정보보호

(5111041)

21장

공개키 암호화와
메시지 인증

보안 해시 함수

- 심플한 해시 함수 예

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

보안 해시 함수

	Bit 1	Bit 2	• • •	Bit n
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
Block m	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

Figure 21.1 Simple Hash Function Using Bitwise XOR

보안 해시 함수

- 각 비트 위치에 대한 단순 패리티 생성
- 세로 중복 검사
- 무결성 검사에 유효 → 메시지 인증에 사용됨
- n-비트 해시값이 균등 분포 : 에러 detection 확률 → 1 / 2^n
- 데이터 포맷팅에 따른 취약성
  - 정형화된 데이터 포맷(예: 각 텍스트 파일에서 최상위 비트는 항상 0)
    - 이 경우 128 해시 함수가 적용된다고 하더라도 2^-128 유효성이 아닌 2^-112 유효성을 가짐
  - Randomizing
    - 데이터의 규칙성에 대한 해시 함수의 취약점을 극복

보안 해시 알고리즘(SHA)

- SHA(Secure Hash Algorithm)는 NIST에서 개발됨
- 1993년 FIPS 180로 공표
- 1995년 SHA-1로 개정됨
  - 160-비트 해시 값 생성
- 2002년 NIST에서 FIPS 180-2 발행
  - 3개의 SHA 추가버전
  - SHA-256, SHA-384, SHA-512
  - 256/384/512-비트 해시 값
  - 기본구조는 SHA-1과 같지만 훨씬 안전해짐
- NIST에서는 2005년에 승인된 SHA-1의 폐지를 발표하고 2010년에 공표된 다른 SHA버전을 따르도록 함

Table 21.1
SHA 파라미터 비교

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80
Security	80	128	192	256

Notes: 1. All sizes are measured in bits.

2. Security refers to the fact that a birthday attack on a message digest of size *n* produces a collision with a work factor of approximately $2^{n/2}$.

SHA-512 구조

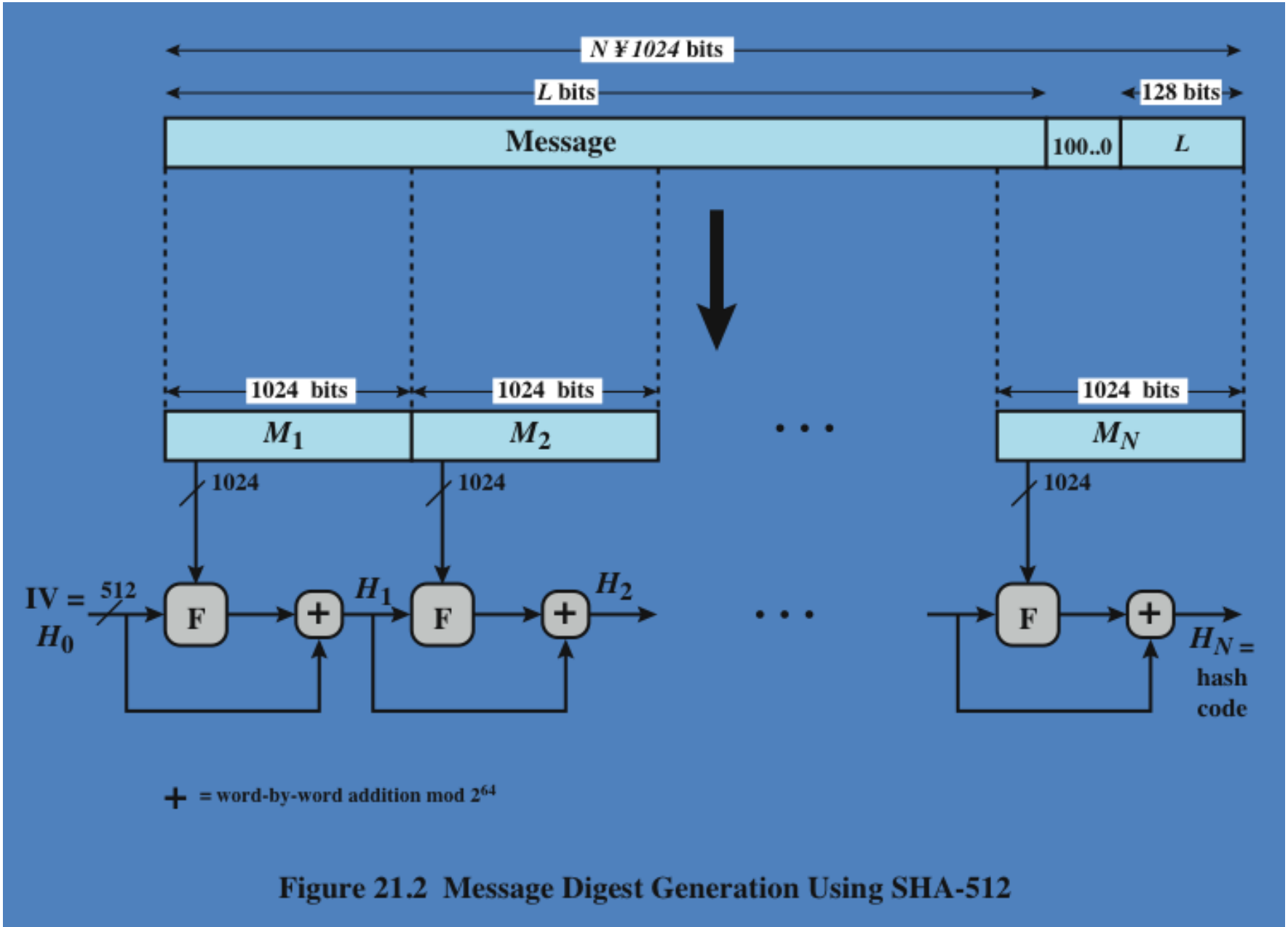


Figure 21.2 Message Digest Generation Using SHA-512

SHA-512 구조

- 1단계 : bit padding
  - 메시지 길이 mod 1024 = 896이 되도록 패딩
    - 첫 비트는 1, 그 이후는 0
    - 1,0,0,0,…
- 2단계 : length 정보L 덧붙임
  - 패딩되기 이전의 원래 길이
  - 총 128bit

➔ 확장된 메시지의 전체 길이는 N*1024

SHA-512 구조

- 3단계 : 해시버퍼 초기화
  - 8개의 64비트 레지스터 존재 ➔ 총 512bit

- **Step 3: Initialize hash buffer.** A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values):

  a = 6A09E667F3BCC908
  b = BB67AE8584CAA73B
  c = 3C6EF372FE94F82B
  d = A54FF53A5F1D36F1
  e = 510E527FADE682D1
  f = 9B05688C2B3E6C1F
  g = 1F83D9ABFB41BD6B
  h = 5BE0CD19137E2179

SHA-512 라운드

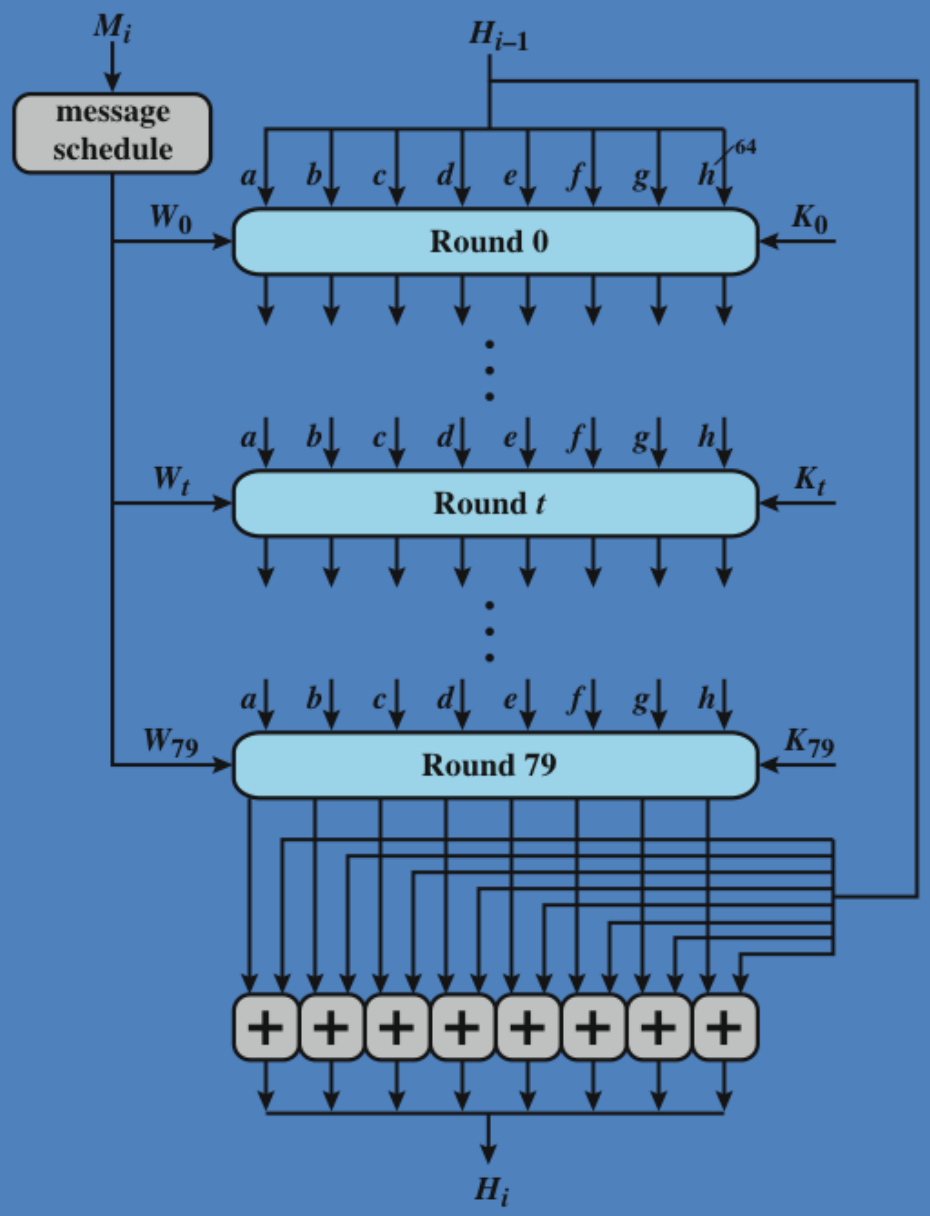


Figure 21.3 SHA-512 Processing of a Single 1024-Bit Block

SHA-512 구조

- 4단계 : 블록 라운드 함수 처리
  - 1024bit 입력(W_i), 512 해쉬버퍼($H_{(i-1)}$), 라운드 덧셈상수 K_i 를 이용하여 다음 단계 해쉬버퍼 중간값 H_i 계산
  - 80라운드 반복
    - AND, OR, NOT, XOR와 같은 프리비티브 부울 함수로 구성
- 5단계 : 출력
  - H_{80} 이 SHA-512 해쉬 알고리즘의 최종 출력

SHA-512 구조

- Message Expansion

4.1 Brief Description of SHA-512

SHA-512 is an iterated hash function that processes 1024-bit input message blocks and produces a 512-bit hash value. In the following, we briefly describe the hash function. It basically consists of two parts: the message expansion and the state update transformation. A detailed description of the hash function is given in [31].

Message Expansion. The message expansion of SHA-512 splits the 1024-bit message block into 16 64-bit words M_i and expands them into 80 expanded message words W_i as follows:

$$W_i = \begin{cases} M_i & 0 \leq i < 16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & 16 \leq i < 80 \end{cases}$$

The functions $\sigma_0(x)$ and $\sigma_1(x)$ are given by

$$\sigma_0(x) = (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7)$$
$$\sigma_1(x) = (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6).$$

Maria Eichlseder and Florian Mendel and Martin Schläffer (2014). "Branching Heuristics in Differential Collision Search with Applications to SHA-512" (PDF). IACR Cryptology ePrint Archive. 2014:302.

SHA-3

- SHA-1이 안전하지 않은 것으로 간주되어 SHA-2로 변환
- SHA-2는 동일한 이전 모델과 동일한 구조와 수학적 연산을 지녀 문제 초래
- SHA-2로 교체하는데 걸리는 시간적 취약성으로, NIST는 2007 Competition에서 SHA-3을 발표

요구사항:

- 224, 256, 384, 512 비트 길이의 해시 값을 지원해야 함
- 알고리즘의 처리 전, 전체 메시지의 버퍼링을 요청하는 대신 알고리즘이 작은 블록을 한번에 처리해야 함

SHA-3 평가 기준

- SHA-2가 지원하는 주요 어플리케이션에 대한 요구사항을 반영하도록 고안됨
  - 디지털 시그니처, 해시 메시지 인증 코드, 키 생성, 의사난수 생성
- 보안
  - 강도는 다양하게 요구되는 해시 사이즈 및 역상 저항성과 충돌저항성 모두에 대하여 이론인 최대치에 가까워야 함
  - SHA-2 함수에 대한 어떠한 잠재적인 공격에도 저항할 수 있도록 설계되어야 함
- 비용
  - 하드웨어 플랫폼 범위에 대하여 시간과 메모리 모두 충분해야 함
- 알고리즘과 구현 특징
  - 유연성과 단순성이 고려되어야 함

HMAC

- 암호화 해시 코드에서 파생된 MAC 개발에 관심
  - 대체로 암호화 해시 함수의 실행이 더 빠름
  - 라이브러리 코드가 널리 이용 됨
  - SHA-1은 비밀키를 필요로 하지 않기 때문에 MAC으로 사용되지 못하게 설계됨

- 보안을 목적으로 의무적 MAC 구현 채택
  - 전송 계층 보안(TLS)나 안전 전자 거래(SET)와 같은 인터넷 프로토콜에서 사용됨

HMAC: Hash-based message authentication code

HMAC 설계 목적

수정작업 없이 이용 가능한 해시함수를 사용하기 위한 목적

성능의 저하 없이 해시함수의 본 성능을 유지하기 위한 목적

더 빠르고 보안 해시함수가 요구되는 경우, 기존 해시함수의 교체를 용이하게 하기 위한 목적

간단한 방법으로 키를 사용하기 위한 목적

내장된 해시함수에 대한 타당한 가정을 기반으로 하는 인증 매커니즘의 강도를 암호학적으로 잘 분석할 수 있도록 하기 위한 목적

HMAC 구조

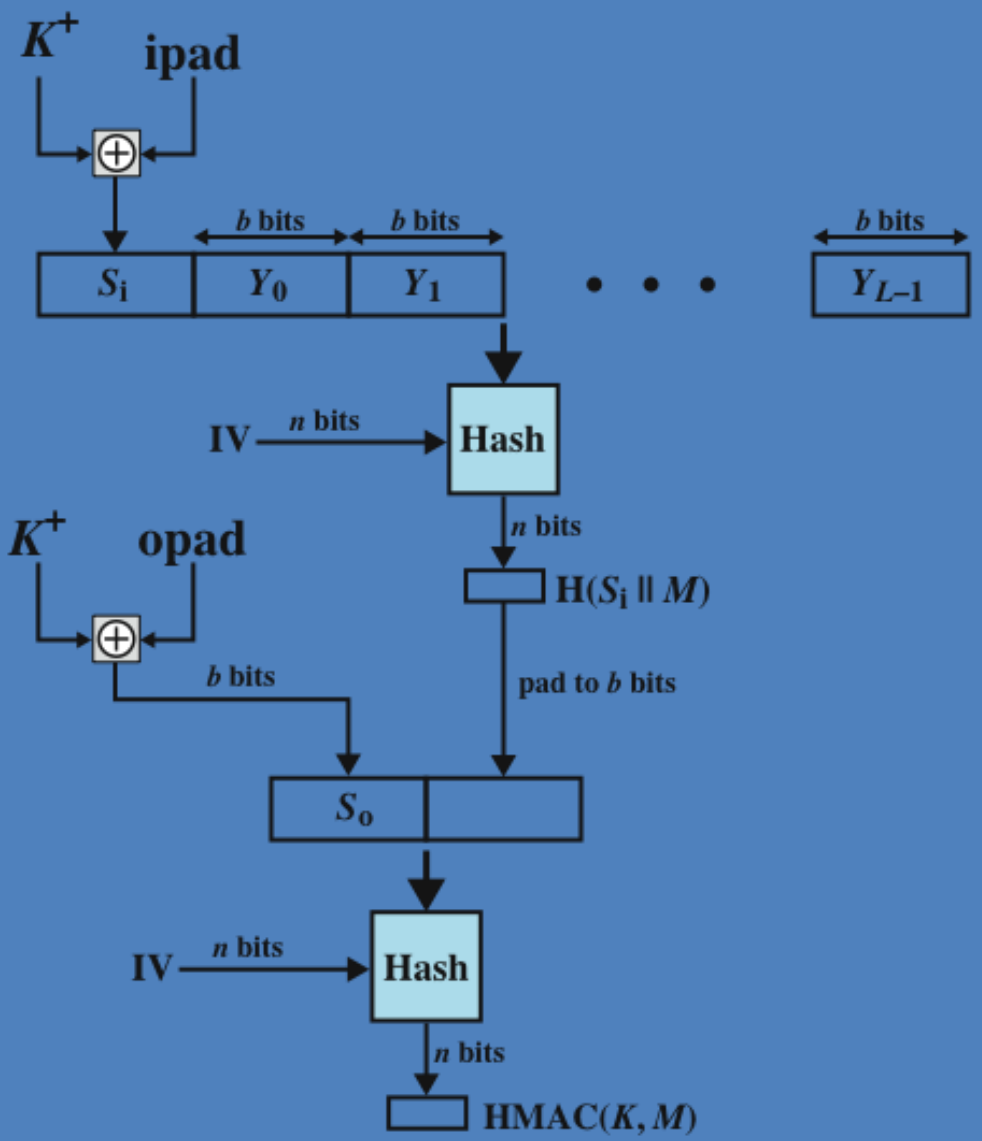


Figure 21.4 HMAC Structure

HMAC

- Notation
  - M : embedded hash function (e.g. SHA)
  - M : input message to HMAC
  - Yi : M의 i번째 블록 (including the padding bits)
  - L : 블록의 수
  - b : 블록 길이
  - n : hash 코드 길이
  - K : 비밀키
  - K+ : 비밀키가 b bit가 되도록 앞에 0을 padding 한 값
  - ipad : 00110110 (0x36) 이 b/8번 반복
  - opad : 01011100 (0x5C) 이 b/8번 반복

HMAC

- HMAC 출력 결과

Then HMAC can be expressed as follows:

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[K^+ \oplus \text{ipad}] \| M]]$$

In words,

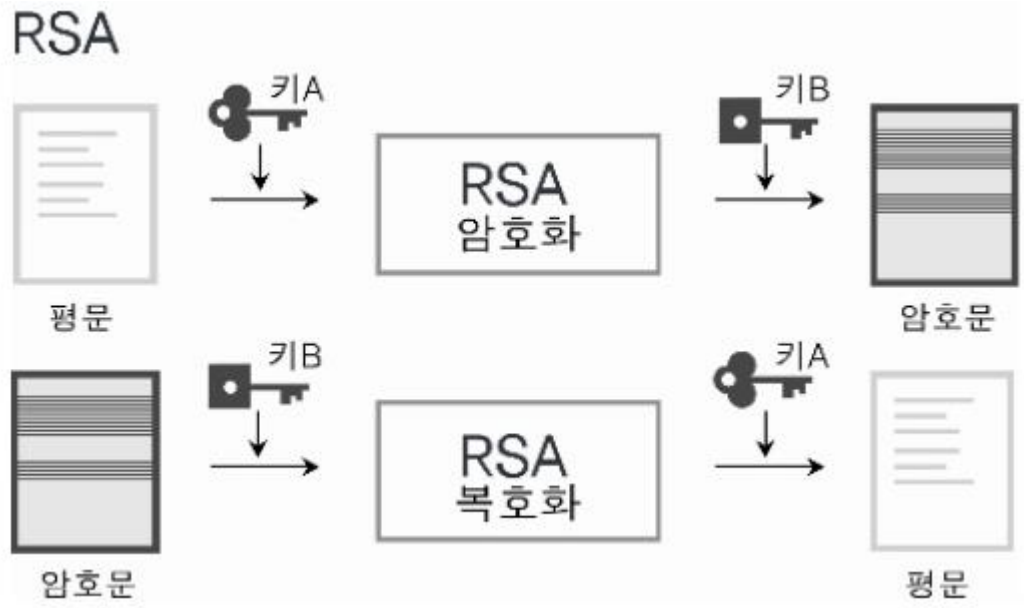
1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zero bytes 0x00).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

HMAC 보안

- 보안은 기본 해시함수의 암호화 강도에 따라 달라짐
- 아래 두 종류 공격이 가능
  - 공격자는 랜덤 시크릿 IV의 결과값을 산출하거나
    - brute force 키 $O(2^n)$, 또는 birthday 공격 사용
  - 또는 공격자는 IV값이 랜덤 값이면서 시크릿 값일 경우 해시함수에서 충돌한 것을 인지
    - ie. find M and M' such that $H(M) = H(M')$
    - birthday 공격 $O(2^{n/2})$

RSA 공개키 암호화

- 데이터의 암호화/복호화뿐만 아니라 키교환, 서명/인증 알고리듬 구현 가능하게 함.



RSA 공개키 암호화

- 일방향 함수를 사용
  - 일방향 함수 : 한쪽으로는 계산이 용이하지만 반대쪽으로는 계산하기 어려운 함수
  - 예1)

> 우리가 종이와 연필을 사용 하여 어떤 숫자의 제곱을 구하는 것은 어렵지 않습니다. 그러나 제곱에서 그 제곱근을 구하는 것은 어렵습니다.
>
> 78 의 제곱은 6084 입니다. 이것을 계산 하는 것은 쉽습니다. 그러나 종이와 연필로만 계산을 했을 때 6084로부터 제곱근인 78을 얻어내는 것은 어렵습니다. 숫자가 커지면 커질수록 계산하는 것은 더 어려워 질 것입니다. 25의 제곱근이 5 라는 것은 쉽게 구할 수 있지만, 19053225 의 제곱근이 4365 라는 것은 쉽게 알기가 어렵습니다.

  - 예2) → RSA 적용 원리

> 두 개의 매우 큰 소수의 곱을 구하는 것은 용이 하지만 그 곱에서 원래의 두 개의 소수를 구하는 것은 매우 어렵습니다.

RSA 공개키 암호화

- n = 1143816257578888676692357799761466120102182967212423625625618429357069352457338978305971235639587050589890751475992900268795435 41

  일 때,

- p = 3490529510847650949147849619903898133417764638493387843990820577
- q = 327691329932667095499619881908344614131776429679929425397982885 33
- 위의 사실을 알아 내기 위해서는 Brute Force로 수십 년 소요

RSA 공개키 암호화

- 1977년 MIT의 Rivest, Shamir 와 Adleman 에 의해 만들어짐
- 가장 잘 알려지고 널리 사용되고 있는 공개키 알고리즘
- 지수화된 정수를 소수(prime)로 모듈로(modulo)연산
- 암호화: $C = M^e \bmod n$
- 복호화: $M = C^d \bmod n = (M^e)^d \bmod n = M$
- 송신자와 수신자 모두 n과 e값을 알고 있음
- 송신자 만이 d값을 앎
- 공개키 암호화 알고리즘
  - public key $PU = \{e, n\}$ and private key $PR = \{d, n\}$.

RSA 알고리즘

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 21.5 The RSA Algorithm

RSA 공개키 암호화

- 키생성

1. 서로 다른 임의의 두개의 소수 p 와 q 를 선택 합니다. p 와 q 는 클수록 암호화의 안정성이 높아 집니다.

2. p 와 q 를 곱한 값 n 을 생성 합니다

n = p * q

3. 오일러 파이 함수 φ(p) 값을 구합니다.

φ(p) = (p-1) * (q-1)

4. 오일러 파이 함수 φ(n)란 무엇일까요?

오일러 파이 함수 φ(n) 은 1부터 n-1 까지의 양의 정수 중에서 n 과 서로 소의 관계에 있는 정수들의 개수를 나타냅니다. 두개 의 정수가 서로 소 라고 하는 것은 두 숫자의 최대 공약수가 1인 것을 말 합니다. 즉 1 이외에는 두 숫자에서 공통적으로 나눌 수 있는 숫자가 없다는 것 입니다.

오일러 파이 함수 φ(n) 의 특별한 경우로서, 다음이 성립 합니다.

l 만약 n 이 소수라면 , φ(n) = n – 1 입니다.

l 또 양의 정수 n 이 두 개의 소수 p 와 q 의 곱으로 이루어져 있다면,

φ(n) = (p-1) * (q-1) 입니다.

RSA 공개키 암호화

- 오일러 함수 예
  - Φ(7) = 6
    - 1,2,3,4,5,6
  - Φ(15) = 8
    - 1,2,4,7,8,10,11,13,14
  - Φ(15) 에 대해 오일러 공식 적용
    - 15=3*5
    - Φ(15) = (3-1) * (5-1) = 2*4 = 8

5. φ(n) 과 서로소의 관계에 있는 e를 구합니다. 단 e 는 1 < e < φ(n) 의 범위에 있는 수이어야 합니다. φ(n) 과 e 는 서로 소 이므로 둘의 최대 공약수는 1 이어야 합니다.

Gcd(e , φ(n)) = 1

RSA 공개키 암호화

6. 법 (modulus)에 대해서 알아 봅시다.

법의 정의에 의하여 80분을 60분인 법을 다음과 같이 수학적으로 표현 합니다.

80 ≡ 20(mod 60)

7. 오일러의 정리에 대해서 알아 봅시다

오일러의 정리는 위에서 소개한 법의 특수한 경우를 공식으로 만든 것입니다. 두 양의 정수 a 와 n 가 서로 소 라면 다음이 성립 합니다.

a^ φ(n) ≡ 1(mod n)

이것을 식으로 표현 하면 다음과 같을 것입니다.

a^ φ(n) mod n = 1

- **예시) a= 3, n = 5**
  - **ϕ(5)=4 이므로 a^ϕ(n)=3^4=81**
  - **a^ϕ(n) mod n ➔ 81 mod 5 = 1 !!!**
  - **위의 오일러 정리가 성립!!!!**

RSA 공개키 암호화

8. e * d ≡ 1(mod φ(n)) 의 식이 성립하는 d 를 구합니다.

즉 (e * d) mod φ(n) = 1 이 되는 d 를 구합니다. 단 d는 d < φ(n) 의 범위에 있는 정수 입니다.

- 앞의 과정에서 구한 e, d를 이용하여 개인키와 공개키를 아래와 다음과 같이 결정!
  - 개인키 : {e, n}
  - 공개키 : {d, n}

RSA 공개키 암호화

- 암호화

평문을 M 이라고 합니다.
M 은 정수 이며 , M < n 의 값 입니다.

암호문은 다음과 같이 계산 될 수 있습니다.

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

- 복호화

암호문을 C 라고 합니다.

암호문은 다음과 같은 식에 의해서 평문으로 계산 될 수 있습니다.

$$M = C^d \pmod n$$

RSA 예제

- p = 5, q = 7
- n = p*q = 5*7 = 35
- Φ(n) = (p-1)(q-1) = (5-1) * (7-1) = 24
- 1<e<n 의 범위에서, Φ(n)=24와 서로 소인 e 값을 구함
  - e = 7로 결정
- d<Φ(n) 의 범위에서, (e*d) mod Φ(n) = 1이 되는 d를 구함
  - (7 * d) mod = 1을 만족하며 d<24인 정수 중 하나는 7

➔ 개인키 : {7, 35}, 공개키 : {7, 35} !!!!

RSA 예제

- 암호화
  - 평문을 3이라고 가정
  - C = M^e (mod n) 이므로,
  - C = 3^7 (mod 35) = 2187 mod 35 = 17
    ➔ 평문 3이 암호문 17로 ciphering!!!
- 복호화
  - M = C^d (mod n) 이므로,
  - M = 17^7 (mod 35) = 410338673 mod 35 = 3

- RSA 알고리즘을 이용하여 암호화 및 복호화 성공

(또 다른) RSA 예제 ➔ HW

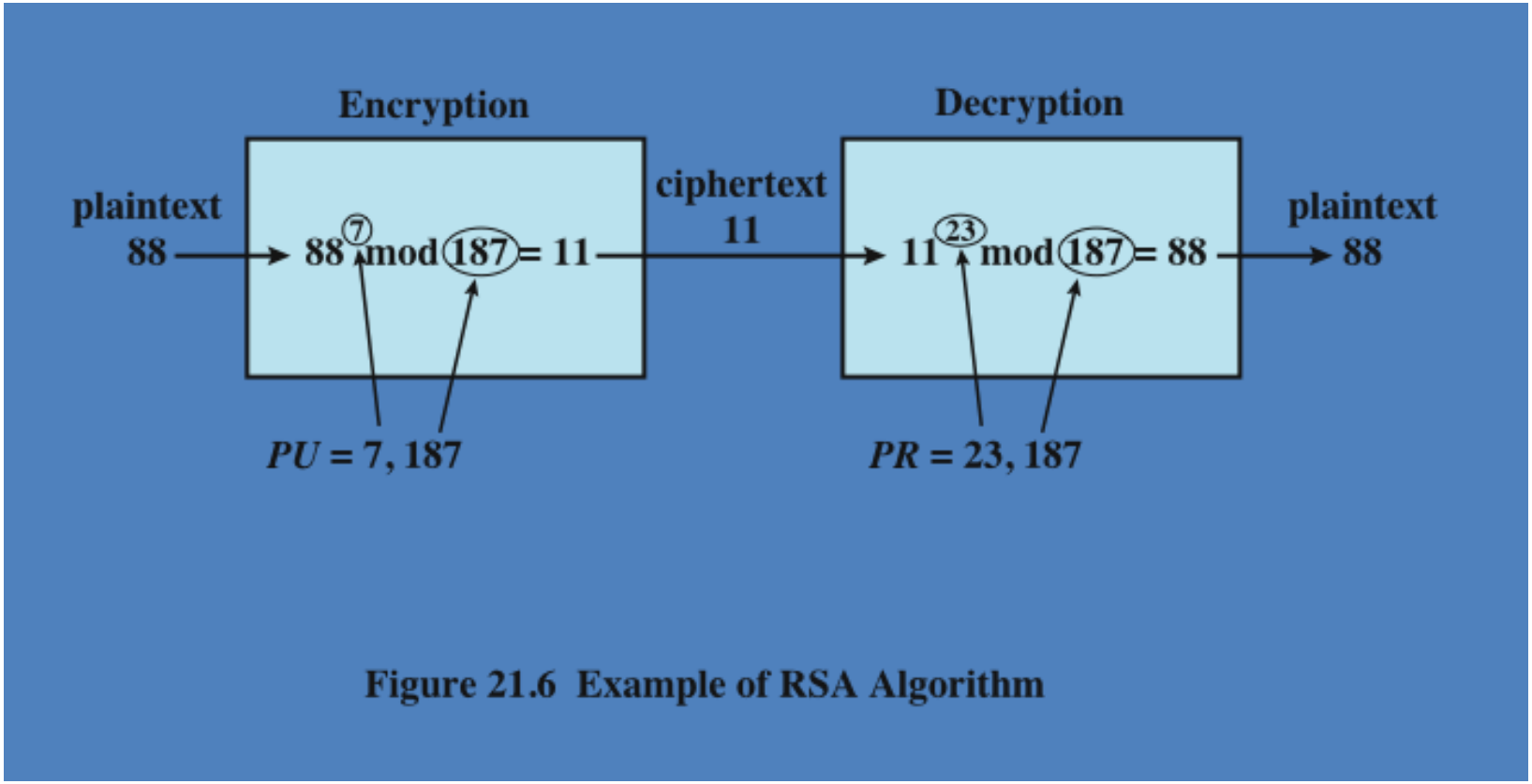


Figure 21.6 Example of RSA Algorithm

RSA 보안

부르트 포스 (brute force)

- 가능한 모든 개인키에 시도
- 커다란 키 공간을 사용하여 방어하지만, 이는 실행속도를 느리게 함

수학적 공격 (mathematical attacks)

- 몇 가지 접근법 ,두 가지 주요 요소의 곱을 인수분해 한 값과 동일

타이밍 공격 (timing attacks)

- 암호 알고리즘의 러닝 타임에 따름
- 예상치 못한 방향으로 나타나며, 암호문 단독 공격
- 대응조치: 일관성 있는 지수화 시간, 임의적 지연, 블라인딩(blinding)

선택된 암호문 공격

- RSA 알고리즘의 속성을 이용한 공격

Table 21.2
인수분해 진척
(Progress in Factorization)

- 주어진 e와 n을 가지고 d를 계산하는 것은 이것을 소인수분해하는 문제의 복잡도와 동치임이 수학적으로 증명

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-Years
100	332	April 1991	7
110	365	April 1992	75
120	398	June 1993	830
129	428	April 1994	5000
130	431	April 1996	1000
140	465	February 1999	2000
155	512	August 1999	8000
160	530	April 2003	—
174	576	December 2003	—
200	663	May 2005	—

21장 #2

공개키 암호화와
메시지 인증

Diffie-Hellman 키 교환

- 공개키 알고리즘의 시초
- 1976년 Diffle과 Hellman이 공개키 컨셉 박람회에서 공개
- 많은 상업 제품에 이용됨
- 비밀키를 안전하게 교환하여 차후의 메시지 암호화에 사용되는 실용적 방법
- 보안은 이산 로그 계산의 난이도에 따라 달라짐

Diffie-Hellman 원리

- 단방향 함수 이용
  - g,x,p을 이용하여 y=g^x mod p를 구하는 것은 쉽지만,
  - g,y,p를 이용하여 원래의 x를 구하는 것은 어렵다는 원리 이용

- 용어정리
  - 비밀키 : A와 B가 공유하고자 하는 암호화에 사용되는 키
  - 공개키 : A와 B가 외부에 공개하는 키
  - 개인키 : A와 B가 각각 개인이 소유

Diffie-Hellman 원리

- 알고리듬의 핵심
  - A의 비밀키 = A의 개인키 [DH연산] B의 공개키
  - B의 비밀키 = B의 개인키 [DH연산] A의 공개키
  - A의 비밀키 = B의 비밀키

- 이산대수 성질 활용
  - 소수q 와 q의 원시근 a
    - 1과 q-1사이의 모든 정수를 a mod q, a^2 mod q, a^3 mod q, … , a^(q-1) mod q 로 나타낼 수 있다
    - 디피-헬만 알고리듬에서 q와 a는 known value

Diffie-Hellman 원리

- A 개인키 및 공개키 생성
  - q보다 작은 임의의 수 X_A를 선택 → 개인키로 지정
  - 자기 자신의 공개를 다음과 같이 계산
    - $Y_A = q^{X_A} \bmod q$
  - 키를 교환하고자 하는 상대방(B)에게 자신의 공개키 Y_A를 전달
- B 개인키 및 공개키 생성
  - q보다 작은 임의의 수 X_B를 선택 → 개인키로 지정
  - 자기 자신의 공개를 다음과 같이 계산
    - $Y_B = q^{X_B} \bmod q$
  - 키를 교환하고자 하는 상대방(A)에게 자신의 공개키 Y_B를 전달

Diffie-Hellman 원리

- A 비밀키 계산
  - 자신(A)의 개인키와 B의 공개키를 이용
  - $K_A = Y_B \text{^} X_A \bmod q$
    $= (q \text{^} X_B \bmod q) \text{^} X_A \bmod q = (q \text{^} X_B) \text{^} X_A \bmod q$
- B 비밀키 계산
  - 자신(B)의 개인키와 A의 공개키를 이용
  - $K_B = Y_A \text{^} X_B \bmod q$
    $= (q \text{^} X_A \bmod q) \text{^} X_B \bmod q = (q \text{^} X_A) \text{^} X_B \bmod q$

- $K_A = q \text{^} X_B X_A \bmod q$, $K_B = q \text{^} X_A X_B \bmod q$
- ➔ $K_A = K_B$!!! ➔ A와 B가 성공적으로 비밀키를 공유

Diffie-Hellman 키 교환 알고리즘

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 21.7 The Diffie-Hellman Key Exchange Algorithm

Diffie-Hellman 예제

다음을 가지고 있음

- prime number $q = 353$
- primitive root $\alpha = 3$

A와 B는 각각 그들의 공개키를 연산함

- A computes $Y_A = 3^{97} \bmod 353 = 40$
- B computes $Y_B = 3^{233} \bmod 353 = 248$

그러고 나서 교환하고 비밀키 연산:

- for A: $K = (Y_B)^{XA} \bmod 353 = 248^{97} \bmod 353 = 160$
- *for B:* $K = (Y_A)^{XB} \bmod 353 = 40^{233} \bmod 353 = 160$

공격자는 다음을 풀어야 함:

- $3^a \bmod 353 = 40$ which is hard
- desired answer is 97, then compute key as B does

키 교환 프로토콜

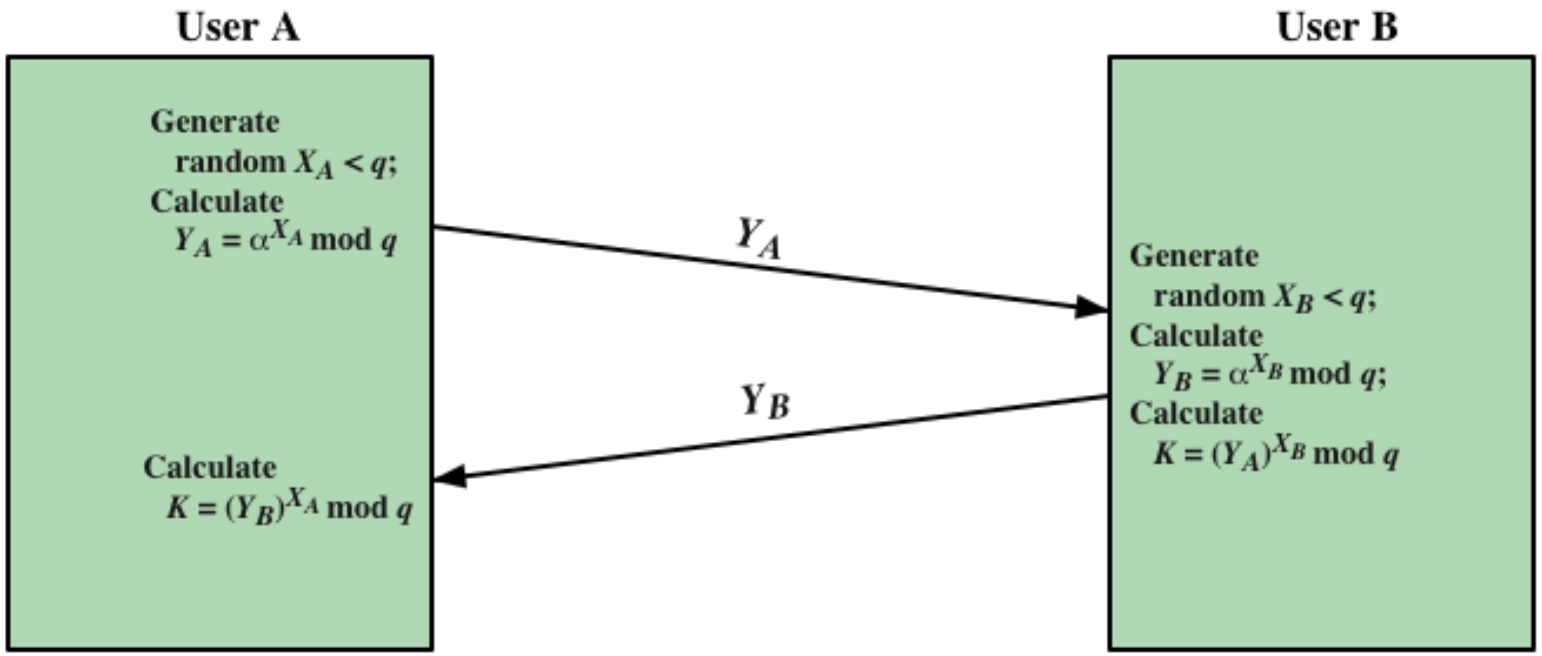


Figure 21.8 Diffie-Hellman Key Exchange

사람이 개입되는 공격

- 공격은 다음과 같음:
  1. Darth 는 개인키X_{D1} & X_{D2}와 공개키 Y_{D1} & Y_{D2}를 생성
  2. Alice는 Bob에게 Y_A 전송
  3. Darth는Y_A를 가로채 Bob에게Y_{D1}를 전송. Darth는 또한 K2 값을 계산
  4. Bob은 Y_{D1}를 갖고 K1값을 계산
  5. Bob X_A를 Alice에게 전송
  6. Darth는 X_A를 가로채 Alice에게 Y_{D2}를 전송. Darth는 K1값을 계산
  7. Alice는 Y_{D2}을 받아 K2값을 계산
- 차후 모든 통신이 차단됨

기타 공개키 알고리즘

DSS
(Digital Signature Standard)

- FIPS PUB 186
- SHA-1과 디지털 서명 알고리즘(DSA)사용
- 1991년에 처음 제안되어 1993년에 보안성 문제로 개정됨. 1996년에 또 다시 교정됨
- 암호화나 키 교환에는 사용될 수 없음
- 디지털 서명 기능만을 제공하도록 고안된 알고리즘 사용

ECC
(Elliptic-Curve Cryptography)

- RSA보다 작은 비트사이즈의 보안
- IEEE P1363에 정의됨
- ECC의 신뢰수준은 아직 RSA만큼 높지 않음
- 타원 곡선으로 알려진 수학적 구조를 기반으로 함

요약

- 보안 해시 함수
  - 단순 해시 함수
  - SHA 보안 해시함수
  - SHA-3
- HMAC
  - HMAC design objectives
  - HMAC algorithm
  - security of HMAC
- RSA public-key encryption algorithm
  - description of the algorithm
  - security of RSA
- Diffie-Hellman key exchange
- elliptic-curve cryptography