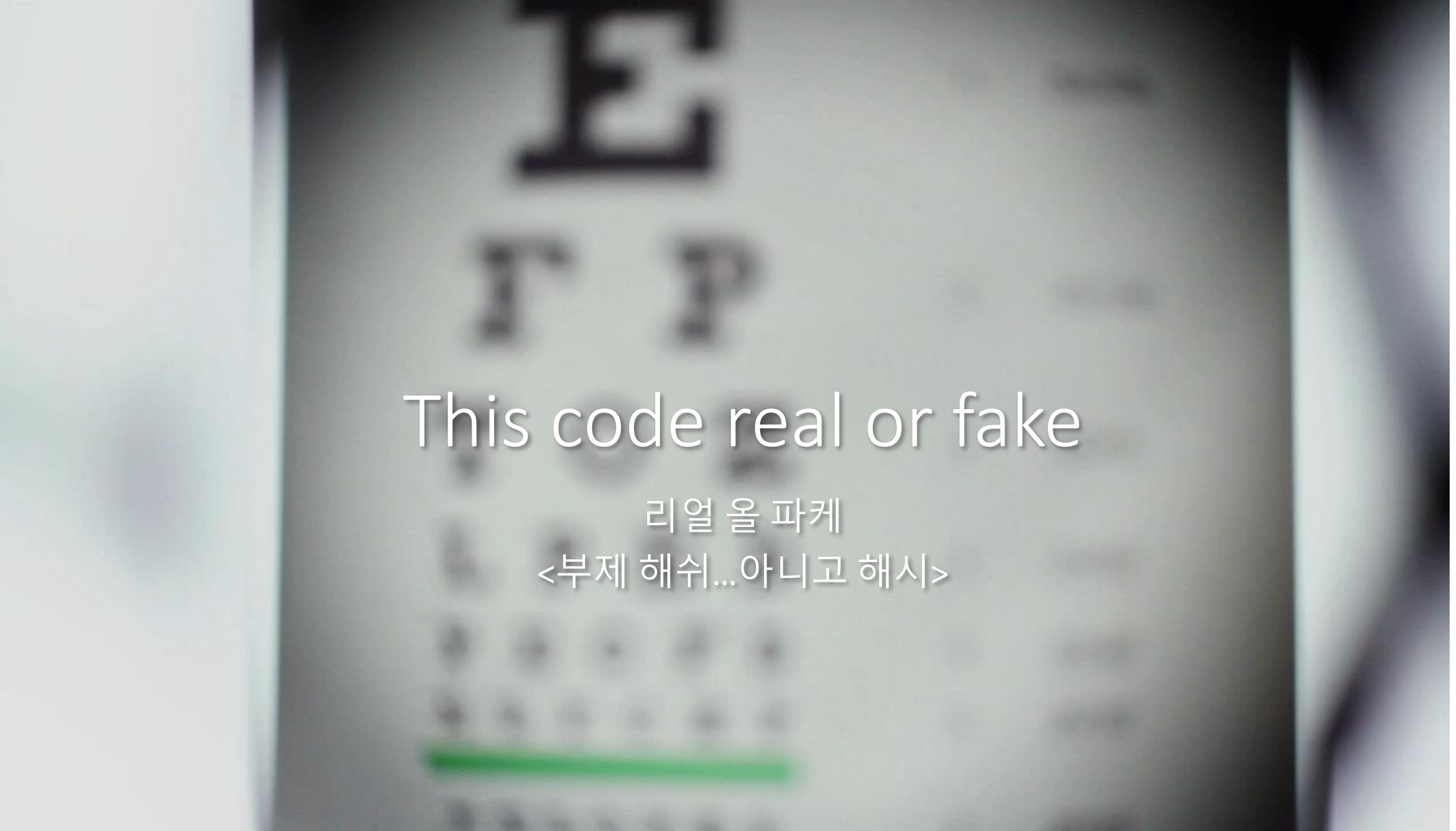


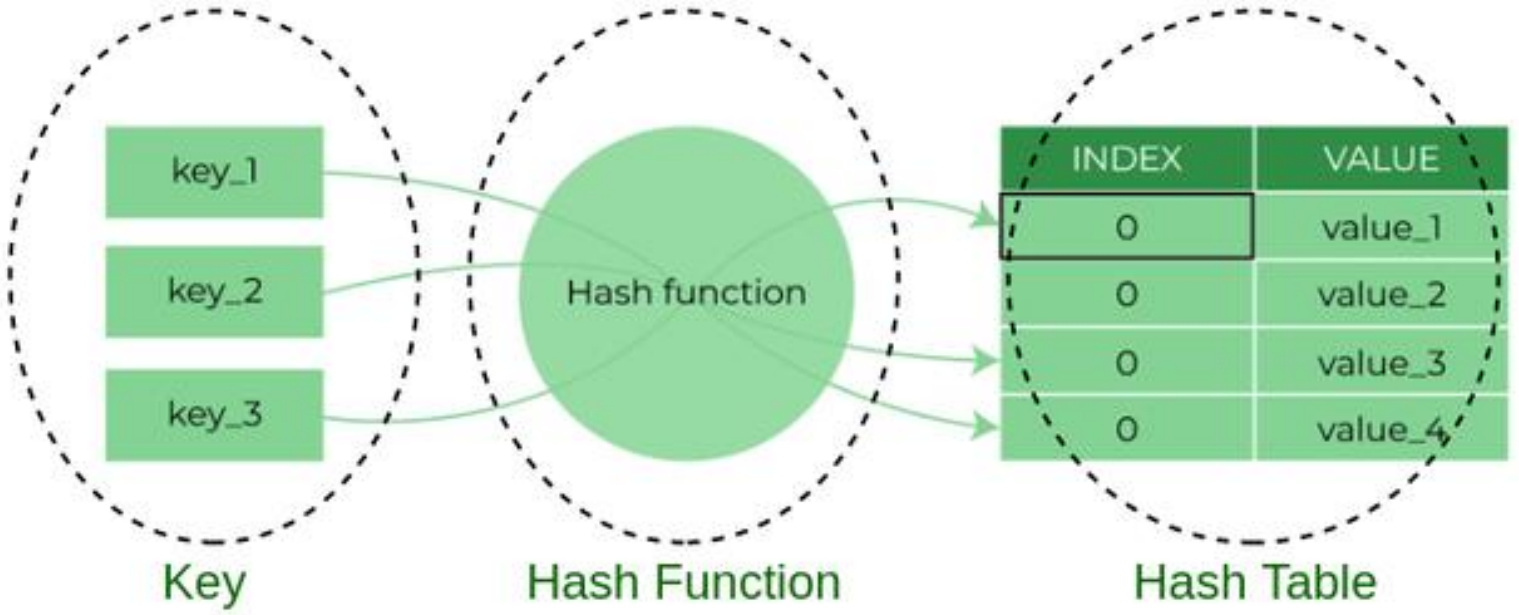
This code real or fake

리얼 올 파케

<부제 해쉬...아니고 해시>

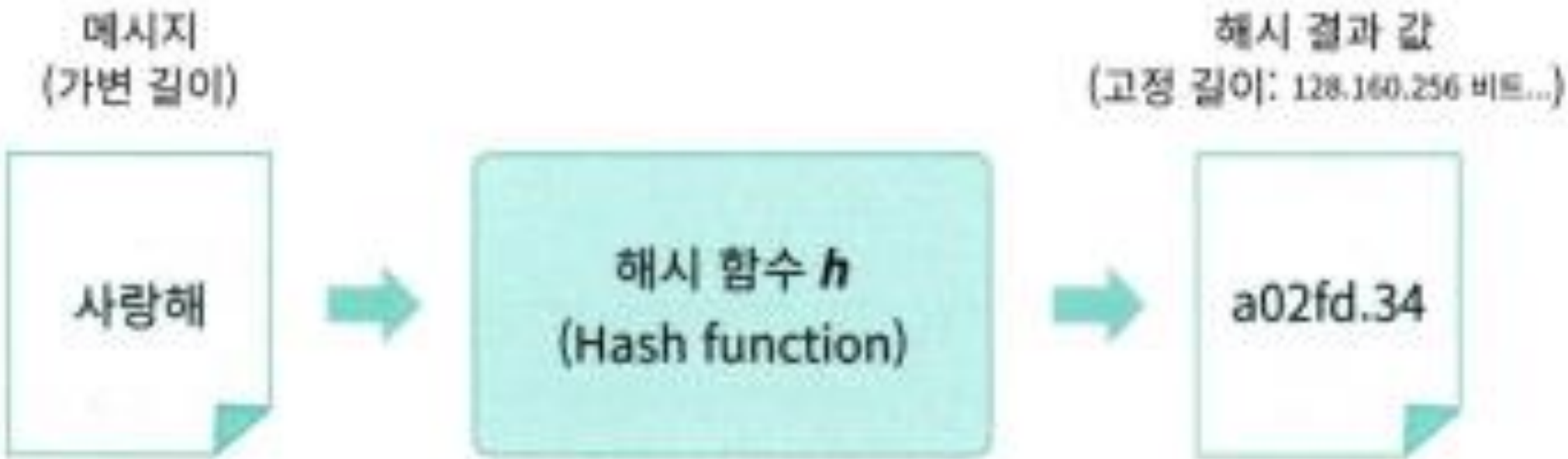
어라...해시?

- 해시 알고리즘(Hash algorithm) – 키를 사용하여 데이터 찾기



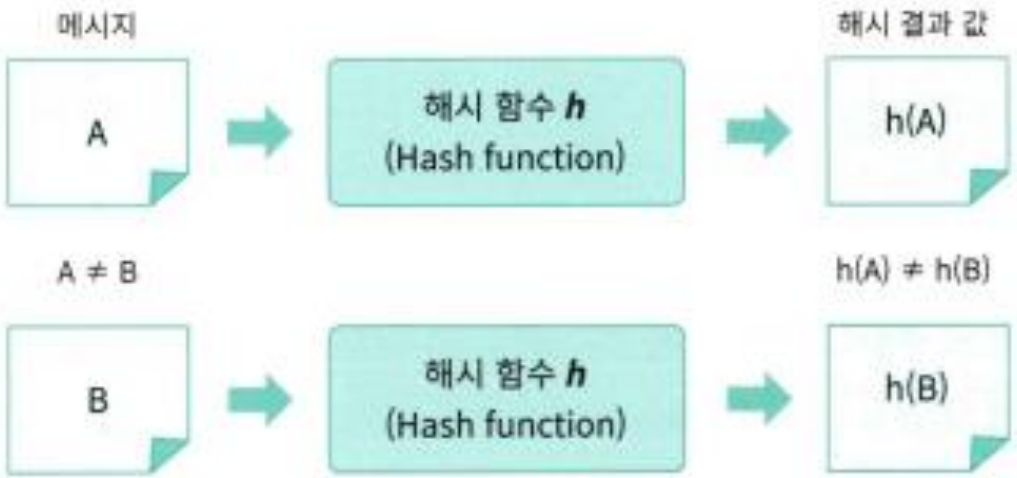
저장 순서 보장 x

->단순히 빠르게 데이터 검사


메시지
(가변 길이)
사랑해
해시 함수 h
(Hash function)
해시 결과 값
(고정 길이: 128.160.256 비트...)
a02fd.34

One-wayness(일방향성)

Collision-free(충돌회피)



효율성 중요

		해시 결괏값 크기 (비트)	최대 입력 메시지 크기(비트)	라운드 수	블록 크기 (비트)
MD5		128	264 - 1	64	512
SHA-1		160	264 - 1	80	512
SHA-2	SHA-256	256	264 - 1	64	512
	SHA-384	384	2128 - 1	80	1024
	SHA-512	512			
SHA-3	SHA3-256	256	∞	24	1088
	SHA3-384	384			832
	SHA3-512	512			576
	SHAKE 128	임의의 크기			1344
	SHAKE 256	임의의 크기			1088

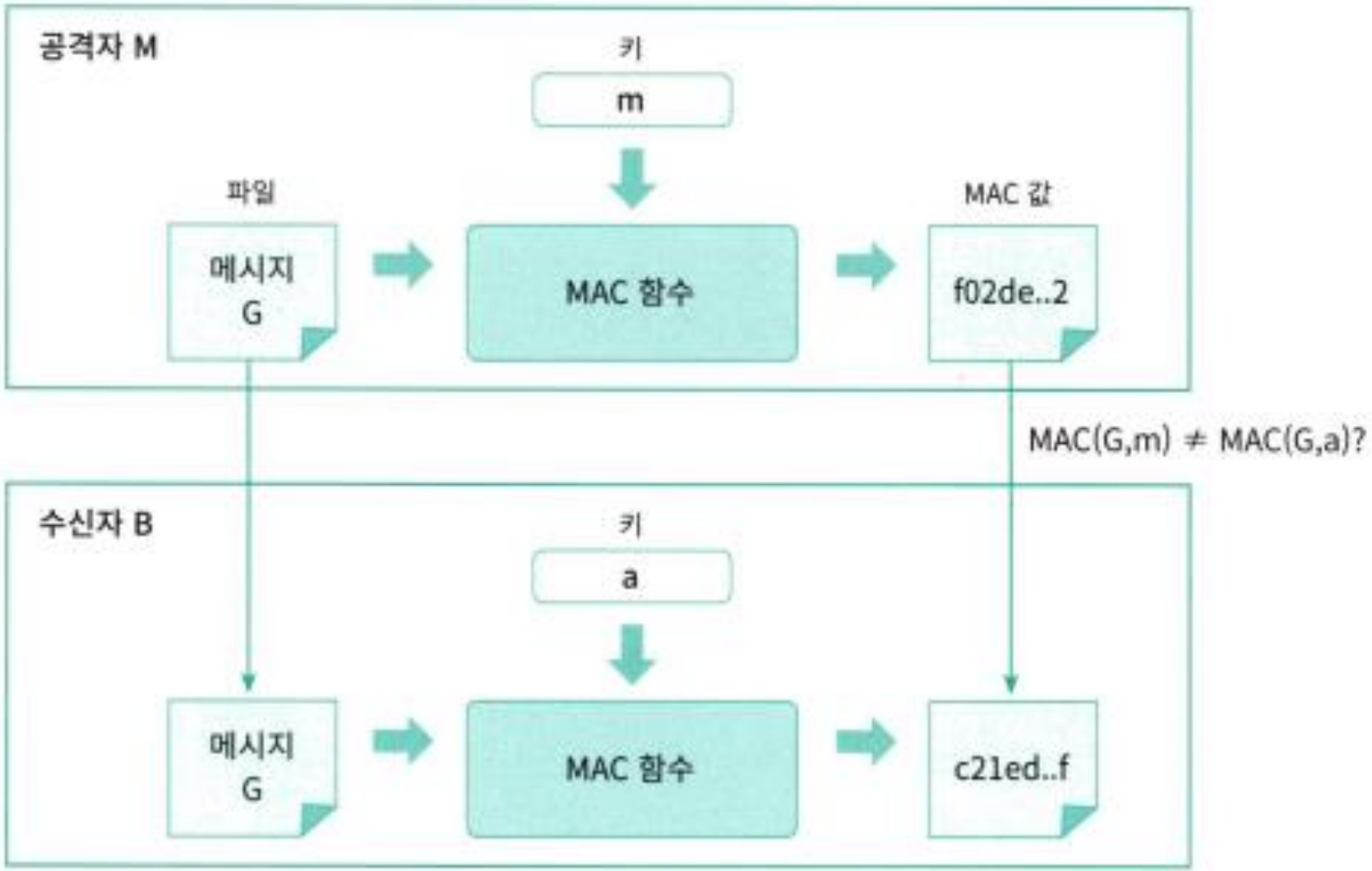
코드

n)

누가 보냈는가?


키
86f01df27
메시지
(가변 길이)
사랑해
MAC 함수
MAC 값
(고정 길이: 128.160.256 비트...)
b1efd..1

어떻게 걸러지는가


공격자 M
키
m
파일
메시지
G
MAC 함수
MAC 값
f02de..2
MAC(G,m) ≠ MAC(G,a)?
수신자 B
키
a
메시지
G
MAC 함수
c21ed..f

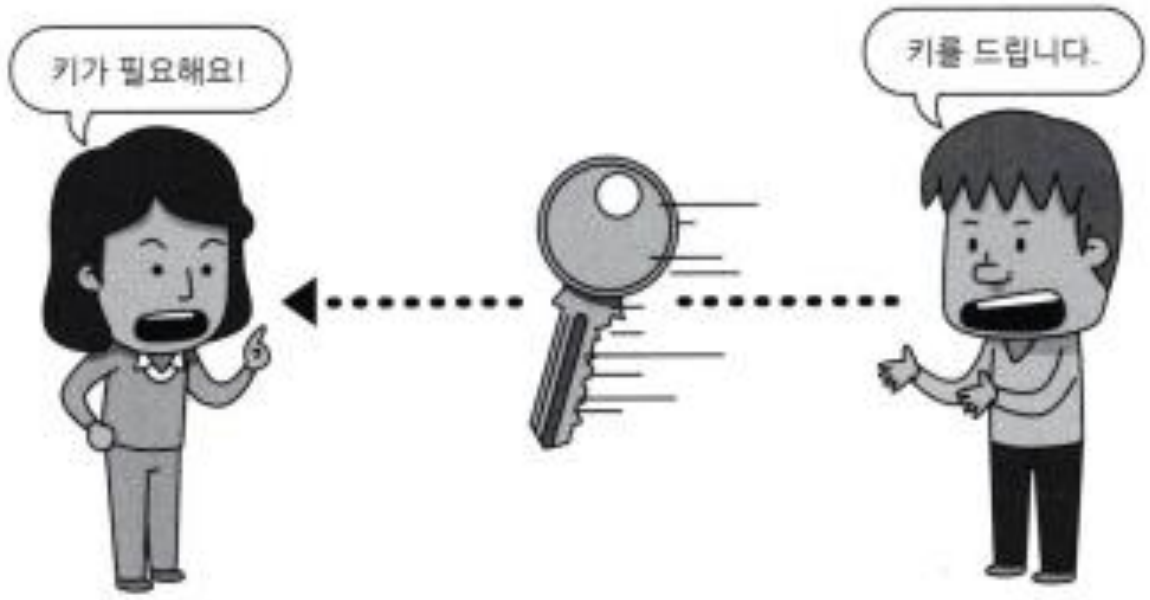
HMAC(Hash Message Authenticaton code)

해시 함수 이용 메시지 인증 코드 구현 방법

```
01: hmac()
02:   입력:
03:    key                                  // 키: 바이트 배열 (byte array)
04:    message                              // 메시지: 바이트 배열 (byte array)
05:
06: // 키의 길이가 블록 길이보다 긴 경우
07: if (length(key) > blockSize) then       // 기존 키에 대해 블록 길이 만큼의 해시 값을 구해,
08:    key ← hash(key)                      // 새로운 키를 만든다
09:
10:
11: //키의 길이가 블록 길이보다 짧은 경우
12: if (length(key) < blockSize) then       // 기존 키에 블록 길이 만큼
13:    key ← Pad(key, blockSize)            // 남은 부분을 0으로 채운 새로운 키를 만든다
14
15:
16: i_key_pad = key ⊕ [0x36 * blockSize]  // ⊕:XOR, Inner padded key
17: o_key_pad = key ⊕ [0x5c * blockSize]  // ⊕:XOR, Outer padded key
18:
19: return hash(o_key_pad || hash(i_key_pad || message))
```

제약 사항 및 공격 방법

1. 키 배송 문제
2. 제 3자에 대한 증명 불가
3. 부인 방지
4. 재전송 공격 대안

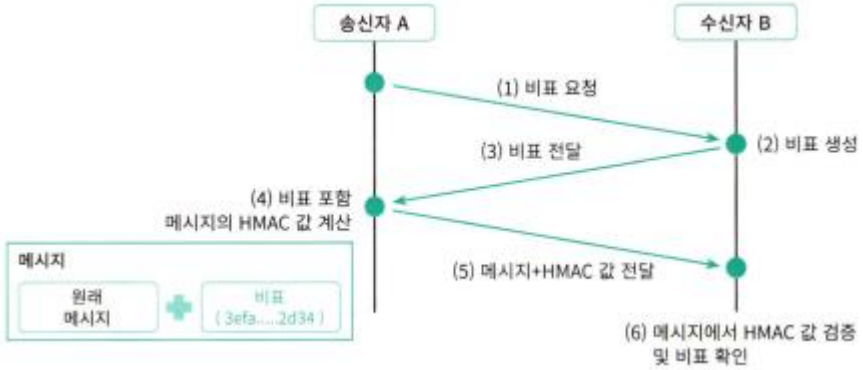


해결방법

- 시퀀스 번호

- 타임 스탬프

- 비표



전자서명 역할

- 메시지 인증
- 메시지 무결성 검증
- 부인방지

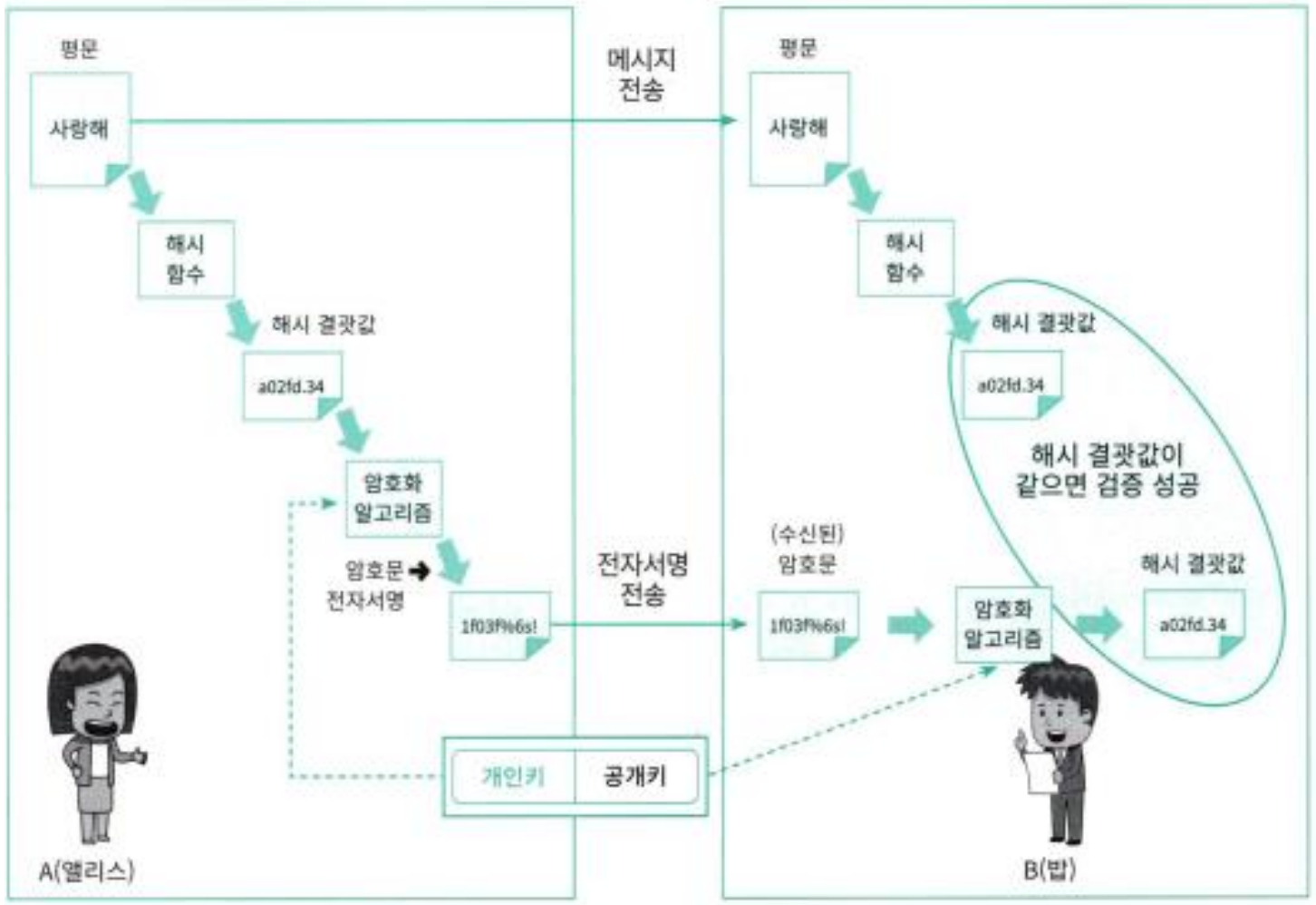
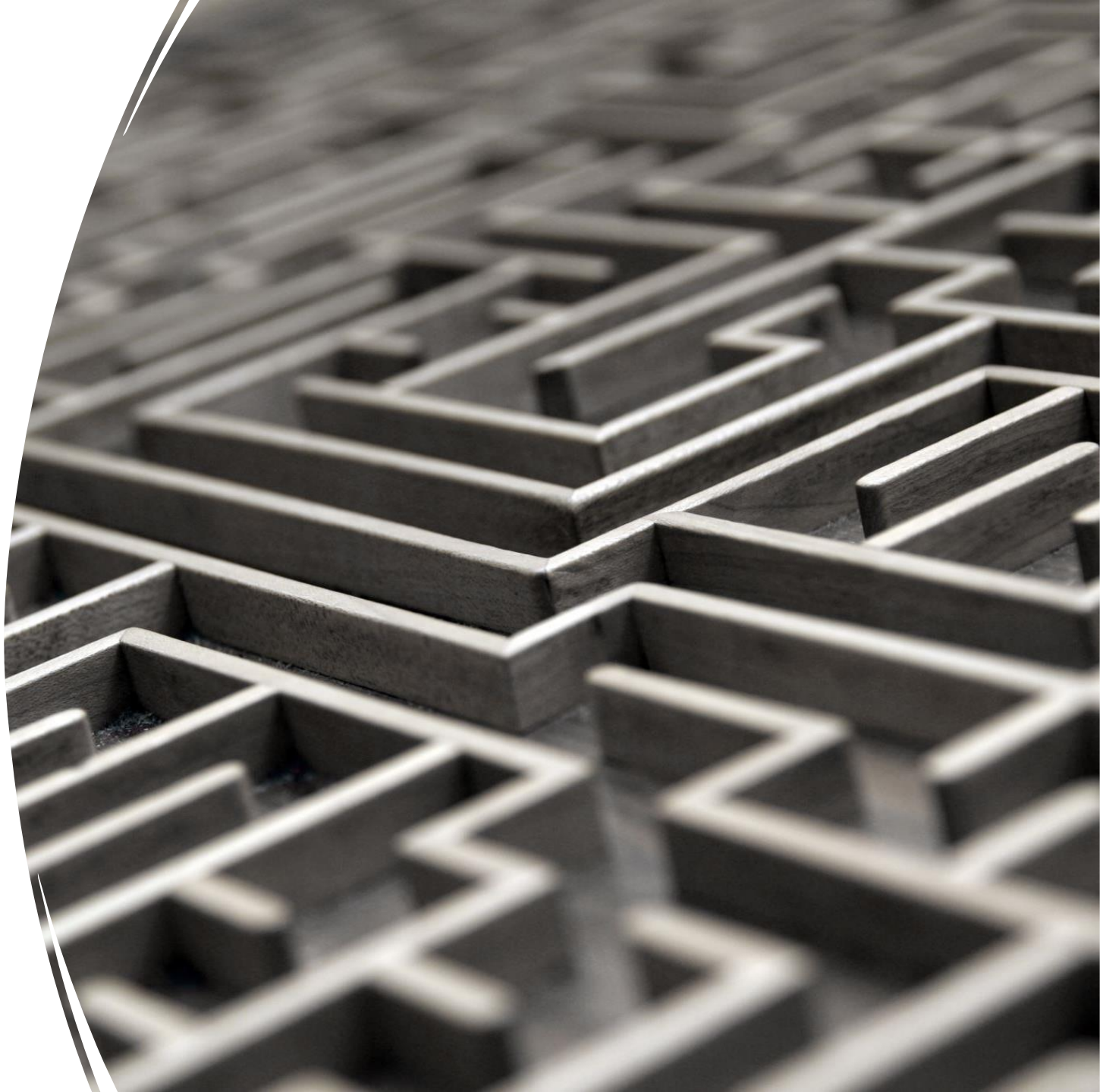


그림 8-23 전자서명을 만들고 검증하는 과정

그렇다면 전자서명은 안전한가?

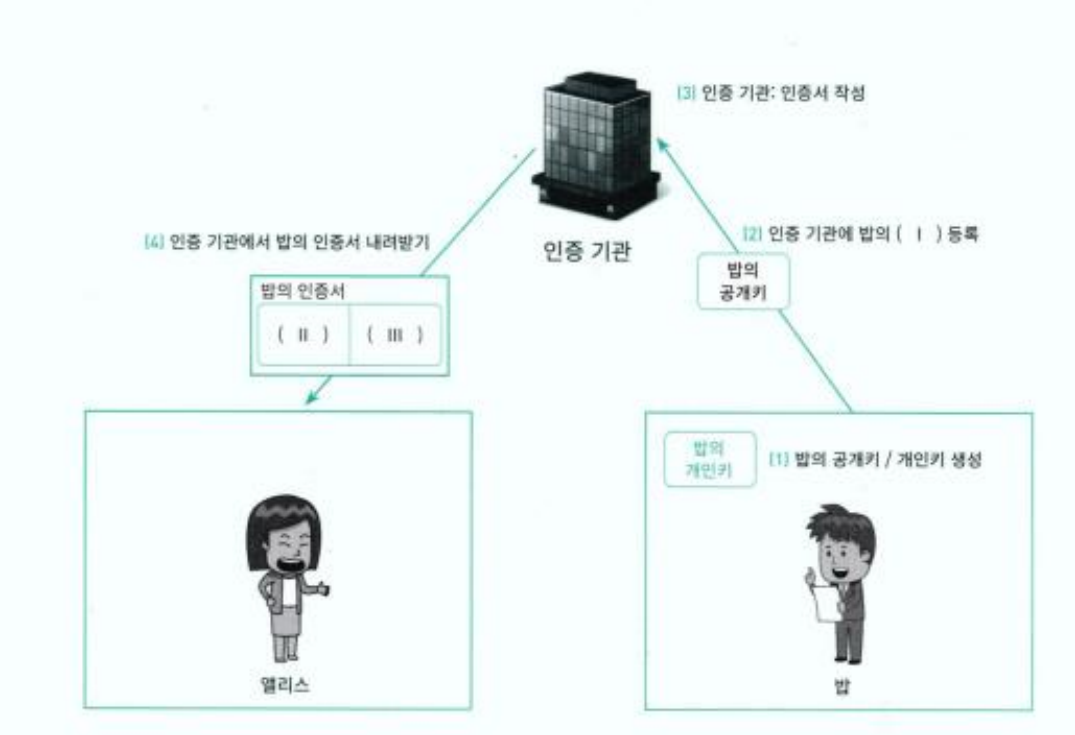
공개키 기반 구조의 한계

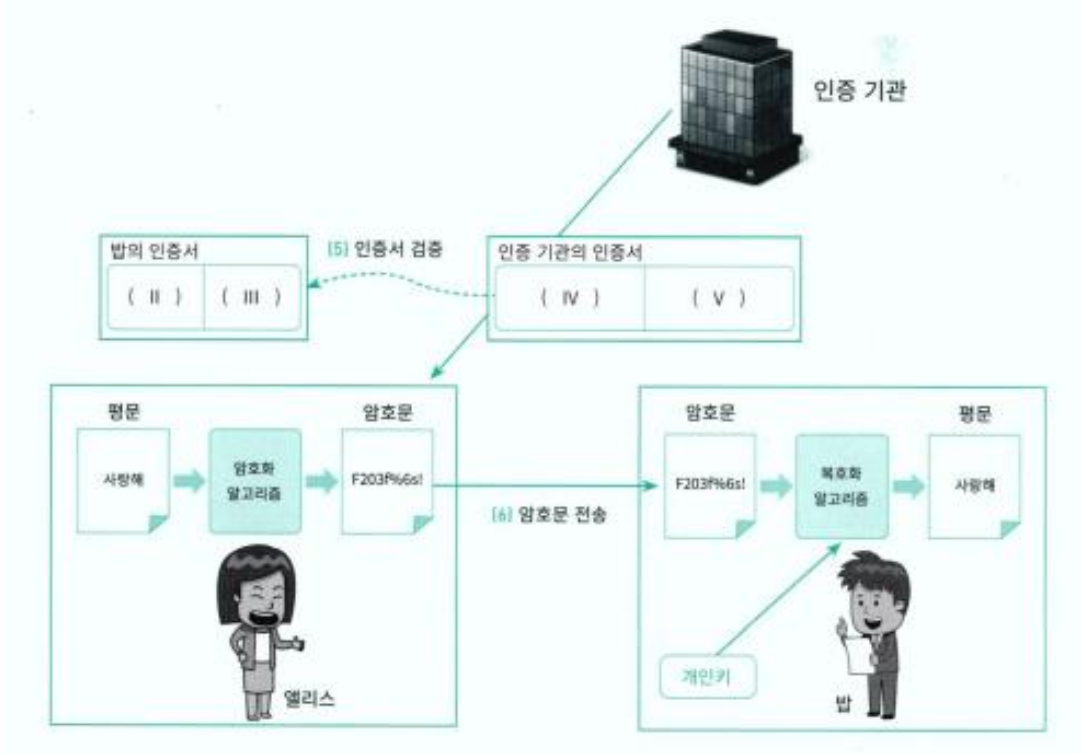


문제 1. 해시의 목적은?

1. 기밀성
2. 무결성
3. 김일성

빈칸채우기 – 인증서 검증 어떻게 할까요?





끝