

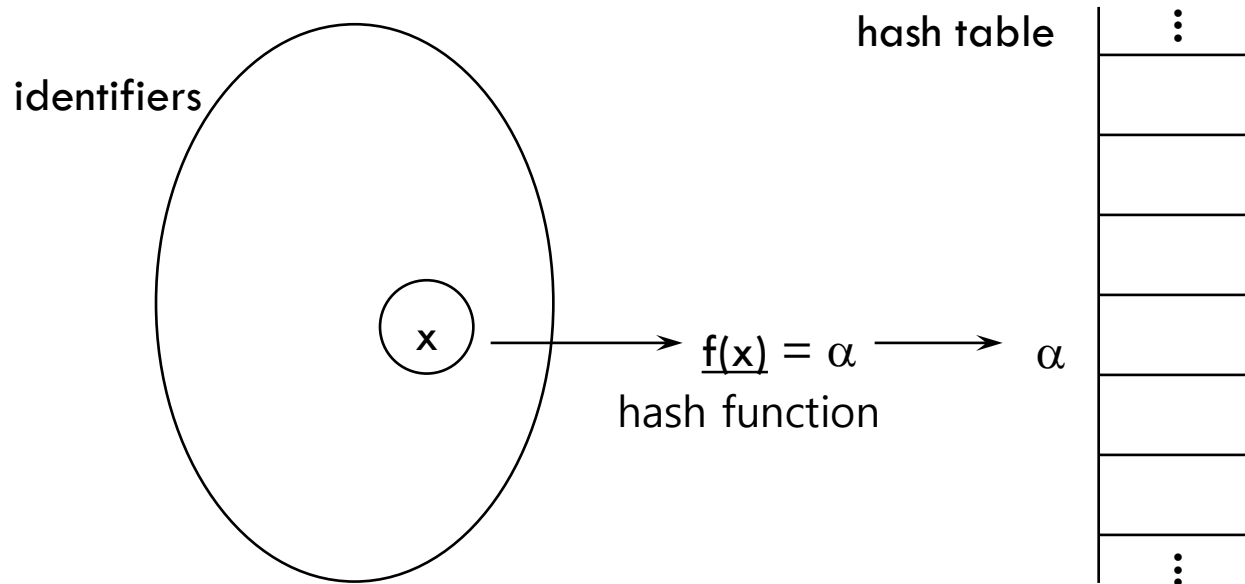
정보보호

05 암호개론 (3)

Hashing (1)

- ▶ dictionary
 - ▶ **symbol table** in computer science
 - ▶ application
 - ▶ spelling checker
 - ▶ thesarus
 - ▶ data dictionary in database application
 - ▶ symbol tables generated by loader, assembler, and compiler
- ▶ operations on symbol table
 - ▶ determine if a particular name is in the table
 - ▶ retrieve the attributes of that name
 - ▶ modify the attributes of that name
 - ▶ insert a new name and its attribute
 - ▶ delete a name and its attributes
- ▶ use hashing
 - ▶ very good expected performance: $O(1)$

Hashing (2)



Hashing (3)

▶ Hash Table

▶ Def)

- ▶ identifier density of a hash table:
 - ▶ n/T where
 - ▶ n : number of identifiers in table
 - ▶ T : total number of possible identifiers
- ▶ loading density or loading factor of a hash table:
 - ▶ $a = n/(s \cdot b)$ where
 - ▶ s : number of slots in each bucket
 - ▶ b : number of bucket
- ▶ two identifiers i_1 and i_2 are *synonyms* with respect to f , if
 - ▶ $f(i_1) = f(i_2)$ where $i_1 \neq i_2$
- ▶ an *overflow* occurs when
 - ▶ we hash a new identifier, i , into a full bucket
- ▶ a *collision* occurs when
 - ▶ we hash two nonidentical identifiers into the same bucket
- ▶ collisions and overflows occur simultaneously iff
 - ▶ bucket size is 1

Hashing (4)

Example) hash table ht with $b=26$, $s=2$, $n=10$

- ▶ hash function f
 - ▶ 1st character of identifier

	slot 0	slot 1
0	acos	atan
1		
2	char	ceil
3	define	
4	exp	
5	float	floor
6		
...		
25		

<u>Identifiers</u>
acos
define
float
exp
char
atan
ceil
floor
clock
ctime

- ▶ hash table with 26 bucket and two slots per bucket

Hashing (5)

▶ Hash Function

▶ requirements for a hash function

- ▶ easy to compute

- ▶ minimizes the number of collision (but, we can not avoid collisions)

▶ uniform hash function

- ▶ for randomly chosen x from the identifier space, $P[f(x)=i] = 1/b$, for all buckets i

- ▶ a random x has an equal chance of hashing into any of the b buckets

Hashing (6)

▶ Hash Function (cont.)

▶ **mid-square**

- ▶ middle of square hash function
- ▶ frequently used in symbol table applications
- ▶ hash function f_m
 - ▶ squaring the identifier
 - ▶ obtain the bucket address by using an appropriate number of bits from the middle of the square
 - ▶ if we use r bits, 2^r buckets are necessary

Hashing (7)

▶ Hash Function (cont.)

▶ **division(modular)**

- ▶ use the modulus(%) operator
- ▶ $f_D(x) = x \% M$ where M : table size
 - ▶ range of bucket address: $0 \sim M-1$
 - ▶ the choice of M is critical
 - ▶ choose M as a prime number such that M does not divide $r^{k \pm a}$ for small k and a
 - ▶ choose M such that it has no prime divisors less than 20

Hashing (8)

▶ Hash Function (cont.)

▶ folding

▶ shift folding

▶ ex) identifier $x = 12320324111220$

x_1	123		123
x_2	203		203
x_3	241		241
x_4	112		112
x_5	20	+	20
			699

▶ folding at the boundaries

$$x_2 \quad \boxed{203} \rightarrow \boxed{302} \quad \quad x_4 \quad \boxed{112} \rightarrow \boxed{211}$$

▶ $\rightarrow 123 + 302 + 241 + 211 + 20 = 897$

Hashing (9)

▶ Hash Function (cont.)

▶ **digit analysis**

- ▶ used in case all the identifiers are known in advance
- ▶ examine the digits of each identifier
- ▶ delete those digits that have skewed distributions
- ▶ select the digit positions to be used to calculate the hash address

Hashing (10)

▶ Overflow Handling

▶ linear open addressing

▶ linear probing

- ▶ when overflow occurs, linear search for the empty slot in the hash table using circular rotation

bucket	x	# of comparisons
0	acos	1
1	atoi	2
2	char	1
3	define	1
4	exp	1
5	ceil	4
6	cos	5
7	float	3
8	atol	9
9	floor	5
10	ctime	9
...		
25		

Hashing (11)

▶ Overflow Handling (cont.)

▶ linear open addressing (cont.)

▶ quadratic probing

- ▶ examine the hash table buckets $ht[f(x)]$, $ht[(f(x) + i^2) \% b]$, $ht[(f(x) - i^2) \% b]$,
 - ▶ for $0 \leq i \leq (b-1)/2$, where b : number of buckets in the table
- ▶ reduce the average number of probes

▶ rehashing

- ▶ use a series of hashing functions f_1, f_2, \dots, f_b
- ▶ bucket $f_i(x)$ is examined for $i = 1, 2, \dots, b$

Hashing (12)

▶ Overflow Handling (cont.)

▶ chaining

- ▶ defect of linear probing
 - ▶ comparison of identifiers with different hash values
- ▶ maintain list of identifiers
 - ▶ one list per one bucket
 - ▶ each list has all the synonyms
 - ▶ requires a head node for each chain



bucket(head node)

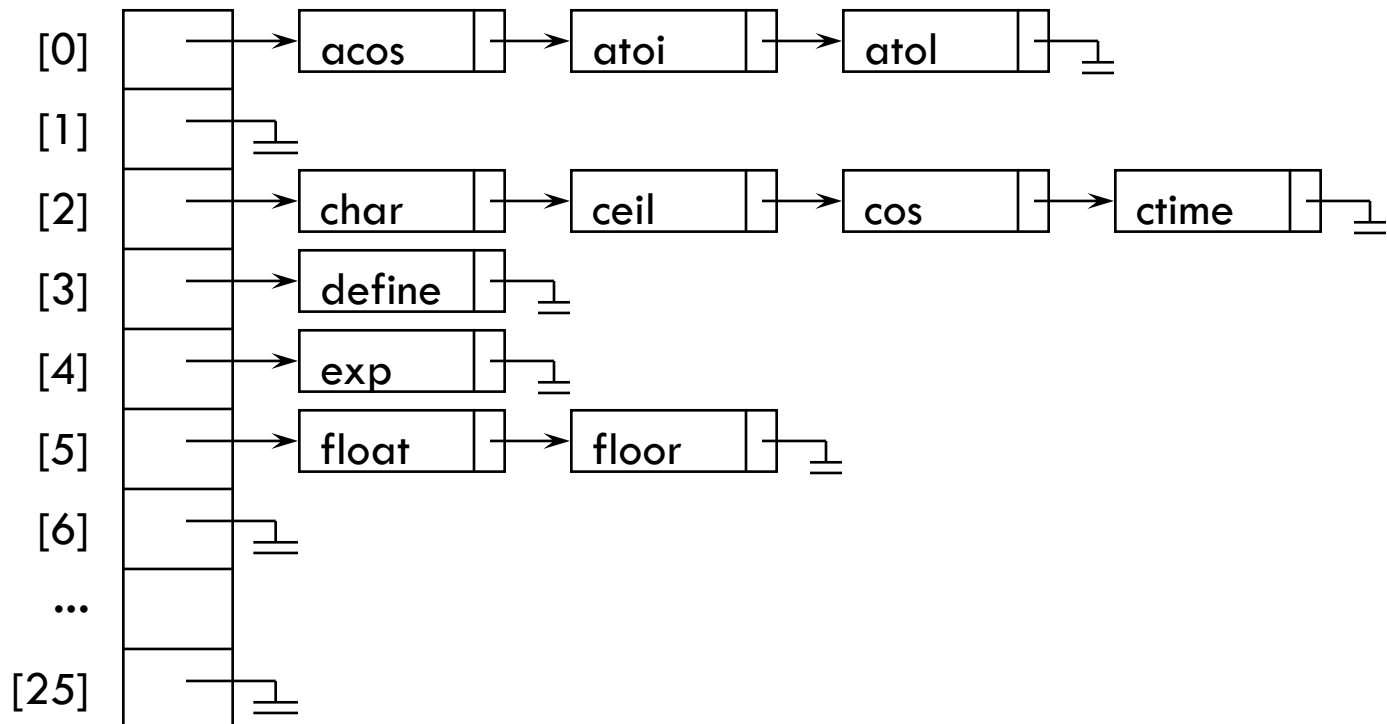


list(linked list)

Hashing (13)

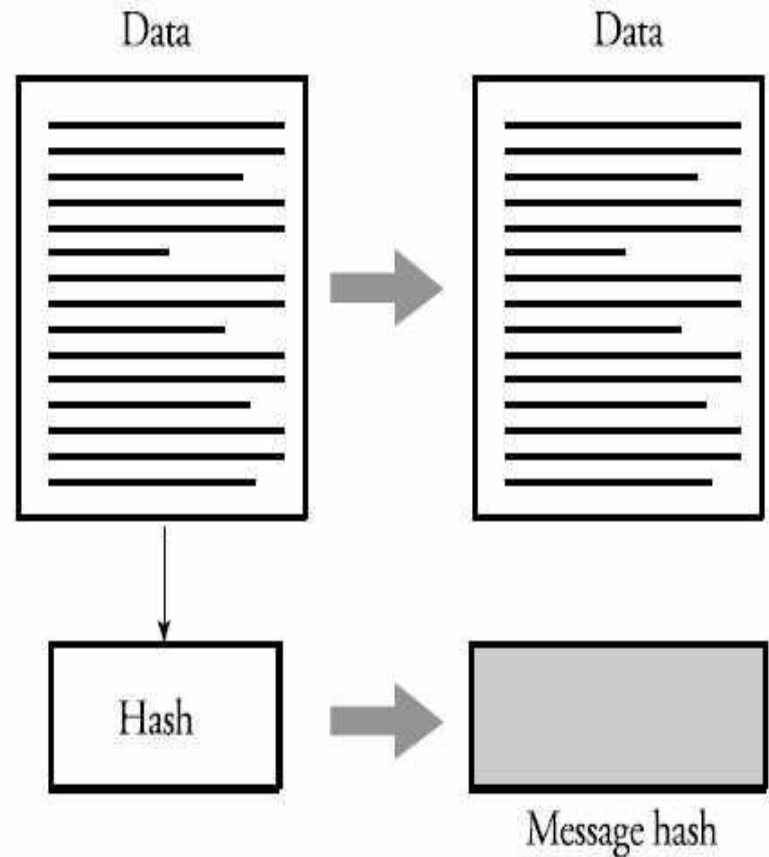
► Overflow Handling (cont.)

► chaining (cont.)



정보보호에서의 해시함수 이용

- ▶ 데이터 무결성을 제공하는 알고리즘 중 하나
 - ▶ 메시지 인증 알고리즘
 - ▶ 단방향 해시함수
- ▶ 임의의 길이의 메시지를 받아들여 특정 길이의 출력 값 생성
- ▶ 출력 값 비교를 통해 무결성 확인



해시함수의 조건

- ▶ H는 임의의 크기의 입력 M을 적용할 수 있어야 한다.
- ▶ H는 일정 크기의 출력 $h = H(M)$ 을 만들어야 한다.
- ▶ H와 M이 주어졌을 때 $h = H(M)$ 계산이 쉬워야 한다
- ▶ H와 h가 주어졌을 때 M을 구하는 계산이 거의 불가능해야 한다.(one way property)
- ▶ H가 주어졌을 때 같은 출력을 갖는 두 입력을 찾지 어려워야 한다. (충돌 회피성)(weak collision resistance)

전자서명의 조건

- ▶ 위조 불가
 - ▶ 서명자만이 서명 생성 가능
- ▶ 서명자 인증
 - ▶ 서명자의 신분 확인 가능
- ▶ 재사용 불가
 - ▶ 다른 문서의 서명으로 사용 불가능
- ▶ 변경 불가
 - ▶ 서명된 문서 내용 변경 불가
- ▶ 부인 불가
 - ▶ 서명한 사실 부인 불가

디지털 서명 알고리즘

- ▶ 공개키 암호방식을 이용한 서명 방식
- ▶ 서명자가 비밀키로 서명을 생성하고, 검증자가 공개키로 확인하는 시스템
- ▶ 직접 서명 방식
 - ▶ 송신자와 수신자 간에 직접 서명 및 검증
- ▶ 중계 서명 방식
 - ▶ 중재자를 통해 확인
 - ▶ 통신 전에 정보 공유가 필요 없고, 외부로부터 공격에 강하며, 시간 확인까지 가능

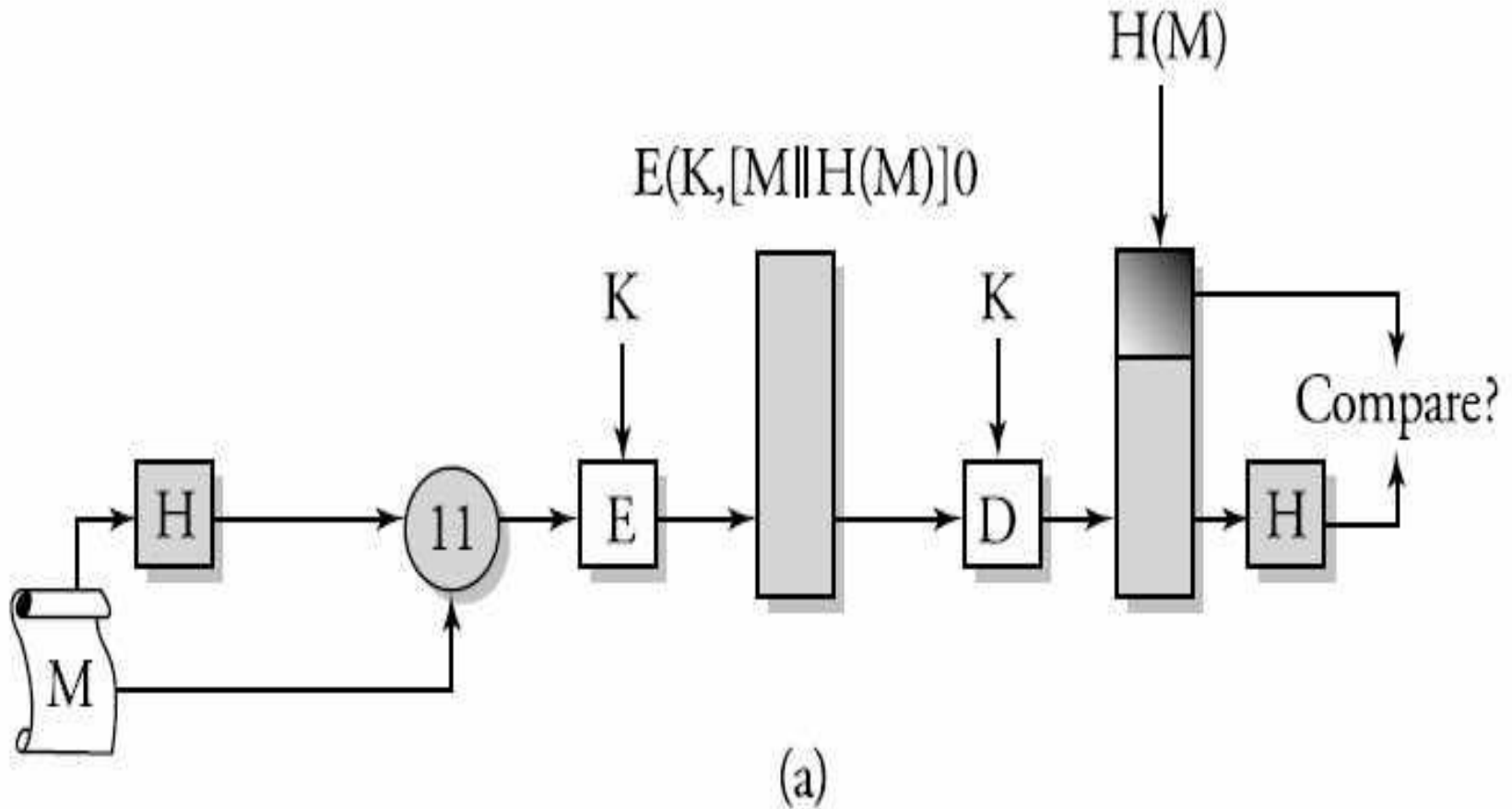
RSA 서명 방식

- ▶ 가장 먼저 실용화
- ▶ 서명 알고리즘의 안전도는 RSA 암호 방식 안전도와 동일
- ▶ 가장 보편적으로 사용
- ▶ 알고리즘
 - ▶ 키 생성 단계
 - ▶ RSA 암호 알고리즘과 동일
 - ▶ 서명 단계
 - ▶ 메시지 M의 해쉬값 H를 구하여 서명값 계산
 - ▶ $S = H^{d_A} \bmod n_A$
 - ▶ 원본 메시지 M과 같이 전송
 - ▶ 검증 단계
 - ▶ 원본 메시지에 대한 H' 계산
 - ▶ 공개키로 S의 H 계산 후 비교

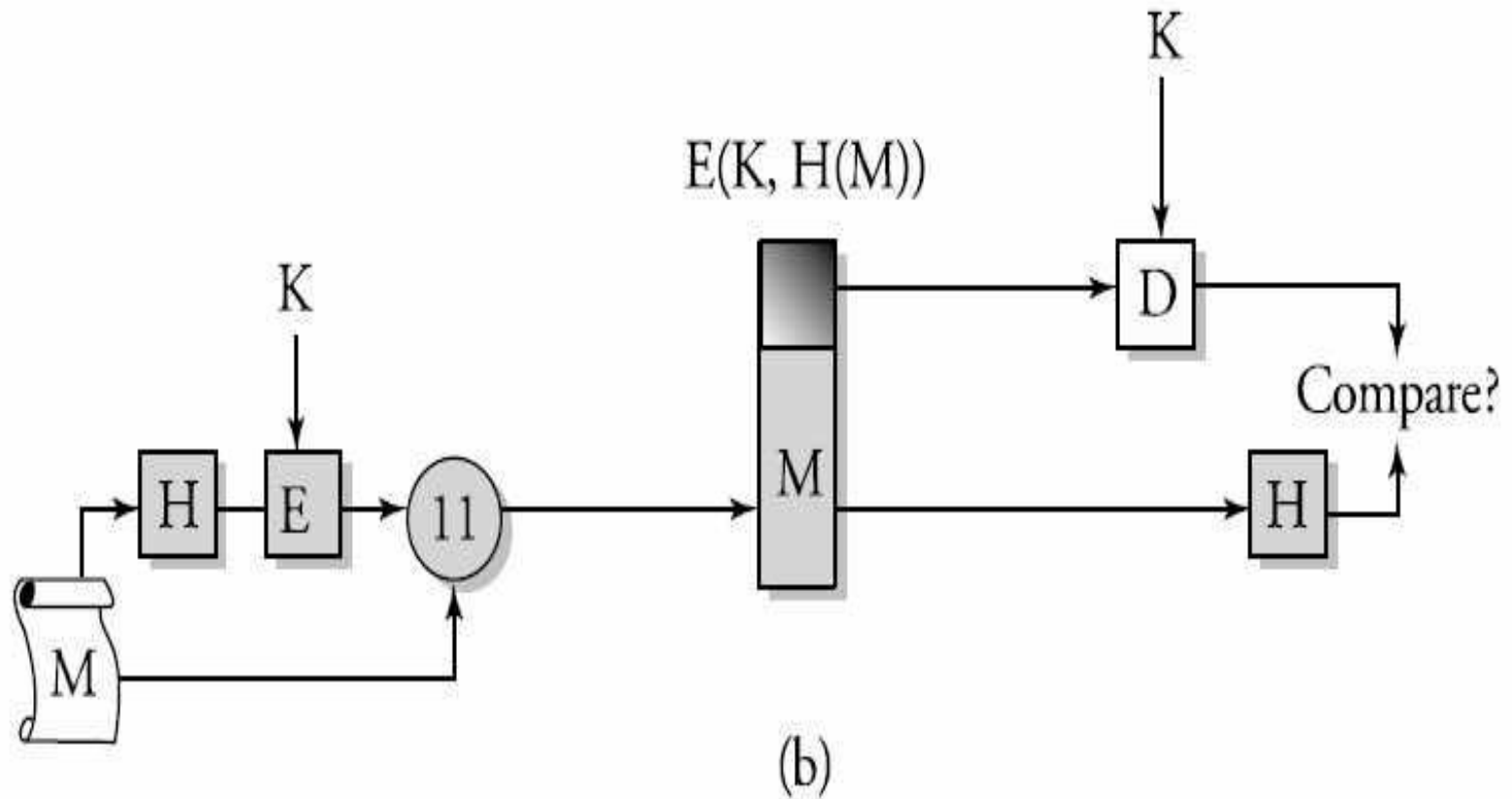
그외 서명 방식

- ▶ ElGamal 서명 알고리즘
 - ▶ 키 생성
 - ▶ 서명
 - ▶ 서명 검증
- ▶ DSS(Digital Signature Standard) 서명 알고리즘
 - ▶ ElGamal 서명 기술 응용
 - ▶ NIST에 의해 1994년 12월 표준 채택
 - ▶ ElGamal(512비트)보다 짧은 서명값(160비트)
 - ▶ 서명
 - ▶ 서명 검증 단계
- ▶ 타원곡선 디지털 서명 알고리즘
 - ▶ EC-DSA
 - ▶ 1985년 N.Koblitz와 V.S Miller가 RSA 방식의 대안으로 제시
 - ▶ 동일한 안전성을 실현하는데 RSA 1024 비트, ECC 160 비트
 - ▶ EC-KCDSA

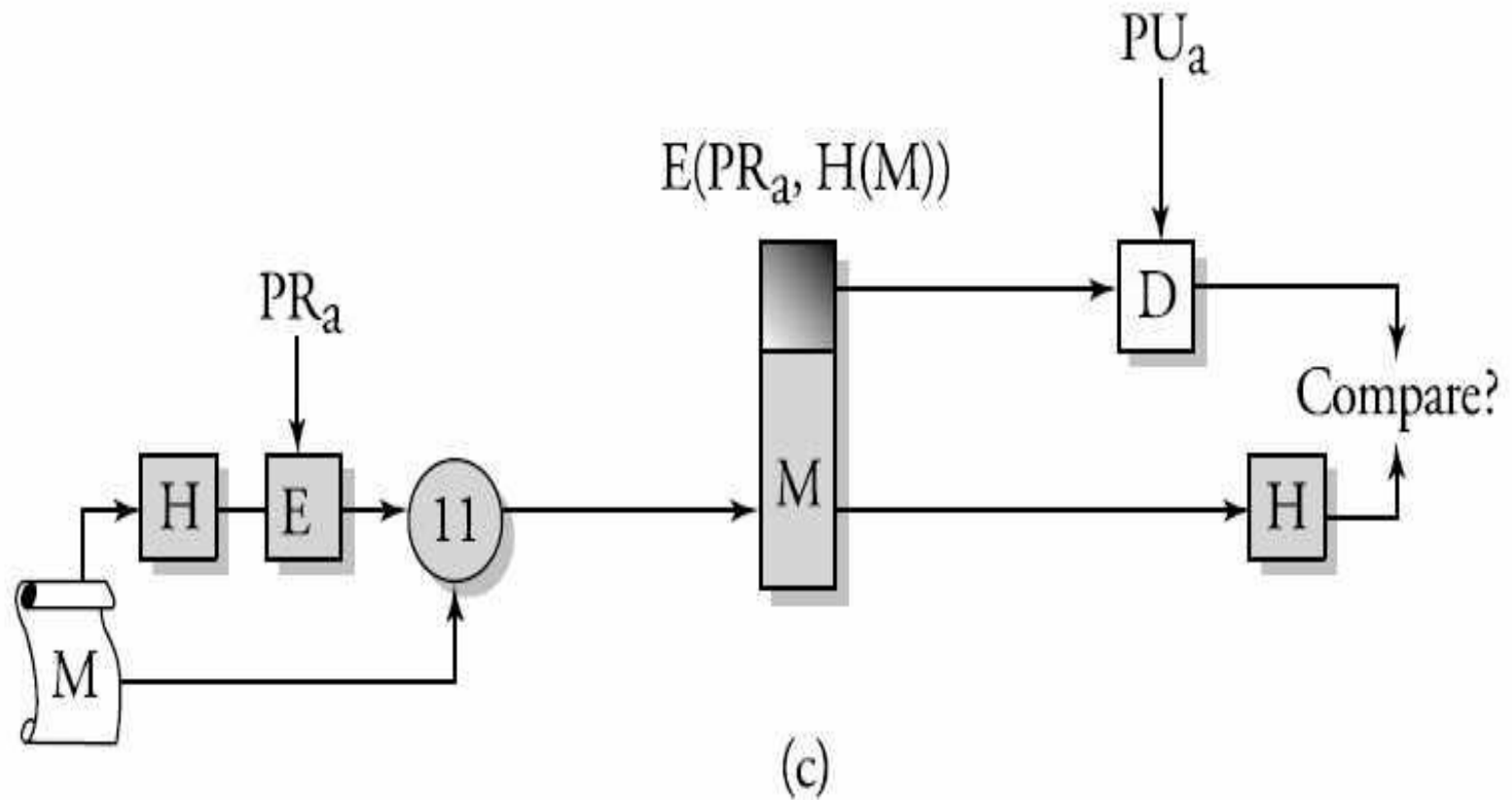
해시함수의 응용 (1)



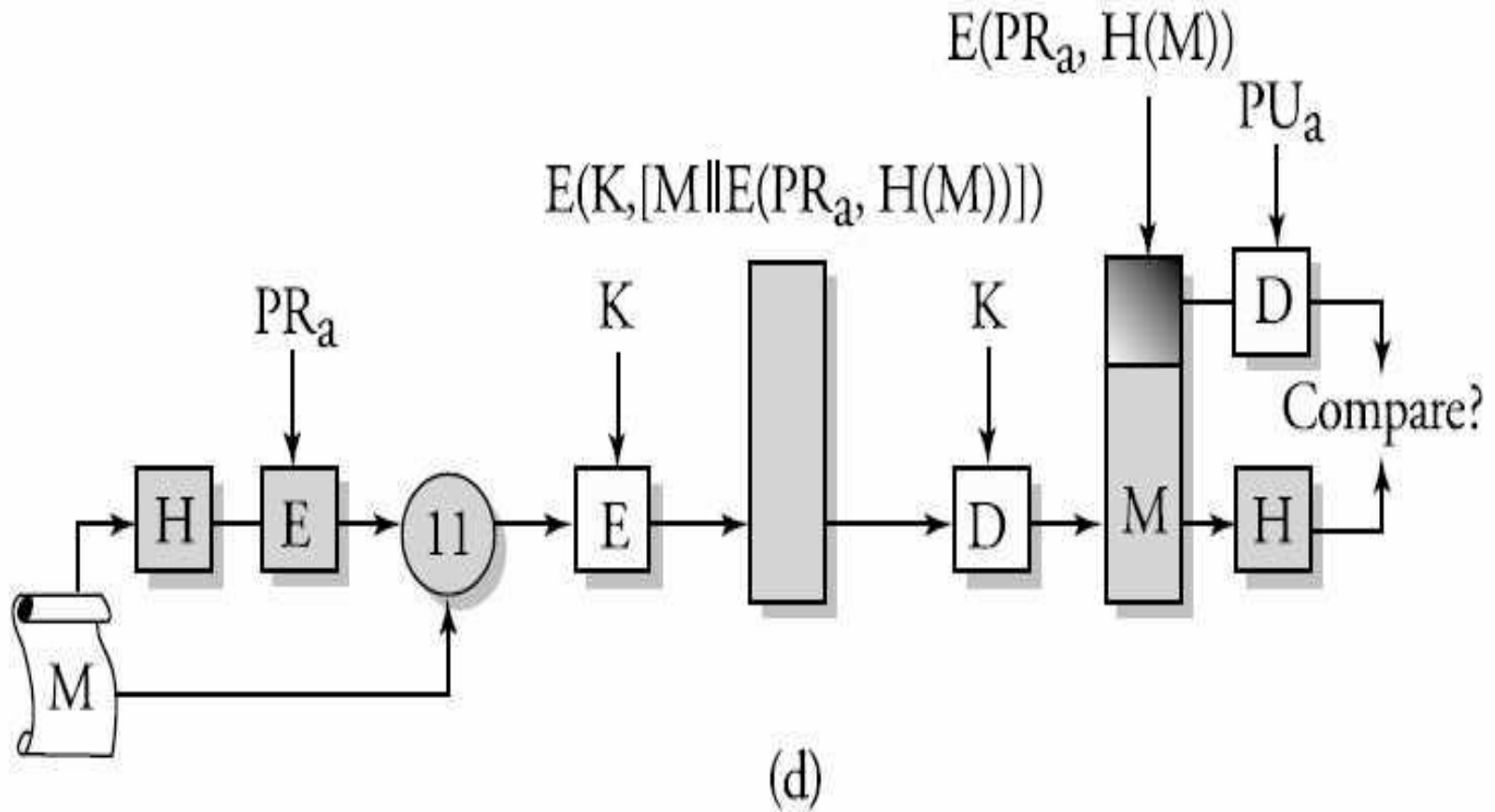
해시함수의 응용 (2)



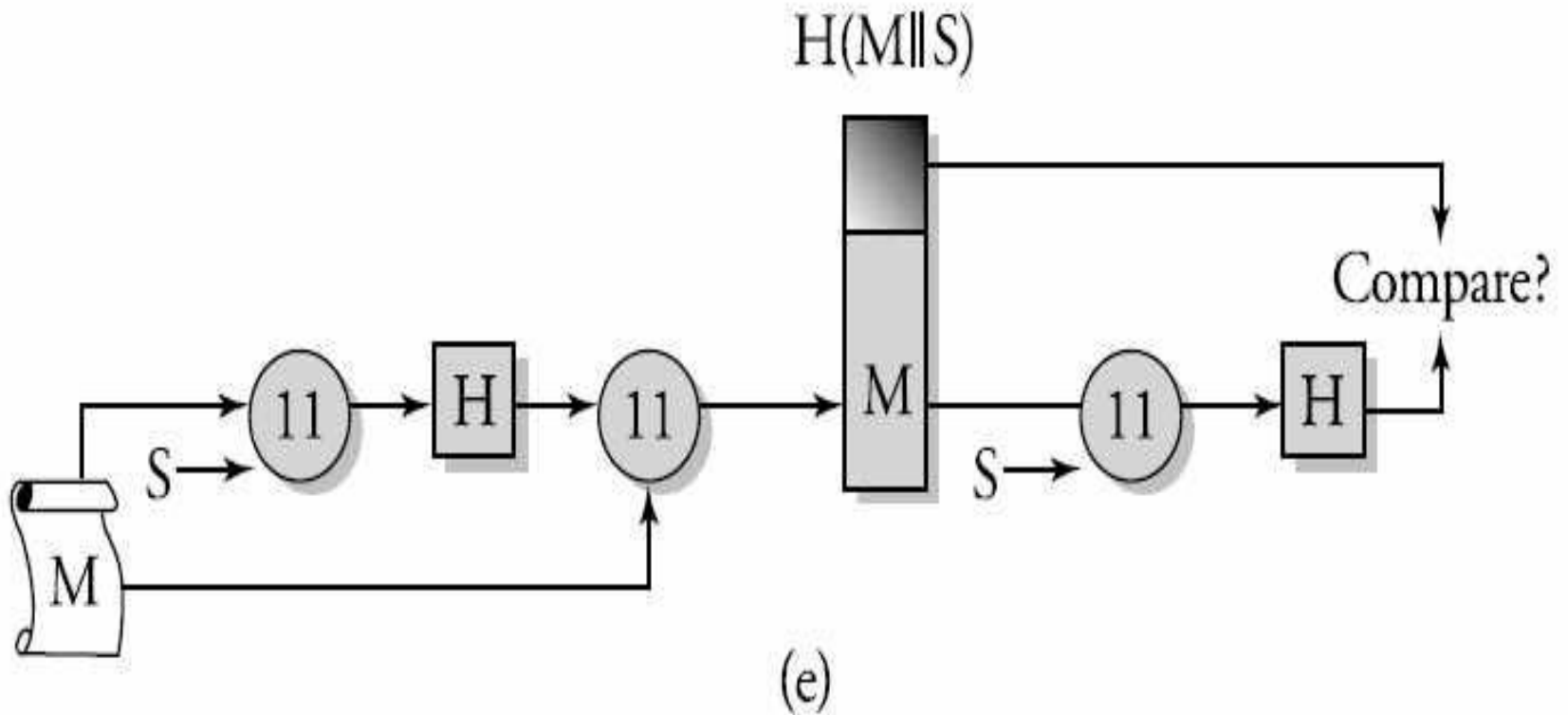
해시함수의 응용 (3)



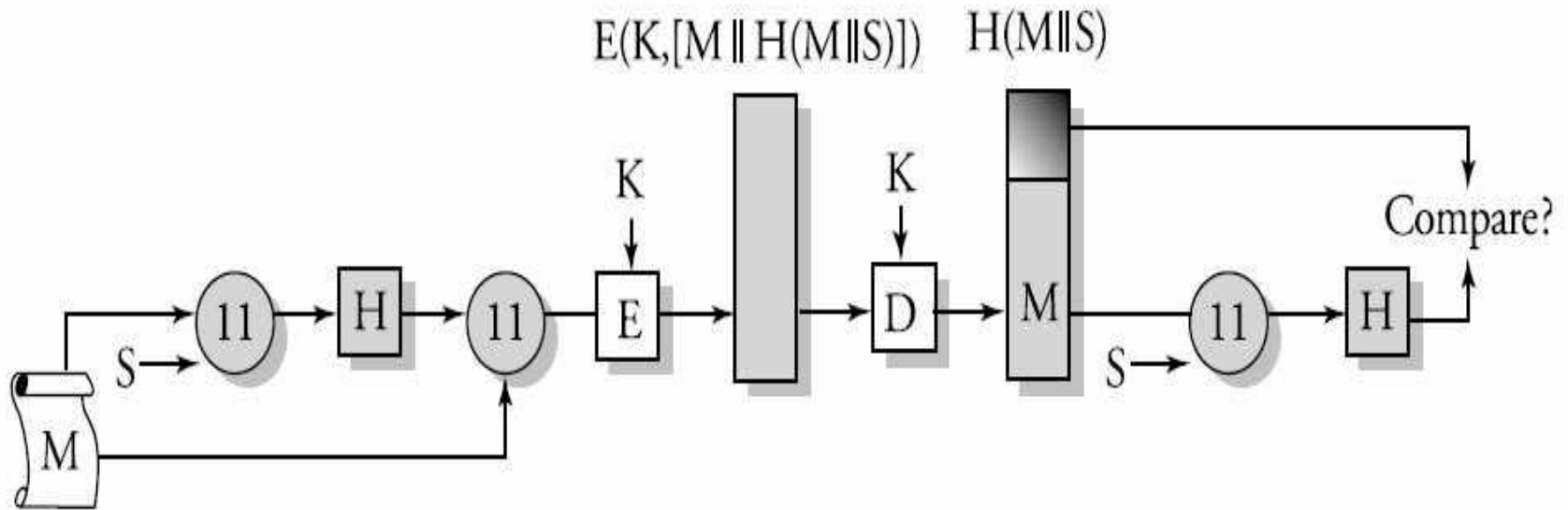
해시함수의 응용 (4)



해시함수의 응용 (5)



해시함수의 응용 (6)



(f)

해시함수 (1)

- ▶ MD5 (Message Digest Version 5)
 - ▶ 512비트 입력 128비트 출력
 - ▶ 충돌회피성에 대한 문제로 인해 기존 응용과 호환으로만 사용 제한
- ▶ MD4 (Message Digest Version 4)
 - ▶ 1990년 Rivest가 개발
 - ▶ 메시지를 128비트로 압축
 - ▶ MD5보다 약간 빠르고, 안전성 측면에서는 다소 떨어짐
- ▶ MD2 (Message Digest Version 2)
 - ▶ PEM 프로토콜에 응용
 - ▶ 보안성은 바이트의 random permutation에 달려있음
 - ▶ 다른 함수보다 다소 느리지만 취약 부분은 아직 발견되지 않음

해시함수 (2)

▶ RIPE-MD

- ▶ 유럽의 RIPE 프로젝트에서 개발된 해시함수
- ▶ SHA-1 못지않은 안전성을 가진 것으로 평가되며, 128, 160, 256, 320 비트 해시 출력값을 제공

▶ HAVAL

- ▶ 1992년 Zheng에 의해 개발
- ▶ 출력값 길이가 가변
- ▶ MD5를 변형하여 1024비트 블록으로 수행
- ▶ 다양한 라운드 수와 7개의 가변 함수, 128, 160, 192, 224, 256 비트 길이 해시값 출력 가능

해시함수 (3)

- ▶ SHA (Secure Hash Algorithm)
 - ▶ NIST에 의해 1993년 FIPS PUB 180으로 표준화
 - ▶ MD4와 유사하게 설계
 - ▶ 512비트 단위로 메시지를 입력하여 160비트 해시값 출력 (입력 전 메시지 길이를 512 비트 정수배로 조정)
- ▶ SHA-1(전자서명용)과 MD5 비교
 - ▶ 각각의 키 길이의 차이가 있음 (160 : 128)
 - ▶ 공격 대응면에서 SHA-1이 더 강하다 (길이 차이)
 - ▶ 안전성 : SHA-1이 비교적 더 안전
 - ▶ 속도 : 연산이 많아 SHA-1이 다소 늦다
 - ▶ 단순성과 간결성 : 두 알고리즘 모두 비교적 간단하며 적용 용이
 - ▶ 바이트 순서 : (big-endian : little-endian)

해쉬함수 (4)

Hash Function	Output Length	Round	Block Size	Speed
MD4	128	48	512	1.00
MD5	128	64	512	0.68
RIPE-MD-128	128	128	512	0.39
SHA-1	160	80	512	0.28
SHA-256	256	64	512	
SHA-384	384	80	1024	
SHA-512	512	80	1024	
RIPE-MD-160	160	160	512	0.24
RIPE-MD-256	256	128	512	
RIPE-MD-320	320	160	512	
Tiger	192	56	512	

해시함수 (5)

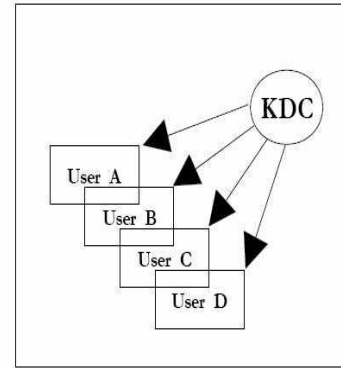
- ▶ 일반적으로 MD5가 많이 사용되고 있음
 - ▶ 취약성이 발견되어 제한적 사용 권고
- ▶ SHA-1은 디지털 서명에 사용하도록 제안됨
- ▶ AES의 128, 192, 256비트에 적용하도록 SHA256, SHA382, SHA512로 확장
- ▶ RIPE-MD-128, RIPE-MD-160, RIPE-MD-256, RIPE-MD-320은 MD5를 대신할 수 있도록 제안
 - ▶ RIPE-MD-128은 충돌저항성 문제가 있음
 - ▶ RIPE-MD-160은 효율성은 낮지만 높은 안전성으로 널리 사용 중
- ▶ Tiger는 64비트 환경에 최적화됨

키 관리(Key Management)

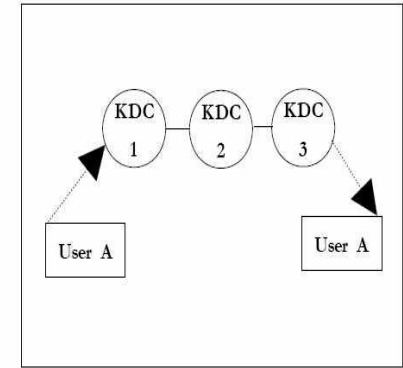
- ▶ 생성
- ▶ **분배**(distribution)
- ▶ 보관
- ▶ 폐기

KDC (Key Distribution Center)

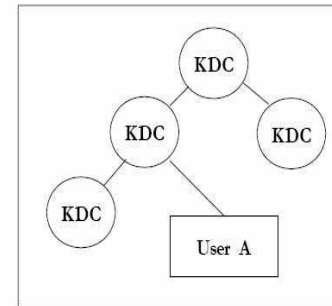
- ▶ 키의 생성과 분배 및 관리를 담당하는 장비 또는 시설
 - ▶ 사용자의 수가 많아질수록 키 관리가 어려워짐
- ▶ 중앙집중식
- ▶ 수평적 다중식 (flat multiple KDC)
- ▶ 계층적 다중식 (hierarchical multiple KDC)



중앙 집중 방식
(Key Distribution Center)



수평적 다중 분배 방식
(flat multiple KDC)



계층적 다중 분배 방식
(hierarchical multiple KDC)

SKA(Symmetric Key Agreement)

- ▶ KDC를 사용하지 않고 두 통신자의 세션키로 운영
- ▶ D-H(Diffe-Hellman) KA(Key Agreement)
 - ▶ 두 사용자가 안전하게 키를 교환하는 방식
 - ▶ 이산대수 알고리즘에 근거
 - ▶ 사용자 A는 임의의 수 x 를 고르고 $R_1 = g^x \text{ mod } p$ 계산
 - ▶ 사용자 B는 임의의 수 y 를 고르고 $R_2 = g^y \text{ mod } p$ 계산
 - ▶ R_1, R_2 교환
 - ▶ 사용자 A는 $(R_2)^x \text{ mod } p$ 계산
 - ▶ 사용자 B는 $(R_1)^y \text{ mod } p$ 계산
 - ▶ $K = (R_2)^x \text{ mod } p = (g^y \text{ mod } p)^x \text{ mod } p = g^{yx} \text{ mod } p = g^{xy} \text{ mod } p = (g^x \text{ mod } p)^y \text{ mod } p = (R_1)^y \text{ mod } p$