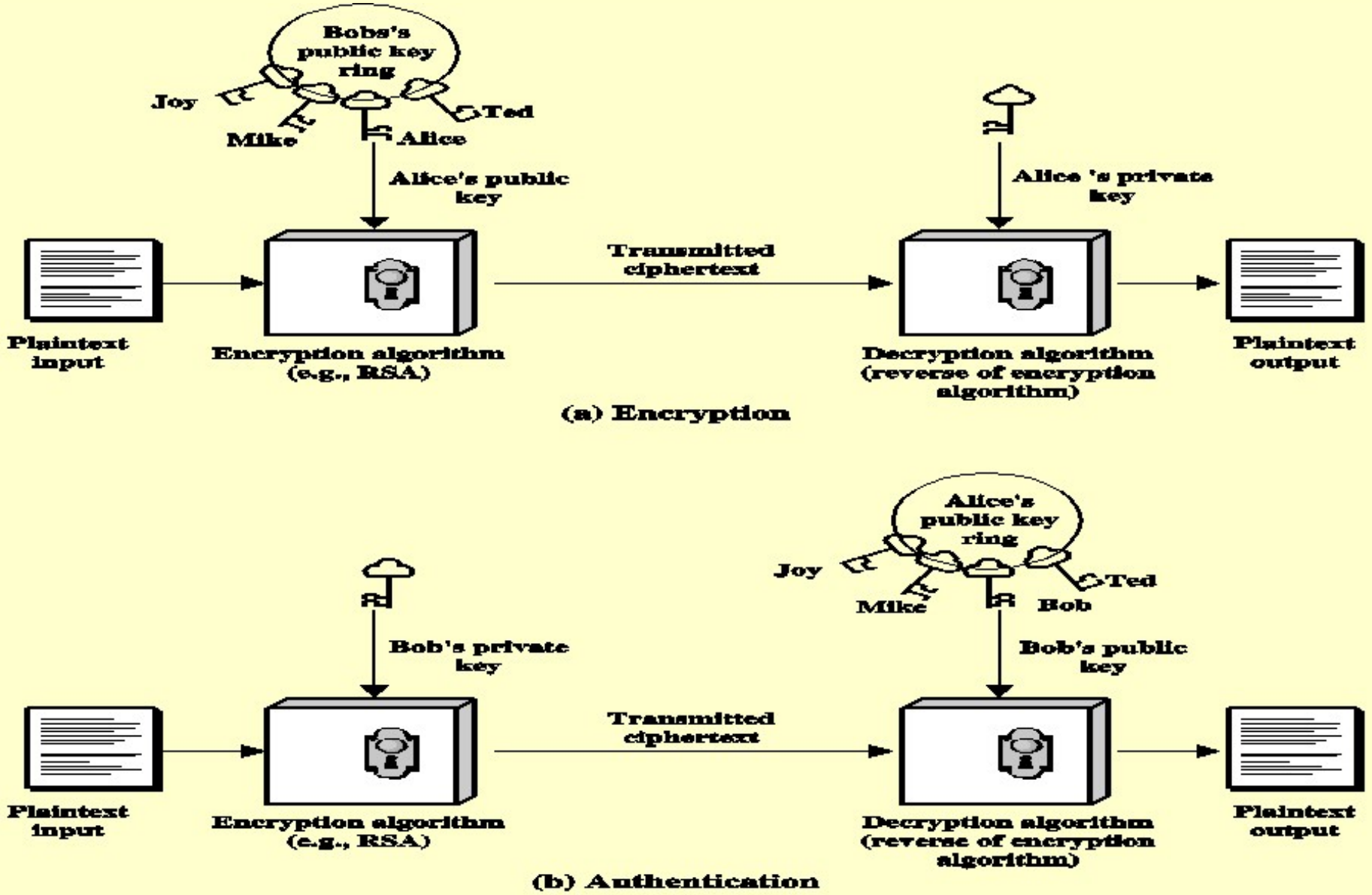
7.1 공개키 암호 시스템의 원리

❑ 공개키 암호 시스템 : 기밀성과 인증

: 개인키로 서명하고 공개키로 암호화하여 기밀성과 인증 제공



공개키 암호


Bobs's public key ring
Joy
Ted
Mike
Alice
Alice's public key
Alice 's private key
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., RSA)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output
(a) Encryption
Alice's public key ring
Joy
Ted
Mike
Bob
Bob's private key
Bob's public key
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., RSA)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output
(b) Authentication

7.1 공개키 암호 시스템의 원리

- 공개키 암호 시스템의 사용
  - **암호/복호** (수신자의 공개키로 메시지 암호)
  - **디지털 서명** (송신자의 개인키로 메시지 서명)
  - **키 교환** (세션키를 교환하기 위해 사용)
- 공개키 암호 시스템의 응용

알고리즘	암호/복호화	디지털 서명	키 교환
RSA	가능	가능	가능
타원 곡선	가능	가능	가능
DSS	불가능	가능	불가능
Diffie-Hellman	불가능	불가능	가능

7.1 공개키 암호 시스템의 원리

❑ 공개키 알고리즘의 조건 (Diffie와 Hellman)

- ❖ 키 쌍(공개키 KU, 개인키 KR)의 생성이 쉽다.
- ❖ 다음 식과 같은 암호문의 생성이 쉽다.
  - ❖ $C = E_{KUb}(M)$
- ❖ 다음식과 같은 암호문의 복구화가 쉽다.
  - ❖ $M = D_{KRb}(C) = D_{KRb}[E_{KUb}(M)]$
- ❖ 공개키 KUb로부터 개인키 KRb를 결정하는 것은 어렵다.
- ❖ 공개키 KUb와 암호문 C로부터 메시지 M의 복구가 어렵다.
- ❖ 암호와 복호 기능이 다음과 같이 적용 가능하다. (추가 사항)
  - ❖ $M = E_{KUb}[D_{KRb}(M)]$

일방향 함수(one-way function)

- 함수 값의 계산이 쉬운 반면 역의 계산은 어렵다는 조건
  - $Y = f(X)$ 쉽다
  - $X = f^{-1}(Y)$ 어렵다.
- 일반적으로 'easy(쉽다)'는 입력 길이에 대한 함수로서 다항식 시간동안 풀릴 수 있는 문제를 의미하는 것으로 정의
- 트랩도어 일방향 함수(trap-door one-way function)
  - 한 방향으로는 계산이 쉽고, 어떤 부가적인 정보가 알려져 있지 않으면 다른 방향에서는 계산이 불가능
  - $Y = f_k(X)$ 만일 k와 X가 알려져 있다면 쉽다
  - $X = f_k^{-1}(Y)$ 만일 k와 Y가 알려져 있다면 쉽다
  - $X = f_k^{-1}(Y)$ 만일 Y는 알려져 있지만 k가 알려져 있지 않으면 어렵다.
- 실제 공개키 기술발전은 적절한 트랩도어 일방향 함수의 개발이 중요
- 임의의 큰 데이터 필드를 취하여 고정된 출력으로 대응시키는 일방향 해쉬함수와의 혼동을 하지 않아야 한다.

7.1 공개키 암호 시스템의 원리

❑ 공격 유형

❖ 전사적 공격에 취약

⇒ 키의 크기를 크게 함으로써 방지

(상대적으로 속도가 느려짐)

❖ 공개키로부터 개인키를 계산하는 방법

⇒ 수학적으로 계산이 불가능함을 증명하지 못함

❖ 가능한 메시지 추측 공격(메시지 길이가 작을 때)

: 모든 가능한 메시지를 공개키로 암호화하여 암호문과 비교

⇒ 메시지에 임의의 비트를 추가함으로써 방지

7.1 공개키 암호 시스템의 원리

❑ Knapsack 암호의 일반적인 개념

: 일정한 크기의 배낭에 어떤 물건을 넣어야 하는지에 관한 문제

❖ 정의

- cargo 벡터 $a = (a_1, a_2, \ldots, a_n)$ a_i :정수
- 평문 메시지 블록 $x = (x_1, x_2, \ldots, x_n)$ x_i :이진수
- 대응하는 암호문 $S = a \cdot x = \sum_{I=1}^{n} (a_i \times x_i)$

❖ 개념

- 공개키 : a
- x_i = 1은 a_i가 Knapsack에 포함된다는 의미
- 송신자는 S=a • x를 계산하여 S 값을 전송
- 수신자는 S와 a에 의해 x를 복구

7.1 공개키 암호 시스템의 원리

- Knapsack의 조건
  - 첫째, S의 값에 유일한 역이 존재
    - ex) 두 가지 역이 존재하는 예

      a = (1, 3, 2, 5), S = 3 일 경우

      x = 1010, x = 0100의 두가지 경우가 존재

    따라서, 원소 a의 조합이 유일한 값을 산출하는 것으로 선택
  - 둘째, 일반적인 복호는 어려우나, 특별한 지식이 있을 경우 쉬움
    - 초증가 벡터(superincreasing vector) 이용
      - 초증가 벡터

        : a의 각 원소가 선행하는 원소들의 합보다 큰 경우
      - a′ = (a1, a2, …, a5) = (171, 197, 459,1191, 2410) 일 경우
      - S′ = a′, x′ = 3798이라면
      - X = $(x_1, x_2, x_3, x_4, x_{5)}$ = (0, 1, 0, 1, 1)

7.1 공개키 암호 시스템의 원리

❑ Knapsack 문제에 초증가 Knapsack 문제를 묶는 방법

- ❖ a′을 임의로 선택 (a′ : 쉬운 초증가 Knapsack 벡터)
- ❖ m과 w를 선택
  - ➢ m : a′원소들의 합보다 큰 정수
  - ➢ w : m과 서로소인 정수
- ❖ a 생성 (a : 어려운 Knapsack 벡터)

  $a = wa' \bmod m$
- ❖ 쉬운 Knapsack 벡터로의 변환 가능

  $w^{-1}a = a' \bmod m$

7.1 공개키 암호 시스템의 원리

❑ Knapsack 알고리즘 사용 예

❖ 키 생성

- a′ (쉬운 Knapsack)

1	3	7	13	26	65	119	267

- w = 467, m = 523, w^{-1} = 28 (mod 523)
- a (어려운 Knapsack)

467	355	131	318	113	21	135	215

$$a = wa' \bmod m$$

- 공개키 : KU = a, 개인키 : KR = {w^{-1}, m, a′)

❖ 암호화

- 평문 x = 01001011
- 암호문 : S = a • x = 818 (0 ×467 + 1 ×355 + 0 ×131 + 0 ×318 + 1 ×113 + 0 ×21 + 1 ×135 + 1 ×215 = 818)

7.1 공개키 암호 시스템의 원리

❑ Knapsack 알고리즘 사용 예 (계속)

❖ 복호화

1	3	7	13	26	65	119	267

❖ $S' = S \cdot W^{-1} \bmod m = a' \cdot x$로 복호

- $818 \times w^{-1} = 818 \times 28 = 415 \pmod{523}$
- $415 \geq 267$ $\Rightarrow x_8 = 1$
- $415 - 267 = 148 \geq 119$ $\Rightarrow x_7 = 1$
- $148 - 119 = 29 < 65$ $\Rightarrow x_6 = 0$
- $29 \geq 26$ $\Rightarrow x_5 = 1$
- $29 - 26 = 3 < 13$ $\Rightarrow x_4 = 0$
- $3 < 7$ $\Rightarrow x_3 = 0$
- $3 \geq 3$ $\Rightarrow x_2 = 1$
- $3 - 3 = 0 < 1$ $\Rightarrow x_1 = 0$

$\Rightarrow$ 평문 : $x_1x_2x_3x_4x_5x_6x_7x_8 = 01001011$

Knapsack 알고리즘 예

- ❖ 키 생성
  - ➢ a′ (쉬운 Knapsack) 2, 4, 7, 13, 31
  - ➢ w = 57, m = 71, $w^{-1} = 5 \pmod{71}$
  - ➢ a (어려운 Knapsack) = wa′ mod m ; , , , , ,
  - ➢ 공개키 : KU = a, 개인키 : KR = {w^{-1}, m, a′)
- ❖ 암호화
  - ➢ 평문 x = 11010
  - ➢ 암호문 : S = a • x =
- ❖ 복호화
  - ➢ S′ = S • W^{-1} mod m =
  - ➢ 평문 :

QUIZ

❖ 키 생성

- a′ (쉬운 Knapsack) 3, 5, 9, 21, 46
- w = 47, m = 87, w^{-1} = ?? (mod 71)
- a (어려운 Knapsack) = wa′ mod m ; ??, ??, ??, ??, ??
- 공개키 : KU = a, 개인키 : KR = {w^{-1}, m, a′)

❖ 암호화

- 평문 x = 10110
- 암호문 : S = a • x = ??

❖ 복호화

- S′ = S • W^{-1} mod m = ?? (mod 71)
- 평문 : ?, ?, ?, ?, ?

7.2 RSA 알고리즘

❑ RSA의 개발 특징

- ❖ 1977년: Ron Rivest, Adi Shamir, Leonard Adleman 공동 개발
- ❖ 1978년: 공개키 암호방식의 요구사항을 만족하는 최초방식으로 공포
- ❖ RSA 구조는 평문과 암호문이 0~(n-1)사이의 블록 암호
- ❖ N의 전형적인 크기는 1024비트 또는 309 십진숫자
- ❖ 비밀키 암호방식(DES)보다 계산이 늦다
- ❖ 안전성은 소인수 분해의 어려움에 근거

7.2 RSA 알고리즘

❑ RSA 암복호화 방식

- ❖ 지수승을 가진 수식을 사용하도록 고안
- ❖ 평문을 블록단위로 암호화(각 블록은 n보다 작은 이진 값)
- ❖ 공개키 : KU = {e, n}, 개인키 : KR = { d, n }
- ❖ 암호화

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

- ❖ 복호화

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

7.2 RSA 알고리즘

❑ RSA 알고리즘

❖ 키 생성

- p, q 선택 (p, q는 솟수)
- n = p × q 계산
- 정수 d 선택 (gcd(ϕ(n),d) =1, 1<d< ϕ(n))
- e 계산 ($e = d^{-1} \bmod \phi(n)$)
- 공개키 (KU = {e, n})
- 개인키 (KR = {d, n})

❖ 암호화

- $C = M^e \pmod n$

❖ 복호화

- $M = C^d \pmod n$

7.2 RSA 알고리즘

❑ 오일러의 정리

❖ 오일러 정리는 서로 소인 모든 a와 n에 대한 관계 표현

$a^{\varphi(n)} \equiv 1 \bmod n$

a = 3;	n = 10;	$\phi(10) = 4;$	$3^4 =$	81	≡	1	mod	10
a = 2;	n = 11;	$\phi(11) = 10;$	$2^{10} =$	1024	≡	1	mod	11

▪ ϕ(n) : n보다 적고 n과 서로소인 양의 정수가 되는 함수

❑ 오일러 정리의 다른 형태

$a^{\varphi(n)+1} \equiv a \bmod n$

❑ 소수 p와 q에 대해 n = pq 일때,

❖ $\varphi(n) = \varphi(pq) = \varphi(p) \times \varphi(q) = (p-1) \times (q-1)$

7.2 RSA 알고리즘

❑ $M^{ed} = M \bmod n$의 증명

❖ 오일러 정리

- p,q (솟수), n = p*q, 정수 n, m (0<m<n), k (임의의 정수) 일 때
- $m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} = m \bmod n$

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

- e 계산 ($e = d^{-1} \bmod \phi(n)$)
- $ed = 1 \bmod \phi(n)$
- $ed = k\phi(n) + 1$

$\Rightarrow$ e와 d는 mod ϕ(n)에 곱셈 역원

(e와 d가 ϕ(n)에 서로소인 경우에만 참)

RSA 알고리즘

키 생성

p, q 선택 p, q는 솟수
$n = p \times q$ 계산
$= (p-1)(q-1)$ 계산
정수 e 선택 $\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
e 계산 $d = e^{-1} \bmod \phi(n)$
공개키 $KU = \{e, n\}$
개인키 $KR = \{d, n\}$

암호화

평문 : $M < n$
암호문 : $C = M^e \pmod{n}$

복호화

암호문 : C
평문 : $M = C^d \pmod{n}$

7.2 RSA 알고리즘

❑ RSA 알고리즘 사용 예

❖ 공개키와 개인키 생성

1. 두 솟수 p = 7, q = 17 을 선택

2. n = pq = 7 × 17 = 119 계산

3. ϕ(n) = (p-1)(q-1) = 96 계산

4. ϕ(n) = 96과 서로소이고 ϕ(n)보다 작은 e 선택 (e = 5)

5. de = 1 mod 96이고 d < 96 인 d를 결정 (d = 77)

⇒ 공개키 KU = {5, 119}, 개인키 KR = {77, 119}

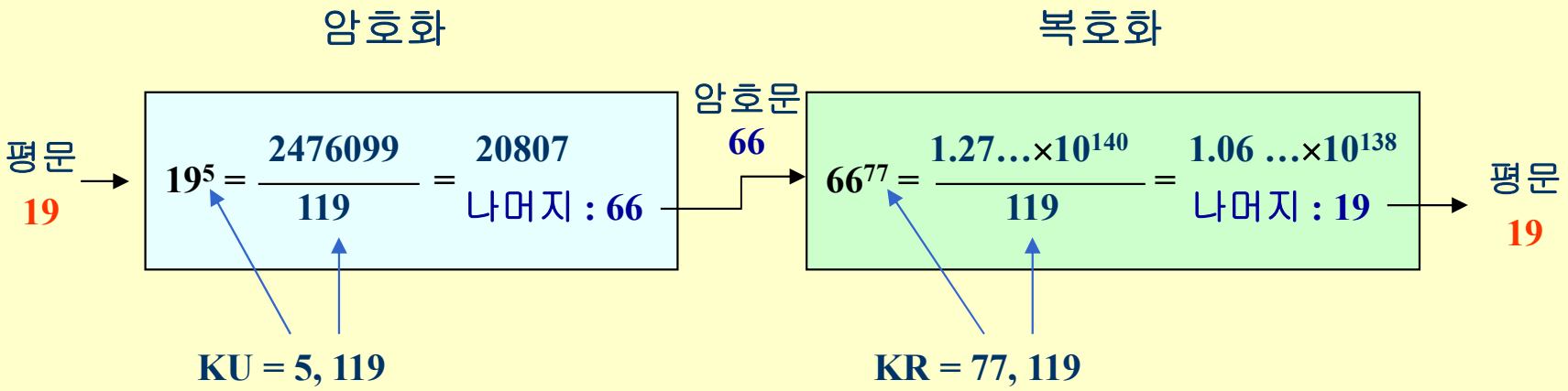
7.2 RSA 알고리즘

❑ RSA 알고리즘 사용 예 (계속)

❖ 암호화와 복호화

: 평문 메시지 M = 19 일 경우

- 암호문 : $19^5 = 66 \bmod 119 \Rightarrow 66$
- 복호문 : $66^{77} = 19 \bmod 119 \Rightarrow 19$



7.2 RSA 알고리즘

1) p=11, q=3
2) n = p. q = 33
3) Φ(n) = Φ(33) = (p-1) (q-1) = 10×2 = 20
4) Φ(n)과 서로소이고 보다 작은 e 선택, e = 3
5) d×e = 1 mod Φ(n) d×e = 1 mod 20 ∴ d=7

KU={3,33} KR={7,33}

메시지 M=5

암호화 : $C = 5^3 \bmod 33 = 125 \bmod 33 = 26$

복호화 : $M = 26^7 \bmod 33 = 8031810176 \bmod 33 = 5$

메시지 M=14

암호화 : $C = 14^3 \bmod 33 = 2744 \bmod 33 = 5$

복호화 : $M = 5^7 \bmod 33 = 78125 \bmod 33 = 14$

7.2 RSA 알고리즘

❑ 암호화와 복호화

- ❖ M^e, C^d : 지수승 계산 ==> 중간과정 이후의 큰 숫자 계산량 증대
- ❖ 중간 결과를 mod n으로 계산하여 숫자 크기 축소
- ❖ 계산량을 줄이는 방법
  - ➢ 모듈러 연산의 특징 이용

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

    ⇒ 중간 결과를 축소
- ❖ 효율적인 지수 승
  - ➢ 직접적인 방법 (x^{16} 일 경우)

    : $x^{16} = x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x$ ⇒ 15번의 곱셈
  - ➢ 효율적인 방법

    : x^2, x^4, x^8, x^{16} ⇒ 4번의 곱셈

Do I=1, 4 x = x^2;

❑ 예) 3^{35} mod 7 ??

35 = 100011

$3^{35} = 3^1 * 3^2 * 3^{32} =$

❑ 예) 3^{45} mod 7 ??

7.2 RSA 알고리즘

❑ 키 생성

- ❖ 암복호에 필요한 키쌍을 생성하기 위하여 선행 작업이 필요
  - ➢ 먼저 2개의 솟수 p와 q를 결정
  - ➢ e와 d중 하나를 선택하고 다른 하나는 계산
- ❖ 전사적 공격을 방어하기 위하여 충분히 큰 소수 p, q가 필요

7.2 RSA 알고리즘

❑ 키 생성

❖ 랜덤한 큰 솟수를 찾는 방법 (확률적 검사법)

1. 랜덤하게 홀수 n을 선택
2. 랜덤하게 a<n인 정수 a를 선택
3. 확률적 솟수 판정법 수행

   (만약 n이 검사에서 실패하면 1단계로)
4. n이 충분한 횟수의 검사에 통과하면 n을 수용,

   그렇지 않으면 2단계로

❖ 솟수 생성을 위한 검사 횟수

- N에 가까운 솟수는 (ln N)에 대하여 평균 1회 생성
- ex) 2^{200} 범위의 솟수 $\Rightarrow$ $\ln(2^{200})/2 = 70$ 회 정도 시행

7.2 RSA 알고리즘

❑ 암호 해독의 고찰

❖ 암호 해독의 접근 방법

- 전사적 공격

  ⇒ 큰 키를 사용하여 방지

- 인수 n를 두 소인수로 인수 분해

  ⇒ 큰 값에 대한 두 솟수의 곱을 인수 분해하는 적당한 알고리즘이 없음

- p와 q을 결정하지 않고 직접 $\phi(n)$을 추측하여 결정

  ⇒ 인수 분해만큼 어려움

- $\phi(n)$을 결정하지 않고 직접 e, d를 추측하여 결정

  ⇒ 인수 분해만큼 어려움

7.2 RSA 알고리즘

❑ 인수분해 시간

자릿수	인수 분해 시간
100	2 주일
150	1 년 (1,000 만 달러 소요)
200	1,500 년 (10 억 달러, 초당 10^{12} 처리)

❑ : 제안 사항(인수 분해를 어렵게)

- $n = 10^{150} \sim 10^{200}$
- $p, q = 10^{75} \sim 10^{100}$
- (p-1), (q-1) 은 큰 솟수를 포함
- gcd(p-1,q-1)은 작아야
- e <n 이고 $d<n^{1/4}$ 일때 d 는 쉽게 결정

RSA에 대한 공격

- 암호해독자가 알고 있는 것
  - 암호문 : 도청해서 얻음
  - 수 e와 n : 공개키로서 공개되어 있으므로, {e, n}을 알고 있음

- 암호해독자가 모르는 것
  - 평문 : 지금부터 해독하려고 하는 내용
  - 수 d : 개인 키 중 적어도 d는 모르고 있음
  - 기타 : 암호해독자는 키 쌍을 만들었을 때의 p, q, ϕ(n)을 모름

RSA에 대한 공격

- 암호문으로부터 평문 구하기
- 암호문 = 평문^e mod n
  - 만약 mod n이 없고,
- 암호문 = 평문^e
  - 이었다면 암호문으로부터 평문을 구하는 것은 쉬움
  - 하지만 mod n이 붙어 있으면 평문을 구하는 것은 이산 대수를 구한다는 매우 곤란한 문제

공개키 암호에 관한 Q&A

- Q1. 공개키 암호는 대칭 암호보다 기밀성이 높은가?
  - 이것만으로는 알 수 없다. 키의 비트 길이에 따라 기밀성의 정도는 변화하기 때문.

- Q2. 1024비트 길이의 키를 갖는 공개 키 암호와, 128비트 길이의 키를 갖는 대칭 암호에서는 비트 길이가 긴 공개 키 암호 쪽이 안전한가?
  - 아니다. 공개 키 암호의 키 길이와, 대칭 암호의 키 길이는 직접 비교할 수 없다.
  - 1024비트 길이의 공개 키 암호와 128비트 길이의 대칭 암호에서는, 128비트 길이의 대칭 암호 쪽이 전사공격에 강하다는 것을 알 수 있음

공개키 암호에 관한 Q&A

- Q3. 공개 키 암호가 생겼기 때문에 앞으로 대칭 암호는 사용되지 않게 되는 것인가?
  - 아니다. 일반적으로 같은 정도의 기밀성을 갖는 키 길이를 선택했을 경우, 공개 키 암호는 대칭 암호보다도 몇 백배나 느려진다. 공개 키 암호는 긴 메시지를 암호화하기에는 적합하지 않다. 목적에 따라 대칭 암호와 공개키 암호 두 가지 모두 사용되게 될 것이다.

- Q4. RSA의 키 쌍을 모두가 자꾸 만들어 가면 그 사이 소수가 없어져 버리는 것은 아닐까?
  - 그럴 염려는 없다. 512비트로 표현할 수 있는 소수의 수는 대략 10의 150거듭제곱으로 전 우주에 존재하는 원자의 개수보다도 많은 수이다.

Report

p=11, q=5 일 경우 공개키를 본인의 학번 끝 두자리를 mod 9하여

0, 4인 경우 : 3

1, 5인 경우 : 7

2, 6인 경우 : 11

3, 7인 경우 : 13

8인 경우 : 17 로 하여

1) 복호화 키를 구하시오.

2) 평문 M=6인 경우의 암호문을 구하시오.

3) 암호문을 복호키로 암호화하여 평문이 나옴을 보이시오.