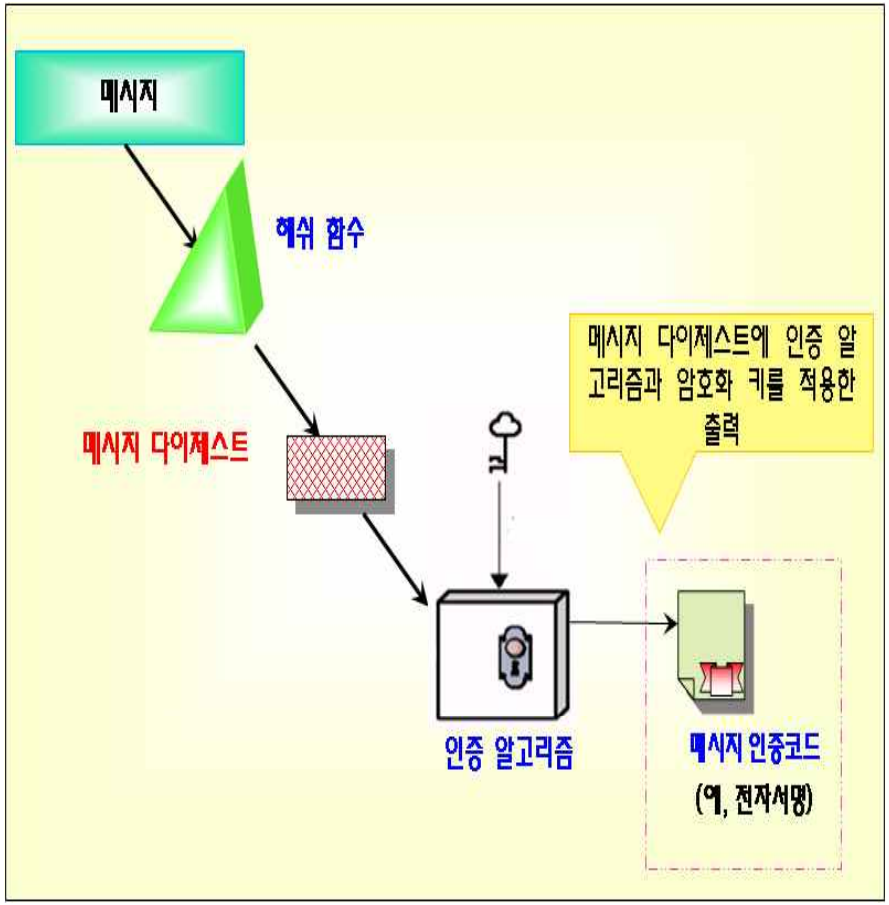
# 4. 해쉬 함수와 메시지 인증 코드

담당교수: 차 영욱
ywcha@andong.ac.kr

안동대학교
ANDONG NATIONAL UNIVERSITY

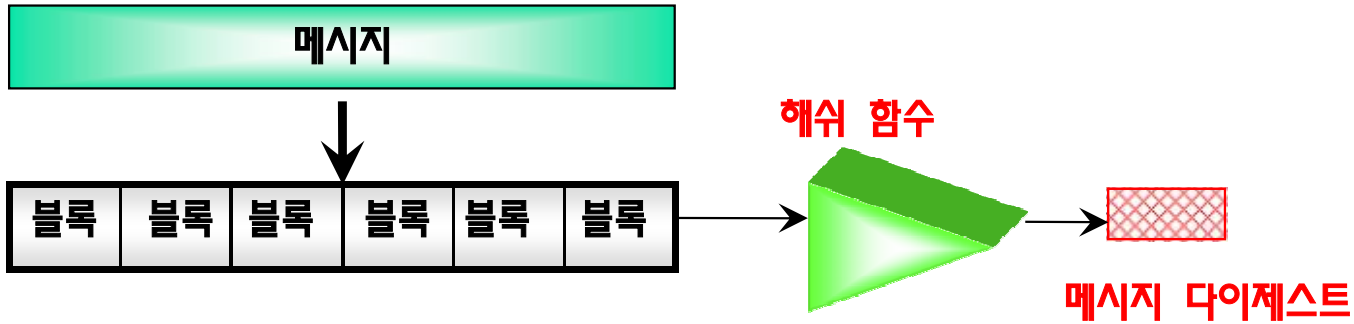
# 목 차

- 해쉬 함수
- 안전 해쉬 알고리즘(SHA)
- 해쉬 함수 비교
- 메시지 인증 코드
  - 대칭키 암호 기반의 메시지 인증
  - 공개키 기반의 메시지 인증
  - 공유 비밀키 기반의 메시지 인증
- HMAC
- 인감과 전자서명



# 해쉬 함수

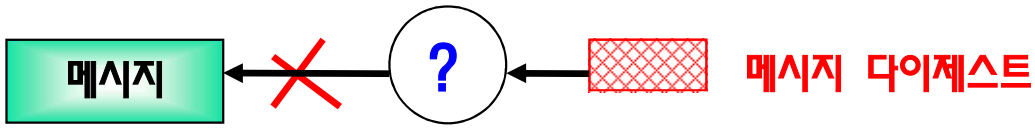
- 메시지를 일정 길이의 블록(예, 512비트)으로 분할 후 해쉬 함수에 입력 ☞ 짧고 일정한 길이의 메시지 다이제스트 생성



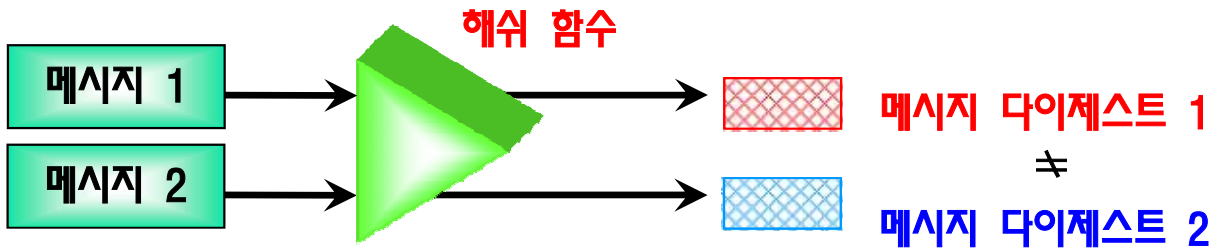

- 메시지 다이제스트는 인증 알고리즘과 결합되어 전자서명 등에 사용되는 메시지 인증코드 생성
- 해쉬함수: MD5, SHA-1, RIPEMD-160, HAS-160(국내 전자서명 인증체계)

# 해쉬 함수의 요구사항

- 메시지 다이제스트의 계산 효율이 좋아야 하며 구현의 실현성이 있어야 함
- 어떤 크기의 데이터 블록이든지 적용 가능하여야 함
- 일방향성: 메시지 다이제스트로부터 원래의 메시지에 대한 계산이 불가능

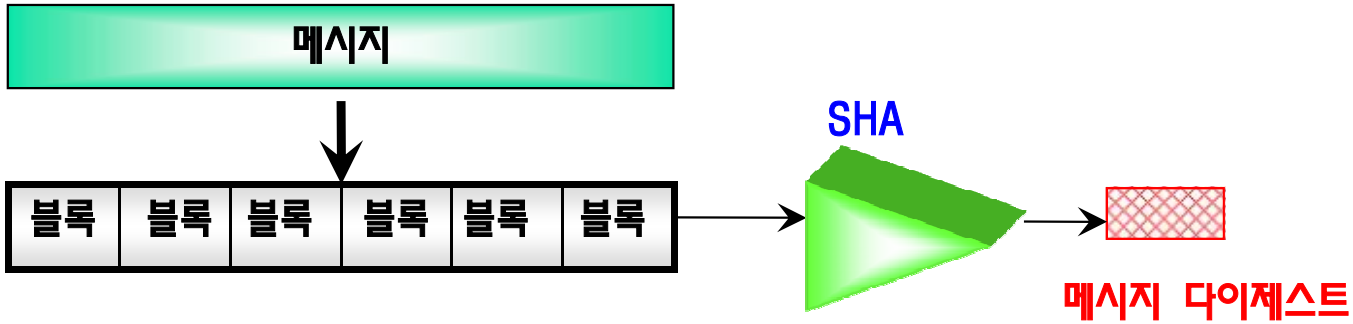



- 강한 충돌 회피성(strongly collision-free)
  - 다른 두 개의 메시지에 대하여 해쉬 함수를 적용한 결과 ☞ 항상 서로 다른 메시지 다이제스트를 출력
  - x1 ≠ x2이면서 h(x1) = h(x2)을 만족하는 두 개의 메시지 x1과 x2를 찾는 일이 계산적으로 불가능



# 안전 해쉬 알고리즘

- 안전 해쉬 알고리즘(SHA: Secure Hash Algorithm)
- 1993년 디지털 서명을 위해서 미국에서 개발된 해쉬 함수




- SHA 개정 판

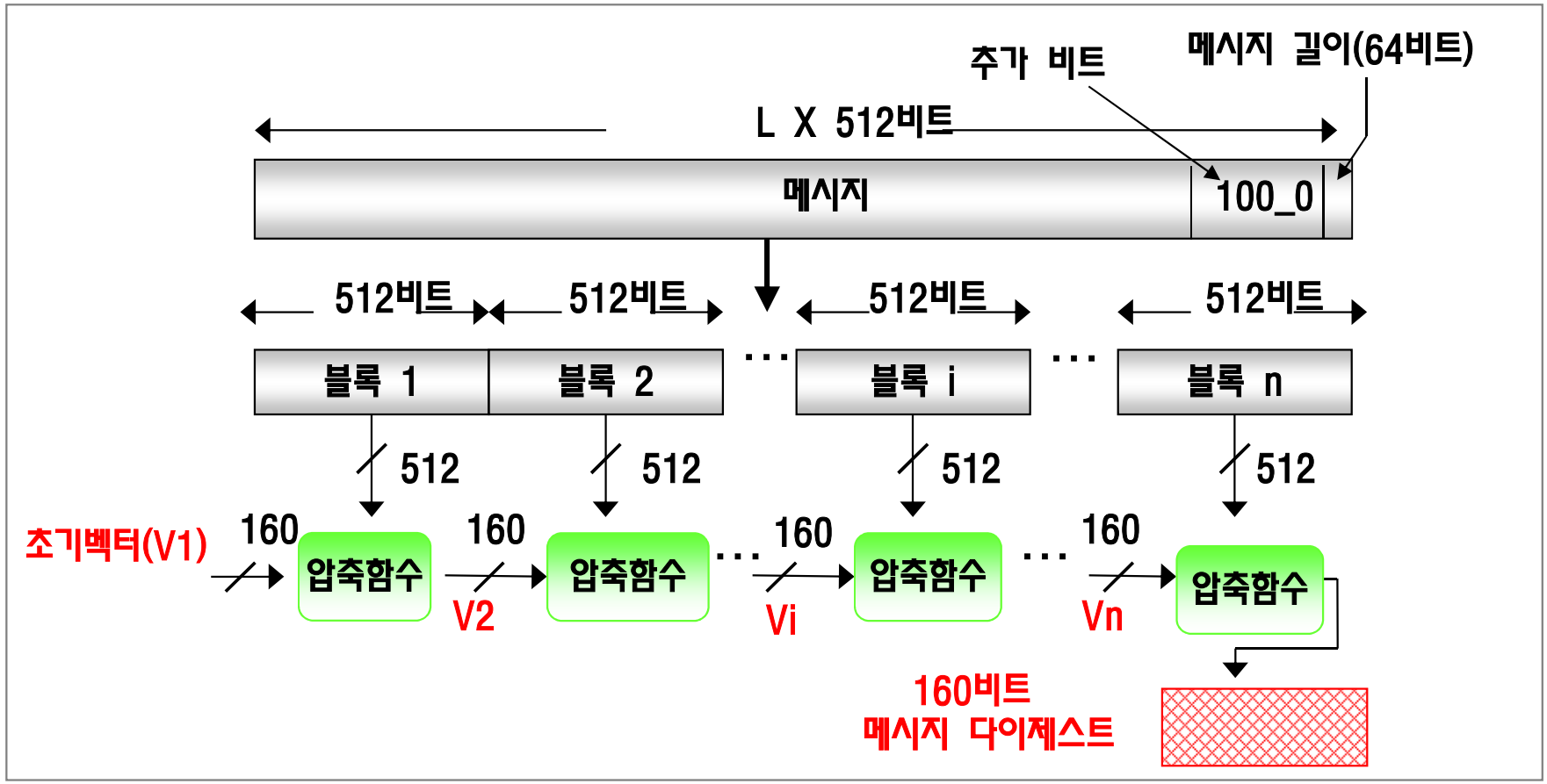
| 알고리즘 | 블럭 길이 (비트) | 메시지 다이제스트 길이 (비트) |
|---|---|---|
| SHA-1 | 512 | 160 |
| SHA-256 | 512 | 256 |
| SHA-384 | 1024 | 384 |
| SHA-512 | 1024 | 512 |

# SHA-1 해쉬 함수

① 메시지에 추가 비트(1000…0) 및 메시지 길이 필드를 추가하여 전체가 512의 배수가 되도록 함

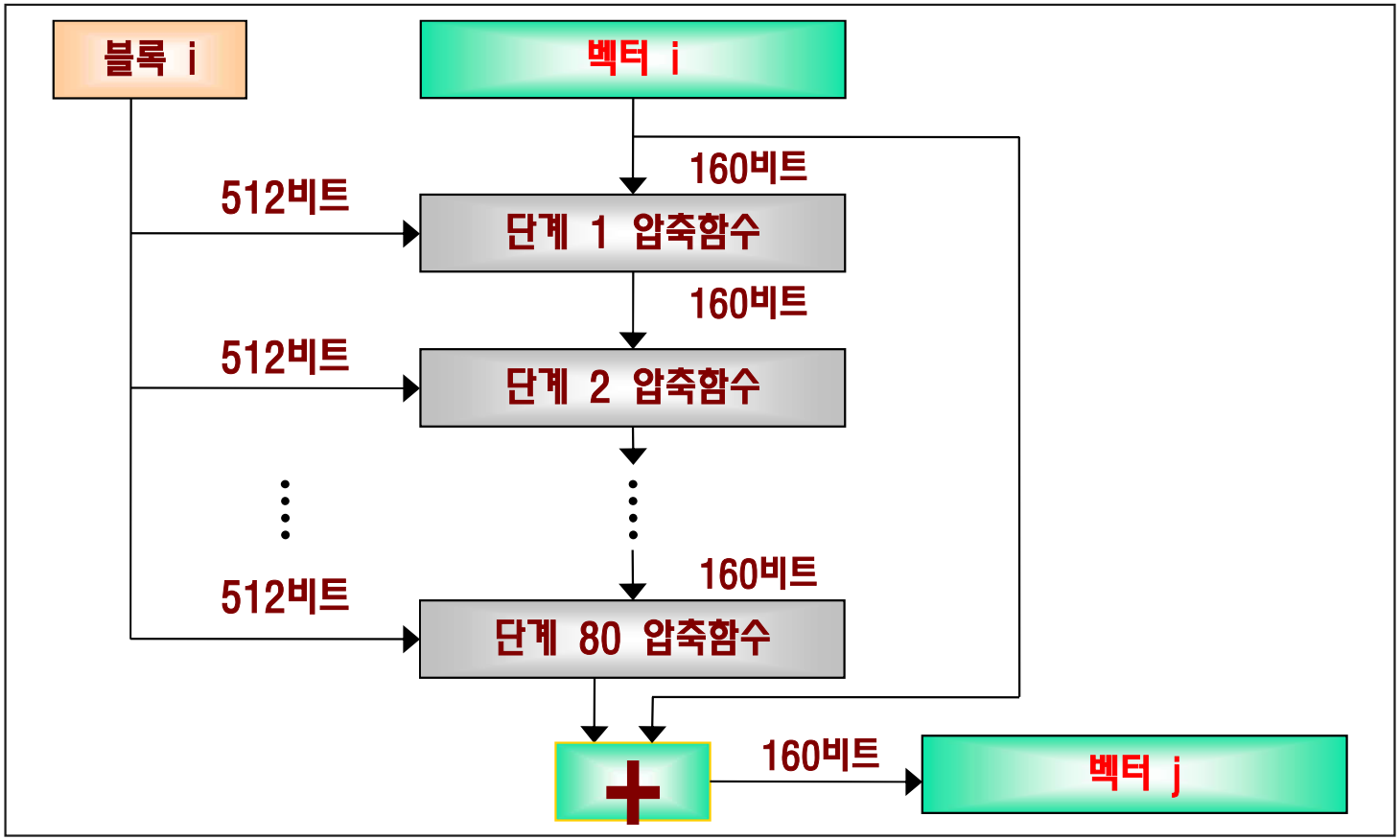
② 160비트의 백터와 512비트 블록을 입력 받아 압축함수 처리

③ 160비트 메시지 다이제스트 생성



# SHA-1의 압축함수

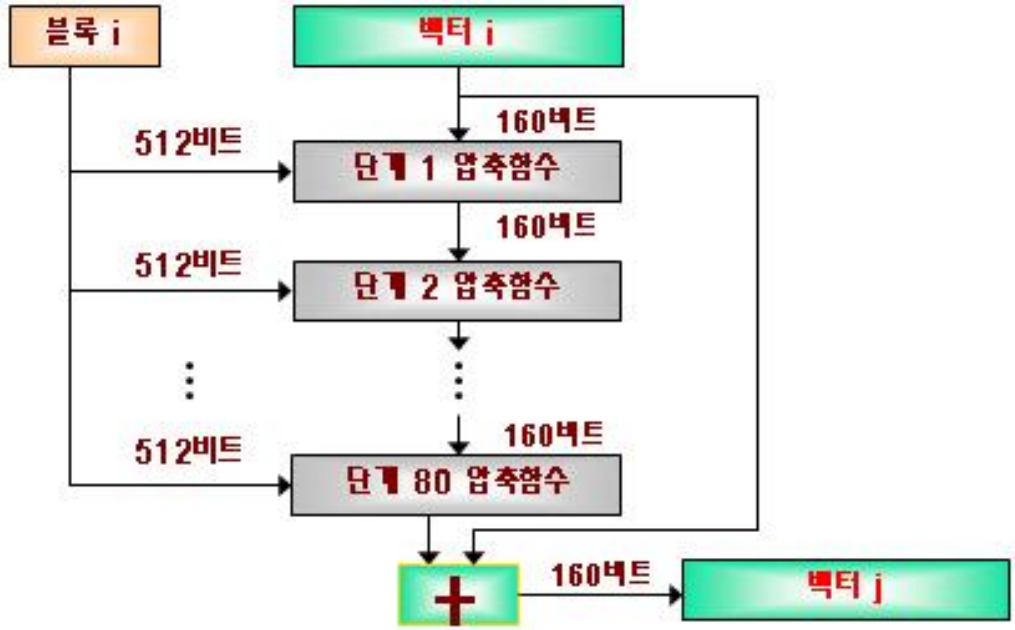
- 입력: 512비트 블록의 메시지, 이전 단계의 160비트 벡터 I
- 절차: 단계 1에서 80까지의 논리 및 압축함수 수행 ➔ 단계 80의 출력과 160비트 벡터 i와 덧셈 수행
- 출력: 다음 단계에서 사용할 160비트 벡터 j 또는 메시지 다이제스트



# 압축함수의 버퍼 및 레지스터

- 압축함수에 사용되는 버퍼 및 레지스터
  - 초기벡터(V1) 및 후속백터(Vi) 160 비트를 구성하는 다섯 레지스터: *A, B, C, D, E*
  - 압축함수의 중간 결과를 저장하는 다섯 레지스터: $H_0$, $H_1$, $H_2$, $H_3$, $H_4$
  - 80개의 32비트 워드 열을 구성하는 단어: $W_0$, …, $W_{79}$ 로 표시
  - 한 개의 워드로 된 TEMP 버퍼

A: 67 45 23 01
B: ef cd ab 89
C: 98 ba dc fe
D: 10 32 54 76
E: c3 d2 e1 f0

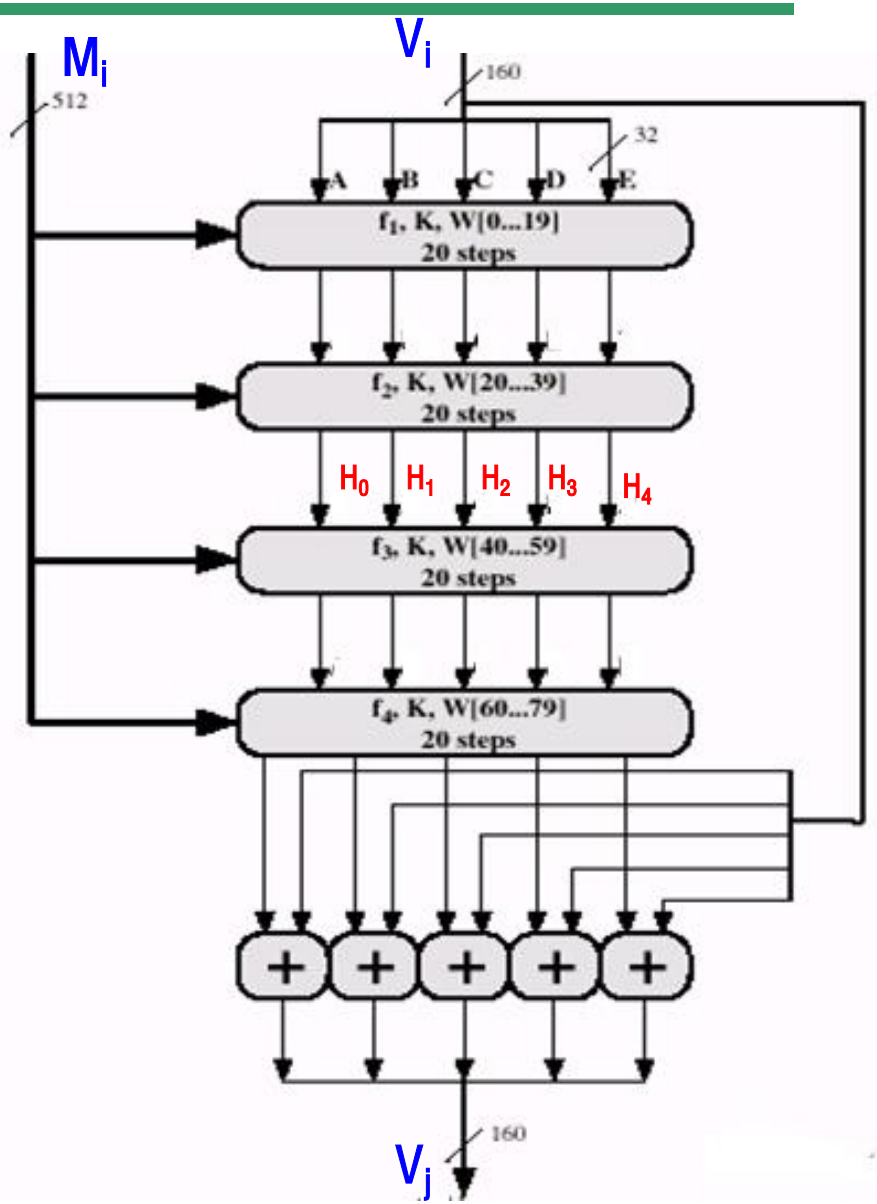


# 압축함수의 기호 정의

- $x \wedge y$ : x와 y의 비트별 AND
- $x \vee y$ : x와 y의 비트별 OR
- $x \oplus y$ : x와 y의 비트별 XOR(exclusive-OR)
- $\neg x$ : x의 비트별 보수
- X+Y : 두 비트 스트링의 모듈로 $2^{32}$ 덧셈
  - $X + Y = X + Y \bmod 2^{32}$
- $S^n(X)$ : 비트 스트링 X를 n비트 만큼 회전좌향 이동
  - $S^n(X) = (X \ll n) \vee (X \gg 32 - n)$
- $M_1, M_2, \cdots, M_i$ : i 번째 블록(512 비트)

# 압축함수의 80단계

- 중간 백터/다이제스트 생성
  - ① $M_i$ 를 16개 단어 $W_0$, $W_1$, ..., $W_{15}$로 분리
  - ② for i = 16 to 79 do
    $W_i = S^1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})$
  - ③ $H_0=A$, $H_1=B$, $H_2=C$, $H_3=D$, $H_4=E$
  - ④ for i = 0 to 79 do
    $TEMP=S^5(H_0)+f_i(H_1,H_2,H_3)+H_4+W_i+K_i$
    $H_4=H_3$, $H_3=H_2$, $H_2=S^{30}(H_1)$
    $H_1=H_0$, $H_0=TEMP$
  - ⑤ $A=A+H_0$, $B=B+H_1$, $C=C+H_2$,
    $D=D+H_3$, $E=E+H_4$
    5개의 레지스터 A, B, C, D, E(160 비트)
    - 중간 블록: 후속 백터($V_j$)
    - 마지막 블록: 메시지 다이제스트



# 압축함수 80 단계의 논리 함수 및 상수 열

- 16단어(512비트) 블록에 대한 압축함수의 80 단계 처리
  - 논리 함수 $f_0, f_1, \cdots, f_{79}$ 를 사용
  - 3개의 32비트 단어 입력, 한 개의 32비트 단어 출력

$$
\begin{array}{lll}
f_i(x,y,z) & = (x \wedge y) \vee (\neg y \wedge z) & (0 \le i \le 19) \\
f_i(x,y,z) & = x \oplus y \oplus z & (20 \le i \le 39) \\
f_i(x,y,z) & = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & (40 \le i \le 59) \\
f_i(x,y,z) & = x \oplus y \oplus z & (60 \le i \le 79)
\end{array}
$$

  - 상수 열 $K_0, K_1, \cdots, K_{79}$ 사용

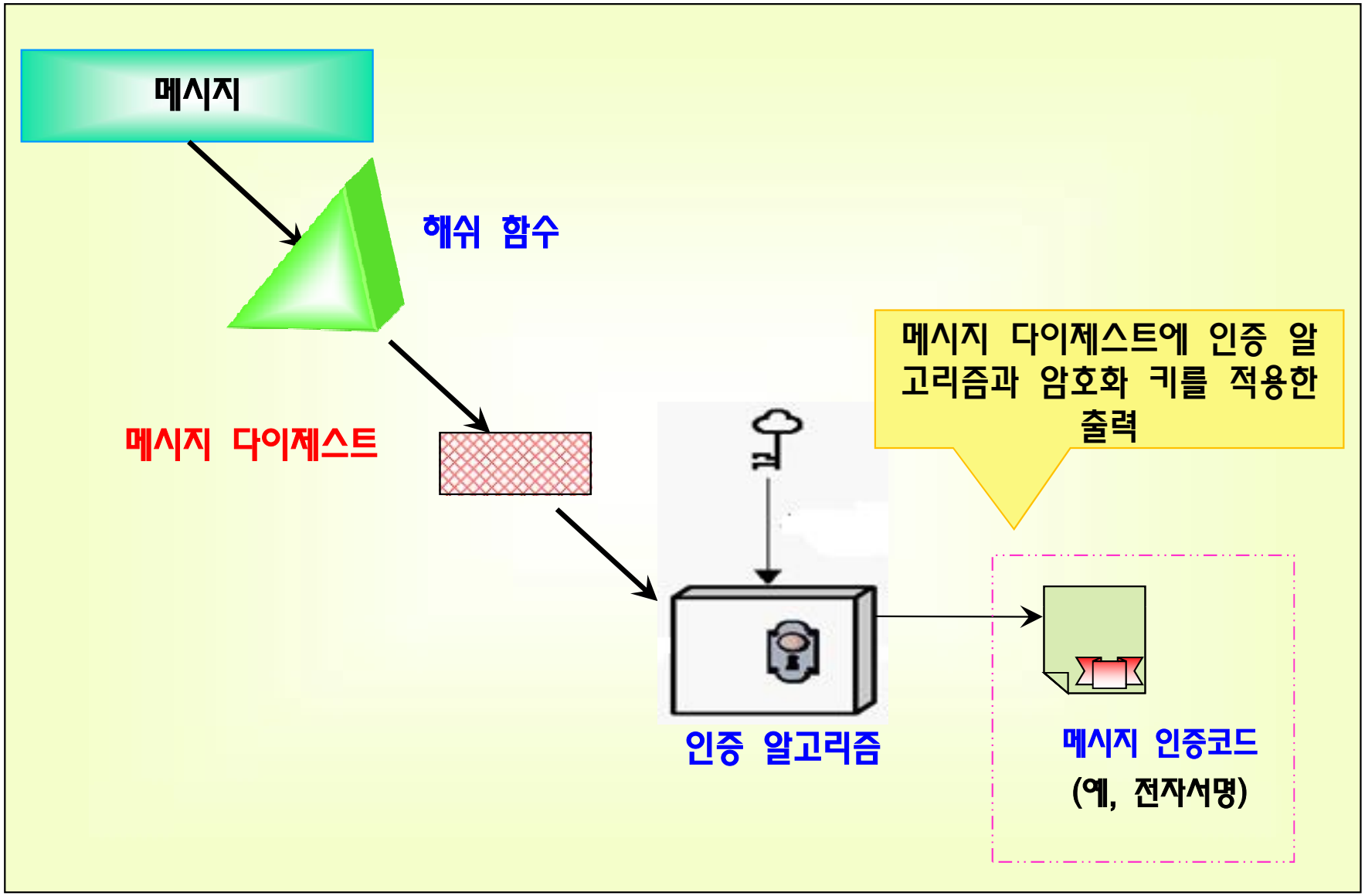
$$
\begin{array}{ll}
K_i = 5a827999 & (0 \le i \le 19) \\
K_i = 6ed9eba1 & (20 \le i \le 39) \\
K_i = 8f1bbcdc & (40 \le i \le 59) \\
K_i = ca62c1d6 & (60 \le i \le 79)
\end{array}
$$

# 해쉬 함수 비교

- MD5: 1992년 MIT에서 개발
  - 계산속도: 단계의 수가 가장 짧은 MD5가 가장 빠름
  - 안전성: 다이제스트 길이가 가장 짧은 MD5가 가장 취약
- SHA-1: 1995년 미국의 디지털 서명을 위해서 개발
- RIPEMD-160: 1996년 유럽연합(EU) 프로젝트에서 개발
- HAS-160: 1998년 한국형 디지털 서명을 위해서 개발

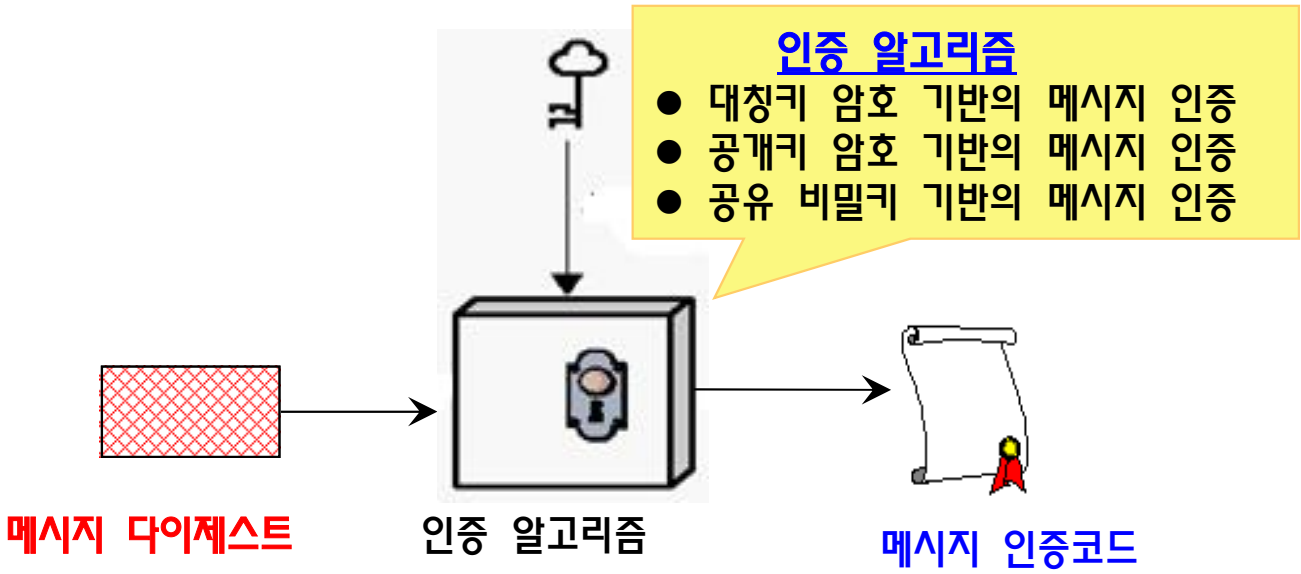
| | MD5 | SHA-1 | RIPEMD-160 | HAS-160 |
|---|---|---|---|---|
| 다이제스트 길이 | 128 비트 | 160 비트 | 160 비트 | 160 비트 |
| 블록 길이 | 512 비트 | 512 비트 | 512 비트 | 512 비트 |
| 단계의 수 | 64 | 80 | 160 | 80 |

# 메시지 인증코드(1/2)

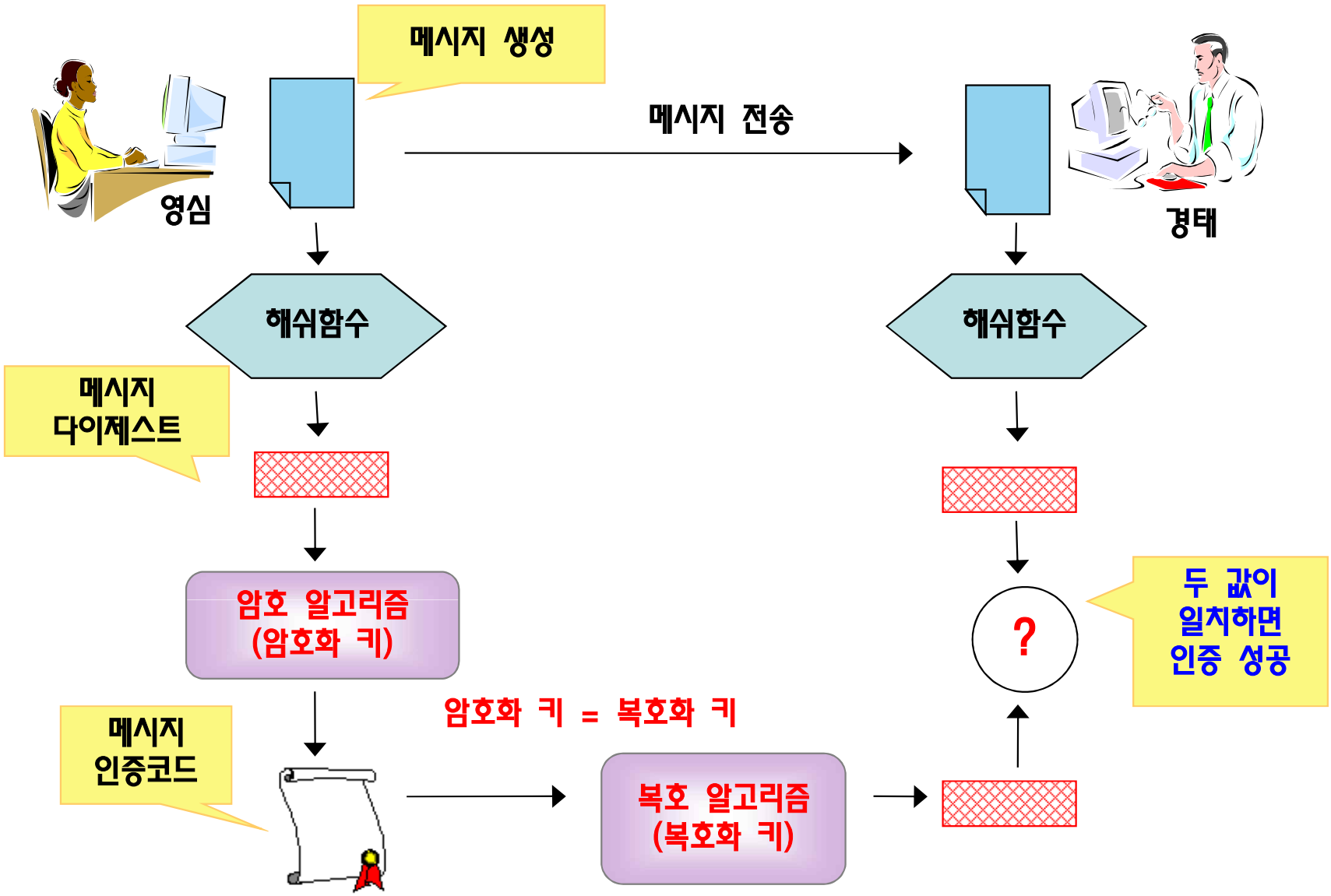

메시지
해쉬 함수
메시지 다이제스트
메시지 다이제스트에 인증 알고리즘과 암호화 키를 적용한 출력
인증 알고리즘
메시지 인증코드
(예, 전자서명)

# 메시지 인증코드(2/2)

- 메시지 인증코드의 응용
  - 전자문서의 전자서명에 적용함으로 송신자의 신원 확인
  - 전자문서의 변조를 방지하는 무결성 서비스
  - 전자상거래에서 거래 사실을 부인하거나 번복하지 못하게 하는 부인 방지



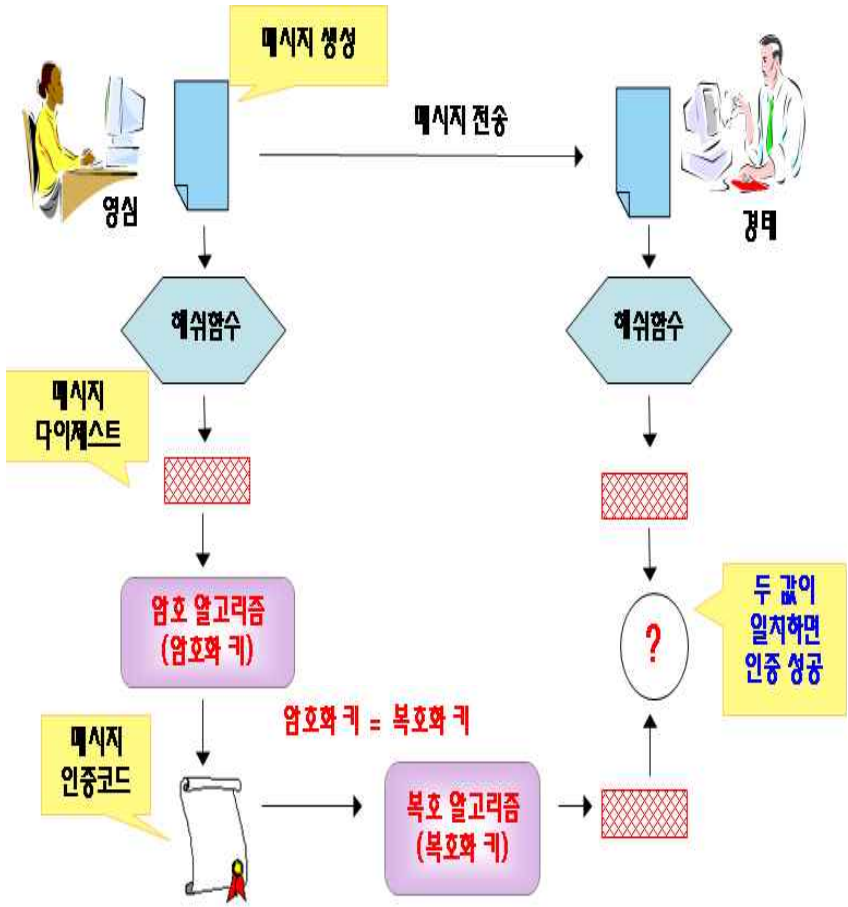
# 대칭키 암호 기반의 메시지 인증(1/2)



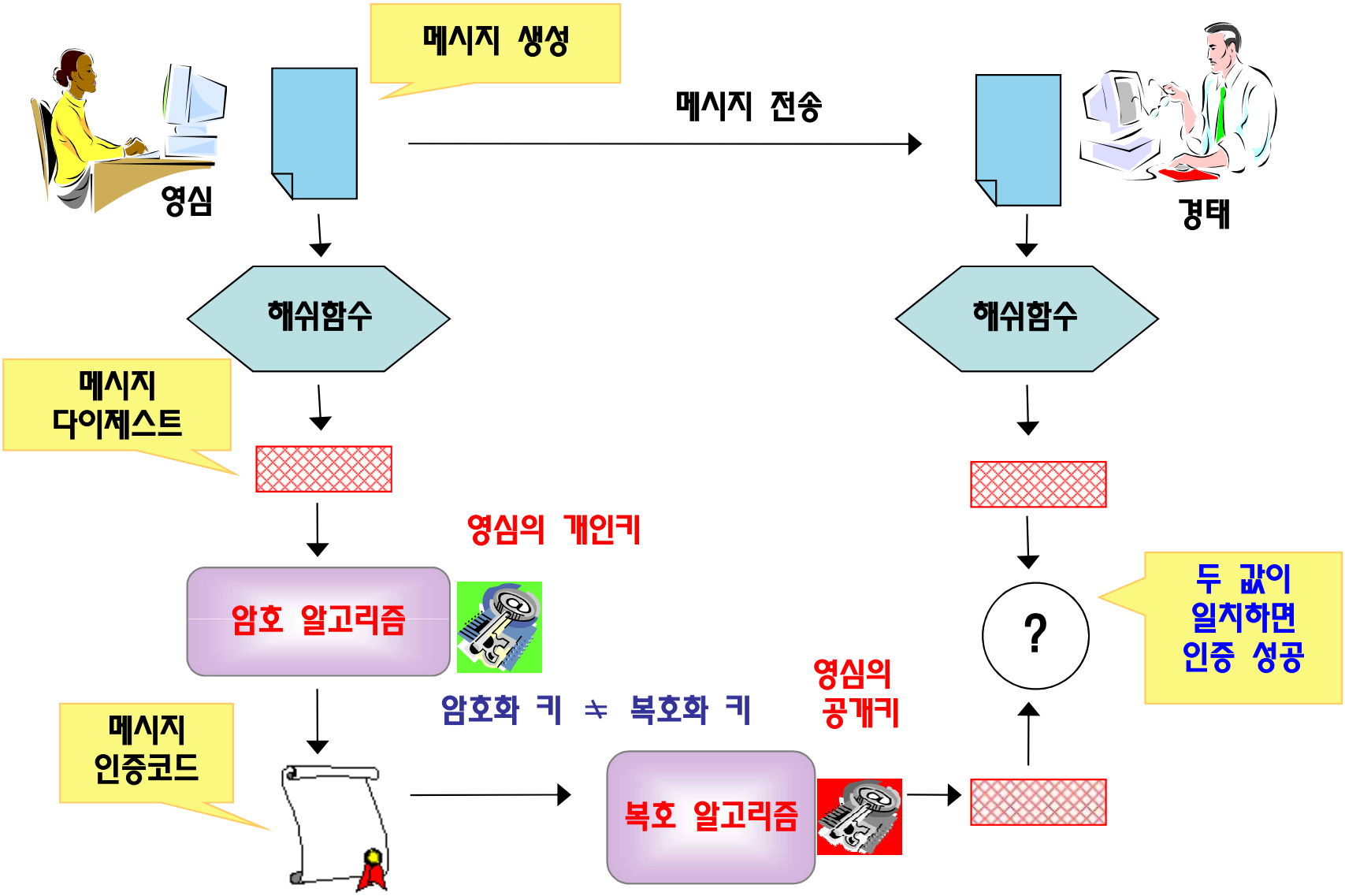
# 대칭키 암호 기반의 메시지 인증(2/2)

- 영심
  - 메시지를 작성한 후 해쉬 함수를 적용하여 메시지 다이제스트 생성
  - 암호화 키로 메시지 다이제스트를 암호화하여 메시지 인증코드 생성
  - 메시지와 메시지 인증코드를 경태에게 동시에 전송

- 경태
  - 수신된 메시지에 해쉬 함수를 적용하여 메시지 다이제스트 생성
  - 수신한 메시지 인증코드를 복호화하여 메시지 다이제스트 유도
  - 두 메시지 다이제스트가 일치하면 영심이가 생성한 메시지 임을 인증

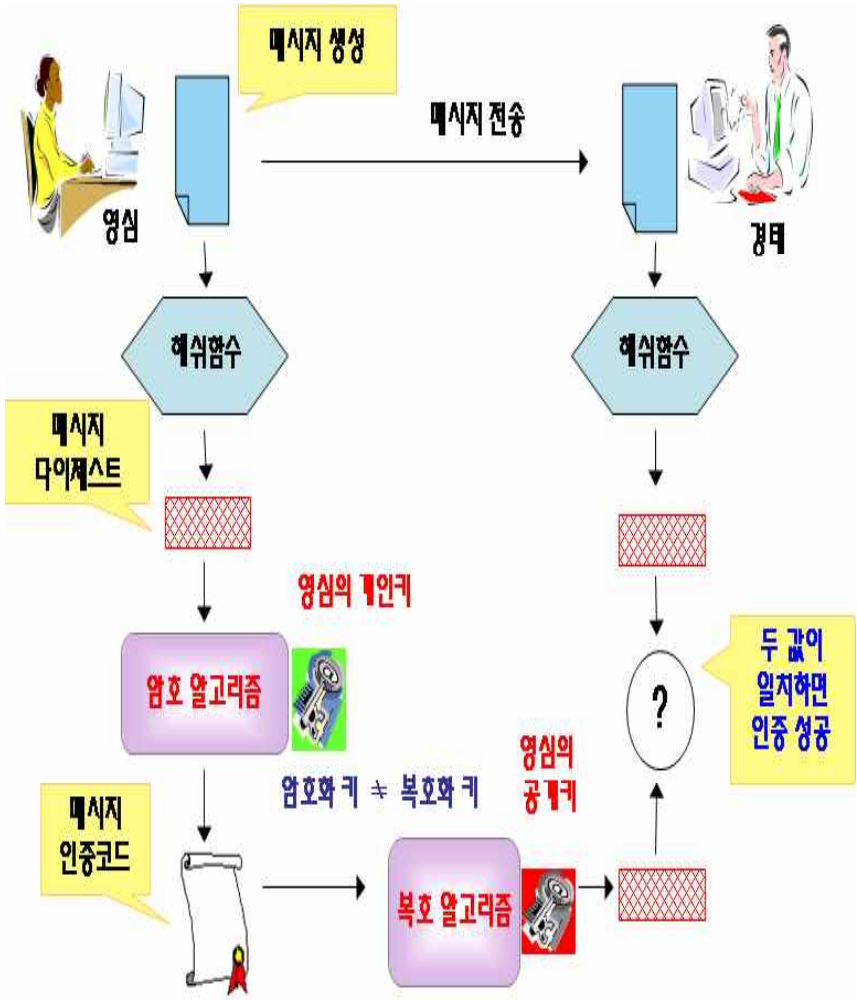


# 해쉬 함수를 이용한 공개키 기반의 메시지 인증(1/2)



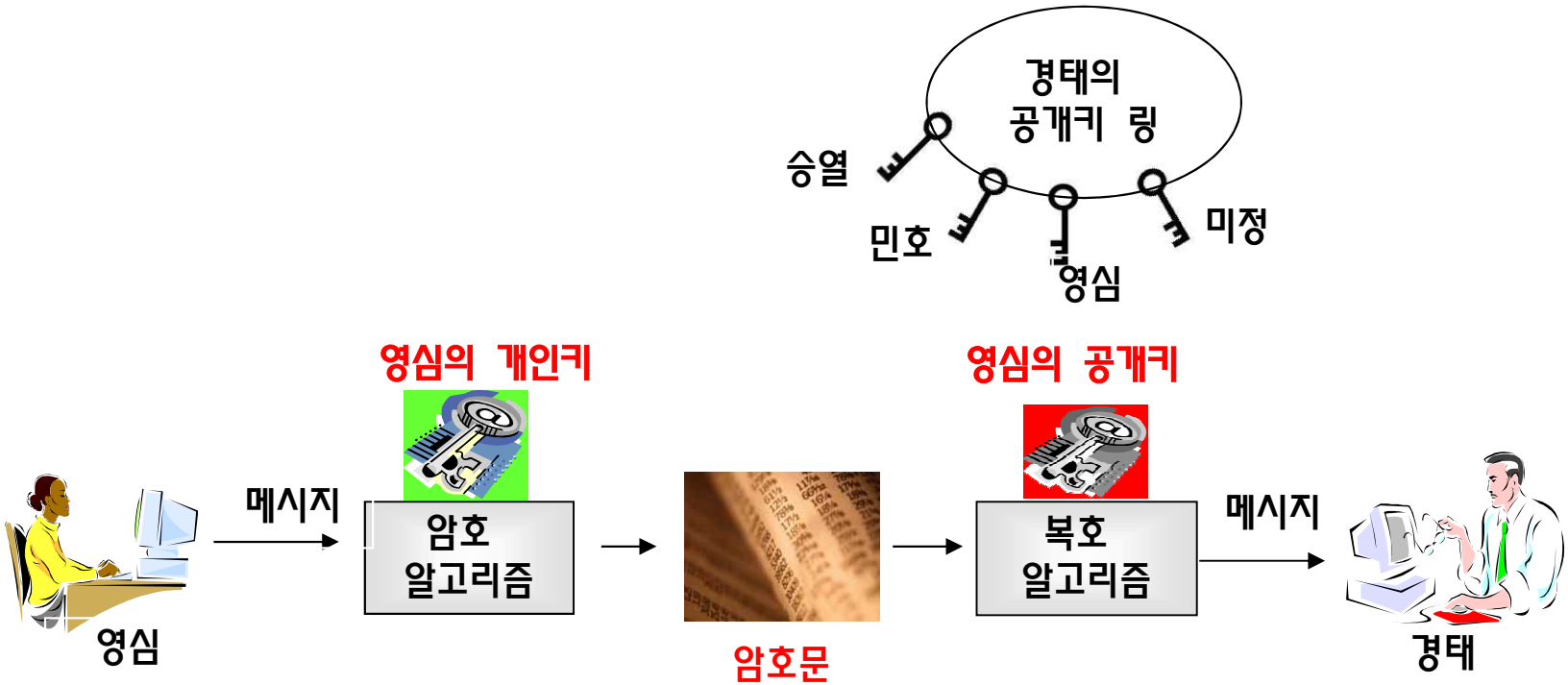
# 해쉬 함수를 이용한 공개키 기반의 메시지 인증(2/2)

- 영심
  - 메시지를 작성 후 해쉬 함수를 적용하여 메시지 다이제스트 생성
  - 영심의 개인키로 메시지 다이제스트를 암호화하여 메시지 인증코드 생성
  - 메시지와 메시지 인증코드를 경태에게 동시에 전송
- 경태
  - 수신한 메시지에 해쉬 함수를 적용하여 메시지 다이제스트 생성
  - 수신한 메시지 인증코드를 영심이의 공개키로 복호화하여 메시지 다이제스트 유도
  - 두 메시지 다이제스트가 일치하면 영심이가 생성한 메시지 임을 인증



# 해쉬 함수가 없는 공개키 기반의 메시지 인증

- 메시지를 작성 후 송신자인 영심의 개인키로 메시지 전체를 암호화
- 경태는 공개키 링에서 영심이의 공개키를 선택하여 수신한 암호문을 복호화
- 해독할 수 있는 평문을 얻으면 영심이가 생성한 메시지 임을 인증

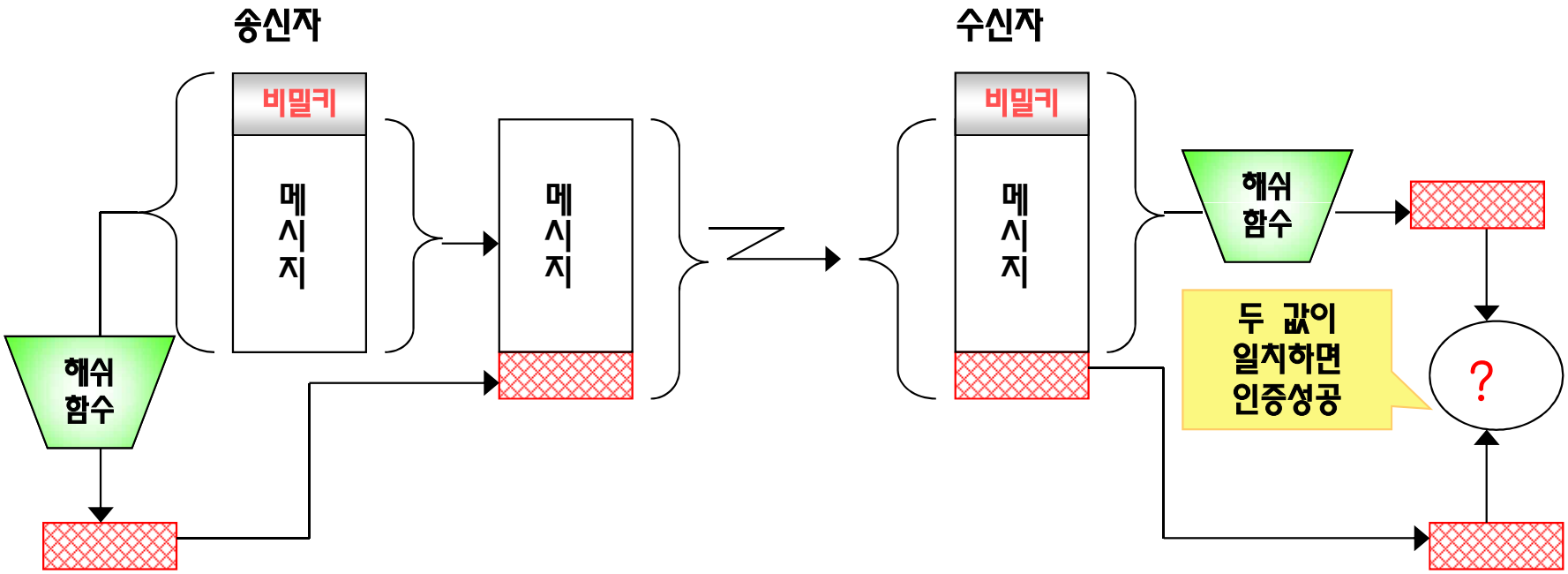


# 공개키 기반의 인증 방식 비교

- 해쉬 함수를 이용하지 않은 메시지 인증
  - 메시지 전체에 대하여 메시지 인증코드의 생성 및 검증을 수행하므로 시간이 많이 소요
  - 별도의 해쉬 함수가 필요 없음

- 해쉬 함수 기반의 메시지 인증
  - 메시지 인증코드의 추가 전송으로 전송량 증가
  - 메시지 다이제스트에만 공개키 암호 알고리즘을 적용하므로 메시지 인증코드의 생성 및 검증이 빠름
  - 현재 많이 사용되는 추세

# 공유 비밀키 기반의 메시지 인증

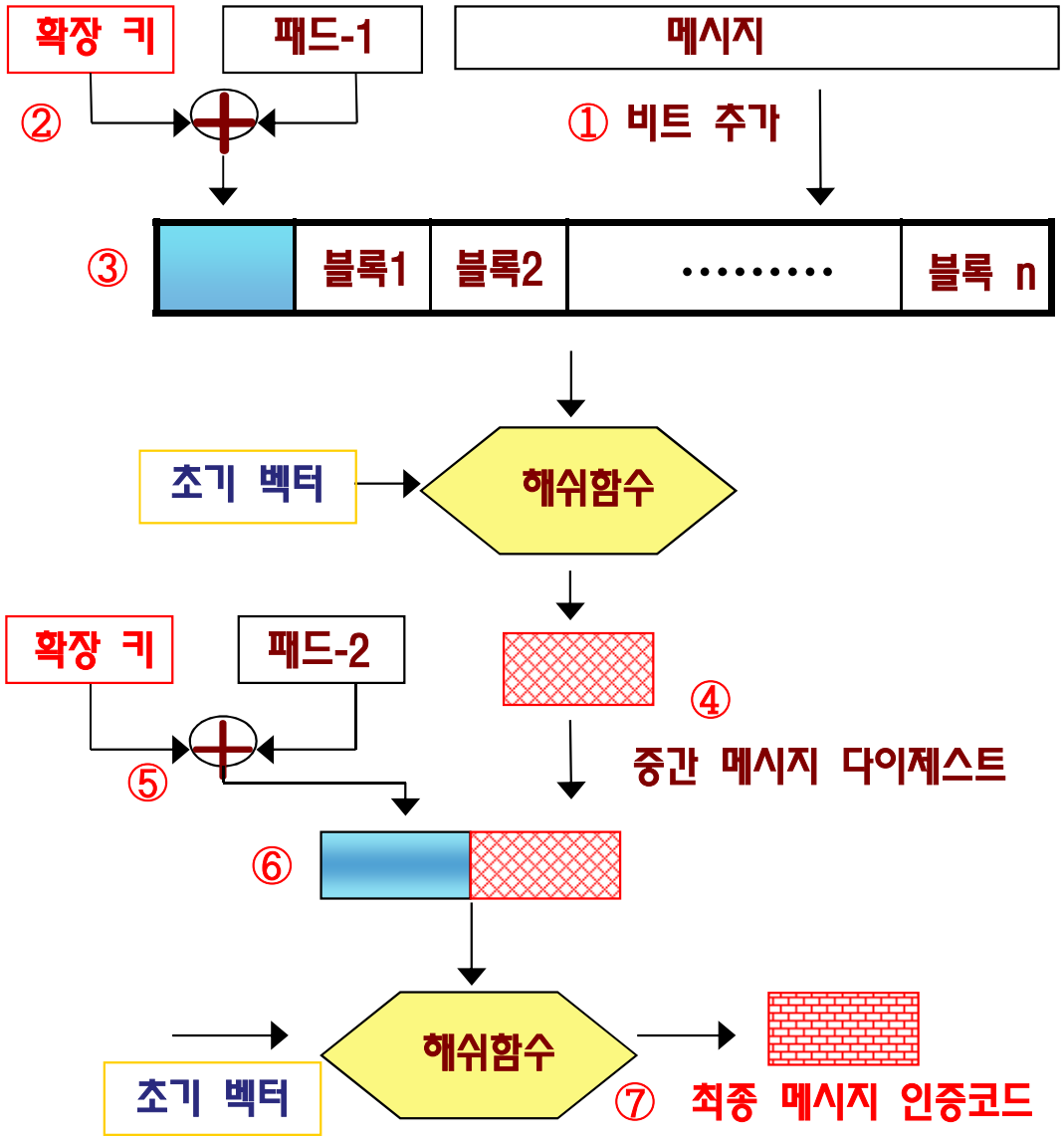



① 송신자와 수신자는 메시지 인증에 사용할 비밀키를 사전에 공유
② 송신자는 비밀키를 메시지에 추가하여 해쉬 함수를 수행
③ 비밀키를 제외한 메시지와 생성된 메시지 다이제스트를 동시에 전송
④ 수신자는 수신된 메시지에 비밀키를 추가하여 해쉬 함수를 수행
⑤ 생성된 메시지 다이제스트와 수신한 메시지 다이제스트가 일치하면 검증된 송신자가 송신한 메시지 임을 인증

# HMAC

- HMAC은 공유 비밀키 기반의 메시지 인증 알고리즘
- HMAC은 MD5, SHA-1, RIPEMD-160 또는 임의의 다른 해쉬 함수와 조합될 수 있음
- HMAC-MD5와 HMAC-SHA-1은 IP 보안 프로토콜의 인증 기능에 사용되는 필수 알고리즘
- HMAC의 안전성: 사용되는 해쉬 함수가 강한 충돌 회피성을 갖고 있는 경우 HMAC도 안전성을 보장
  - 강한 충돌 회피성: 다른 두 개의 메시지에 대하여 해쉬 함수를 적용한 결과는 서로 다른 메시지 다이제스트를 출력

# HMAC 절차




- 확장 키: 블록 길이가 되도록 비밀키에 0 추가
- 패드-1: 00110110 이 블록 길이 만큼 반복
- 패드-2: 01011100 이 블록 길이 만큼 반복

① 메시지를 해쉬 함수가 요구하는 블록의 배수가 되도록 비트 추가

② 비밀키의 길이가 블록 길이와 같도록 뒤에 0을 추가하여 확장 키를 생성하여 패드-1과 XOR

③ 단계 1과 2의 바이트 스트링들을 서로 연결

④ 단계 3의 결과에 해쉬 함수를 적용하여 중간 메시지 다이제스트 생성

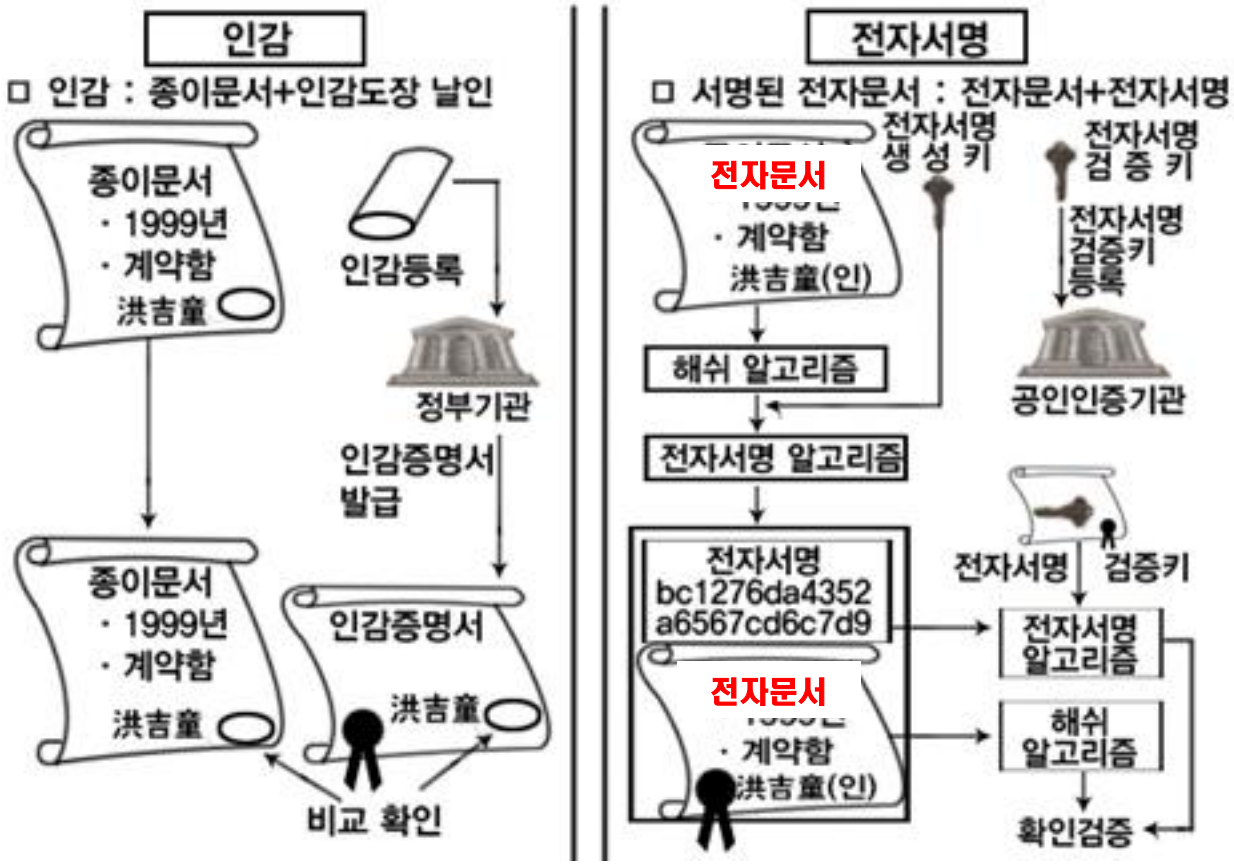
⑤ 확장 키와 패드-2의 XOR

⑥ 단계 5의 결과와 중간 메시지 다이제스트를 서로 연결

⑦ 단계 6의 결과에 해쉬 함수를 적용하여 송신할 최종 메시지 인증코드 생성

# 인감과 전자서명

❑ 전자서명 생성 및 검증키: 대칭키, 개인키/공개키, 공유 비밀키



# 요점 정리(1/3)

- 해쉬 함수: 긴 메시지를 일정 길이의 블록으로 분할 후 압축 기능을 통하여 짧고 일정한 길이의 메시지 다이제스트 생성

- 해쉬 함수의 요구사항
  - 고정길이의 출력인 메시지 다이제스트 생성
  - 계산 효율이 좋아야 하며 구현의 실현성이 있어야 함
  - 일방향성: 메시지 다이제스트로 부터 원래의 메시지에 대한 계산이 불가능
  - 강한 충돌 회피성

- 안전 해쉬 알고리즘(SHA: Secure Hash Algorithm)
  - 1993년 디지털 서명을 위해서 미국에서 개발된 해쉬 함수
  - 80단계의 논리 및 압축함수 수행: 160비트의 백터와 512비트 블록을 입력 받아 160비트 메시지 다이제스트 생성

# 요점 정리(2/3)

- 해쉬함수의 비교
  - MD5(1992년 MIT): 512비트 블록, 128비트 다이제스트, 64단계
    - 계산속도: 단계의 수가 가장 짧은 MD5가 가장 빠름
    - 안전성: 다이제스트 길이가 가장 짧은 MD5가 가장 취약
  - SHA-1(1995년 미국): 512비트 블록, 160비트 다이제스트, 80단계
  - RIPEMD-160(1996년 EU): 512비트 블록, 160비트 다이제스트, 80단계
  - HAS-160(1998년 한국); 512비트 블록, 160비트 다이제스트, 80단계

- 메시지 인증코드: 메시지 다이제스트에 인증 알고리즘과 암호화 키를 적용한 출력

- 메시지 인증코드의 응용
  - 전자 문서의 전자서명에 적용함으로 송신자의 신원 확인
  - 전자 문서의 변조나 위조를 방지하는 무결성 서비스
  - 전자상거래 사실을 부인하거나 번복하지 못하게 하는 부인 방지

# 요점 정리(3/3)

- 메시지 인증코드의 생성 방식
  - 대칭키 암호 기반의 메시지 인증
  - 해쉬함수를 이용하지 않는 공개키 암호 기반의 메시지 인증
    - 별도의 해쉬 함수가 필요 없으나, 메시지 전체에 대하여 메시지 인증코드의 생성 및 검증을 수행하므로 시간이 많이 소요
  - 해쉬함수를 이용하는 공개키 암호 기반의 메시지 인증
    - 메시지 인증코드의 추가 전송으로 전송량 증가
    - 메시지 다이제스트에만 공개키 암호 알고리즘을 적용하므로 메시지 인증 코드의 생성 및 검증이 빨리 수행되므로 현재 많이 사용되는 추세
  - 공유 비밀키 기반의 메시지 인증
    - HMAC은 공유 비밀키 기반의 메시지 인증 알고리즘으로 MD5, SHA-1, RIPEMD-160 또는 임의의 다른 해쉬 함수와 조합될 수 있음