

8장 해쉬 함수 및 메시지 인증

정보보호이론

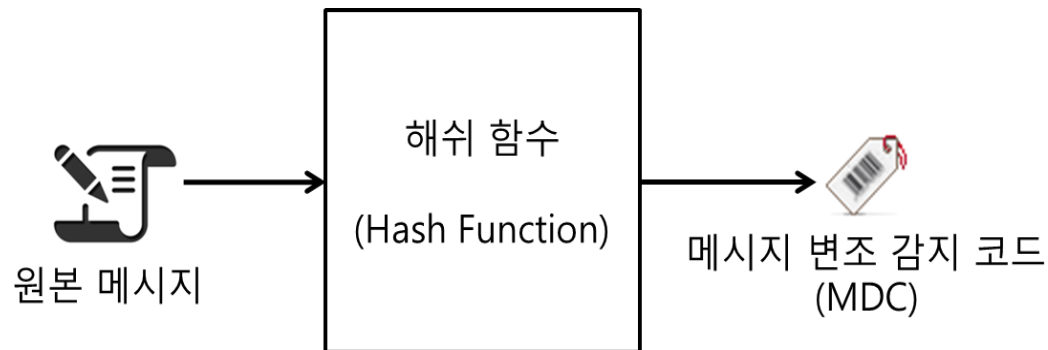
Spring 2015



고려대학교
KOREA UNIVERSITY

8.1 메시지 무결성

- 메시지 무결성(Message Integrity) : 원본 메시지의 변조를 방지하기 위한 서비스
 - ✕ 대칭키 & 공개키 암호시스템은 기밀성을 제공
 - ✕ 무결성을 위해 Modification Detection Code(MDC) 사용
 - ▶ 변조 감지 코드를 생성하기 위한 방법으로 해쉬 함수 (Hash Function, 8.3절에서 설명) 사용



8.2 해쉬 함수(Hash Function)

■ 해쉬 함수 : $h = H(M)$

1. 임의의 크기의 데이터를 입력 값
2. 고정된 크기의 출력 값

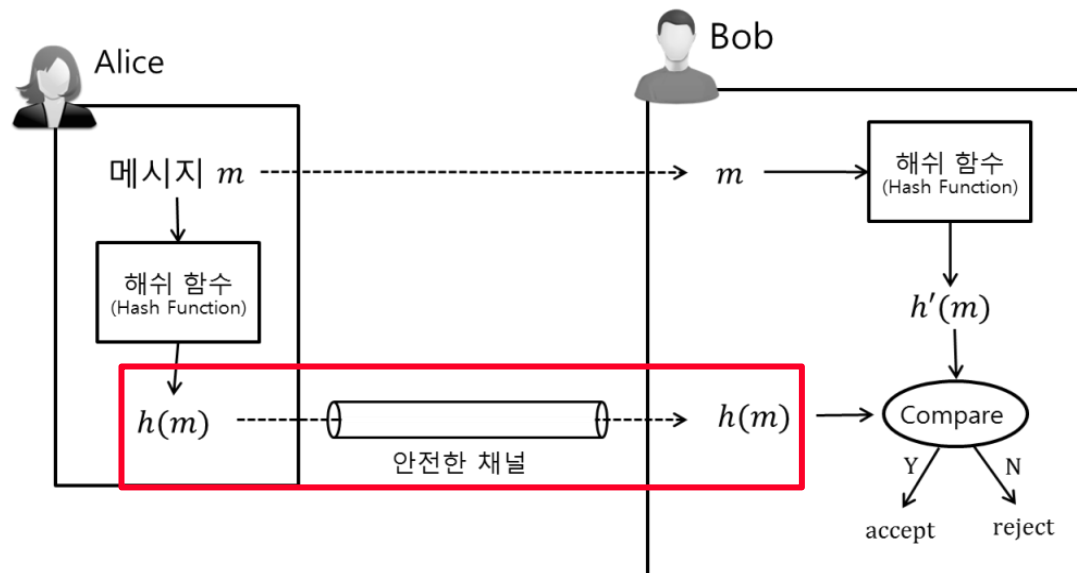
- ✕ 하드웨어 및 소프트웨어의 적용이 쉬워야 하며, 어떤 입력 데이터에 대해서도 출력 값을 계산하기 용이
- ✕ 해쉬 함수는 공개된 함수이며 키가 사용되지 않음
 - ▶ 키를 사용한 해쉬 함수: $h = H(k, M) \rightarrow \text{MAC (Message Authentication Code, 8.4절에서 설명)}$
- ✕ 전자 서명을 생성하기 위하여 사용됨
 - ▶ 메시지 m 에 대한 서명은 $h(m)$ 을 생성 후 서명 (9장에서 설명)

8.2 메시지 변조 감지 코드(MDC)

■ MDC

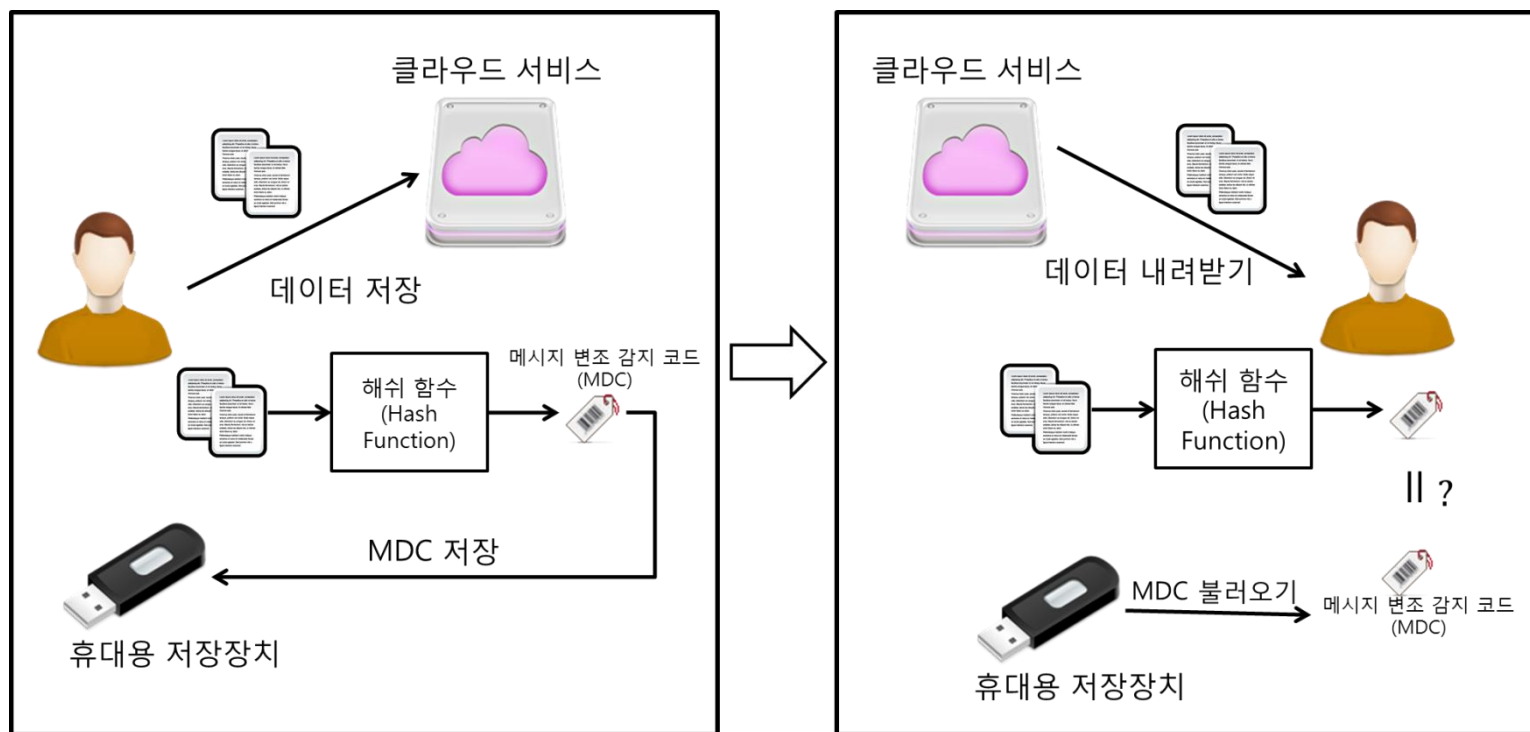
✕ 메시지 m 과 MDC $h(m)$ 은 분리될 수 있으며, $h(m)$ 은 변조되지 않도록 하는 것이 중요!

1. Alice는 메시지 m 에 대한 MDC $h(m)$ 생성
2. Alice \rightarrow Bob : $m, h(m)$, 단 $h(m)$ 은 안전한 채널로 전송



8.2 메시지 변조 감지 코드(MDC)

- MDC를 이용한 메시지 변조 감지 코드 활용 예
 - ✕ 클라우드 서비스에서 원본 메시지에 대한 무결성을 제공



8.3 암호학적 해쉬 함수(Cryptographic Hash Function)

■ 암호학적으로 안전한 해쉬 함수의 성질

1. 역상 저항성(Preimage Resistance): Given h , infeasible to find x s.t. $H(x)=h$
2. 제 2 역상 저항성(Second Preimage Resistance): Given x , infeasible to find y s.t. $H(y)=H(x)$
3. 충돌 저항성(Collision Resistance): infeasible to find any (x,y) s.t. $H(y)=H(x)$

Note : ZIP 등의 압축함수와 Checksum은 역상 저항성을 만족하지 못함

x	$h(x)$	x	$h(x)$
34A2BFE41	A535	BBDEE123	AEC7
53EF95	ABC4	5200DE	8DF9
824A74CD	7601	A6EEF1123	1B2A
2572AB4	AB12	A84F	0EF2
294AE9500D	5234	C1	52F3

x	$h(x)$	x	$h(x)$
180A	190D	AEF7D8C	8DF9
100F	58FE	32178FE88	802C
53EF95	8DF9	8000127D	EF19
C1	52F3	2572AB4	6A9D
824A74CD	1B2A	A6EEF1123	1B2A
A84F	1B2A	34A2BFE41	A535
BBDEE123	AEC7	5200DE	8DF9
294AE9500D	802C	A	ABC4

x	$h(x)$	x	$h(x)$
17AB8AF023	190D	9EFAD58	154A
102F	58FE	F2178FE88	A75F
53EF95	802C	50D00127D	535D
C1	52F3	AB	800E
824AA4CD	1B2A	A6EEF1123	AA5D
12A3	1B2A	34A2BFE41	C3F0
67BAD13	AEC7	320410DE	1189
A54F2	D812	112A	89DF

8.3 암호학적 해쉬 함수(Cryptographic Hash Function)

■ 랜덤 오라클(Random Oracle)

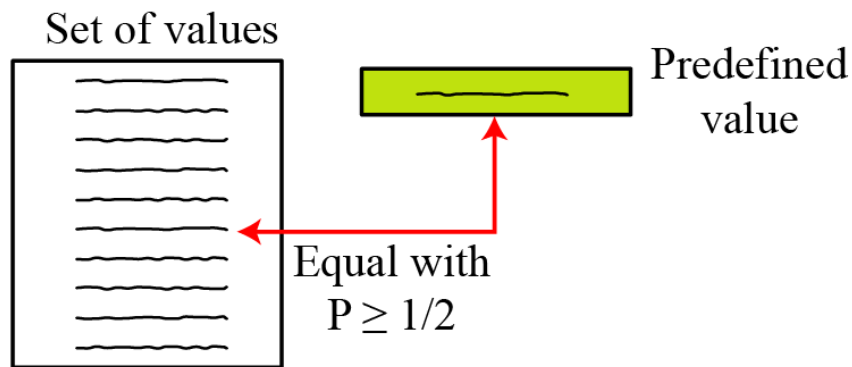
- ✕ 오라클 : 이론적인 추상 머신(theoretical abstract machine)으로 어떠한 complexity class(심지어 undecidable 문제)도 사용 가능함
- ✕ 랜덤 오라클 : 이상적인 해쉬 함수로 Bellare와 Rogaway에 의해 제안
 - ▶ 동작 방식
 1. 질의에 대해 출력 domain에서 uniform하게 선택된 난수 출력
 2. 과거 질의에 대해서는 동일한 난수를 출력
- ✕ 즉 각 질의를 출력 domain의 난수와 mapping

8.3 암호학적 해시 함수(Cryptographic Hash Function)

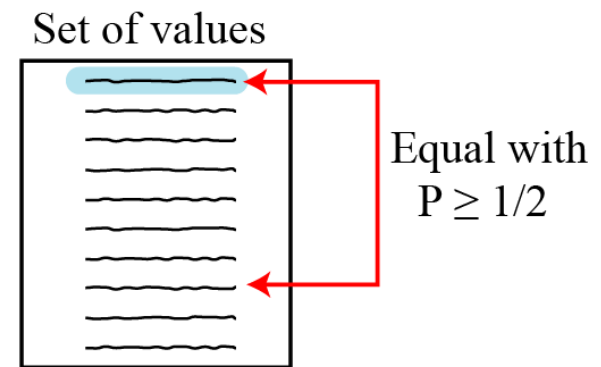
- 랜덤 오라클을 이용한 암호학적 증명
 - ✗ 증명을 위하여 필요한 수학적 성질을 제공하는 구현 가능한 함수가 존재하지 않을 때 사용
 - ▶ 랜덤 오라클을 사용하여 안전하다고 증명된 시스템은 secure in the random oracle model라 하며 반대 개념은 secure in the standard model이라 한다.
 - ✗ 해시 함수의 출력이 true random이라는 가정이 필요할 때 암호학적 해시 함수를 모델링 하기 위하여 사용
 - ▶ 프로토콜 A의 안전성을 보이기 위해, 해시 함수를 랜덤 오라클이라 가정하고, A를 풀 수 있다면 NP에 속한 B를 풀게 됨 (reduction)으로 보임 → 즉 대우로 증명 : B는 어려운 문제이므로 A는 어려움
 - ✗ Further reading : Random oracles are practical: A paradigm for designing efficient protocols
 - ▶ <http://cseweb.ucsd.edu/~mihir/papers/ro.html>

8.3 암호학적 해시 함수(Cryptographic Hash Function)

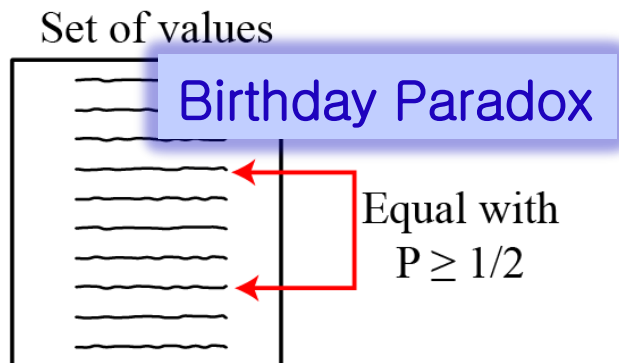
■ 생일 문제(Birthday Problems)



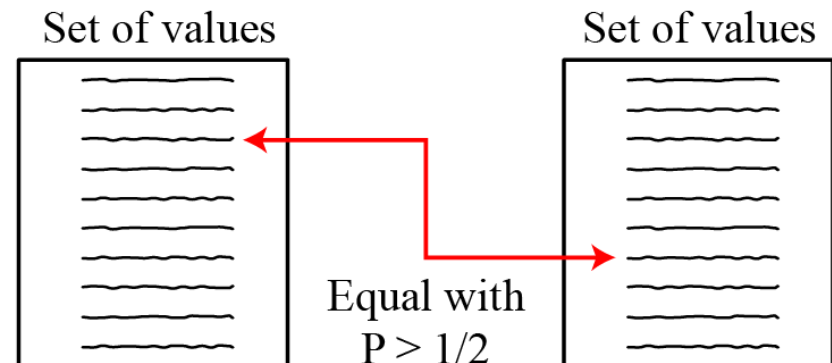
a. First problem (**Preimage Resistance**)



b. Second problem (**Second Preimage Resistance**)



c. Third problem (**Collision Resistance**)



d. Fourth problem

8.3 암호학적 해쉬 함수(Cryptographic Hash Function)

■ 생일 문제(Birthday Problems)

Table 11.3 Summarized solutions to four birthday problems

<i>Problem</i>	<i>Probability</i>	<i>General value for k</i>	<i>Value of k with $P = 1/2$</i>	<i>Number of students ($N = 365$)</i>
1	$P \approx 1 - e^{-k/N}$	$k \approx \ln[1/(1 - P)] \times N$	$k \approx 0.69 \times N$	253
2	$P \approx 1 - e^{-(k-1)/N}$	$k \approx \ln[1/(1 - P)] \times N + 1$	$k \approx 0.69 \times N + 1$	254
3	$P \approx 1 - e^{k(k-1)/2N}$	$k \approx \{2 \ln [1/(1 - P)]\}^{1/2} \times N^{1/2}$	$k \approx 1.18 \times N^{1/2}$	23
4	$P \approx 1 - e^{-k^2/2N}$	$k \approx \{\ln [1/(1 - P)]\}^{1/2} \times N^{1/2}$	$k \approx 0.83 \times N^{1/2}$	16

8.3 암호학적 해시 함수(Cryptographic Hash Function)

■ 생일 문제(Birthday Problems) : 1st and 2nd 문제

- ▶ n 비트인 $h (= H(x))$ 가 주어진 경우 50%이상의 확률로 x 을 찾는 문제

= “고정된 생일 h ”을 갖는 학생이 50% 이상의 확률로 존재하기 위하여 필요한 학생의 수

= 해수 함수를 평가해야 하는 횟수 $\rightarrow 0.69 \times 2^n$

- ▶ n 비트인 $h (= H(x))$ 가 주어진 경우 50%이상의 확률로 $h = H(y)$ 인 $y(\neq x)$ 를 찾는 문제

= “고정된 생일 h ”를 갖는 학생이 있는 경우 동일한 생일을 갖는 학생이 50% 이상의 확률로 존재하기 위하여 필요한 학생의 수

= 해수 함수를 평가해야 하는 횟수 $= 0.69 \times 2^n + 1$

8.3 암호학적 해시 함수(Cryptographic Hash Function)

■ 생일 역설(Birthday Paradox) : 3rd 문제

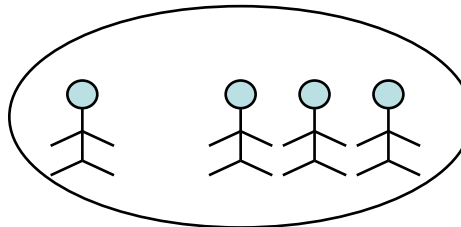
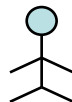
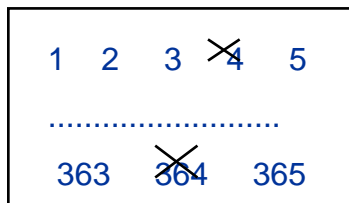
- ▶ 50%이상의 확률로 $h = H(x) = H(y)$ 인 쌍 (x, y) ($y \neq x$)를 찾는 문제
= 같은 생일을 갖는 두 학생이 50% 이상의 확률로 존재하기 위하여 필요한 학생의 수

$$\rightarrow p = 1 - 1(1 - 1/n)(1 - 2/n) \dots (1 - \{r-1\}/n) = 1.18 \times 2^{n/2}$$

$$\rightarrow 50\% = 1.18 \times 365^{1/2} \approx 23$$

$$\rightarrow 120 \text{ 비트인 해시 값 } h \text{의 안전성은 } 2^{60}$$

→ Running on a 1 GHz computer (with 10^9 cycles per second), 2^{60} CPU cycles requires $2^{60}/10^9$, or about 35 years. → BUT the fastest existing supercomputer (as of 2008) can execute 4.78×10^{14} per second, and 2^{60} would require only **40 minutes**. Taking 2^{80} is more prudent choice (80 years on the supercomputer mentioned)



8.3 암호학적 해쉬 함수(Cryptographic Hash Function)

■ 생일 문제(Birthday Problems) : 4th 문제

✕ 서명 공격 : $S = \text{Sig}(H(m))$

- ▶ $m \neq m'$ 이면서 $H(m) = H(m')$ 인 쌍 (m, m') 을 발견하여 공격
- ▶ 서명자에게 $S = \text{Sig}(H(m))$ 를 받고 m' 에 대한 서명이라 주장

1. 공격자는 정당한 문장 m 과 동일한 의미를 갖는 $2^{n/2}$ 개의 문장을 생성 $\rightarrow M$
2. 공격자는 자신이 원하는 문장 m' 과 동일한 의미를 갖는 $2^{n/2}$ 개의 문장을 생성 $\rightarrow M'$
3. M 과 M' 에서 동일한 해쉬 값을 갖는 (m, m') 을 50%이상의 확률로 발견

Dear Anthony,

{ This letter is, I am willing } to introduce { you to, to you } { Mr., _ } Lee, the
{ new, newly appointed } { chief, senior } jewellery buyer for { you, the } Northern
{ European, Europe } { area, division }.

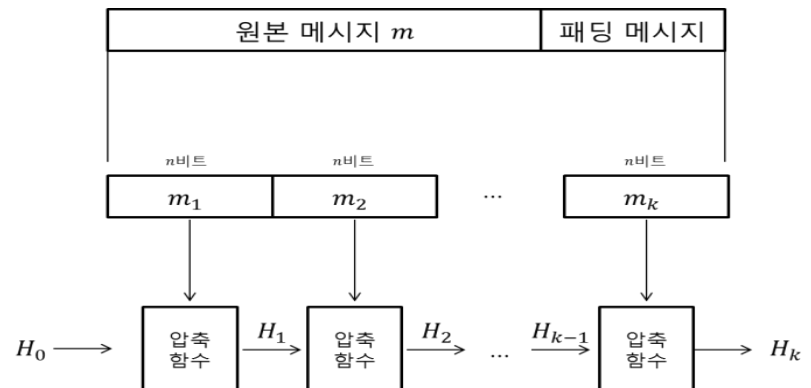
8.4 암호학적 해시 함수 설계 원리

■ 반복 구조의 해시 함수(Iterated Hash Function)

✧ Merkle-Damgard 구조: MD5, SHA-1, SHA-2 등

▶ 취약점 발견 → SHA-3은 다른 구조를 채택

1. n 비트의 배수가 되도록 ($m \parallel$ 패딩)
2. ($m \parallel$ 패딩)을 한 블록이 n 비트가 되도록 분할 m_1, m_2, \dots, m_k
3. $f(H_0, m_1) = H_1$
- 초기 값 H_0 , 압축 함수 $f(\cdot)$
4. k 번 수행하여 H_k 를 생성 $\rightarrow h(m) = H_k$

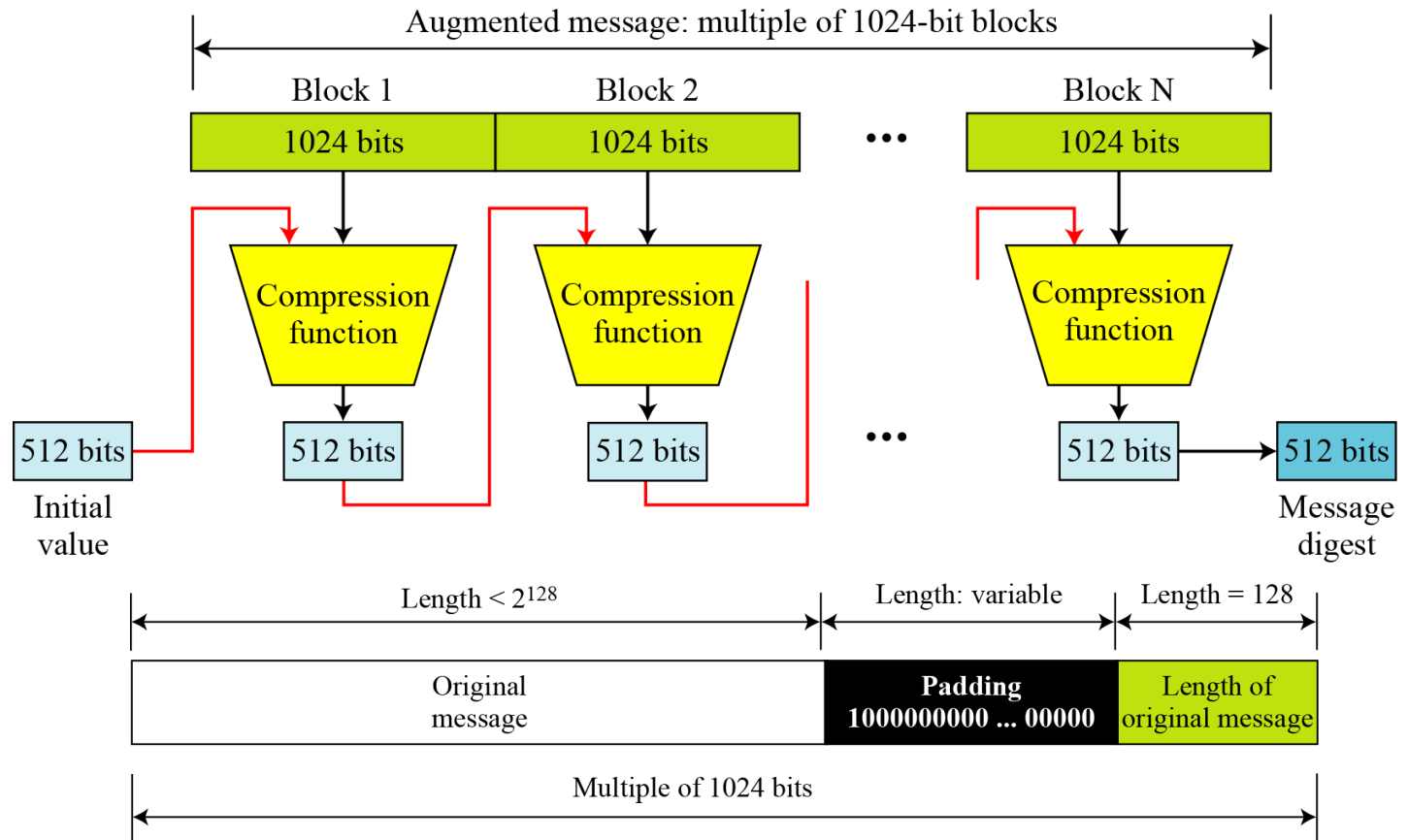


H_0 : 초기값
 H_t : 메시지 다이제스트(출력값)

8.4 암호학적 해시 함수 설계 원리

■ 반복 구조의 해시 함수(Iterated Hash Function)

✧ Merkle-Damgard 구조 예 SHA-512



8.5 다양한 해쉬 함수

- Sotirov et al., MD5 considered harmful today (2008)
 - ✗ MD5를 사용하는 공개키 인증서 위조에 성공 → PKI의 취약성으로 직결
 - ✗ Web site phishing 공격 가능

	HAS-160	SHA-1	SHA-2			
			SHA-224	SHA-256	SHA-384	SHA-512
암호문(해쉬값) 길이(비트)	160	160	224	256	384	512
참조규격	TTA ¹⁾ TAS.KO.12.0011/R2	NIST ²⁾ FIPS 180-3				

알고리즘	해쉬길이	안전성 분석 내용
SHA-1	160	2^{63} 의 연산량으로 충돌쌍 생성 가능
SHA-2	224, 256, 384, 512	현재 생일공격보다 효율적인 공격방식이 없음
MD5	128	1) 일반 노트북 컴퓨터에서 약 1분정도에 충돌쌍 생성 가능 2) X.509인증서에 사용 시 2^{52} 의 연산량으로 위조 공격 가능 3) 해쉬함수기반 메시지 인증 코드인 HMAC에서 사용할 경우, 2^{96} 의 연산량으로 키복구 공격 가능
HAS-160	160	현재 생일공격보다 효율적인 공격방식이 없음

8.5 다양한 해쉬 함수

■ SHA(Secure Hash Algorithm)

- ✕ 1993년 미국 국가 안전 보장국(NSA)과 미국 표준 기술연구소(NIST)가 함께 설계
 - ▶ SHA-1 : SHA-0의 취약점을 보강, 160 비트 해쉬 값
 - ▶ 2002년 SHA-2계열 (SHA-256, SHA-384, SHA-512) 설계
 - ▶ In 2005, security flaws were identified in SHA-1. The SHA-2 variants are similar to SHA-1. → SHA-3 공모, 2012년 Keccak (pronounced "catch-ack") 알고리즘 최종 선정

8.5 다양한 해쉬 함수

■ SHA-3

✧ SHA-3 공모

- ▶ 2008년 : 총 51개의 해쉬 함수가 1차 후보로 발표
- ▶ 2009년 : 2차 라운드를 통해 14개 후보로 압축
- ▶ 2010년 12월 : 5개의 해쉬 함수가 최종 후보로 발표
- ▶ 2012년 10월 : Guido Bertoni, Joan Daemen, Gilles Van Assche (STMicroelectronics), , Michaël Peeters (NXP Semiconductors)가 제안한 Keccak (pronounced "catch-ack") 알고리즘 최종 선정
- ▶ 160, 224, 256, 384, 512비트 해쉬 값
- ▶ <http://csrc.nist.gov/groups/ST/hash/sha-3/index.html>

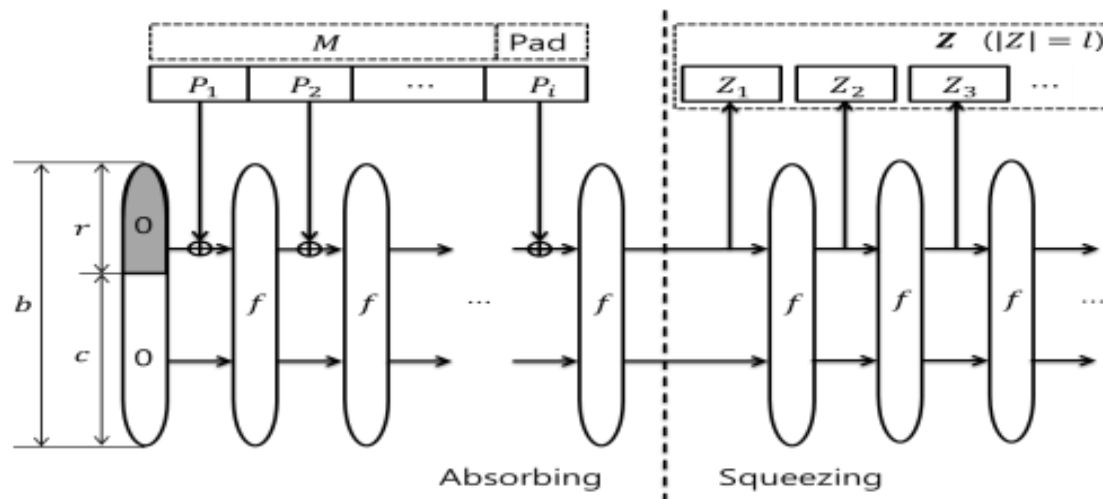
8.5 다양한 해쉬 함수

✧ 스펀지 구조 : SHA-3




▶ 스펀지 구조(Sponge Construction)

- 입력 메시지를 조금씩 흡수하여 뒤섞어 주는 과정을 반복하는 absorbing 단계와 absorbing 단계 이후 Squeezing 단계를 통하여 출력 메시지를 조금씩 뽑아내는 구조
- "Keccak has the added advantage of not being vulnerable in the same ways SHA-2 might be," says NIST computer security expert Tim Polk. "An attack that could work on SHA-2 most likely would not work on Keccak because the two algorithms are designed so differently."

Width b =
Bitrate r +
Capacity c



Note : 키 길이 권고

Method	Date	Symmetric	Asymmetric		Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash
[1] Lenstra / Verheul 	2076	129	6790	5888	230	6790	245	257
[2] Lenstra Updated 	2090	128	4440	6974	256	4440	256	256
[3] ECRYPT II	2031 - 2040	128	3248		256	3248	256	256
[4] NIST	> 2030	128	3072		256	3072	256	256
[5] FNISA	> 2020	128	4096		200	4096	256	256
[6] NSA	-	128	-		-	-	256	256
[7] RFC3766 	-	128	3253		256	3253	242	-
[8] BSI (signature only)	-	-	-		-	-	-	-

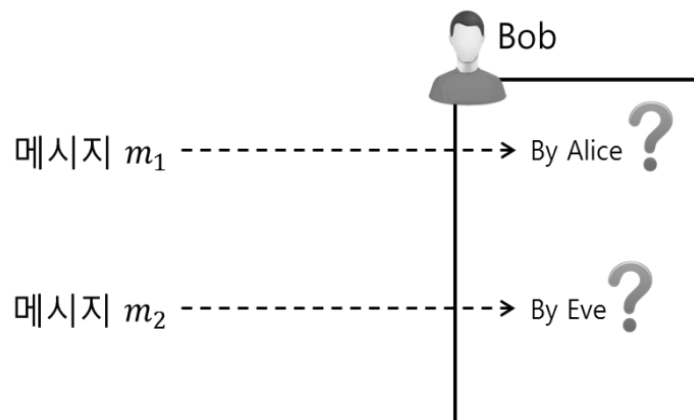
All key sizes are provided in bits. These are the minimal sizes for security.

Lenstra and Verheul Equations (2000)
 Lenstra Updated Equations (2004)
 ECRYPT II Recommendations (2011)
 NIST Recommendations (2011)
 FNISA Recommendations (2010)
 Fact Sheet NSA Suite B Cryptography (2010)
 Network Working Group RFC3766 (2004)
 BSI Recommendations (2011)

8.6 메시지 인증코드 (MAC)

■ Message Authentication Code(MAC) :

- ✕ 메시지 근원 인증 (Message origin authentication)을 제공
 - ▶ 해쉬 함수를 이용한 MDC는 무결성만을 제공 → 메시지의 출처를 확인할 수 없음
 - ▶ MAC은 무결성도 함께 제공

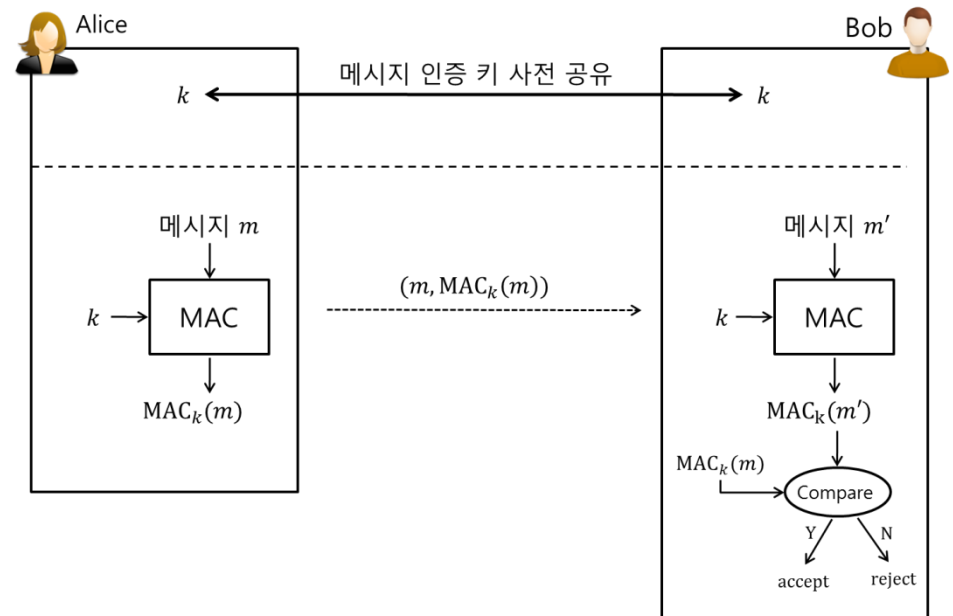


8.6 메시지 인증코드 (MAC)

■ Message Authentication Code(MAC) :

✕ $MAC_k(m) = h(k, m)$

1. Alice와 Bob은 MAC 키 k 를 사전공유
2. Alice는 메시지 m 에 대한 $MAC_k(m)$ 생성
3. Alice \rightarrow Bob : $m, MAC_k(m)$
4. Bob은 수신 메시지 m' 에 대한 $MAC_k(m')$ 생성
5. $MAC_k(m) = ? MAC_k(m')$



8.6 메시지 인증코드 (MAC)

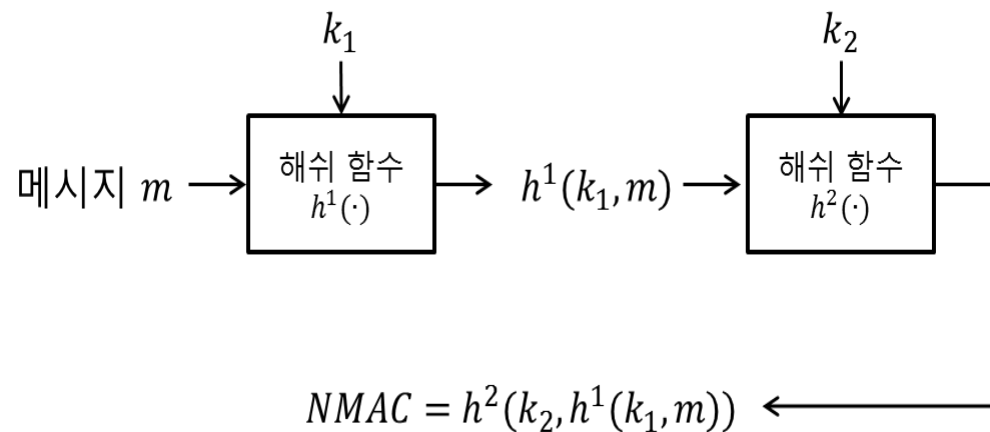
■ MAC 생성 방법 :

해쉬 함수 기반 메시지 인증 코드	Nested MAC
	HMAC
블록 암호 알고리즘 기반 메시지 인증 코드	CBC-MAC
	CMAC

8.6 메시지 인증코드 (MAC)

■ Nested MAC

✧ $NMAC_{k_1, k_2}(m) = h^2(k_2, h^1(k_1, m))$

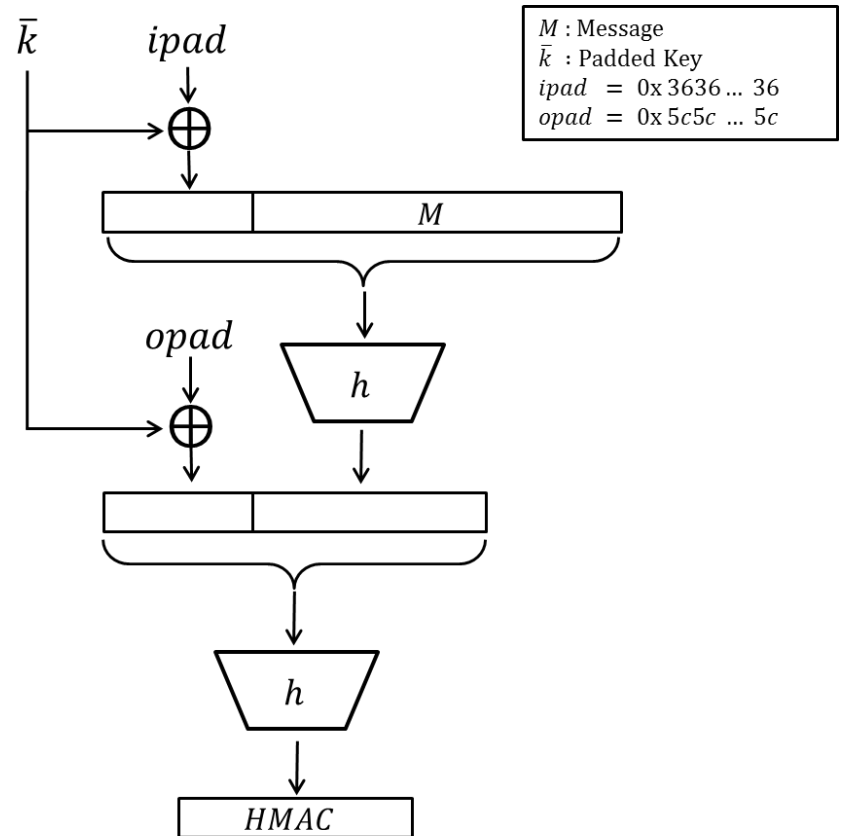


8.6 메시지 인증코드 (MAC)

■ HMAC(Hashed MAC)

✖ NIST 표준 FIPS198

1. 메시지 m 을 b 비트
사이즈로 분할
2. 키 k 앞에 0으로
패딩 하여 b 비트로 맞추어
 \bar{k} 생성
3. $HMAC_K = h[(\bar{k} \oplus opad) ||$
 $Hash[(\bar{k} \oplus ipad) || M)]]$



$$HMAC = h(\bar{k} \oplus opad || h(\bar{k} \oplus ipad || M))$$

8.6 메시지 인증코드 (MAC)

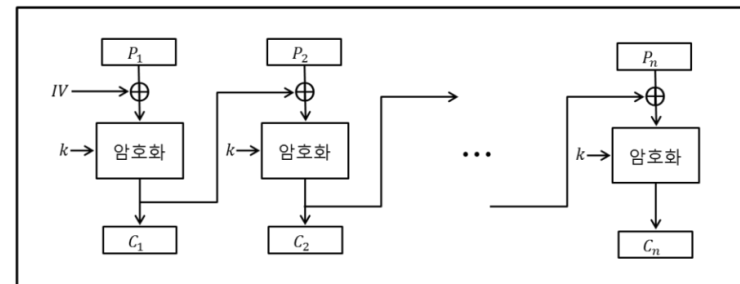
- HMAC(Hashed MAC)의 안전성
 - ✕ 사용된 해쉬 함수의 안전성에 기반
 - ✕ HMAC에 대한 공격
 - ▶ 사용된 MAC 키 전수 공격
 - ▶ 생일 공격
 - 키가 사용되었기 때문에 동일한 키로 생성된 $2^{n/2}$ 개의 HMAC값을 수집
 - 해쉬 함수 H는 공개되었기 때문에 MDC는 자유로이 생성할 수 있지만 MAC은 키가 사용되기 때문에 수동적으로 관찰해서 수집해야 함!!!

8.6 메시지 인증코드 (MAC)

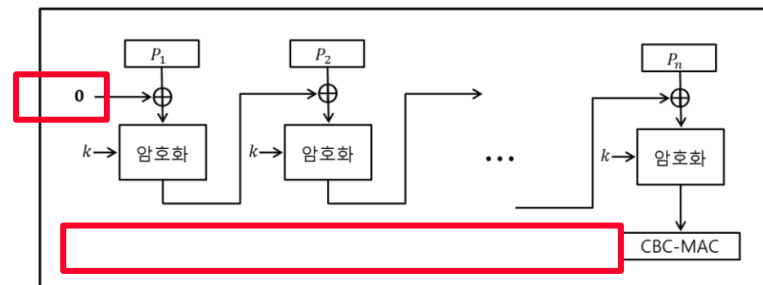
■ CBC-MAC(Cipher Block Chaining MAC)

- ✧ 블록 암호의 운영모드 중 CBC모드를 사용하여 메시지 인증 코드를 생성 → 실제 환경에서 편리
 - ▶ 고정된 초기벡터($IV = 0$)를 사용
 - ▶ 고정된 메시지에 대한 MAC

CBC모드 암호화



CBC-MAC

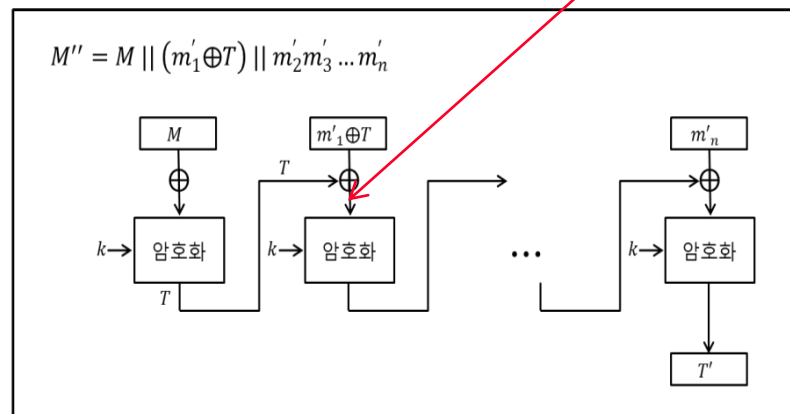
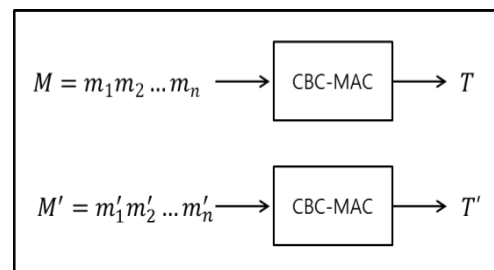


8.6 메시지 인증코드 (MAC)

■ CBC-MAC(Cipher Block Chaining MAC)

✕ 가변길이의 메시지에 대한 메시지 인증 코드를 생성하는 경우 → 제2역상을 계산

1. $M = m_1m_2 \dots m_n, M' = m'_1m'_2 \dots m'_n$ 인 $(M, T), (M', T')$ 를 수집
2. $M'' = M \parallel (m'_1 \oplus T) \parallel m'_2m'_3 \dots m'_n$ 을 계산
3. $CBC - MAC_k(M') = CBC - MAC_k(M'')$



8.6 메시지 인증코드 (MAC)

■ CMAC(Cipher-based MAC) : NIST SP800-38B

- ✕ 고정된 길이의 메시지만을 사용해야 하는 CBC-MAC에서의 제약 사항을 제거
 - ▶ 2개의 키 k_1, k_2 사용
 - ▶ 두 메시지 M' 과 M'' 의 메시지 인증 코드 값이 같아지도록 하기 위해서는 k_2 에 대한 값을 알고 있어야만 함

