공개키 암호

https://youtu.be/mgl9x2_kNPg

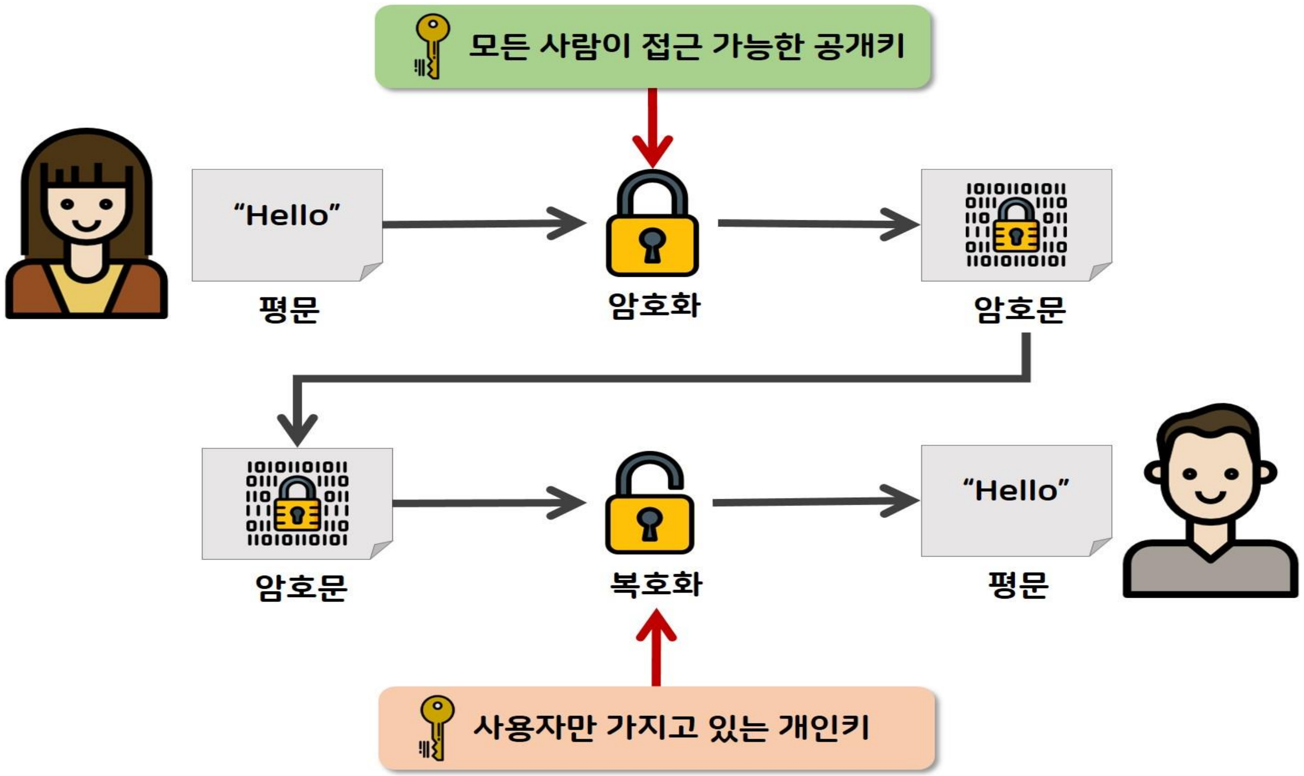
HANSUNG UNIVERSITY 한성대학교

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

공개키 암호란?



모든 사람이 접근 가능한 공개키
"Hello"
평문
암호화
암호문
암호문
복호화
"Hello"
평문
사용자만 가지고 있는 개인키

공개키 암호란?

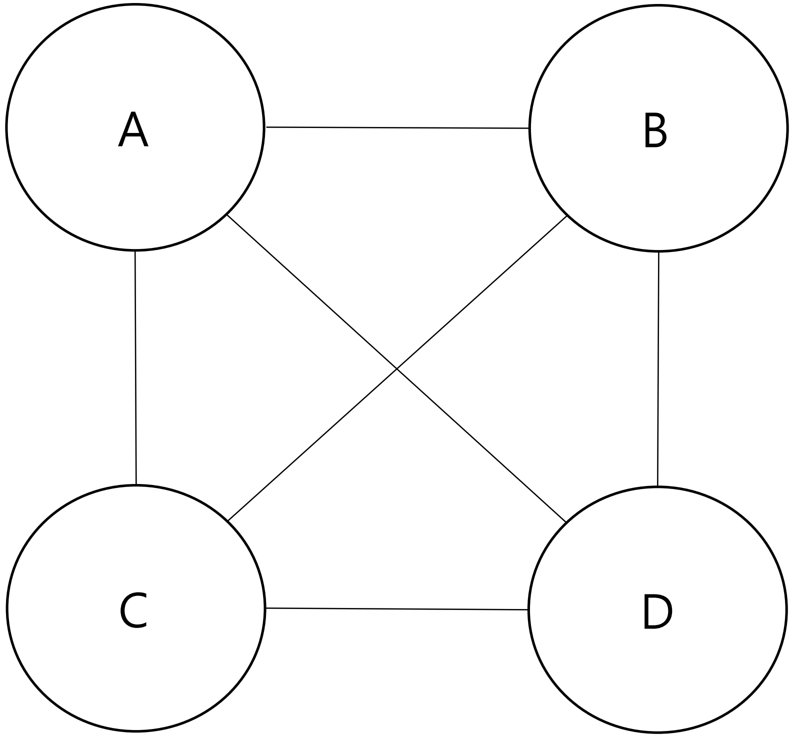
- 수학적 난제를 기반으로 구성됨
  - 인수분해 : RSA
  - 이산대수 : DH, KCDSA
  - 타원곡선 이산대수 : ECDH, ECDSA, EC-KCDSA

  => 연산에 많은 시간을 소요

공개키 암호란?

- 공개키가 필요한 이유!

  => 키 분배에 사용



A = key_{AB}, key_{AC}, key_{AD}

B = key_{AB}, key_{BC}, key_{BD}

C = key_{AC}, key_{BC}, key_{CD}

D = key_{AD}, key_{BD}, key_{CD}

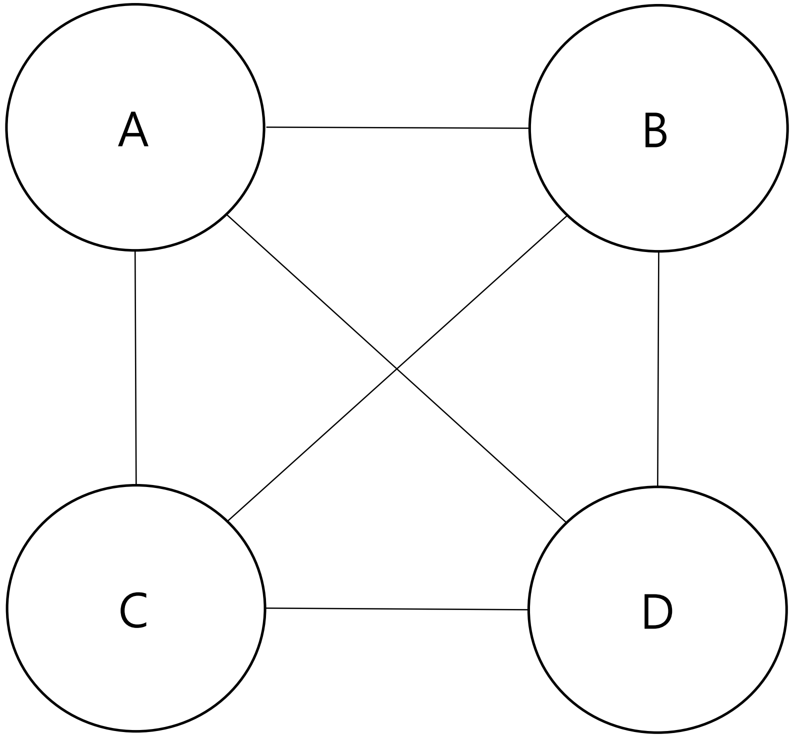
${}_nC_2 = n(n-1) / 2$ 개

$\fallingdotseq n^2 / 2$

=> 총 6개의 키 필요!

공개키 암호란?

- 공개키가 필요한 이유!
  => 키 분배에 사용



A = (PublicKey$_A$, PrivateKey$_A$)
B = (PublicKey$_B$, PrivateKey$_B$)
C = (PublicKey$_C$, PrivateKey$_C$)
D = (PublicKey$_D$, PrivateKey$_D$)

2n개
=> 총 8개의 키 필요!

공개키 암호란?

- 대칭키와의 비교

	공개키	대칭키
키의 관계	암호화키≠복호화키	암호화키=복호화키
암호화 키	공개	비공개
복호화 키	비공개	비공개
키전송	불필요	필요
키의 개수	2n	$n^2/2$
속도	느림	빠름

공개키 암호란?

- 결론
  - 공개키를 이용하여 키를 공유
  - 대칭키를 이용하여 암, 복호화 등의 핵심 연산 수행

공개키 암호란?

- 일방향 함수
  $y = f(x)$에서 x가 주어지면 y에 대해서 찾기는 쉬움
  But, y가 주어졌을 때 $x = f^{-1}(y)$를 찾기는 어려운 함수

- 트랩도어 일방향 함수 (공개키 암호)
  추가적인 정보가 주어졌을 때 $x = f^{-1}(y)$를 찾기 쉬워지는 함수
  인수분해, 이산대수, 타원곡선이산대수으로 구성

공개키 암호란?

- Ex) RSA (인수분해 기반)
  n = p*q, (p, q는 2048bit 이상의 소수)

$$\boxed{y = f(x) = x^e (\mathrm{mod}\ n)}$$

ed = 1 mod ø(n) 인 d를 알고있으면,

$$\boxed{y^d = f(x)^d = x^{ed} (\mathrm{mod}\ n)}$$

공개키 암호란?

- 인수분해 문제

  값이 큰 두 소수가 있을 때, p * q = n 을 구하기는 쉽지만

  n이 주어졌을 때 p, q를 알아내긴 어렵다.

RSA

- RSA 개요
  - 가장 많이 사용하고 있는 공개키 암호
  - 큰 두 소수의 곱 (n = p*q)의 인수분해 문제에 안정성을 기반하고 있음
  - p * q 를 계산하여 n을 얻는 것은 쉬우나, n이 주어졌을 때 p, q를 알아내기는 어려움 (일방향 함수)
  - p, q 를 안다면 개인키를 알아내기 쉬움 (트랩도어 일방향 함수)
  - 안정성을 위해 n은 2048bit 이상의 길이 (112bit의 보안강도)
  - 안정성과 효율성을 적절히 고려하여 공개키를 2^{16}+1로 고정 (0x10001)

RSA

- RSA 키 생성

  1) 큰 소수 p, q 선택 (n이 2048bit 이어야하므로 p, q는 1024bit)

  난수 생성 -> 소수 판정 (확률론적 소수판정, MR 등등)

  -> 난수가 아니면 다시 생성, 난수라면 p, q로 사용

  2) $n = p * q$ 계산

  3) 자연수 e를 선택

  $(\gcd(e, \varnothing(n)) = 1$ 또는 $\gcd(e, (p-1)(q-1)) = 1$

  But, 사실상 거의 모든 공개키는 $2^{16}+1$로 고정

  => $\gcd(p-1, e) = 1$ 이 아니면 난수 생성

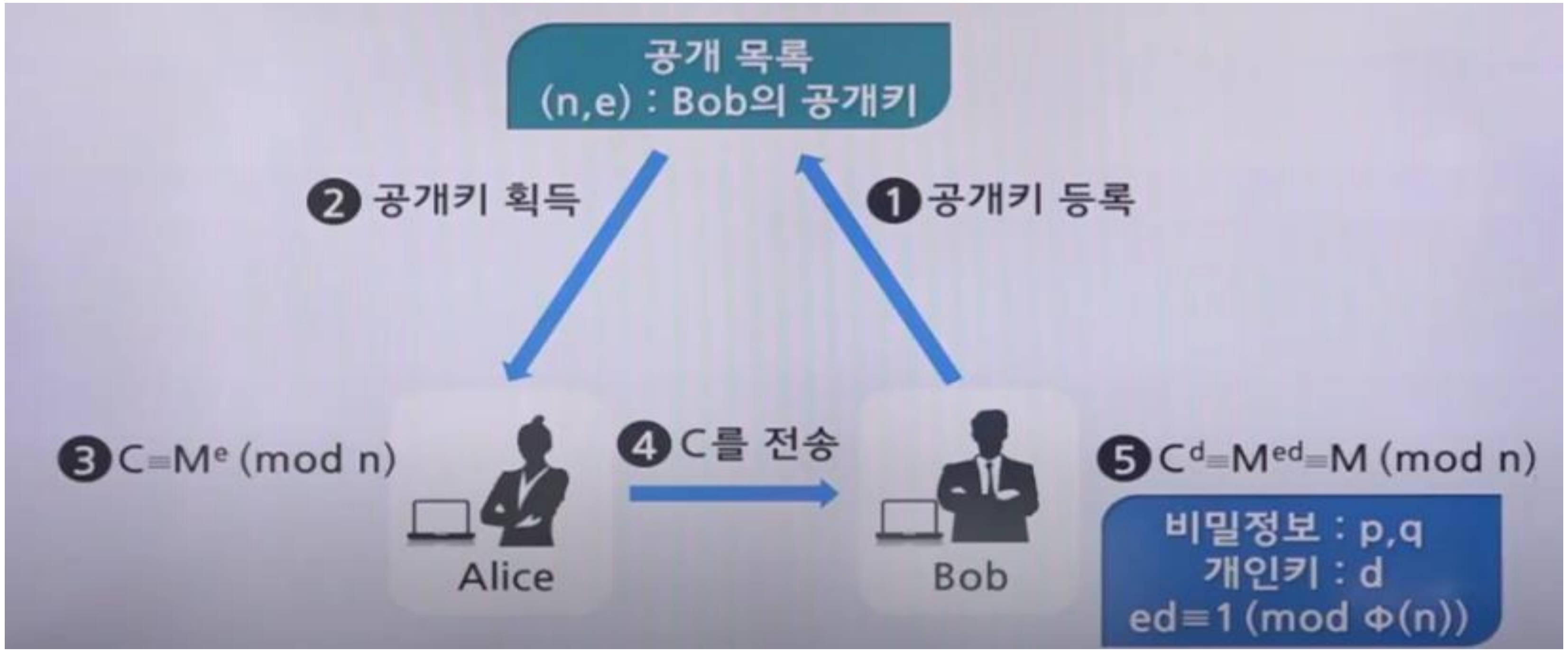
  4) 자연수 d를 선택

  $ed \equiv 1 \bmod(\varnothing(n))$ 또는 $ed \equiv 1 \bmod((p-1)(q-1))$

  5) 공개키 (n, e) 개인키 (p, q, d)

RSA

- RSA 과정



4)에서 C의 패턴을 보고 공격자가 평문을 알아낼 수 있다.

-> 블록암호의 운영모드에서 IV 값을 넣듯이 랜덤성을 추가해줘야 한다.

RSA

- RSA 암복호화 개요

옵션: 해시함수(hLen: 해시함수 출력의 바이트 수,

MGF(Mask Generation Function)

입력: 공개키(n, e),

M: 메시지(mLen: 메시지의 바이트 수, mLen ≤ k - 2hLen - 2),

L: 옵션레이블(제공되지 않는 경우 empty 스트링)

출력: 암호문 C (k 바이트)

에러: 메시지 또는 레이블이 해쉬함수의 범위를 벗어난 경우

Steps:

1. 길이 체크:
   a. 만약 레이블 L이 해시함수의 입력 길이보다 긴 경우 에러 (2^{61}-1 보다 큰 경우)
   b. 만약 mLen > k - 2hLen - 2인 경우 에러
2. **EM = EME - OAEP encode(M, mLen, L, lLen)**
3. RSA 암호화:
   A. m = OS2IP (EM)
   B. c = RSAEP((n, e), m)
   C. C = I2OSP(c ,k)
4. 암호문 C 출력

2)의 과정이 랜덤성 주입!

RSA

- RSA 표준 암복호화 과정

    1) 만약 레이블 L이 제공되지 않는 경우

        L은 empty 스트링이고,

        hLen을 해시함수의 출력 바이트수라 하면

        lHash = Hash(L)임

    2) PS는 0을 패딩하는 값으로

        k - mLen - 2hLen - 2개의 제로 옥텟으로 구성됨

    3) lHash, PS, 0x01, 메시지 M을 연접하여

        k - hLen - 1 옥텟 길이를 가지는 DB를 생성함

        => DB = | Hash || PS || 0x01 || M

    4) hLen 길이를 가지는 seed를 랜덤하게 생성함

    5) dbMask = MGF (seed, k - hLen - 1)

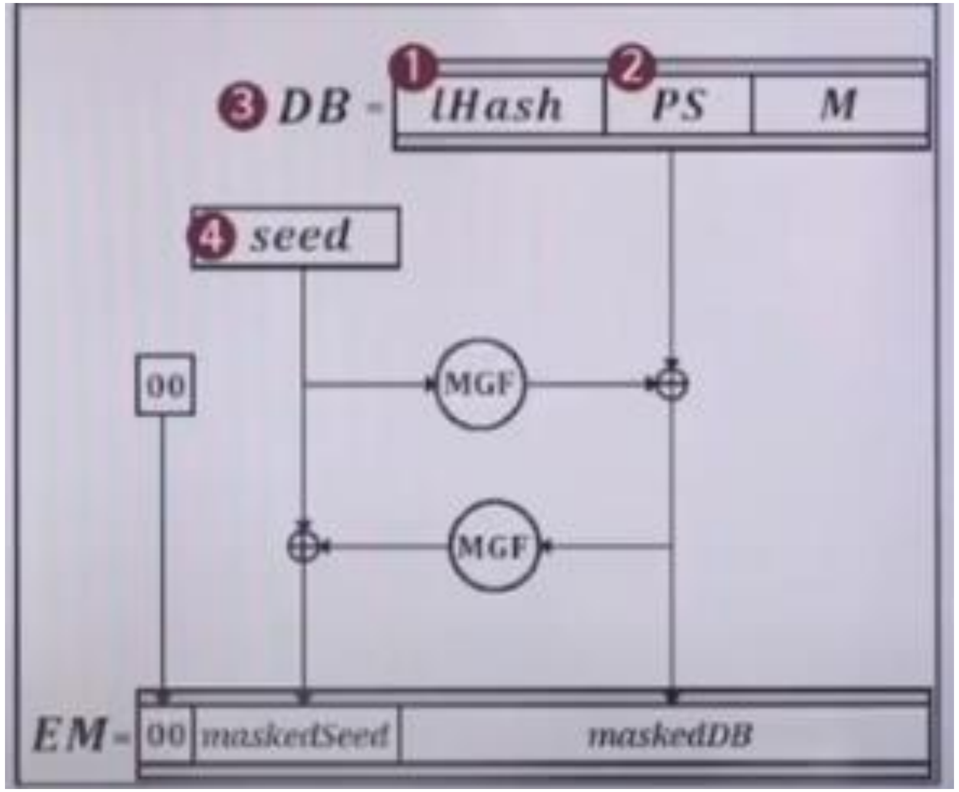
    6) maskedDB = DB xor dbMask

    7) seedMask = MGF (maskedDB, hLen)

    8) maskedSeed = seed xor seedMask

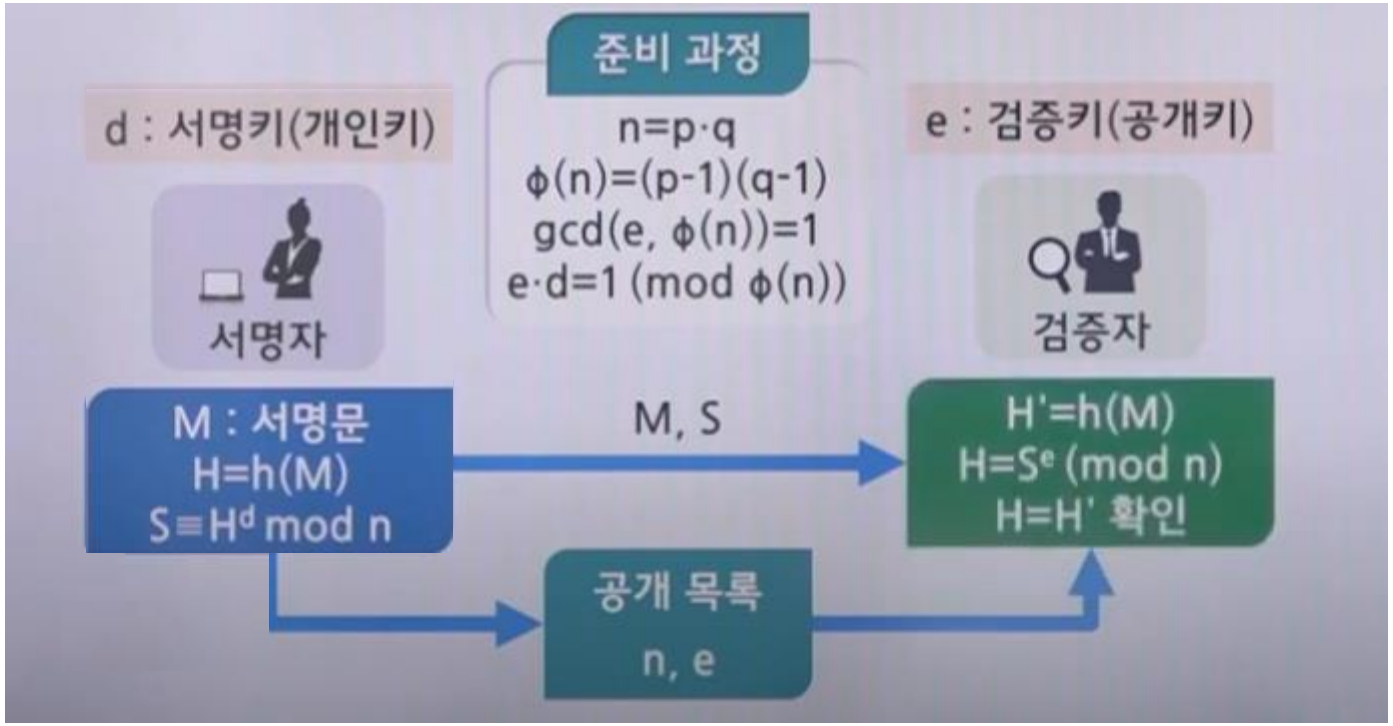
    9) 0x00, maskedSeed, maskedDB를 연접하여 k 옥텟 길이를 가지는 EM을 생성함

        => EM = 0x00 || maskedSeed || maskedDB



RSA

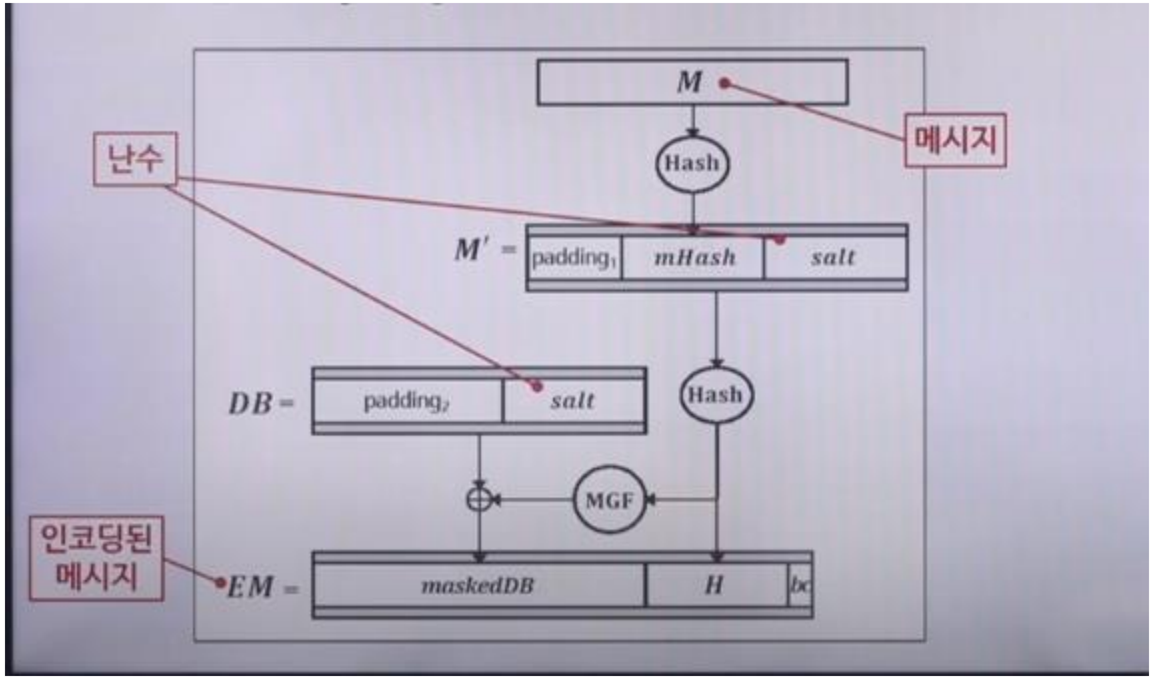
- 전자서명 개요



준비 과정
d : 서명키(개인키)
n=p·q
ϕ(n)=(p-1)(q-1)
gcd(e, ϕ(n))=1
e·d=1 (mod ϕ(n))
e : 검증키(공개키)
서명자
검증자
M : 서명문
H=h(M)
S≡H^d mod n
M, S
H'=h(M)
H=S^e (mod n)
H=H' 확인
공개 목록
n, e

RSA

- 전자서명 암복호화



RSA 성능 개선

- RSA 성능

$$S \equiv M^d \pmod{n}$$

$ed \equiv 1 \pmod{\varnothing(n)}$

d => 대략 2048bit

M, S, n => 대략 2048bit => 매우 느림

=> 해결하기 위한 방법 존재!

RSA 성능 개선

- 기본 개념

  이름 = 지수를 읽어가는 방법

  - Left-to-Right Exponentiation (최상위 -> 최하위)
  - Right-to-Left Exponentiation (최하위 -> 최상위)
  - Left-to-Right k-ary Exponentiation (최상위 -> 최하위, k개씩)

  지수 => 이진법으로 표현됨

  Ex) A^E을 계산할 때 $E = (e_k, e_{k-1}, \ldots, e_1, e_0)_2$ 로 표현

RSA 성능 개선

- 성능 개선 방법

$$A^E \bmod M$$

1) 다음과 같은 A의 지수(E)를 사전계산 해둠.

$A \bmod M$, $A^2 \bmod M$, $A^{2^2} \bmod M$, ..., $A^{2^k} \bmod M$

2) 미리 계산하여 2진법으로 표현된 E를 이용하여 모듈러 지수 연산 수행

RSA 성능 개선

- 성능 개선 방법 예시

(A = 2, E = 644, M = 645)
$A^E \bmod M = ?$

$E = 644 = (1010000100)_2$

$k = 9$

$2^1 = 2 \pmod{645}$

$2^2 = 4 \pmod{645}$

$2^4 = 16 \pmod{645}$

$2^8 = 256 \pmod{645}$

$2^{16} = 391 \pmod{645}$

$2^{32} = 16 \pmod{645}$

$2^{64} = 256 \pmod{645}$

$2^{128} = 391 \pmod{645}$

$2^{256} = 16 \pmod{645}$

$2^{512} = 256 \pmod{645}$

$2^{644} = 2^{512+128+4 =} 2^{512} \times 2^{128} \times 2^4 = 256 \times 391 \times 16 = 1 \pmod{645}$

2^{644} => 총 13번의 곱셈 연산

RSA 성능 개선

- Left-to-Right Exponentiation

Input: g, E = (e_t, e_{t-1}, ..., e_1, e_0), M
Output: g^E mod M

1. A <- 1
2. For i from t to 0 do the following:
    1. A <- A x A
    2. If e_i = 1, then A <- A x g

$A = 1$

$A \times A = A^2 = A^{(10)}$

$A^{2\wedge 2} = A^4 = A^{(100)}$

$A^4 \times A = A^{(100)} \times A^{(1)} = A^{(101)}$

=> 제곱은 뒤에 0을 붙여주는 효과!

=> A를 곱해주는 건 뒤에 1을 채워주는 효과!

RSA 성능 개선

- Left-to-Right Exponentiation

$E = (1011000)_2 \Rightarrow g^E \bmod M$

A = 1

A <- A x g => g^1

A <- A x A => $g^{(10)}$

A <- A x A => $g^{(100)}$ A <- A x g => $g^{(101)}$

A <- A x A => $g^{(1010)}$ A <- A x g => $g^{(1011)}$

A <- A x A => $g^{(10110)}$

A <- A x A => $g^{(101100)}$

A <- A x A => $g^{(1011000)}$

- t bit라면 t번의 제곱
- 1이 있을 때 곱함. 1이 있을 확률 = 1/2 => t/2번
- => 3t / 2 번
- => t가 2048bit이면 => 3072번의 연산

RSA 성능 개선

- Right-to-Left exponentiation

  input: g, $E = (e_t, e_{t-1}, \ldots, e_1, e_0)$, M

  Output: g^E mod M

  1. A <- 1, S <- g
  2. While E ≠ 0 do the following:
     1. if E is odd, then A <- A x S
     2. E <- E / 2
     3. If E ≠ 0, then S <- S x S

RSA 성능 개선

- Right-to-Left Exponentiation

$A = 1$	$S = g$	$E = 283$	1
$A = g$	$S = g^2$	$E = 141$	1
$A = g \times g^2 = g^3$	$S = g^4$	$E = 70$	0
$A = g^3$	$S = g^8$	$E = 35$	1
$A = g^3 \times g^8 = g^{11}$	$S = g^{16}$	$E = 17$	1
$A = g^{11} \times g^{16} = g^{27}$	$S = g^{32}$	$E = 8$	0
$A = g^{27}$	$S = g^{64}$	$E = 4$	0
$A = g^{27}$	$S = g^{128}$	$E = 2$	0
$A = g^{27}$	$S = g^{256}$	$E = 1$	1
$A = g^{27} \times g^{256} = g^{283}$			

RSA 성능 개선

- Left-to-Right K-ary Exponentiation

  $g^E \bmod M$

  $E = (e_t, e_{t-1}, ..., e_1, e_0)_2 = (k_t, k_{t-1}, ..., k_1, k_0)_B$

  $E = (1011011101110101011)_2$

  $= (10\ 110\ 111\ 011\ 101\ 011)_{2^3}$

  $k_5 = 10, k_4 = 110, k_3 = 111, k_2 = 011, k_1 = 101, k_0 = 011$

  g^{000} g^{001}, ..., g^{110}, g^{111}을 사전 계산

-> g^{10}	-> g^{10000}	-> g^{10000} x g^{110}
-> g^{10110}	-> $g^{10110000}$	-> $g^{10110000}$ x g^{111}
-> $g^{10110111}$	-> $g^{10110111000}$	-> $g^{10110111000}$ x g^{011}
-> $g^{10110111011}$	-> $g^{10110111011000}$	-> $g^{10110111011000}$ x g^{101}
-> $g^{10110111011101}$	-> $g^{10110111011101000}$	-> $g^{10110111011101000}$ x g^{011}
-> $g^{10110111011101011}$		

RSA 성능 개선

- 암호화

  $g^e \bmod n$

  => $e = 2^{16}+1$ = 0x10001 =$(10000000000000001)_2$

  => 16번의 제곱 + 1번의 곱셈 => 17번의 연산

- 복호화

  $g^d \bmod n$

  => g = 2048bit => (3x2048)/2 = 3072번의 연산

  ※ 암호화와 복호화가 약 200배 차이!

  => 새로운 방법 필요!

RSA 성능 개선

- 중국인의 나머지 정리(CRT, Chinese Remainder Theorem)
  : 연립 일차 합동방정식의 공통해를 찾는 문제를 유용하게 풀 수 있게 해주는 정리

Ex) 3으로 나누면 2가 남고,
5로 나누면 3이 남고,
7로 나누면 2가 남는 수 중에서
제일 작은 수는?

$X = 2 \bmod 3$
$X = 3 \bmod 5$
$X = 2 \bmod 7$
$X = ? \bmod (3 \times 5 \times 7)$

RSA 성능 개선

- 중국인의 나머지 정리(CRT, Chinese Remainder Theorem)

$m_1, m_2, \ldots m_k$가 쌍마다 서로소일 때

$x = a_1 \pmod{m_1}$
$x = a_2 \pmod{m_2}$
...
$x = a_k \pmod{m_k}$

을 만족하는 해는 $m = m_1 m_2 \ldots m_k$에서 유일함

따라서, x = ? mod m을 얻을 수 있음

→

p, q는 소수이므로 서로소

$c^d = a \pmod{p}$
$c^d = b \pmod{q}$

을 만족하는 해는 $n = pq$에서 유일함

c^d = ? mod n을 얻을 수 있음 ($n = pq$)

- 복호화 값을 연립일차 합동방정식? 으로 표현 => crt 사용 가능
- 어떤 공통된 값이 나오게 된다! 그게 바로 복호화 된 값
- 모듈러 값의 모든 쌍이 서로소! 여야 한다 (소수여야함)

RSA 성능 개선

X = a mod p와 x = b mod q를 만족하는 x 구하기

1) x = a*(u) + b*(v)라 하면

2) 1)식은 x = a mod p를 만족해야 하므로 모듈러 p에 대하여 a가 되어야 한다.

1)식의 b*(v)는 모듈러 p에 대하여 0이 되어야 하므로 v는 p의 배수이다.

그러므로 p가 곱해진 v = p * r이 된다

3) 마찬가지로 1)식의 x = b mod q를 만족해야 하므로 모듈러 q에 대하여 b가 되어야 한다.

따라서 1)식의 a*(u)는 모듈러 q에 대하여 0이 되어야 하므로 u는 q의 배수이다.

그러므로 q가 곱해진 u = q * s가 된다.

4) 식을 정리하면 x = a*(q*s) + b*(p*r) 이 된다.

RAS 성능 개선

5) 정리한 식에 모듈러 p를 취하면 x = a(q*s) mod p가 된다.

- 이것이 a와 같아야 하므로 q*s는 mod p 상에서 1이 되어야 한다.
- 따라서 $q*s = q*(q^{-1} \bmod p)$ 가 된다.
- 6) 정리한 식에 모듈러 q를 취하면 x = b*(p*r) mod q 가 된다.
- 이것이 b와 같아야 하므로 p*r은 mod q 상에서 1이 되어야 한다.
- 따라서 $p*r = p*(p^{-1} \bmod q)$가 된다.
- 7) 최종적으로 식을 정리하면
- $X = a*(p*(p^{-1} \bmod q)) + b*(q*(q^{-1} \bmod p))$ 를 얻게된다.

- => 4번의 곱셈과 2번의 역원 계산!

Q & A