

정보보호와 암호

컴퓨터공학과 장대희



경희대학교
KYUNG HEE UNIVERSITY



PWNJAB
@KYUNGHEE UNIVERSITY

정보보호와 암호의 관계

❖ 분석방해

- CNC 서버와의 통신시 암호화
- 분석당하기 싫은 데이터를 암호화
 - ✓ 인코딩과 암호화의 차이?

❖ 랜섬웨어

- 사용자의 파일을 암호화하여 인질로 잡음
- 비트코인 지불시 복호화 키 제공
 - ✓ 왜 비트코인으로 지불하는가?



모든 데이터는 숫자의 집합

❖ 모든 바이너리는 HEXDUMP 가능

- Text 문서 -> 바이너리 문서의 부분집합

| | | | |
|----------|-------------------------|-------------------------|------------------|
| 00000000 | 00 a2 00 01 03 00 00 00 | 00 73 77 72 63 6f 6d 6d |swrcomm |
| 00000010 | 61 6e 64 6f 00 73 77 72 | 63 6f 6d 6d 61 6e 64 6f | ando.swrcommando |
| 00000020 | 00 71 53 4e 41 54 57 2a | 33 00 5c 68 6f 73 74 6e | .qSNATW*3.\hostn |
| 00000030 | 61 6d 65 5c 68 6f 73 74 | 70 6f 72 74 5c 6e 75 6d | ame\hostport\num |
| 00000040 | 70 6c 61 79 65 72 73 5c | 6d 61 78 70 6c 61 79 65 | players\maxplaye |
| 00000050 | 72 73 5c 6d 61 70 6e 61 | 6d 65 5c 67 61 6d 65 74 | rs\mapname\gamet |
| 00000060 | 79 70 65 5c 66 72 61 67 | 6c 69 6d 69 74 5c 74 69 | ype\fraglimit\ti |
| 00000070 | 6d 65 6c 69 6d 69 74 5c | 6e 75 6d 74 65 61 6d 73 | melimit\numteams |
| 00000080 | 5c 64 65 64 69 63 61 74 | 65 64 73 65 72 76 65 72 | \dedicatedserver |
| 00000090 | 5c 66 72 69 65 6e 64 6c | 79 66 69 72 65 00 00 00 | \friendlyfire... |
| 000000a0 | 00 00 0a | | ... |
| 000000a3 | | | |
| Offset | hex | | ASCII |

암호의 종류

❖ 고전암호

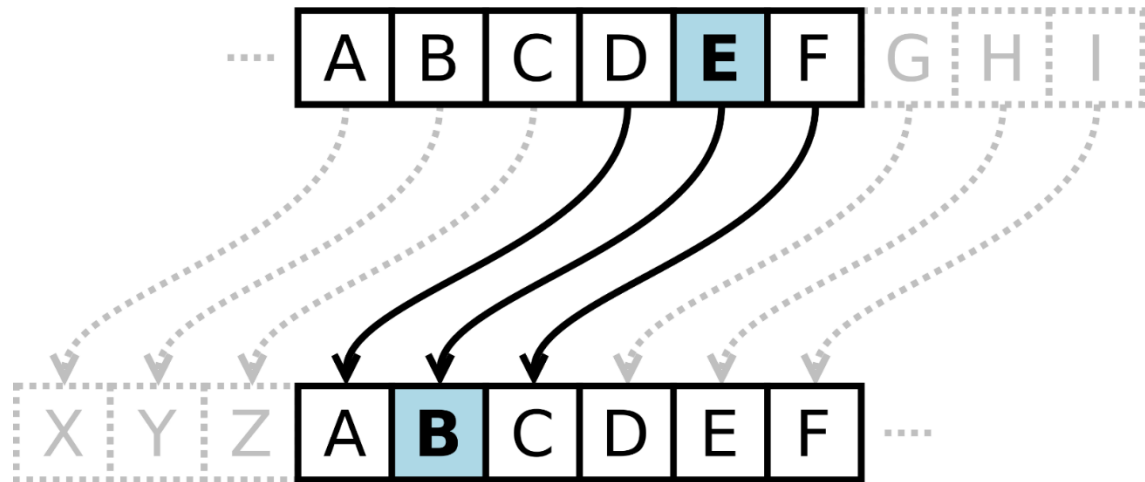
- Caesar Cipher
- Vignere

❖ 대칭암호

- AES

❖ 비대칭암호

- RSA



고전암호

| Plain | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|--------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cipher | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

The encryption can also be represented using modular arithmetic

$$E_n(x) = (x + n) \mod 26.$$

Decryption is performed similarly,

$$D_n(x) = (x - n) \mod 26.$$



고전암호 방식

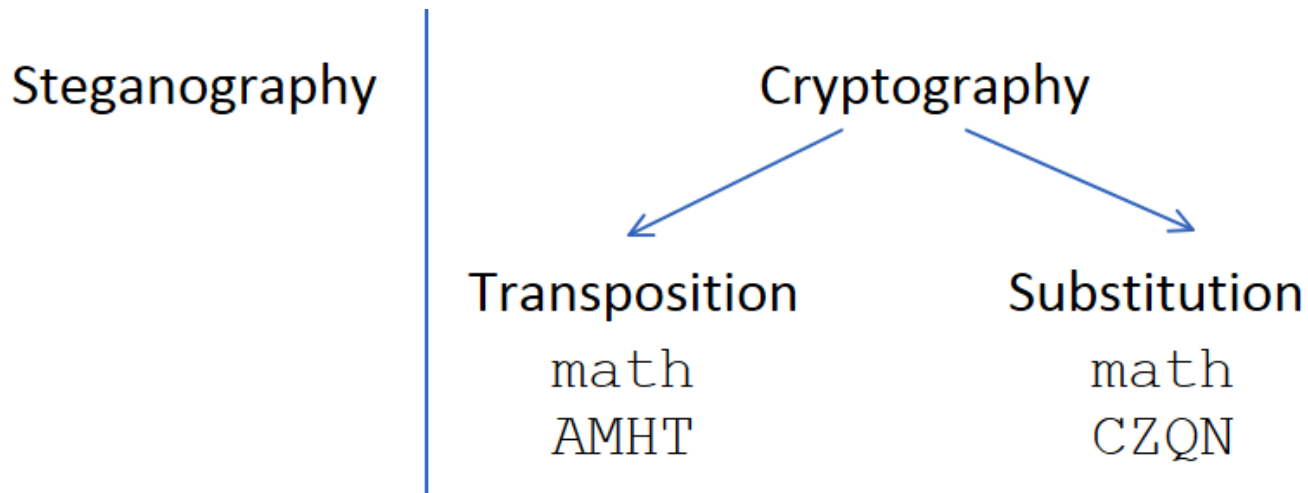
❖ Substitution Cipher (치환암호)

- 글자를 약속된 규칙에 따라 다른글자로 치환
 - ✓ 약속된 규칙? -> 보통 Table 로서 존재
 - ✓ Base64 인코딩?

❖ Transposition Cipher (전치암호)

- 글자들의 순서만 변경

❖ Steganography 와 암호화의 차이는?



Vignere (비그니어)

❖ 고전 암호 중 Key 를 사용하는 단순한 암호화

Plaintext: ATTACKATDAWN

Key: LEMONLEMONLE

Ciphertext: LXFOPVEFRNHR

Vigenère can also be described algebraically. If the

$$C_i = E_K(M_i) = (M_i + K_i) \bmod 26$$

and decryption D using the key K as

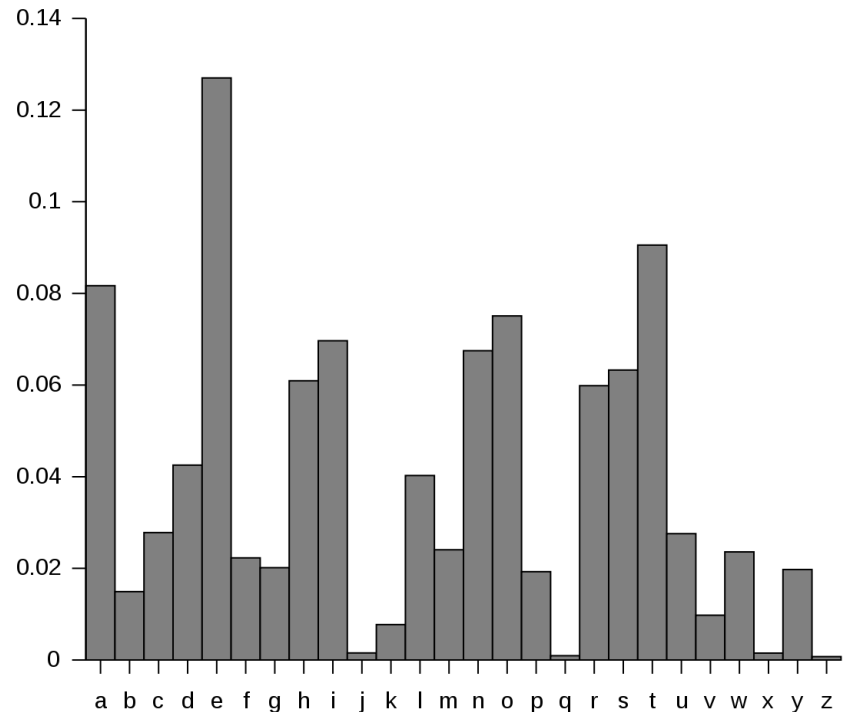
$$M_i = D_K(C_i) = (C_i - K_i + 26) \bmod 26,$$

고전암호의 취약성

❖ Frequency 분석

❖ 영문알파벳에는 Frequency 가 존재

- 가장 많이 사용되는 알파벳은? → E

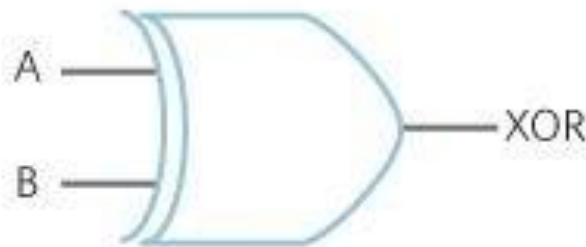


중요 연산: XOR

❖ XOR

- eXclusive OR
- 프로그래밍 언어에서 ^ 기호로 연산자 표기
 - ✓ $A = b \wedge c;$
- 수학적/전자회로 에서 아래와같이 표현

$$X = A \oplus B$$



| A | B | XOR |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

왜 XOR 이 중요한가?

❖ 인코딩

▪ 인코딩/암호화 의 기반

- L1. 교환법칙: $A \otimes B = B \otimes A$
- L2. 결합법칙: $(A \otimes B) \otimes C = A \otimes (B \otimes C)$
- L3. 항등원의 존재: 어떤 A 에 대하여서도, $A \otimes Z = A$ 가 되는 값 $Z = 0$ 이 존재한다.
- L4. 각 원소에 대해 유일한 역원의 존재: 각 A 에 대하여, $A \otimes A^{-1} = Z$ 가 되는 A^{-1} 가 존재한다.
 - L4a. 각 원소의 역원은 사실 자기 자신임: $A \otimes A = 0$

❖ 대표적 연산특성

- $A \oplus B \oplus B = A$
 - ✓ 한번 적용 \rightarrow 값이 변화함. 두번 적용 \rightarrow 원래대로 복원
- XOR Swap?

왜 XOR 이 중요한가?

❖ 덧셈 뺄셈도 마찬가지로 아닌가?

- $A + B - B = A$
- 값을 더함 \rightarrow 값이 변함
- 값을 뺌 \rightarrow 값이 원래대로 돌아옴

❖ 덧셈 뺄셈보다 왜 XOR 을 사용하는가?

- 덧셈 뺄셈의 문제점
 - ✓ Carry propagation ($4 + 7 = 11$, $11 - 7 = 4$)
 - ✓ 연산의 중간에 추가 bit 공간이 필요해짐
- XOR 은 비트단위의 연산이기때문에 Carry propagation 이 없음
 - ✓ $4 \wedge 7 = 3$, $3 \wedge 7 = 4$
 - ✓ 연산 내내 비트의 길이가 고정
 - ✓ CPU 입장에서 빠르게 처리 가능한 연산

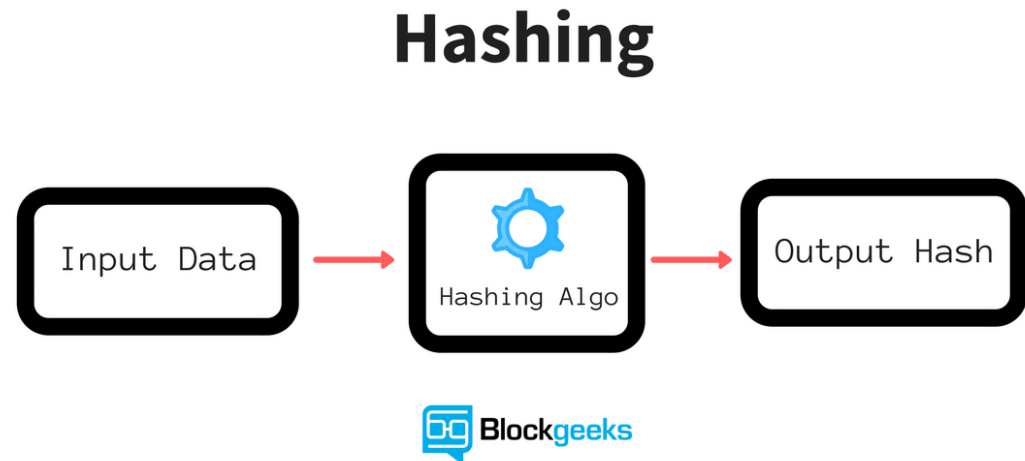
Hashing이란 무엇인가

❖알고리즘의 종류

- 가변길이 데이터를 고정길이 데이터로 변환
- N:1 단방향 함수

❖해싱의 종류

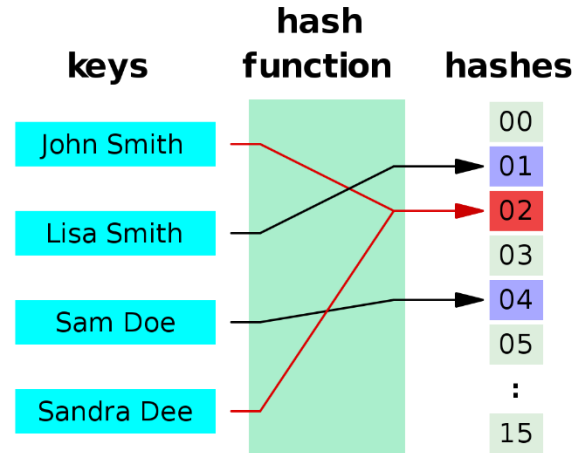
- CRC
- MD5, SHA1



Hashing이란 무엇인가

❖ 이론적인 예

- $H(x) = 1$
 - ✓ 해시 함수
- $H(x) = \text{length}(x) \% 10$
 - ✓ 해시 함수



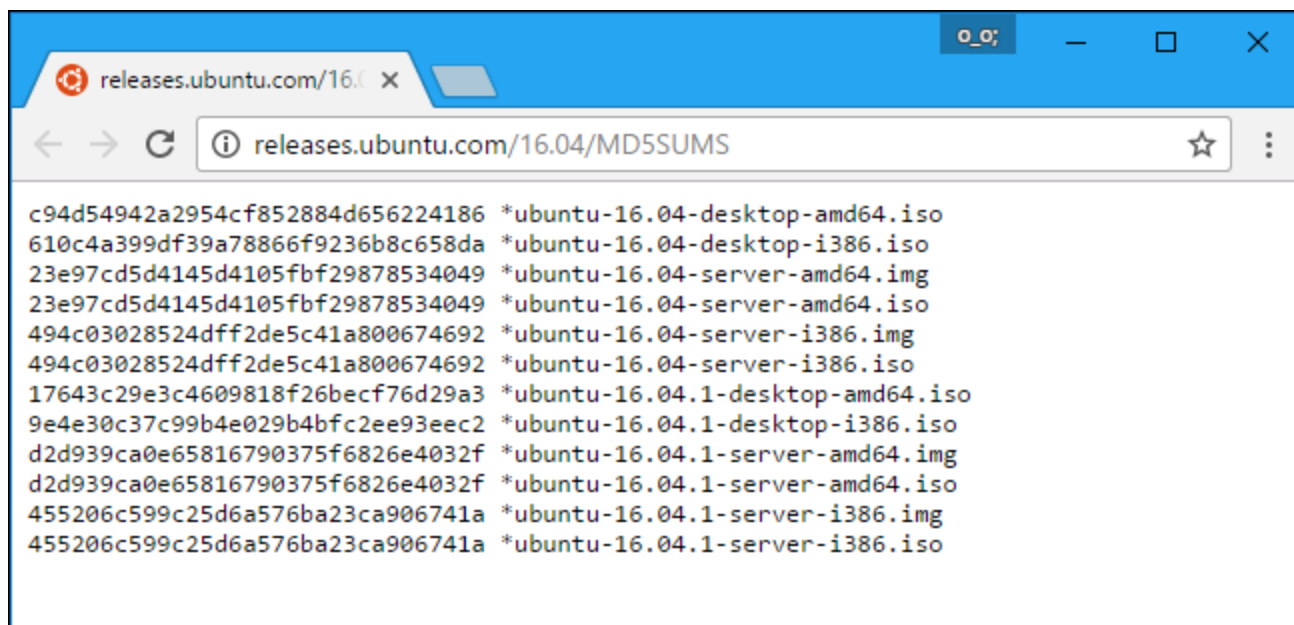
❖ 실제 예

- $\text{md5}('a') = 0cc175b9c0f1b6a831c399e269772661$
- $\text{md5}('abcde') = ab56b4d92b40713acc5af89985d4b786$
- $\text{md5}('abcdefghijklmnopqrstuvwxyz') = c3fcd3d76192e4007dfb496cca67e13b$
- $\text{md5}('ABCDEFGHIJKLMNOPQRSTUVWXYZ') = e751daaa1f79d23c10e3953f106ac1a9$

Hashing 의 특성 및 활용 (데이터 무결성)

❖ 고정길이 출력

- 데이터를 식별하기 위한 ID 로 활용
 - ✓ Collision 이슈
- 파일의 무결성 검사



The screenshot shows a web browser window with the address bar displaying "releases.ubuntu.com/16.04/MD5SUMS". The page content lists MD5 hashes for various Ubuntu 16.04 release images, including desktop and server versions for amd64 and i386 architectures. The hashes are listed in two columns, with the file names on the right.

```
c94d54942a2954cf852884d656224186 *ubuntu-16.04-desktop-amd64.iso
610c4a399df39a78866f9236b8c658da *ubuntu-16.04-desktop-i386.iso
23e97cd5d4145d4105fbf29878534049 *ubuntu-16.04-server-amd64.img
23e97cd5d4145d4105fbf29878534049 *ubuntu-16.04-server-amd64.iso
494c03028524dff2de5c41a800674692 *ubuntu-16.04-server-i386.img
494c03028524dff2de5c41a800674692 *ubuntu-16.04-server-i386.iso
17643c29e3c4609818f26becf76d29a3 *ubuntu-16.04.1-desktop-amd64.iso
9e4e30c37c99b4e029b4bfc2ee93eec2 *ubuntu-16.04.1-desktop-i386.iso
d2d939ca0e65816790375f6826e4032f *ubuntu-16.04.1-server-amd64.img
d2d939ca0e65816790375f6826e4032f *ubuntu-16.04.1-server-amd64.iso
455206c599c25d6a576ba23ca906741a *ubuntu-16.04.1-server-i386.img
455206c599c25d6a576ba23ca906741a *ubuntu-16.04.1-server-i386.iso
```

좋은 Hashing 의 조건

❖ Collision 이 없어야 함

- 입력이 1비트라도 다른경우 출력이 크게 달라짐
 - ✓ input 과 output 사이의 의존성이 없어야함
- 파일이 깨진경우 (1비트 오류) 탐지 가능

❖ 빠른 계산속도

- 성능 이슈

Hash 크랙

❖역함수가 가능한가?

- 해시값으로부터, 입력을 계산 할 수 있는가?

```
root@et:~/hashcat# ./hashcat -m 15100 -a 3 -w 3 hash.txt ?1?1?1?1?1?1t
hashcat (v3.5.0) starting...

OpenCL Platform #1: NVIDIA Corporation
=====
* Device #1: GeForce GTX 1080, 2026/8107 MB allocatable, 20MCU
* Device #2: GeForce GTX 1080, 2028/8114 MB allocatable, 20MCU
* Device #3: GeForce GTX 1080, 2028/8114 MB allocatable, 20MCU
* Device #4: GeForce GTX 1080, 2028/8114 MB allocatable, 20MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Applicable optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force

Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 75c

$sha1$20000$46487265$5EZZAHIEWP.Hwufke.kRNWOA.sg1:hashcat

Session.....: hashcat
Status.....: Cracked
Hash.Type.....: Juniper/NetBSD sha1crypt
Hash.Target.....: $sha1$20000$46487265$5EZZAHIEWP.Hwufke.kRNWOA.sg1
Time.Started....: Wed Apr 5 11:37:22 2017 (2 mins, 8 secs)
Time.Estimated...: Wed Apr 5 11:39:30 2017 (0 secs)
Guess.Mask.....: ?1?1?1?1?1?1t [7]
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 184.8 kH/s (91.44ms)
Speed.Dev.#2.....: 184.9 kH/s (91.24ms)
Speed.Dev.#3.....: 184.8 kH/s (91.30ms)
Speed.Dev.#4.....: 184.8 kH/s (91.53ms)
Speed.Dev.*.....: 739.3 kH/s
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) salts
Progress.....: 93716480/308915776 (30.34%)
Rejected.....: 0/93716480 (0.00%)
Restore.Point....: 0/11881376 (0.00%)
Candidates.#1....: harinat -> hmldcot
Candidates.#2....: waktltt -> wltxs1t
Candidates.#3....: wkjytrt -> wqcdudt
Candidates.#4....: wswsbut -> wwpfsit
HwMon.Dev.#1.....: Temp: 53c Fan:100% Util:100% Core:1936MHz Mem:4513MHz Bus:1
HwMon.Dev.#2.....: Temp: 59c Fan:100% Util:100% Core:1949MHz Mem:4513MHz Bus:1
HwMon.Dev.#3.....: Temp: 54c Fan:100% Util:100% Core:1949MHz Mem:4513MHz Bus:1
HwMon.Dev.#4.....: Temp: 55c Fan:100% Util:100% Core:1936MHz Mem:4513MHz Bus:1

Started: Wed Apr 5 11:37:17 2017
Stopped: Wed Apr 5 11:39:30 2017
```


Hashing 의 특성 및 활용 (데이터 보안)

❖ 단방향성

- 보안의 목적으로 활용
 - ✓ 암호 저장

| id | username | password | passwordHint |
|----|-----------|----------------------------------|------------------------|
| 1 | admin | 7E7274BAC45E467C5AB832170F12E418 | k3wl dud |
| 2 | pumpkin22 | 5377DBF76D995CC213ED76924A31CB13 | my favorite holiday |
| 3 | johndoe | 512239D9AE0C3B5567DE188739F689F2 | Freddie Mercury's band |
| 4 | alexa45 | 2FE5421E49061F8225C2FB7CB81980FD | password |
| 5 | guy | ABE35E2827DDA834C9612FE9E9C92CE0 | NULL |
| 6 | maryjane | 198670893B2781C83F3DA5D45150123D | I'm one! |
| 7 | dudson123 | 59E2113217E65B9885F9DA73FDC5697B | scary movie! |

좋은 Hashing 의 조건

❖역변환이 불가능해야함

- abcde 를 md5 hashing 하면? → ab56b4d92b40713acc5af89985d4b786
 - ✓ 누구나 쉽고 빠르게 알수있음
- ab56b4d92b40713acc5af89985d4b786 를 만드는 md5 의 입력은?
 - ✓ 계산이 (현실적으로) 불가능

❖추가적 조건들

- Avalanche Effect
- Collision Resistance
- Pre-image Resistance

❖해싱에 대한 공격

- 1. 브루트포스 (크랙)
- 2. 암호학적 공격

좋은 Hashing 의 조건

❖ Avalanche Effect

- An important and desirable feature of a good hash function is the non-correlation of input and output, or so-called "avalanche effect", which means that **a small change in the input results in a significant change in the output, making it statistically indistinguishable from random**

❖ Collision Resistance

- Collision resistance is **the property of a hash function that it is computationally infeasible to find two colliding inputs.**

❖ Preimage resistance

- Preimage resistance is **the property of a hash function that it is hard to invert**, that is, given an element in the range of a hash function, it should be computationally infeasible to find an input that maps to that element.

Hashing 과 보안

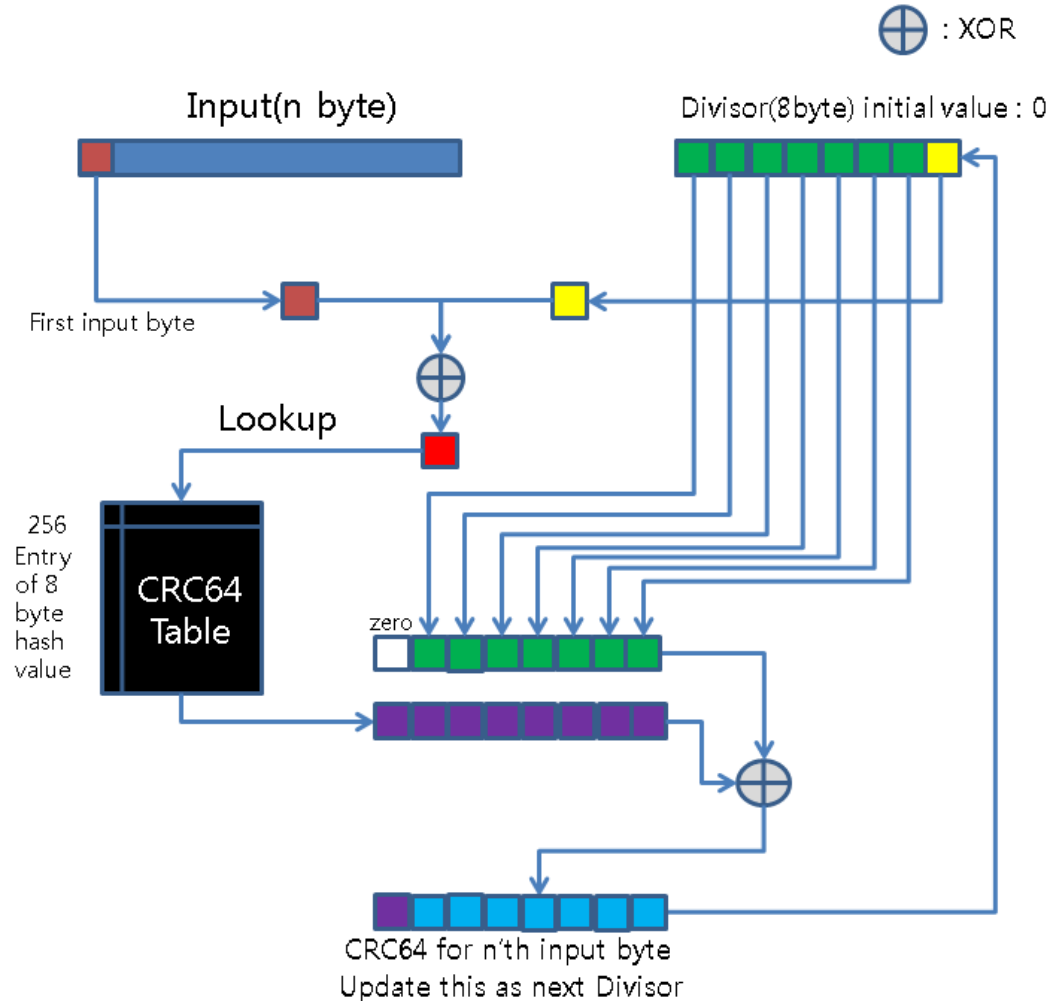
❖비-암호학적 해시함수들

- 보안을 고려하지 않고 디자인 (성능위주로만)
 - ✓ 대표적인 예: CRC

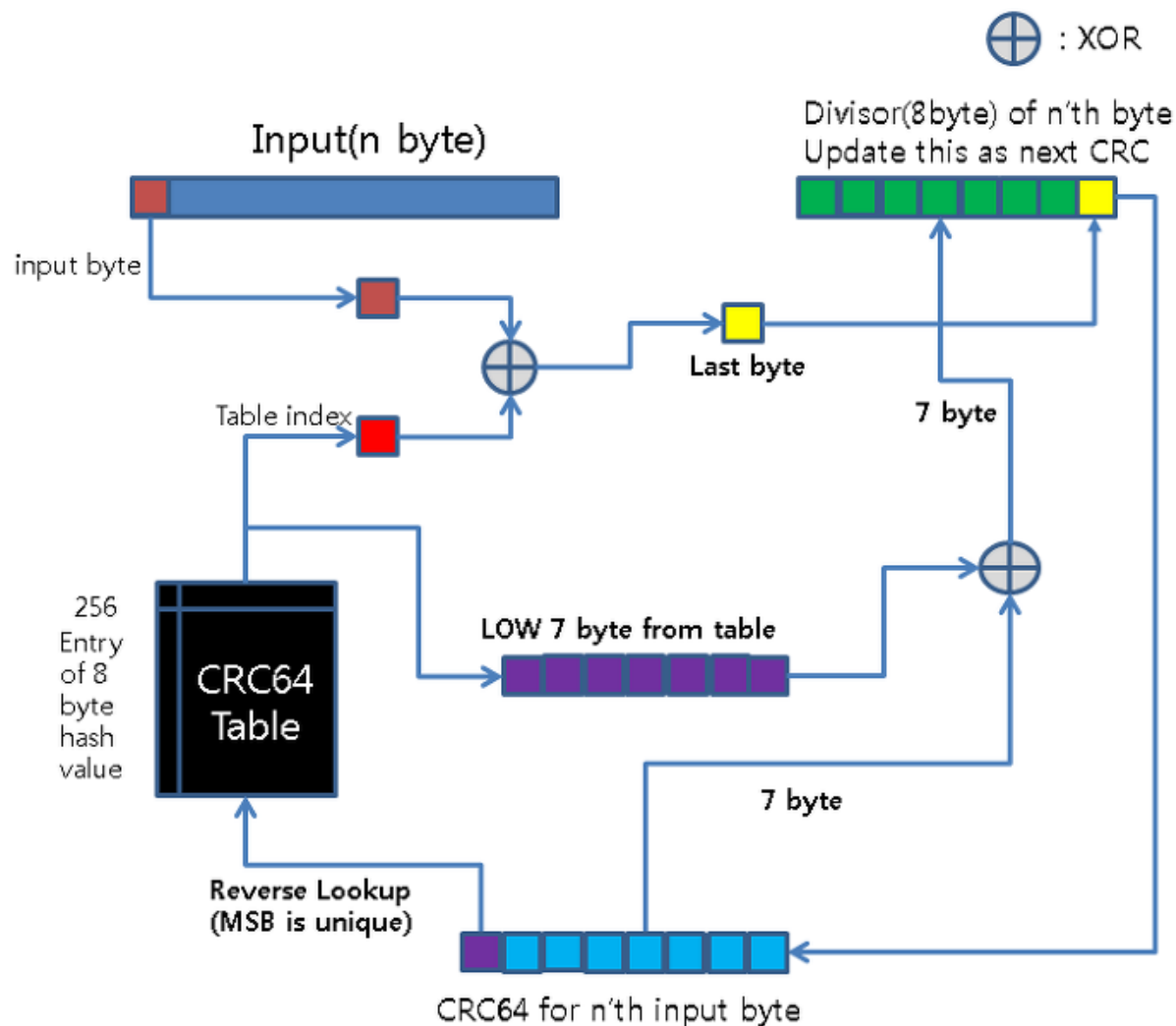
❖암호학적 해시함수들

- 보안을 고려하여 디자인
 - ✓ MD5, SHA1 (현재는 broken)
 - ✓ SHA256, SHA512, ECC

CRC 해시 알고리즘



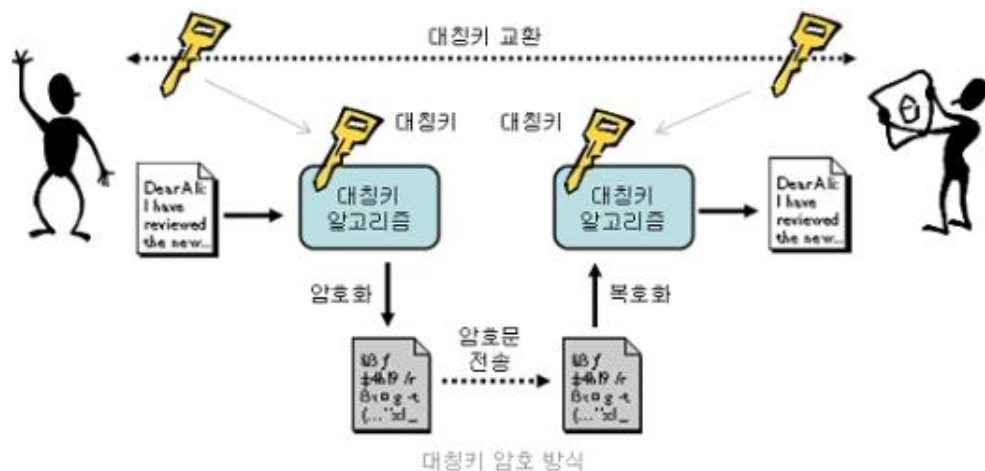
CRC 해시 알고리즘 리버싱



대칭암호 vs 비대칭암호

❖ Key의 사용형태에 따라 구분

- 대칭: 암호화/복호화 시 동일한 키 사용
 - ✓ 연산 속도가 빠름
- 비대칭: 암호화/복호화 시 서로 다른 키를 사용
 - ✓ 연산 속도가 느림
 - ✓ 키가 2개 이므로 관리가 복잡해짐



AES

❖ Advanced Encryption Standard

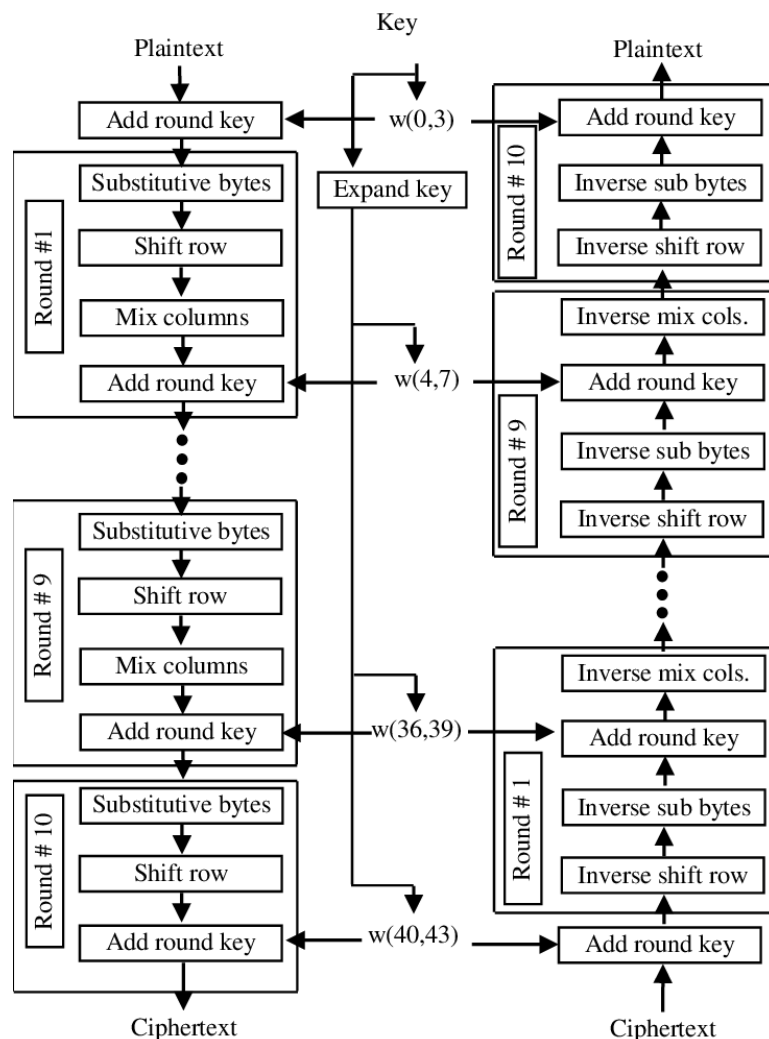
- 대표적인 대칭암호

❖ 핵심요소

- 키
- IV (초기화 벡터)
- 블록사이즈
- 모드

❖ 키 길이

- 128bit = 16 byte
- 192bit = 24 byte
- 256bit = 32 byte
- ...



키, IV, 블록 사이즈 관계

❖ In AES, the key size and the block size are not always the same, while the **initial vector (IV) size is always the same as the block size**.

- Key size
 - ✓ AES supports key sizes of 128, 192, and 256 bits. These different key sizes lead to AES-128, AES-192, and AES-256 variants
- Block size
 - ✓ AES operates on fixed block sizes of 128 bits. This is true regardless of the key size chosen (128, 192, or 256 bits).
- Initial Vector (IV) size
 - ✓ The IV size in AES is always equal to the block size
 - ✓ The IV is used in various **chaining modes (such as CBC or CFB)** to ensure that encrypting the same plaintext with the same key produces different ciphertexts, thus increasing security.

❖ Key size in AES can vary (128, 192, or 256 bits), while the block size and IV size are always fixed at 128 bits.

- AES256 의 경우도 IV/Block 은 128비트 고정.

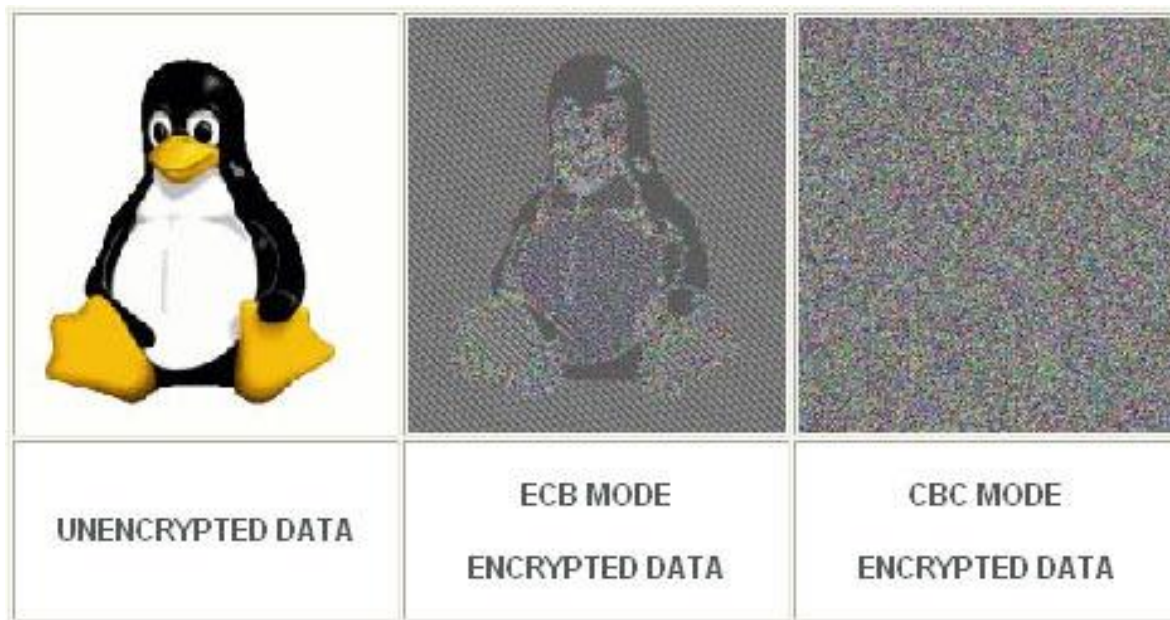
대칭암호의 모드

❖ ECB

- Electric Code Book

❖ CBC

- Cipher Block Chaining



RSA

❖ Rivest-Shamir-Adleman

- 대표적인 비대칭암호



Practical implementations use the Chinese remainder theorem to speed

The values d_p , d_q and q_{inv} , which are part of the private key are computed

$$d_p = d \bmod (p - 1) = 413 \bmod (61 - 1) = 53$$

$$d_q = d \bmod (q - 1) = 413 \bmod (53 - 1) = 49$$

$$q_{inv} = q^{-1} \bmod p = 53^{-1} \bmod 61 = 38$$

$$\Rightarrow (q_{inv} \times q) \bmod p = 38 \times 53 \bmod 61 = 1$$

Here is how d_p , d_q and q_{inv} are used for efficient decryption. (Encryption

$$m_1 = c^{d_p} \bmod p = 2790^{53} \bmod 61 = 4$$

$$m_2 = c^{d_q} \bmod q = 2790^{49} \bmod 53 = 12$$

$$h = (q_{inv} \times (m_1 - m_2)) \bmod p = (38 \times -8) \bmod 61 = 1$$

$$m = m_2 + h \times q = 12 + 1 \times 53 = 65$$

핵심원리요약: 큰수의 소인수분해가 어렵다

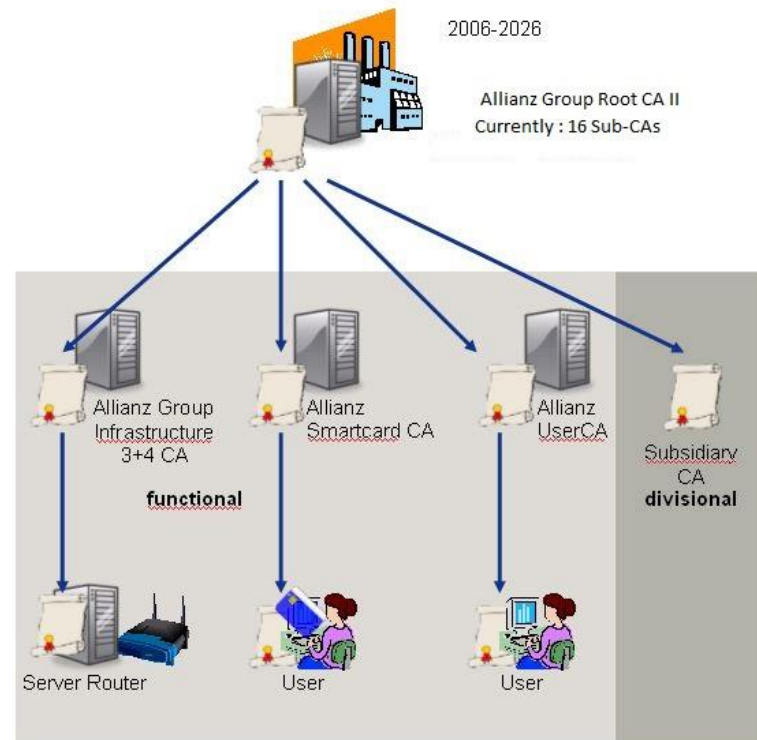
비대칭키 활용예시

❖키교환

- 통신 감청 하에서도 비밀리에 대칭키 공유가 가능

❖전자서명 (공인인증서 등)

- 공개키에 대한 소유권 증명
 - ✓ 공인인증서
 - ✓ CA: Certificate Authority
 - ✓ Root CA?



키 교환

❖ 디피헬만 (Diffie-Hellman)

- The Diffie-Hellman key exchange algorithm is a cryptographic method that allows two parties to establish a shared secret key over an insecure communication channel with following steps (alice/bob communicating).
 - ✓ 1. Agreement on public parameters: 큰 소수 P , 그보다 작은 정수 G 선정 (공개).
 - ✓ 2. Generate private keys: P 보다 작은 정수 a, b (랜덤) 을 각자 선정
 - ✓ 3. Calculate public keys: P, G , 각자의 비밀을 통해 공개키 A, B 계산
 - $A = g^a \text{ mod } p$
 - $B = g^b \text{ mod } p$
 - ✓ 4. Exchange public keys: A 와 B 를 서로 공개교환 (a, b 아님)
 - ✓ 5. Calculate shared secret key: 공유되는 비밀키 계산
 - $K = B^a \text{ mod } p$ (Alice 의 경우)
 - $K = A^b \text{ mod } p$ (Bob 의 경우)
- The security of the Diffie-Hellman key exchange relies on the difficulty of solving the discrete logarithm problem

키 교환 – 디피헬만 (Diffie-Hellman)

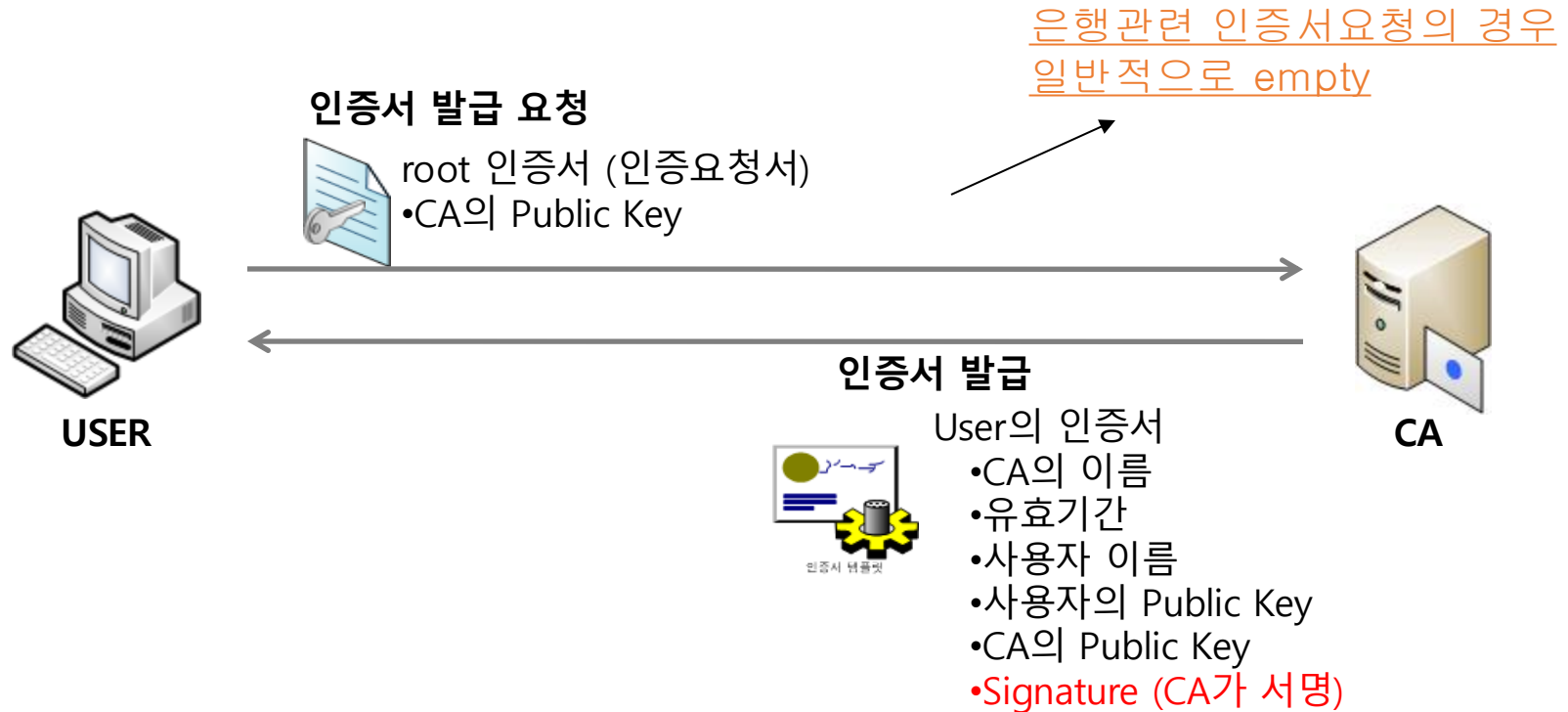
❖ RSA 과 디피헬만의 차이

▪ Key exchange approach:

- ✓ RSA: The key exchange process using RSA typically involves one party (usually the server) generating a public-private key pair. The public key is shared with the other party (the client), who uses it to encrypt a session key. The encrypted session key is then sent back to the server, which decrypts it using its private key. Both parties now have a shared secret session key to encrypt and decrypt messages. (주로 신원인증)
- ✓ Diffie-Hellman: In Diffie-Hellman, both parties actively participate in the key exchange process by generating private keys and public keys, then exchanging public keys. They each independently compute the shared secret key using their own private key and the other party's public key. No encryption or decryption of the session key is required. (주로 키교환용)

❖ RSA is often used for authentication and digital signatures, while Diffie-Hellman is used for key exchange.

공인인증서 발급

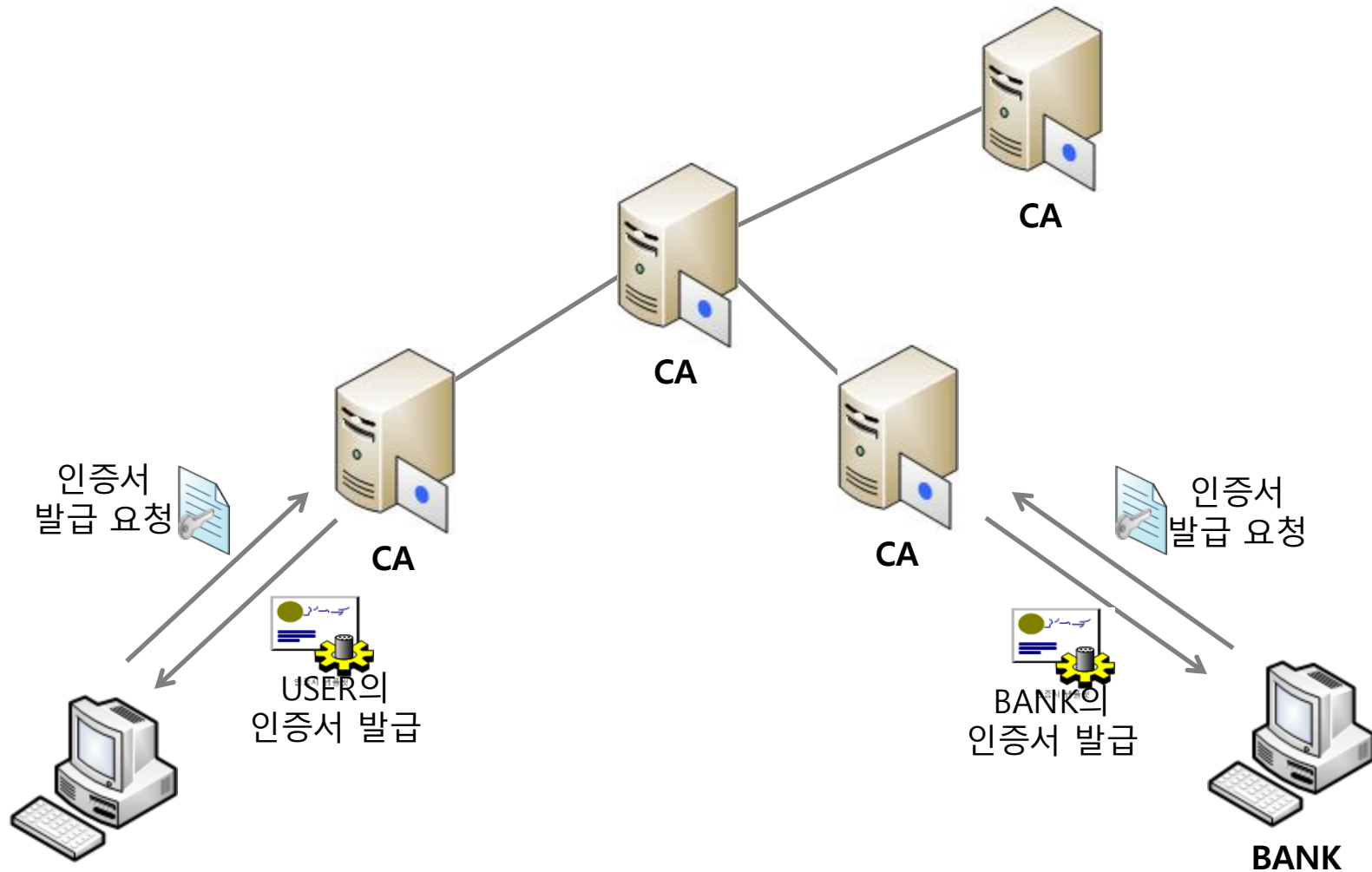


• 인증서의 Signature

Signature를 제외한 모든 필드로 digest를 만들고, 이 digest를 CA의 Private Key로 암호화함.

일종의 전자서명으로 인증서의 내용이 사실임을 CA가 공증함.

CA 계층구조



전자서명

❖전자서명의 활용 예시

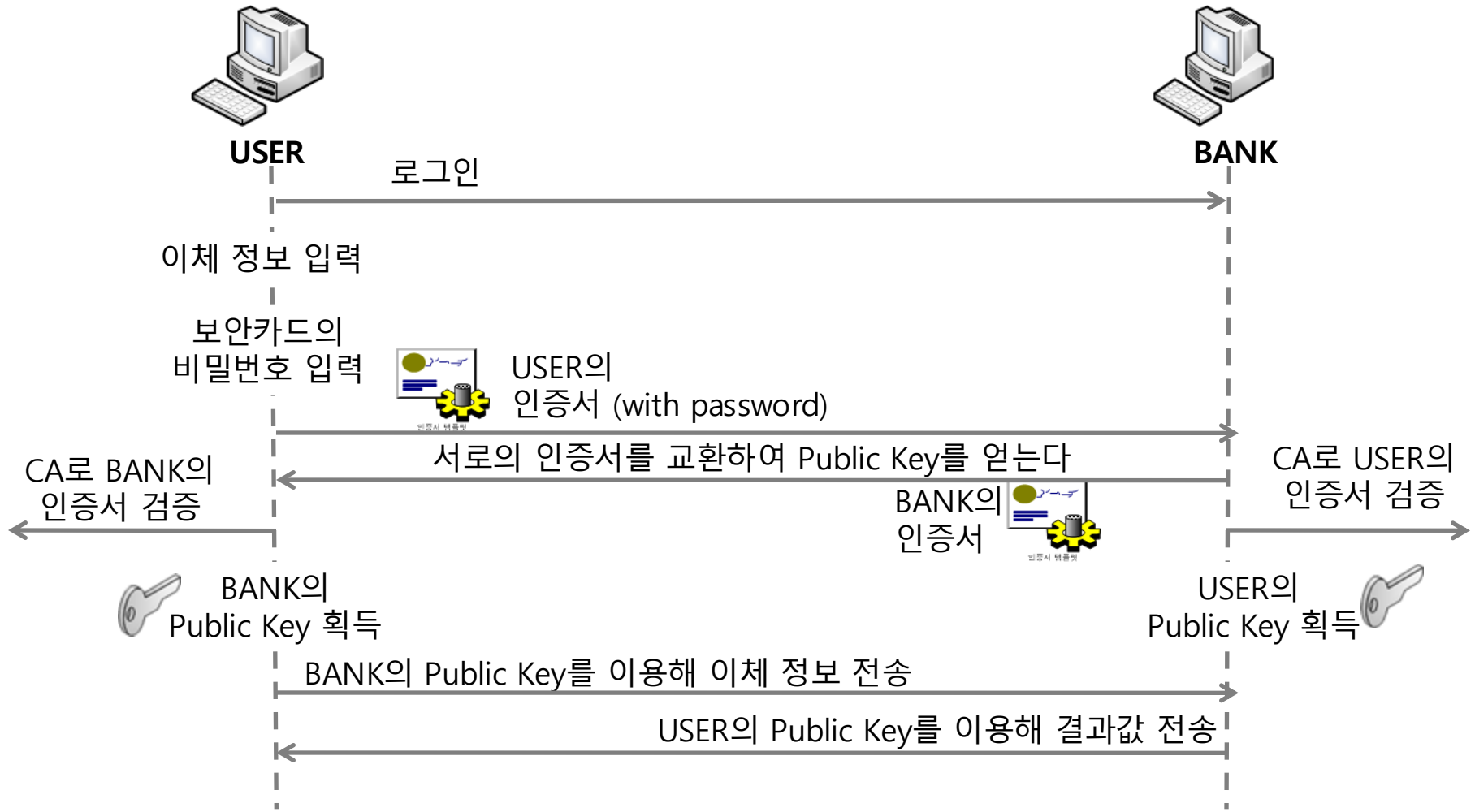
1. 영희는 셀카를 촬영 후 해당 사진의 인물이 자신임을 인터넷에 알리고자 함
2. 그러나 인터넷의 **다른 사람이** 자신의 사진을 합성하여 생김새를 **변조하고 재 배포** 하는 것이 우려됨
3. 영희는 셀카 사진파일의 해시값을 자신의 비밀키로 암호화하고, 해당 암호+공개키를 사진과 함께 배포.
4. 증명을 하고싶은 사람은 영희의 공개키로 해시값을 해독한 뒤, 사진에 대한 해시를 계산하여 변조 유무 확인.
5. 영희의 공개키로 올바른 해시값이 복호화 되었다는것은 사진 배포자가 영희의 비밀키를 가지고 있다는 증거.
6. 영희의 비밀키는 영희만 가지고있으므로, 사진 배포자가 영희임이 증명됨.

전자서명

❖전자서명의 활용 예시

1. 철수가 이의를 제기함: 영희의 사진과 함께 배포된 공개키가 영희의 것임을 어떻게 믿을 수 있나?
2. 영희는 자신의 공개키가 실제로 자신의 것임을 주장하는 문서를 작성하여 제 3기관에 제출.
3. 제3의 기관이 해당 문서를 검토하고 사실임을 확인함.
4. 제3기관에서 해당 문서의 해시값을 기관의 비밀키로 암호화하고, 기관의 공개키와함께 동봉 (공인인증서)
5. 영희는 제3기관으로부터 공인인증서를 받고, 사진+사진의암호화된해시+공인인증서 를 배포
6. 철수는 제3기관의 공개키로 영희의 주장이 담긴 문서의 해시값이 올바르게 복호화 되는지 검증
7. 철수가 제3기관의 공개키를 신뢰한다면, 이 시점에서 영희의 공개키도 함께 믿을수있음.
8. 제3기관 (CA) 의 주장을 어떻게 신뢰 할 것인가? -> 제3기관들의 공개키를 증명하는 또다른 기관을 신뢰
9. CA 들을 인증해주는 상위 CA 신뢰의 반복 -> Root CA -> 최 상위 신뢰 기관

SSL 암호화 통신 과정



암호화 통신

❖일반적으로 아래와 같이 진행

- 서버가 클라이언트로 서버의 공인인증서를 전달
- 서버의 RSA 공개키를 통해 서버의 진위 인증 (피싱방지)
- 이후 디피헬만으로 대칭키 공유
- HTTP 데이터의 암호화/복호화 통신은 대칭암호이용
 - ✓ 이유: 비대칭암호화는 성능이 느림

❖대표적인 암호화 라이브러리

- OpenSSL
- Key 의 글자들?
 - ✓ Base64 인코딩된 모습

```
-rw-rw-r-- 1 joshua joshua 3311 Dec 20 16:14 private.pem
-rw-rw-r-- 1 joshua joshua 800 Dec 20 16:14 public.pem
joshua@blackbox:~/nxp_tests/OpenSSL_Generated_Keys$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAzeEbLhHzbVDL7mIfat6n
Wp29omyxSdIFsb3RGkLmCbaFcsoU+f7ld50HPTc3a9jSGYBEURhTBJHeUm6aRnAW
BIep3WOTK4m6IS7lAdt7Bj2Mb6GvkqeIXHCgUsN+GhUvdTLB01JGGbAuGibpWvVT
WCcPyFEXyYla8VJLzJlnJfcOGT7mKovpn27uwWbbTTPdDQsMyImZZP29RDZ+IvDF
90S/qrkWW00ip+L3XTs+HWB5yQ3iXL3KLWkFtfb0SBwmtHsLQrzunS328G0gwwNn
6NDSbNW5sJTkWCQsslW520Bo2eZILnV8bVLnph9lTPg6aUnkn7w1sCDE5lx6di
/dsH4e70VPXxq838C1EaFPCAvu7ErTqqmUxPgU44+usz55/uMsLU00bg8vveXcWG
jYYxTZZfLV9fTf4ENxAwWcMl8/qeDP3ZcnTd2JQCCzqzJGW+Cf9G9LTKHIitfdx6
PY7EvUCvjdfJJ1Sbo/kd2xroFSTW140LYWQZ9HoL80p94K2qSyS7qOK7ATgttqa0
0QeuRI0sexPyqNT++4cU/HYR8NUqUKAYQe2LoIzSbQfAyB8hg1B0GDNNfXRXsxp
XDK/cmyhAs0i83eyMrmeCZNCiV9IPdphcZptdu9hT3l0gFcpIDatsJWR+72xCM2
0mbG8JrupVUL2TLWIyP6iAMCAwEAAQ==
-----END PUBLIC KEY-----
```

HTTPS 의 작동과정과 순서

How does HTTPS work: SSL explained

This presumes that SSL has already been issued by SSL issuing authority.



