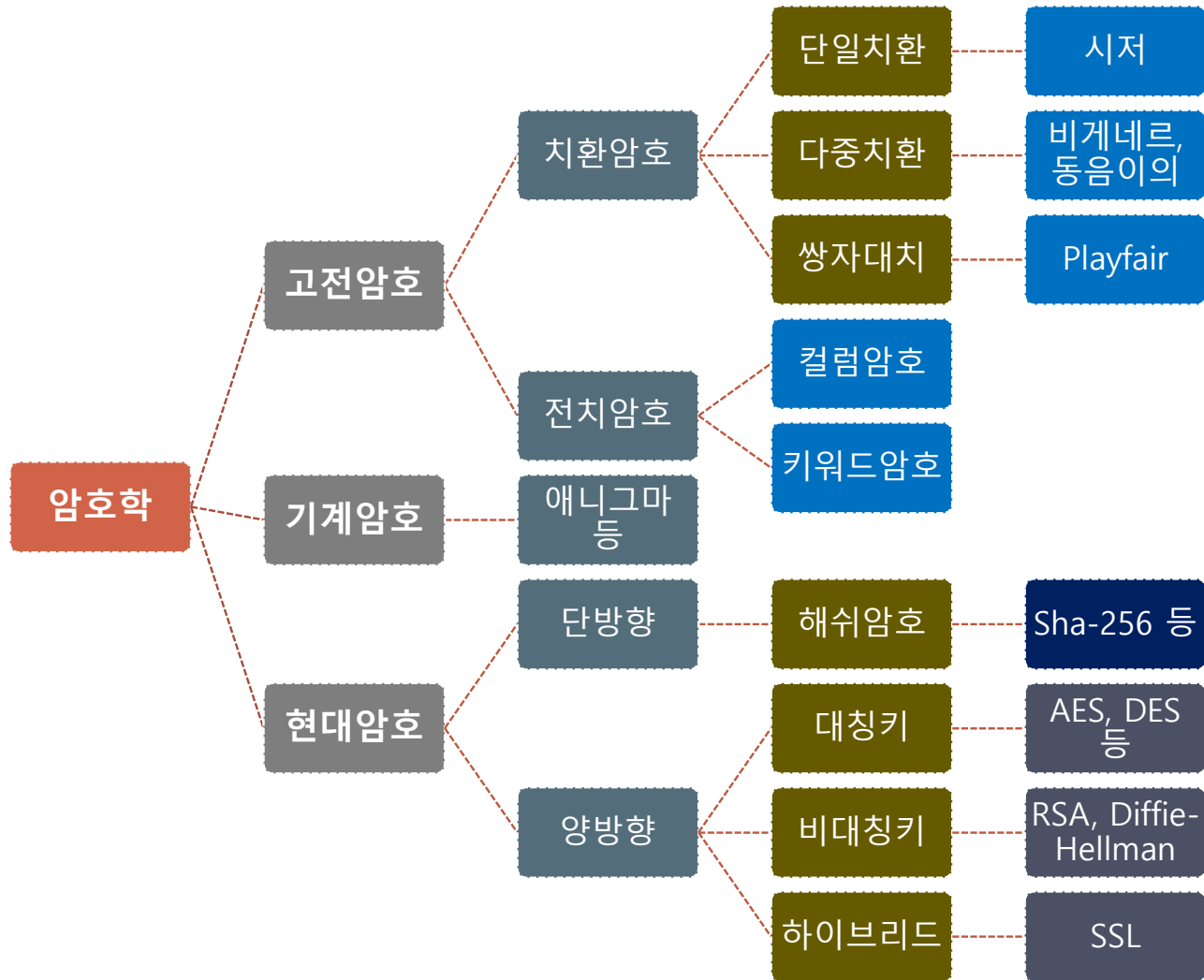


암 호 학 II

-현대 암호(블록암호, 비대칭키)-



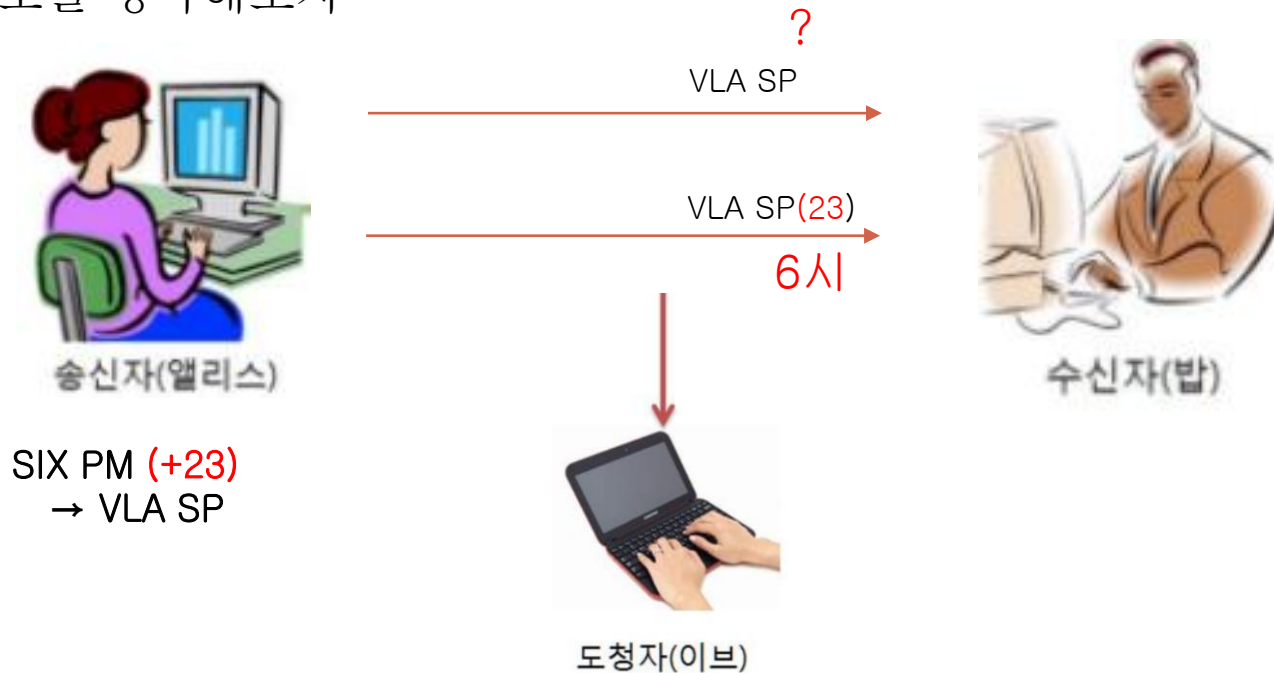
암호학의 분류



키배송의 중요성



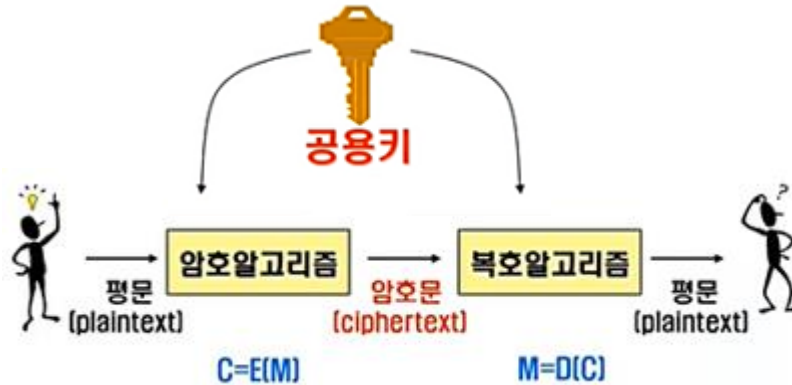
- 시저암호를 생각해보자



- 키값을 안전하게 전달할 수 있는 방법이 있을까?

대칭키(비밀키) 암호화의 단점

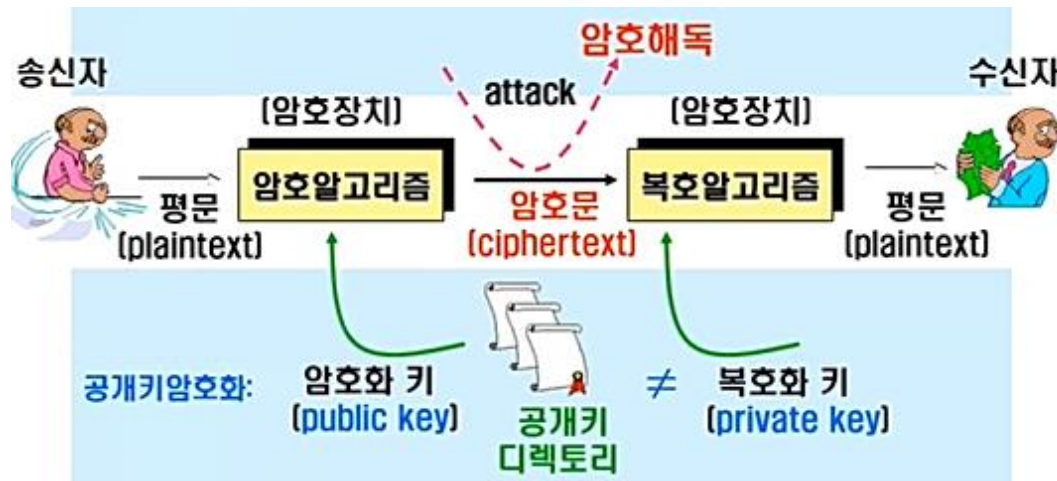
- 대칭키 암호화, 비밀키 암호화, 단일키 암호화



- 장점
 - 다양한 알고리즘
 - 알고리즘 수행속도가 빠름
- 단점
 - 키 배송이 어려움
 - 서로 같은 키를 가지고 있어야 하므로 n 명의 상대가 있는 경우 n 개의 비밀키가 있어야 함.
 - 만약 여러 상대방에게 같은 키를 사용한다면 그들은 서로의 메시지를 읽을 수 있게 됨
 - 부인방지를 막을 수 없음
 - ✦ 송신자와 수신자를 증명할 수 있는 인증을 할 수 없음.
 - ✦ A와 B가 같은 키를 가지고 있을 때 그 두 사람이 메시지를 만들고 암호화한 다음, 서로 상대방이 그 메시지를 보냈다고 주장할 수 있음.

공개키(비대칭키) 암호 시스템

- 개념
 - 1976년 디피(Diffie)와 헬만(Hellman)에 의해 제안된 공개키 암호기법의 개념
 - 키에 관한 정보를 공개함으로써 키 관리의 어려움을 해결하고자 하는 방식
- 공개키와 비공개키 (비대칭 키 시스템)
 - 공개키(public key)
 - 암호화할 때 사용하는 키 (누구에게나 공개되어 있음)
 - 비공개키(private key)
 - 복호화할 때 사용하는 키



디피-헬만 키 교환 알고리즘(Diffie-Hellman key exchange)

- 예제

A는 임의의 큰 수 11를 선택 $2^{11} \bmod 17 = 8$

B는 임의의 큰 수 13를 선택 $2^{13} \bmod 17 = 15$ (2와 17은 서로 약속; 도청되어도 무방)

A는 B에게 8을 보낸다.

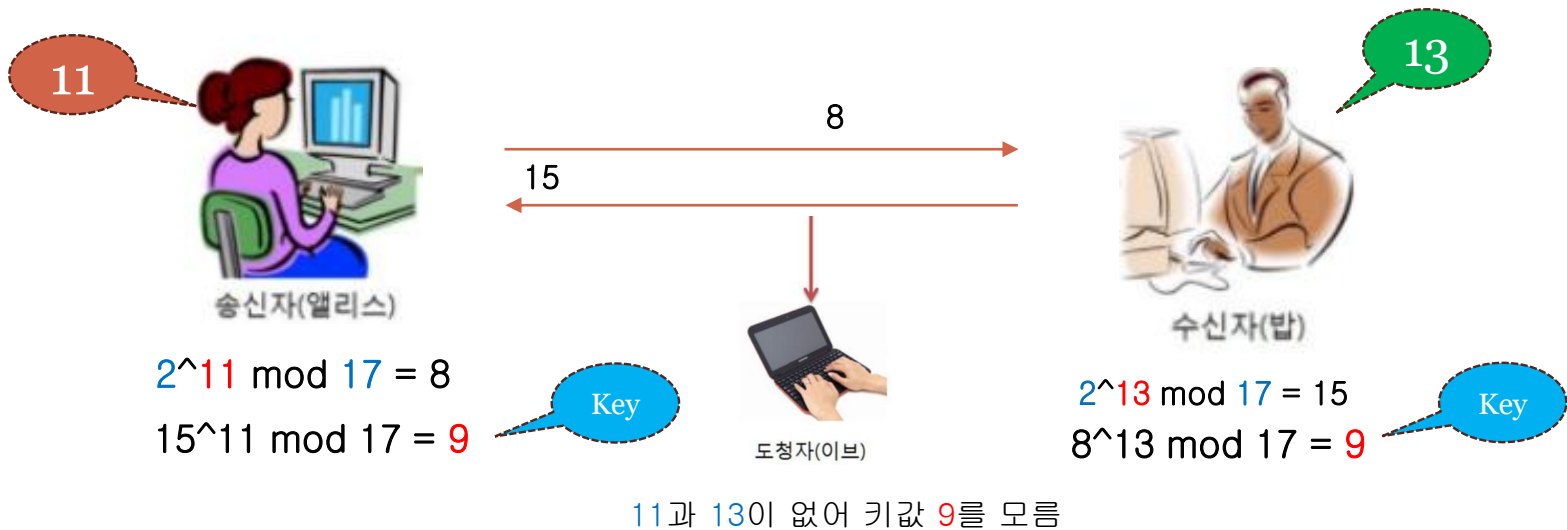
B는 A에게 15을 보낸다.

A는 받은 15으로 계산한다. $15^{11} \bmod 17 = 9$

B는 받은 8로 계산한다. $8^{13} \bmod 17 = 9$

A만 알고 있는 키는 11, B만 알고 있는 키는 13, 둘 다 알고 있는 키는 2, 17, 8, 15, 9
그런데 도청하는 사람은 2, 17, 8, 15 밖에 모른다

9를 비밀 키로 하여 암호화한 후 메시지 전달

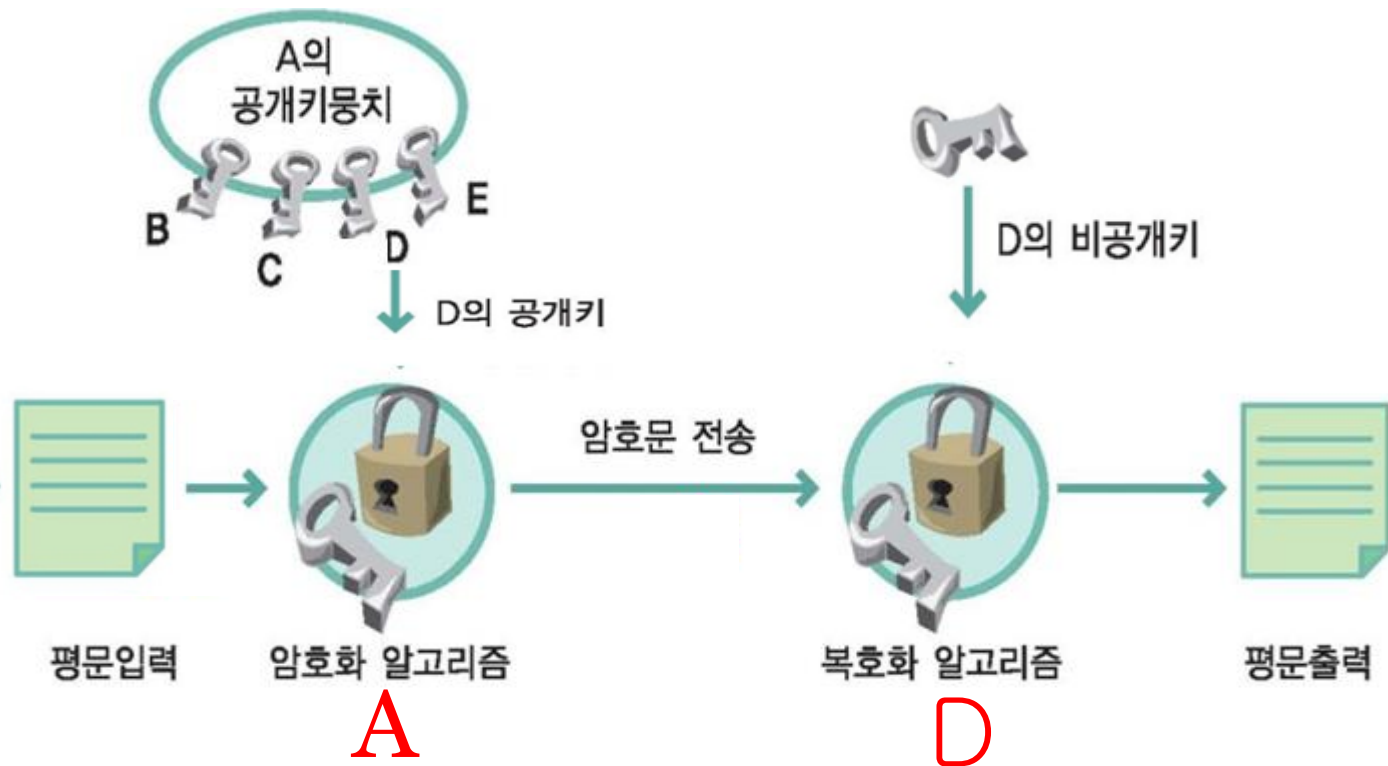


디피-헬만 키 교환 알고리즘 (Diffie-Hellman key exchange)

- 비대칭키(공개키) 알고리즘에서 사용되는 **키 교환 방식**
 - 암호문 작성, 인증 등에는 이용 못함
- 절차
 - 상대방의 공개키와 나의 비밀키를 이용하여 비밀키를 생성
 - A의 공개키와 B의 개인키를 이용하여 B의 비밀키 생성
 - B의 공개키와 A의 개인키를 이용하여 A의 비밀키 생성
 - 이산대수법에 의거한 수학적 공식에 의해 **A의 비밀키와 B의 비밀키가 같아짐**
 - 송신자와 수신자는 이 비밀키를 사용하여 데이터를 암호화하여 메시지 전송
- 취약점
 - 신분위장 공격에 취약
 - 애초에 B가 아닌 사람이 B인척 하거나
 - 연산 된 결과인 비밀키만 취득한 제 3자가 B 행세를 할 수도 있다.
 - 재전송 공격에 취약
 - 단순히 비밀키로 잘 암호화 되어 있는지만 확인
 - 예를 들어 아이디/비번 입력 정보를 스니핑(도청)
 - 이를 재전송하면? 사용자를 조작할 수 있다.
 - 해결책
 - 메시지 인증(MAC) 등

공개키(비대칭키) 암호 시스템

- 절차
 - A가 D에게 암호문을 과정을 살펴보자
 - A는 여러사람의 공개키를 가질 수 있다
 - 그 중, D의 공개키로 암호문을 만들어 전송한다
 - D는 자기의 비밀키로 복호화 한다



RSA 암호 시스템의 탄생

- MIT의 Rivest, Shamir, Adleman이 제안
- 특성
 - 공개키 방식
 - 키 배송이 용이
 - 전자 서명 및 인증이 용이



RSA 암호화의 기본원리

- 두개의 키 (암호화 키, 복호화키)를 사용
 - 암호화 $C=E(M) = M^e \bmod n$
 - 복호화 $M=E(C) = C^d \bmod n$
 - 페르마의 소정리(Fermat's little Theorem)에 근거함
 - 암호화키 (e, n)와 복호화키 (d, n)를 구하는 것이 문제!!!
- 일반적으로 e 와 n 은 공개되어도 무방
 - 그러나 이를 주어도 d 를 구하는 것을 매우 어렵게
 - e : 공개키
 - d : 개인키
 - n 은 노출이 불가피함

n 이 노출 되어도 소인수 분해가 되지 않으면 d 가 안전함

RSA의 키 생성 알고리즘

- ① 임의의 큰 소수 **p, q**를 선택,
 $n = p \times q$
- ② $L = \text{LCM}\{(p-1), (q-1)\}$
- ③ $\text{GCD}(d, L) = 1$ 이 되도록 **d**를 정함
LCM은 최소공배수, GCD는 최대공약수
- ④ $(e \times d) \bmod \{(p-1)(q-1)\} = 1$ 이 되도록 **e**를 정함

- ① $p=19, q=23, n=437$
- ② $L = \text{LCM}\{18, 22\} = 198$
- ③ $d=61$ 로 정함(비밀키)
- ④ $e \times 61 \bmod 396 = 1$
 $e=13$ (공개키)

평문 $M=2$

$$\begin{aligned}\text{암호문 } C &= m^e \bmod n \\ &= 2^{13} \bmod 437 = 326\end{aligned}$$

$$\begin{aligned}\text{복호문 } M &= C^d \bmod n \\ &= 326^{61} \bmod 437 = 2\end{aligned}$$

RSA 암호문 전송 과정(1)

- ① $p=19, q=23, n=437$
- ② $L = \text{LCM}\{18, 22\} = 198$
- ③ $d=61$ 로 정함(비밀키)
- ④ $e \times 61 \bmod 396 = 1$
 $e=13$ (공개키)

평문 $M=2$

암호문 $C = m^e \bmod n = 2^{13} \bmod 437 = 326$

복호문 $M = C^d \bmod n = 326^{61} \bmod 437 = 2$

비밀키 : 61
공개키 : 13



수신자(밥)

$e=13, n=437$

326



송신자(앨리스)

$$326^{61} \bmod 437 = 2$$



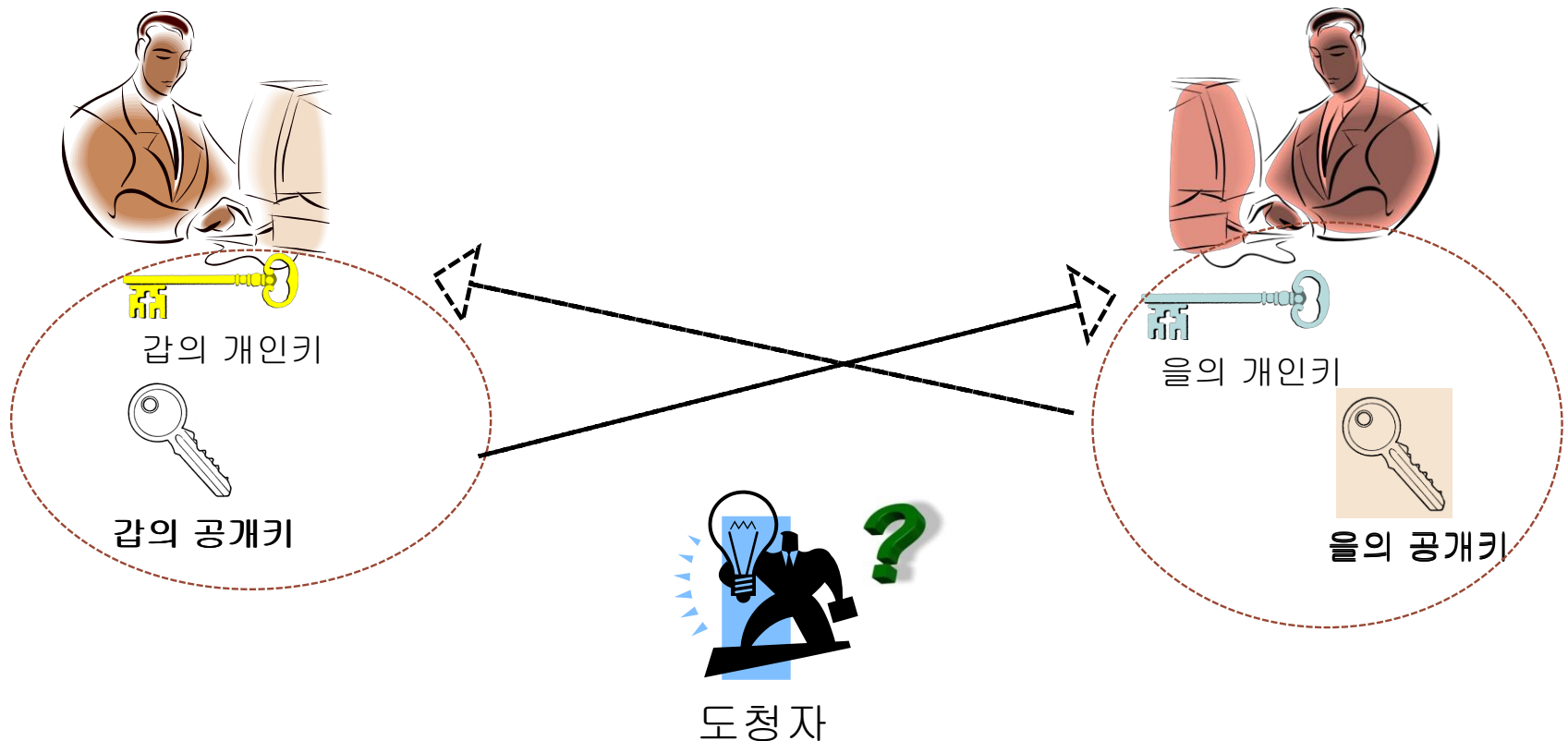
도청자(이브)

?

$$2^{13} \bmod 437 = 326$$

RSA 암호의 키 배송 특성

- A와 B는 각자의 public key(공개키)를 서로에게 알려줌(단, 비밀키는 본인이 보관)
 - A는 PubA, PriA, PubB를, B는 PubB, PriB, PubA를 가지게 됨.
 - 공격자 C는 PubA, PubB만 가질 수 있으나 복호에 필요한 비밀키는 가지지 못함
- A와 B는 서로의 공개키로 암호화하여 송신하고 수신자는 자기의 비밀키로 복호화



RSA 온라인예제

The image shows a web interface for RSA encryption and decryption. It is divided into two main sections: 'Generate RSA Key Pair' and 'RSA Encryption/Decryption'. The 'Generate RSA Key Pair' section has a button 'Generate RSA Key Pair' and displays a 'Public Key' and a 'Private Key'. The 'RSA Encryption' section has a text input 'Enter Plain Text to Encrypt' (containing 'Boy'), a text area 'Enter Public/Private key' (containing the public key), a radio button 'RSA Key Type: Public key' (selected), a dropdown 'Select Cipher Type' (set to 'RSA'), an 'Encrypt' button, and a text area 'Encrypted Output (Base64):' (containing the encrypted text). The 'RSA Decryption' section has a text input 'Enter Encrypted Text to Decrypt (Base64)' (containing the encrypted text), a text area 'Enter Public/Private key' (containing the private key), a radio button 'RSA Key Type: Private key' (selected), a dropdown 'Select Cipher Type' (set to 'RSA'), a 'Decrypt' button, and a text input 'Decrypted Output:' (containing 'Boy'). A yellow arrow points from the 'Encrypted Output' of the encryption section to the 'Enter Encrypted Text' of the decryption section. Red and blue curved arrows show the flow of data between the key generation, encryption, and decryption steps.

Generate RSA Key Pair

Public Key

```
MIGfMA0GCsQGSib3DQEBAQUAA4GNADCBiQKBgQCyFf1QuNslfzJudNS/DA5u3TpW39j8q9A3gao/Wr0nRS4Fy+VpN0G19wwnezPigPERodljr3og3/189e2AuF+J/Bf15BhiSySlwgdSvPlaMs8lc7Cvvz3lqyio5OcsoWbF3lJmkXNeGqONmKQ
```

Private Key

```
MIICeQIBADANBgkqhkiG9w0BAQEFAASCAmMwgGJfAgEAAoGBALIV/VC42yv/Mm50lL8MDm7dOnDf2Pyr0DeBqj9avSdFLgXL5Wk3QbX3DCd7M+KA8RGh2WOveiDf/Xz17YC4X4n8F/XkGGJLJjCB1K88toyzwhzsk+/PcirKKjk5yyhZsXf
```

RSA Encryption

Enter Plain Text to Encrypt

Boy

Enter Public/Private key

```
MIGfMA0GCsQGSib3DQEBAQUAA4GNADCBiQKBgQCyFf1QuNslfzJudNS/DA5u3TpW39j8q9A3gao/Wr0nRS4Fy+VpN0G19wwnezPigPERodljr3og3/189e2AuF+J/Bf15BhiSySlwgdSvPlaMs8lc7Cvvz3lqyio5OcsoWbF3lJmkXNeGqONmKQ
```

RSA Key Type: ☒ Public key ☐ Private Key

Select Cipher Type

RSA

Encrypt

Encrypted Output (Base64):

```
b1vEd89QycyDTToBYCzRRyFYnpJgV4NAcwNpzyCQA4diauGTf1UGcWd1818Utsu6MykOV3rKsh9WR6jl0ljhSG/ydh3AhVD3X4cXw7sTzMzP6M
```

RSA Decryption

Enter Encrypted Text to Decrypt (Base64)

```
b1vEd89QycyDTToBYCzRRyFYnpJgV4NAcwNpzyCQA4diauGTf1UGcWd1818Utsu6MykOV3rKsh9WR6jl0ljhSG/ydh3AhVD3X4cXw7sTzMzP6M
```

Enter Public/Private key

```
MIICeQIBADANBgkqhkiG9w0BAQEFAASCAmMwgGJfAgEAAoGBALIV/VC42yv/Mm50lL8MDm7dOnDf2Pyr0DeBqj9avSdFLgXL5Wk3QbX3DCd7M+KA8RGh2WOveiDf/Xz17YC4X4n8F/XkGGJLJjCB1K88toyzwhzsk+/PcirKKjk5yyhZsXf
```

RSA Key Type: ☐ Public key ☒ Private Key

Select Cipher Type

RSA

Decrypt

Decrypted Output:

Boy

RSA 암호화 예제

- Bob이 “MATH”를 Alice에게 보내고자 한다.

- 키 생성 (Alice)

- $p = 41$ 과 $q = 53$ 을 택한다.
- $n = pq = 2173$ 을 계산한다.
- $(p-1)(q-1) = 2080$ 과 서로소인 정수 $e = 623$ 을 택한다.
 $623 \cdot 207 \equiv 1 \pmod{2080}$ 이므로, $d = 207$ 이 된다.

- 암호문 작성 (Bob)

- 먼저 “MATH”를 숫자화 한다
 - 예를 들어 $M = 77, A = 65, T = 84, H = 72 \rightarrow 77658472$
- 위 수를 세 자리 씩 나눈다. m_3 는 세자리가 되도록 9를 채움

$$m_1 = 776, \quad m_2 = 584, \quad m_3 = 729$$

- 공개키 $e = 623, n = 2173$ 을 이용하여 나머지를 구함

$$m_1^e = 776^{623} \equiv 2159 = r_1 \pmod{2173}$$

$$m_2^e = 584^{623} \equiv 1697 = r_2 \pmod{2173}$$

$$m_3^e = 729^{623} \equiv 2018 = r_3 \pmod{2173}$$

- Alice에게 $r_1 = 2159, r_2 = 1697, r_3 = 2018$ 을 송신한다.

- 복호화 (Alice)

- Alice는 수신한 암호문을 복호화하기 위해, 비밀키 $d = 207, n = 2173$ 을 이용하여 나머지를 계산함

$$2159^{207} \equiv 776 \pmod{2173}$$

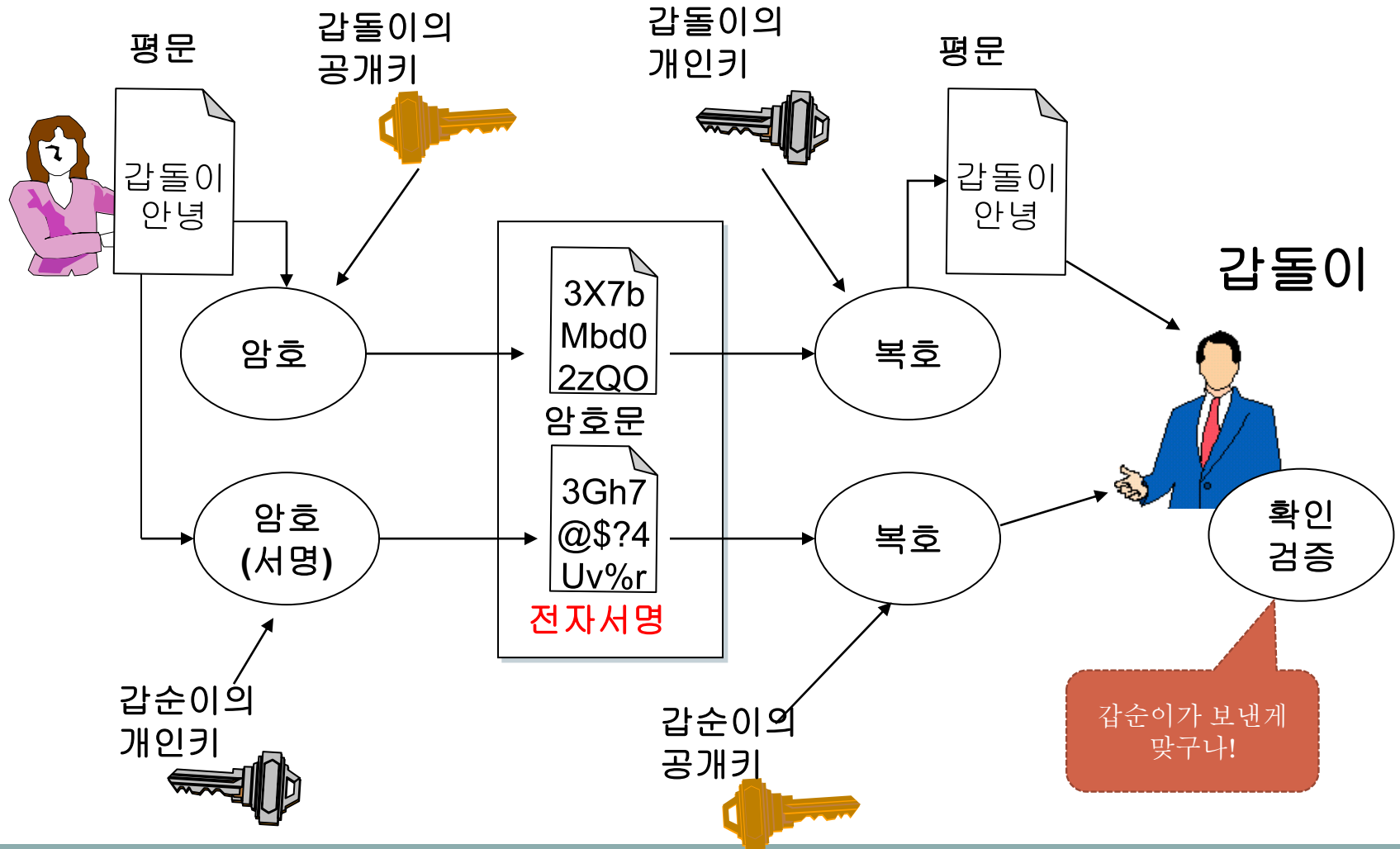
$$1697^{207} \equiv 584 \pmod{2173} \quad m_1 = 776, \quad m_2 = 584, \quad m_3 = 729$$

$$2018^{207} \equiv 729 \pmod{2173}$$

- 세 수를 붙여 776584729 를 만들고 두 자리 씩 끊어서 $M = 77, A = 65, T = 84, H = 72$
- 원래의 문장 “MATH”를 얻음

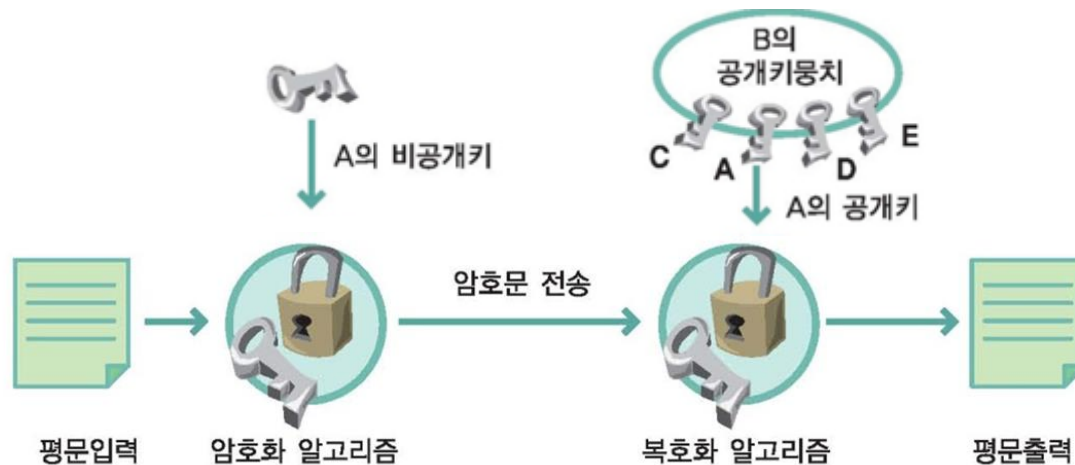
전자서명의 원리

갑순이

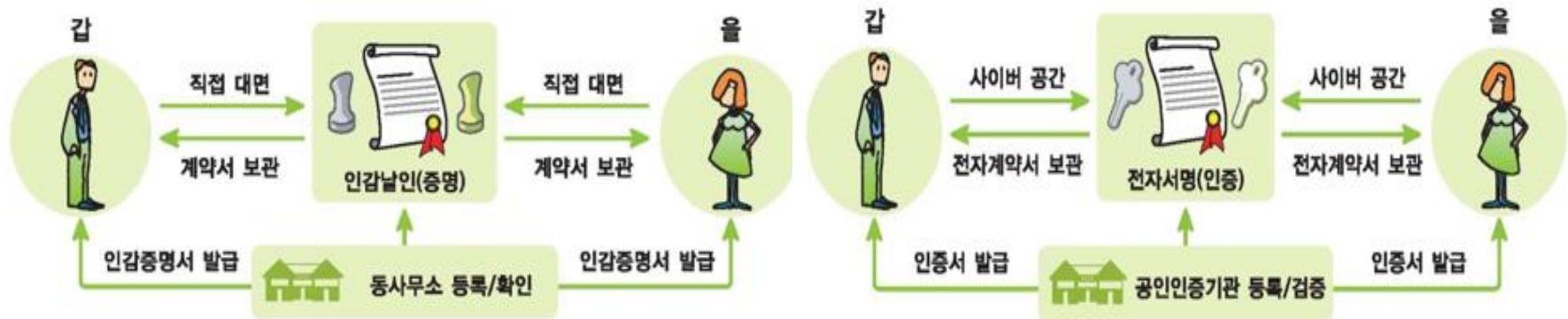


전자 서명

- 전자 서명(Digital Signature)
 - A가 B에게 메시지를 보낼 때
 - ✧ A의 비공개키로 암호화
 - ✧ B는 A의 공개키로 이를 해독
 - A의 공개키로 해독이 가능하다면
 - ✧ 이 암호문은 A의 비공개키로 암호화한 것임이 틀림없음
 - ✧ A는 보낸 사실을 부인 못함
 - ✧ 전자서명



전자서명의 활용



최상위 인증기관	전자서명인증관리센터	http://www.rootca.or.kr
전자서명인증관리센터	한국정보인증(주)	http://www.sigmgate.com
	한국증권전산(주)	http://www.signkorea.com
	금융결제원	http://www.yessign.or.kr
	한국전산원	http://sign.nca.or.kr
	한국전자인증	http://gca.crosscert.com
	한국무역정보통신	http://www.tradesign.net

인증이란?

- 인증(authentication)
 - 정보의 교류 속에서 전송 받은 정보의 내용이 변조 또는 삭제되지 않았는지와 주체가 되는 송/수신자가 정당한지를 확인하는 방법
- 사용자 인증
 - 사용자가 터미널을 통해 컴퓨터 시스템에 들어가기를 원하거나 또는 정보의 전송에서 필요한 송/수신자, 이용자, 관리자 등이 자신이 신분을 증명하기 위한 방법
- 메시지 인증
 - 전송되는 메시지의 내용이 변경이나 수정이 되지 않고 본래의 정보를 그대로 가지고 있다는 것을 확인하는 과정

공인인증서



- 공인인증서 개요
 - 인터넷 상에서 여러 활동을 할 때 신원을 확인하고, 문서의 위조와 변조, 거래 사실의 부인 방지 등을 목적으로 공인인증기관(CA: Certificate Authority)이 발행하는 전자적 정보로서, 일종의 사이버 거래용 인감증명서
- 공인인증서 발행기관
 - 정보통신부가 지정한 6곳

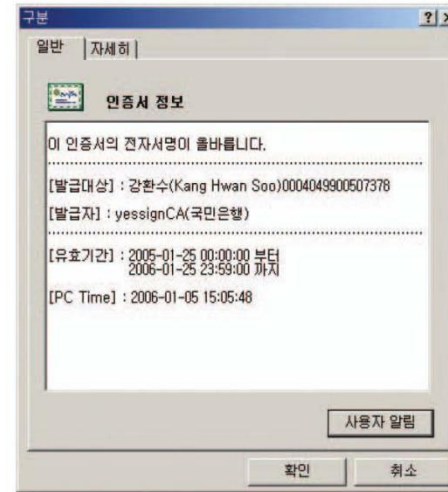


그림 12-21 공인인증서의 예

최상위 인증기관	전자서명인증관리센터	http://www.rootca.or.kr
전자서명인증관리센터	한국정보인증(주)	http://www.sigmgate.com
	한국증권전산(주)	http://www.signkorea.com
	금융결제원	http://www.yesign.or.kr
	한국전산원	http://sign.nca.or.kr
	한국전자인증	http://gca.crosscert.com
	한국무역정보통신	http://www.tradesign.net

공인인증서 활용

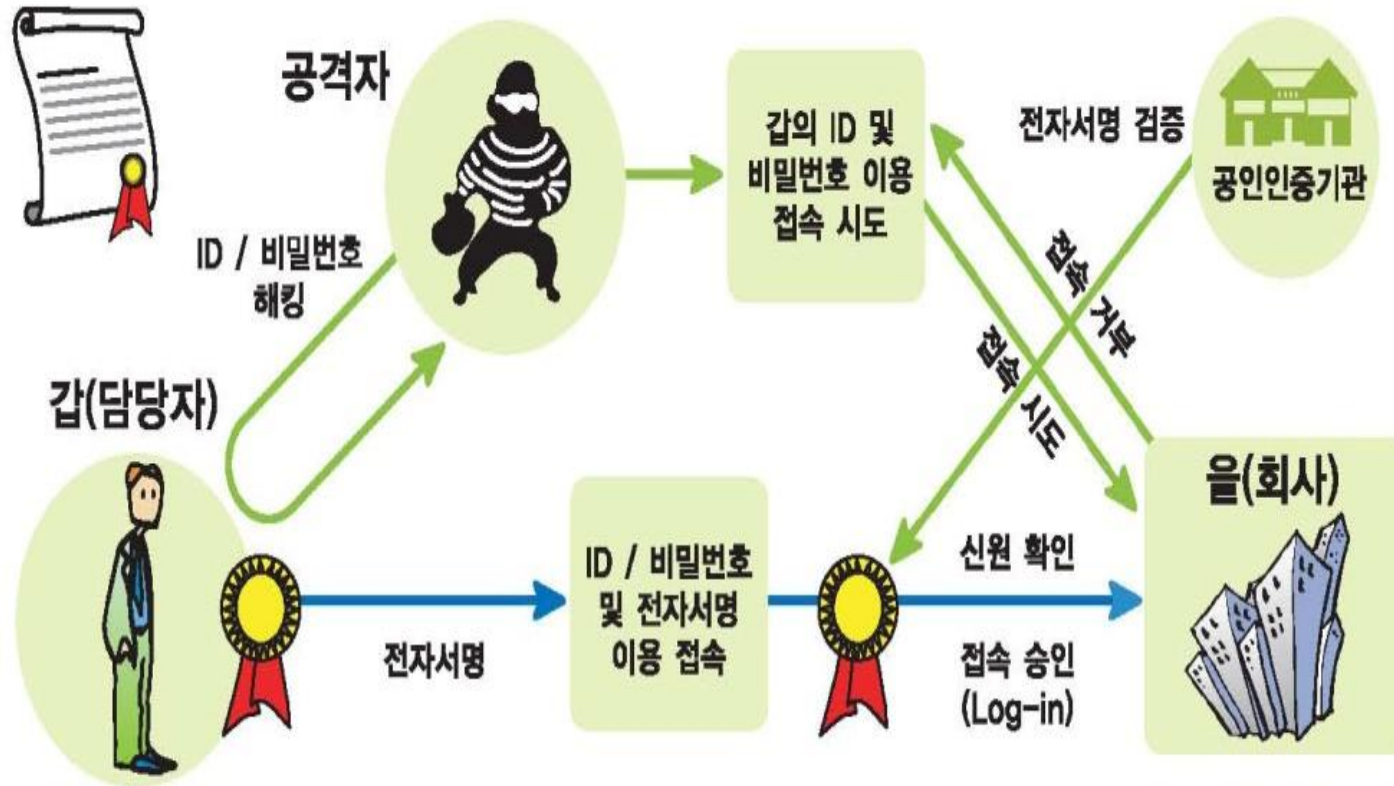


그림 12-22 공인인증서를 이용한 해킹 방지

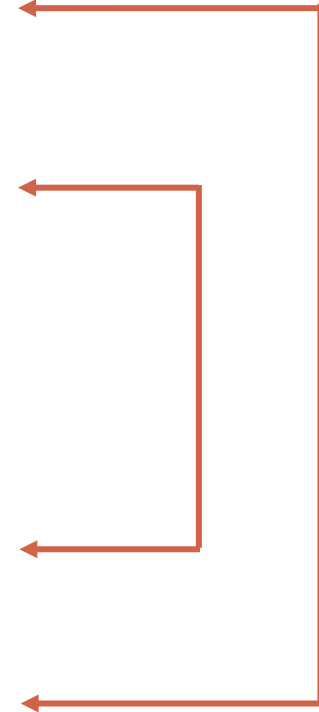
공개키와 비밀키 시스템의 비교

1) 공개키 시스템(비대칭키 시스템)

- 장점
 - 키배송 등 관리가 용이
 - 개인키만 비밀화
- 단점
 - 처리속도가 느림
 - 안전성 증명이 어려움.
- 이용 : 효율적인 서명(부인 방지, 키 배송)

2) 비밀키 시스템(대칭키 시스템)

- 장점
 - 처리속도가 빠름(짧은 키 사용)
- 단점
 - 키를 사용자 모두가 비밀화
 - 안전성을 위한 잦은 키 교체 필요
- 이용 : 대용량 데이터 전송



RSA 암호 시스템의 안전성(1)

- 매우 큰 정수의 소인수 분해가 어렵다는 가정에서 설계
- 예제
 - $p=3,490,529,510,847,650,949,147,849,619,903,898,133,417,764,638,493,387,843,990,820,577$ (63자리 정수)
 - $q=327,691,329,932,667,095,499,619,881,908,334,461,413,177,642,967,992,942,539,798,288,533$ (22자리 정수)
 - $n=1,143,816,257,578,888,676,692,357,799,761,466,120,102,182,967,212,423,625,625,618,429,357,069,352,457,338,978,830,597,123,563,958,705,058,989,075,147,599,290,026,879,543,541$ (130자리정수)
 - **p, q를 알고 n을 구하는 것은 매우 간단**
 - **그러나 n을 소인수 분해하여 p, q를 알아내는 것은 숫자가 클수록 어려워짐**
- 보통 2^{1024} 크기(300자리)의 소수를 사용함
 - 2^{1024} 크기의 두 소수의 곱을 소인수분해하는데 3,486년 걸림
 - $2^{2048} : 10^{11}$ 년
 - $2^{3072} : 10^{18}$ 년

251959084756578934940271832400483985714292821262040320277771378360436620207075955562640185258
807844069182906412495150821892985591491761845028084891200728449926873928072877767359714183472
702618963750149718246911650776133798590957000973304597488084284017974291006424586918171951187
461215151726546322822168699875491824224336372590851418654620435767984233871847744479207399342
365848238242811981638150106748104516603773060562016196762561338441436038339044149526344321901
146575444541784240209246165157233507787077498171257724679629263863563732899121548314381678998
85040445364023527381951378636564391212010397122822120720357 →617자리, 소인수분해 현상금

RSA 암호 시스템의 안전성 (2)

- **RSA 공개키 암호 시스템의 안전성**

- 공개키 n 을 소인수 분해할 수 있으면 해독이 가능하나, **RSA** 공개키 암호 시스템이 해독 가능하다고 해서 공개키 n 의 소인수 분해가 가능하다는 것은 증명되지 않음.

즉, 소인수 분해 이외의 방법으로도 **RSA** 공개키 암호 시스템은 해독 가능

- **RSA** 공개키 암호 시스템을 안전하게 구성하기 위하여 소수 p 와 q 는 다음 조건을 만족해야 한다.

- ① p 와 q 는 거의 같은 크기의 수이다.
- ② $p-1$ 과 $q-1$ 은 큰 소인수를 갖는다.
- ③ $\gcd(p-1, q-1)$ 은 작은 수이다.
- ④ $p+1$ 과 $q+1$ 이 큰 소인수를 갖는다.
- ⑤ $|p-q|$ 는 충분히 커야 한다.

공개키 암호 시스템의 종류

- 소인수 분해의 어려움에 기반을 둔 공개키 암호 시스템
 - **RSA** 암호 시스템
 - **Robin** 공개키 암호 시스템
 - **Williams** 공개키 암호 시스템
- 이산대수 문제의 어려움에 기반을 둔 공개키 암호 시스템
 - **Diffie-Hellman** 키 분배 방식
 - **ElGamal** 공개키 암호 시스템
- 배낭 문제에 기반을 둔 공개키 암호 시스템
 - **Merkle-Hellman** I 형 배낭 암호
 - **Merkle-Hellman** II 형 배낭 암호
- 타원 곡선 암호 시스템
 - 타원 곡선상의 **Diffie-Hellman** 키 분배 방식
 - 타원 곡선상의 **ElGamal** 공개키 암호 시스템
- 확률 암호 시스템
 - **Goldwasser-Micali** 확률 공개키 암호 방식
 - **Blum-Goldwasser** 확률 공개키 암호 방식
 - **OAEP(Optimal Asymmetric Encryption Padding)** 확률 공개키 암호 방식

국내외 권고 암호알고리즘

▮▮▮ <표 1> 국내외 권고 암호 알고리즘

분류		NIST(미국) (2015)	CRYPTREC(일본) (2013)	ECRYPT(유럽) (2018)	국내 ¹⁾ (2018)
대칭키 암호 알고리즘 (블록암호)		AES 3TDEA ²⁾	AES Camellia	AES Camellia Serpent	SEED HIGHT ARIA LEA
해시함수		SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 SHA3-224 SHA3-256 SHA3-384 SHA3-512	SHA-256 SHA-384 SHA-512	SHA-256 SHA-384 SHA-512 SHA-512/256 SHA3-256 SHA3-384 SHA3-512 SHA3-shake128 ³⁾ SHA3-shake256 ³⁾ Whirlpool-512 BLAKE-256 BLAKE-384 BLAKE-512	SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 SHA3-224 SHA3-256 SHA3-384 SHA3-512 LSH-224 LSH-256 LSH-384 LSH-512 LSH-512-224 LSH-512-256
공개키 암호 알고 리즘	키 공유용	DH ECDH MQV ECMQV	DH ECDH	ECIES-KEM PSEC-KEM RSA-KEM	DH ECDH
	암·복호 화용	RSA	RSA-OAEP	RSA-OAEP	RSAs



대칭키 암호화(블럭암호)

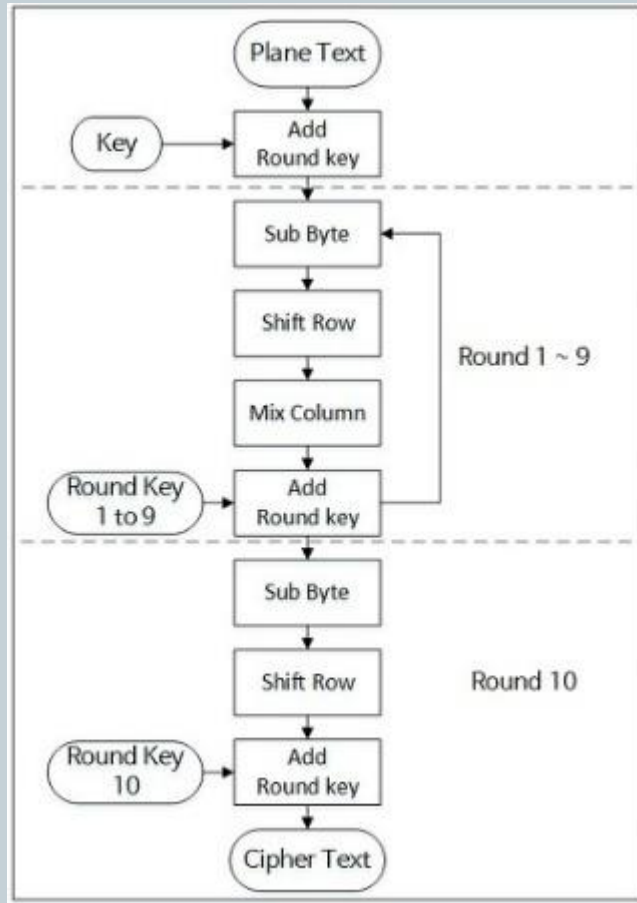
-AES (Advanced Encryption Standard) 중심으로

AES 암호시스템의 개요

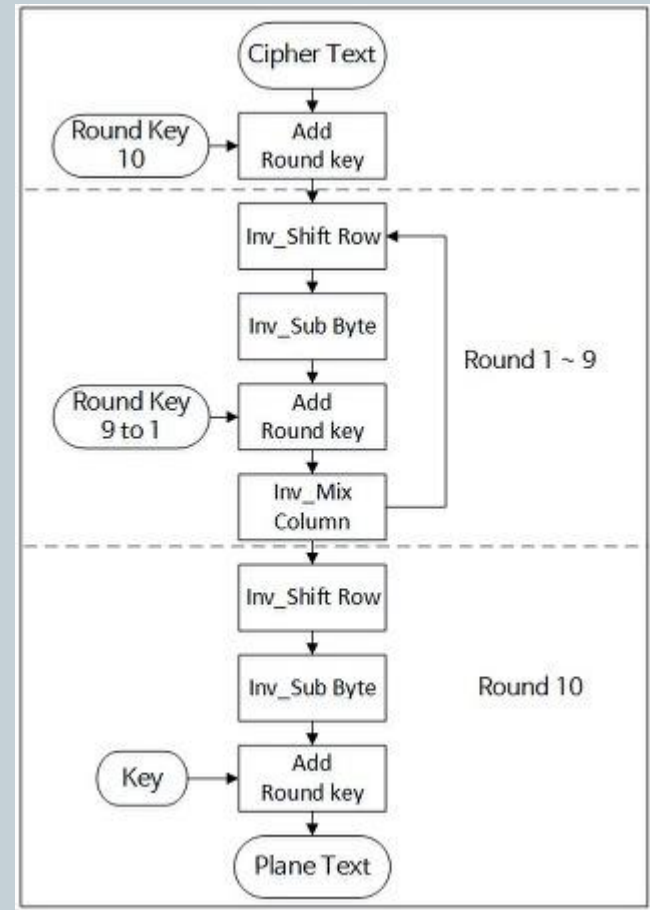
- Advanced Encryption Standard(AES)
- 2001년 미국 표준 기술 연구소(NIST)에 의해 제정된 암호화 방식
- AES는 두 명의 벨기에 암호학자인 **존 대먼**과 **빈센트 라이먼**에 의해 개발된 **Rijndael(레인달)**에 기반하며 AES 공모전에서 선정되었다.
- AES 표준은 여러 레인달 알고리즘 중 블록 크기가 128비트인 알고리즘을 말한다.

항목	DES	AES
키 크기	56	128/192/256
평문 블록 크기	64	128
암호문 블록 크기	64	128
라운드 수	16	12/14/16
전체 구조	Feistel	SPN
개발 기관	미국 표준 기술 연구소	미국 표준 기술 연구소
보안	키 크기가 작기 때문에 AES보다 안전하지 않음	DES보다 훨씬 안전
속도	AES보다 느림	DES보다 비교적 빠름

Rijndael 알고리즘(128bit Algorithm)



암호과정



복호과정

AES 암호시스템의 구현



- AES는 국제표준규격의 128bit 블록암호이다.
- 키 길이는 128, 192, 256bit를 지원할 수 있다.
- AES는 SPN 구조를 따른다
- AES 알고리즘 안에는 총 4가지의 연산이 존재한다.
 - 1. Add Round Key
 - 2. Sub Byte
 - 3. Shift Row
 - 4. Mix Column

1. Add Round Key



- Data와 라운드 키를 XOR연산을 한다
- AddRoundKey는 각 라운드마다 갱신됨
- 총 round+1 번 수행
 - AES128 는 10 round이므로 11번 수행됨

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

data (128bit)

$\text{XOR} \wedge$

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

round key (128bit)

=

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

(128bit)

2. Sub Bytes (Substitute bytes)

- Sub bytes 연산은 8비트 단위(1바이트)로 데이터를 **치환**하는 방식
- 연산은 1바이트를 $GF(2^8)$ 상에서 역원을 구한 후, 아핀 변환
- 편의 상 **S-BOX**를 만들어서 이를 이용해 치환하는 방식을 많이 사용한다
- 전방향 S-box는 다음과 같은 수학적 원리로 적용이 된다:

- 1) Invert in $GF(2^8)$
- 2) Multiply by a matrix X
- 3) Add a constant y

Galois
Field

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

Foward S-Box(암호화 시)

대 체 예 제



19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

D4

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

Foward S-Box(암호화 시)

Inverse S-Box (복호화 시)



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

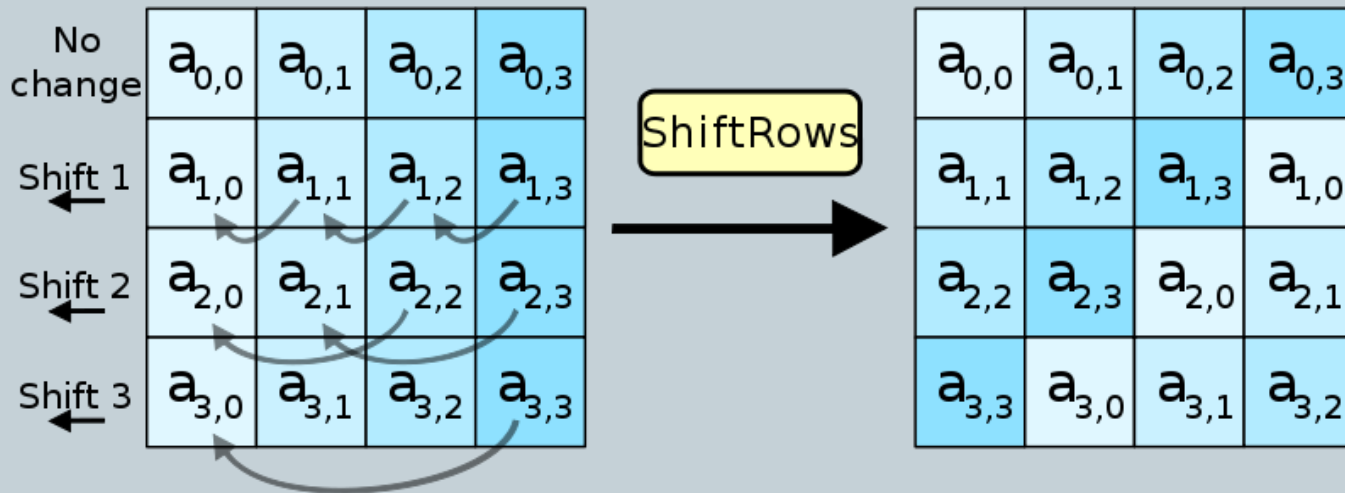
D4

19

3. Shift Row



- Shift row 연산은 그림과 같이 행단위로 left shift rotate 를 진행한다



4. Mix Column

- Permutation 과정
- 각각 바이트에 특정 행렬의 행과 데이터의 column 별로 곱연산을 진행
 - 행렬의 곱)
- 특정 행렬은 미리 정의되어 있으며 다음과 같다

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$



$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$	$S'_{1,0}$
$S'_{2,2}$	$S'_{2,3}$	$S'_{2,0}$	$S'_{2,1}$
$S'_{3,3}$	$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$

Foward Mix (암호화 시)

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$	$S'_{1,0}$
$S'_{2,2}$	$S'_{2,3}$	$S'_{2,0}$	$S'_{2,1}$
$S'_{3,3}$	$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$



$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Inverse Mix(복호화 시)

Mix Column 예시



- 행렬의 곱을 수행
- $2*d4 \text{ xor } 3*bf \text{ xor } 1*5d \text{ xor } 1*30 = 04$
- $1*d4 \text{ xor } 2*bf \text{ xor } 3*5d \text{ xor } 1*30 = 66$
- $1*d4 \text{ xor } 1*bf \text{ xor } 2*5d \text{ xor } 3*30 = 81$
- $3*d4 \text{ xor } 1*bf \text{ xor } 1*5d \text{ xor } 2*30 = 04$

02	03	01	01	d4	04
01	02	03	01	bf	66
01	01	02	03	5d	81
03	01	01	02	30	e5

Key Schedule(Key Expansion)



- AES 암호화에서 사용되는 키는 각 라운드 별로 round key를 생성하여 사용한다.
- 최초의 키가 다음과 같다고 하고 예제를 보자

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

Key

라운드 키생성 1단계



- 마지막 column 값을 위쪽으로 shift rotate 를 적용하고 Sub byte를 통해서 해당 값을 치환한다

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

cf
4f
3c
09

shift rotate

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box



8a
84
eb
01

라운드 키생성 2단계



- 이것을 RCON으로 정의 되어있는 행렬을 이용해서 RCON의 첫번째 column과 XOR 연산을 수행한다.
- 추가로 원래 키의 첫번째 column과도 XOR을 진행하면 첫 라운드 키의 첫번째 column이 된다

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon

2b		8a		01		a0
7e	\oplus	84	\oplus	00	=	fa
15		eb		00		fe
16		01		00		17

원래 키의 첫번째 column



- 다음으로 계산된 첫번째 column과 원래 키의 두번째 column을 XOR 하여 라운드 키의 두번째 column을 계산한다

28	\oplus	a0	$=$	88
ae		fa		54
d2		fe		2c
a6		17		b1

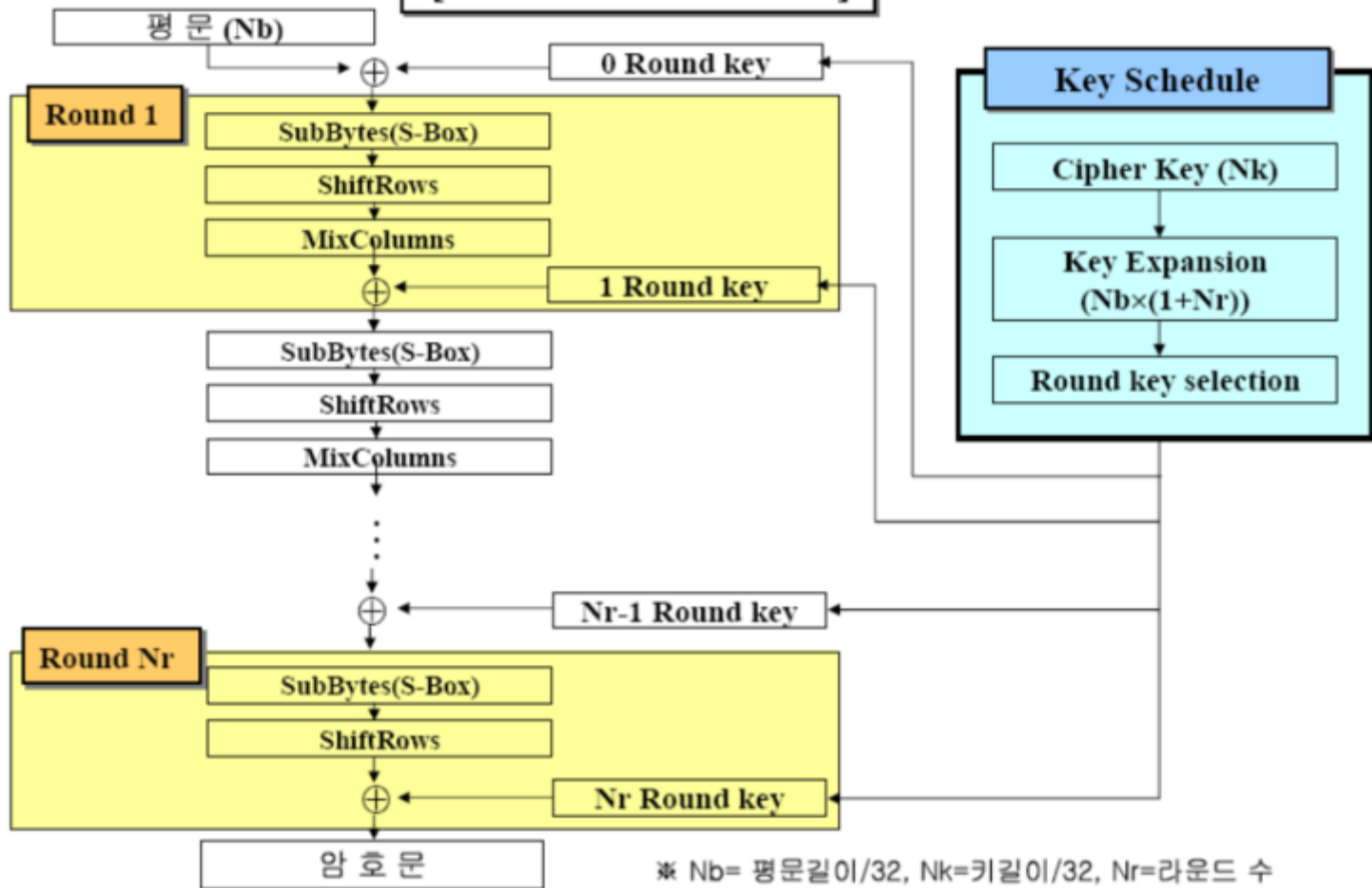
- 같은 방식으로 나머지 세번째, 네번째 column을 계산하면 최종적으로 첫번째 라운드 키가 완성된다

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

RoundKey1

- 이렇게 생성된 첫번째 라운드키로 다음 라운드 키를 계산하는 방식으로 10라운드 키까지 계산할 수 있다

[AES의 암호화 과정]



AES 온라인예제

AES Online Encryption

Enter text to be Encrypted

Boy

Select Cipher Mode of Encryption

ECB

Key Size in Bits

128

Enter IV (Optional)

Enter Initialization vector

Enter Secret Key

hkleehkleehklee!

Output Text Format: ☒Base64 ☐Hex

Encrypt

AES Encrypted Output:

mIQW26cofHHXygdvf8DkoQ==

AES Online Decryption

Enter text to be Decrypted

mIQW26cofHHXygdvf8DkoQ==

Input Text Format: ☒Base64 ☐Hex

Select Cipher Mode of Decryption

ECB

Key Size in Bits

128

Enter Secret Key used for Encryption

hkleehkleehklee!

Decrypt

AES Decrypted Output (**Base64**):

Qm95

Decode to Plain Text

Boy

ECB CBC

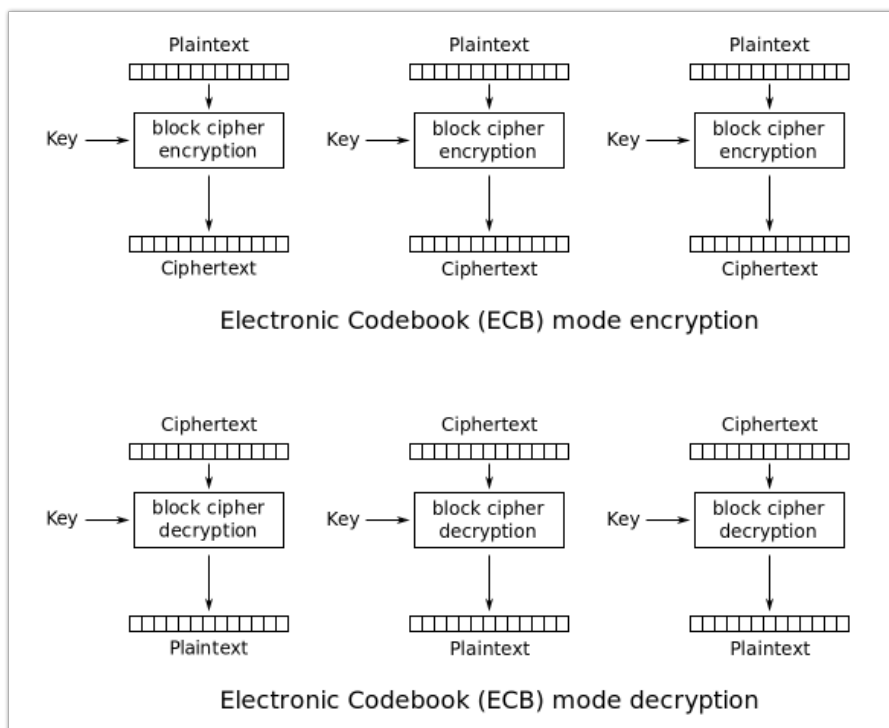
128bit=16byte

AES 암호화 운영모드(ECB, CBC 등)



• 1. ECB (Electronic Code Block) Mode

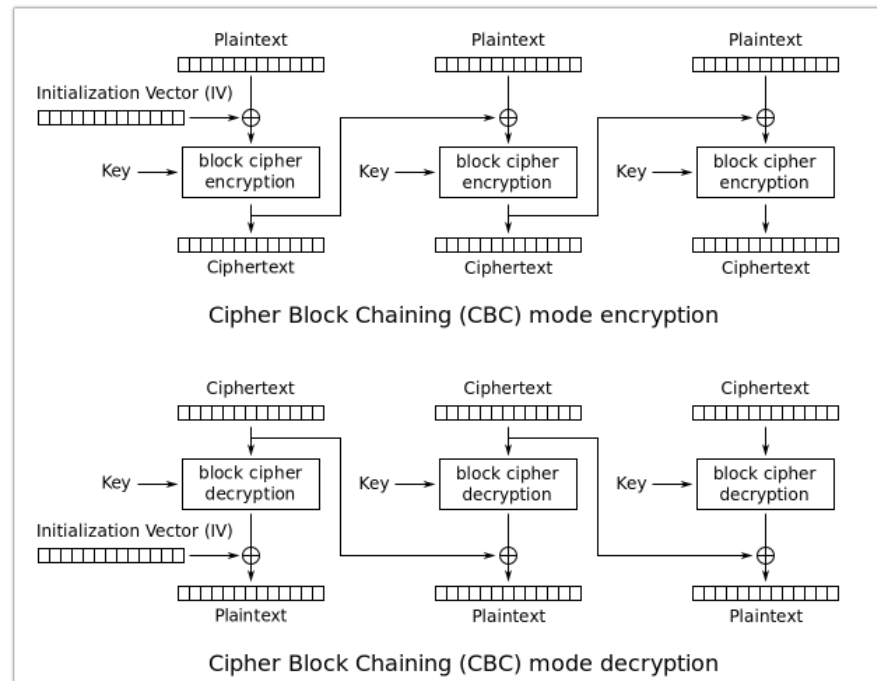
- 가장 단순한 모드로 블록단위로 순차적으로 암호화 하는 구조
- 한개의 블록만 해독되면 나머지 블록도 해독이 되는 단점이 있음
- 각 블록이 독립적으로 동작, 한블록에서 에러가 난다고 해도 다른 블록에 영향 없음



AES 암호화 운영 모드(ECB, CBC 등)

2. CBC(Cipher Block Chaining) Mode

- 블록 암호화 운영 모드 중 보안 성이 제일 높은 암호화 방법으로 가장 많이 사용됨.
- 평문의 각 블록은 XOR연산을 통해 이전 암호문과 연산 됨
 - ✦ 첫번째 암호문에 대해서는 IV(Initial Vector)가 암호문 대신 사용됨(또 다른 키로 간주)
- 암호화가 병렬처리가 아닌 순차적으로 수행됨
 - ✦ 깨진 암호문의 해당블록과 다음블록의 평문까지 영향을 미침





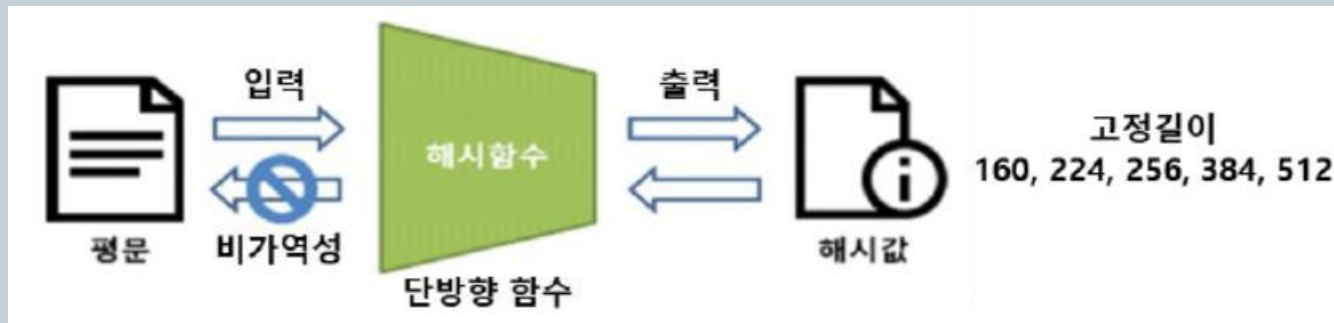
해시 함수 (Hash Function)

해시 함수



- 해시 함수란?

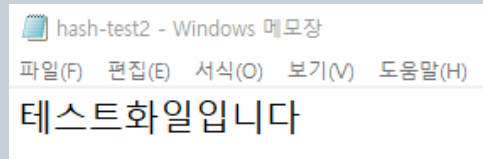
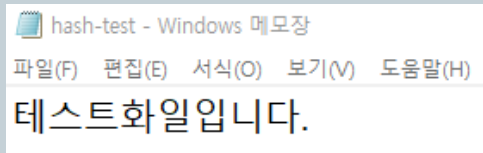
- 원문을 **고정길이**의 의사난수로 변환해주는 단방향함수
 - ✦ 해시 값은 해시 함수에서 정해진 고정 길이가 나오게 된다.
 - ✦ 입력 값을 메시지라고 함
 - ✦ 해시 함수를 **Message Digest Function** 라고도 함
 - ✦ 메시지는 꼭 텍스트가 아니어도 됨.
 - 즉, 화상 파일이거나 음성 파일이어도 상관없음.
- **역함수가 존재하지 않음**



해싱의 예 -SHA256



- 다음과 같은 두개의 파일에 대하여 해시값을 구해보자



```
cmd 명령 프롬프트

D:\>powershell get-filehash hash-test.txt


Algorithm      Hash
-----
SHA256         493197D20797EC7065673F681FB54B130DA7A5445AAE58F7E1FFDC8E4EB114AB

D:\>powershell get-filehash hash-test2.txt

Algorithm      Hash
-----
SHA256         E1DC2985F45B2977816B2F298F42BB439A4E379983949F7D35B850B5D452898E
```

- 출력 값은 64개의 16진수(=256비트) - 고정길이
- 파일은 마침표 하나 차이이지만 해시 값은 전혀 다름
- 파일에 대한 해시 값으로 파일의 변형 유무를 파악할 수 있다
- 해시 값은 일종의 지문과 같은 역할을 함

sha256 온라인예제



Enter Plain Text to Compute Hash

Boy

Enter the Secret Key

asdd

Select Cryptographic Hash Function

SHA-256

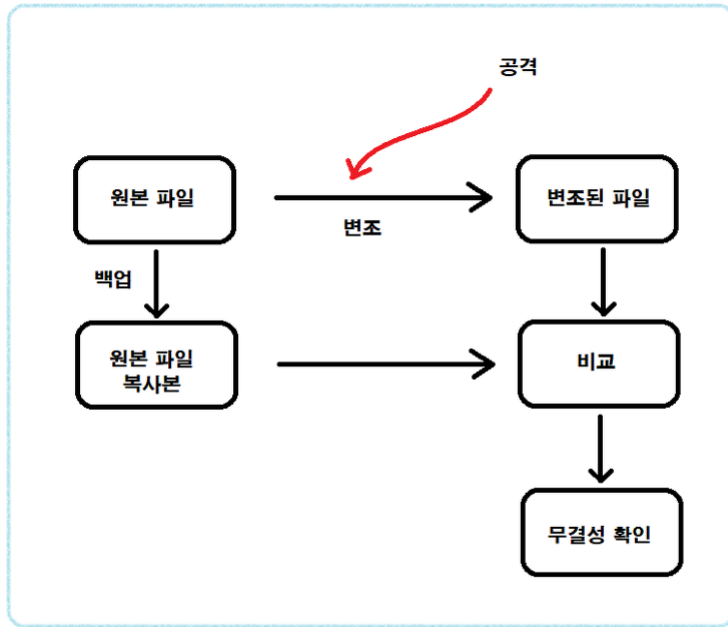
Output Text Format: ☒Hex ☐Base64

Compute Hash

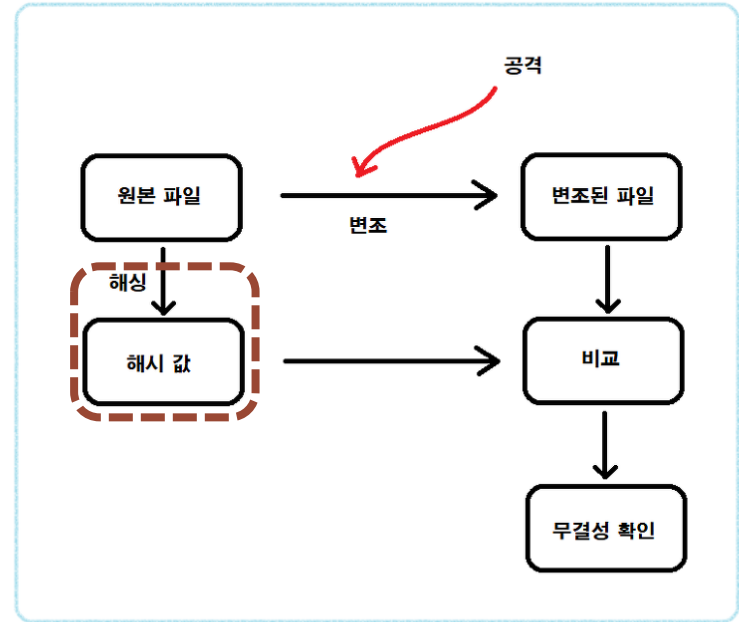
Hashed Output:

91996ae9a753cdd43ac82a0ffb656cd508bc44d7f4652ba1d2bbb77cb47fb599

해시 함수를 사용하는 이유



원본 파일을 복사하게 된다면 상당한
시간과 엄청난 용량이 필요함



256비트의 해시 값만으로 변조 여부를
판단할 수 있음

암호학적 해시함수의 예



- MD(메시지 다이제스트) 방식
 - MD2, 4(뿔림), 5가 있으며, 공개키 기반 구조를 만들기 위해 RSA와 함께 개발됨.
 - MD2는 8비트 4, 5는 32비트 컴퓨터에 최적화.
 - MD5는 메시지를 512비트로 된 블록으로 나누고, 128 다이제스트를 출력. 하지만 너무 짧아 충돌 공격에 내성이 약함.
- SHA(Secure Hash Algorithm)
 - NIST가 개발.
 - SHA-1은 SHA의 약점을 보완한 것.
 - SHA-256, SHA-384, SHA-512 등이 있음.
 - 다이제스트 생성 전 오리지널 메시지를 일정한 길이의 블록으로 구성.
 - 길이가 모자라면 패딩으로 채움
 - 예를 들어 **SHA-512** 블록은 1024비트. 다이제스트 생성 전 128비트 정수 필드에 오리지널 메시지 길이를 넣고, 나머지 공간에 패딩을 넣음.
즉, 1024 블록에서 2^{128} 이 오리지날 메시지 비트, 128비트는 오리지널 메시지 길이 정보, 나머지는 패딩.
* 블록체인에 SHA 256 사용됨

SHA 알고리즘의 종류와 특징

알고리즘	출력 해시 값 크기	블록 크기	충돌성
HAVAL	256/224/192/160/128	1024	있음
MD2	128	128	있음
MD4	128	512	있음
MD5	128	512	있음
PANAMA	256	256	있음
RIPEMD	128	512	있음
RIPEMD-128/256	128/256	512	없음
RIPEMD-160/320	160/320	512	없음
SHA-0	160	512	있음
SHA-1	160	512	있음
SHA-256/224	256/224	512	없음
SHA-512/384	512/384	1024	없음
Tiger(2)-192/160/128	192/160/128	512	없음
VEST-4/8 (hash mode)	160/256	8	없음
VEST-16/32 (hash mode)	320/512	8	없음
WHIRLPOOL	512	512	없음

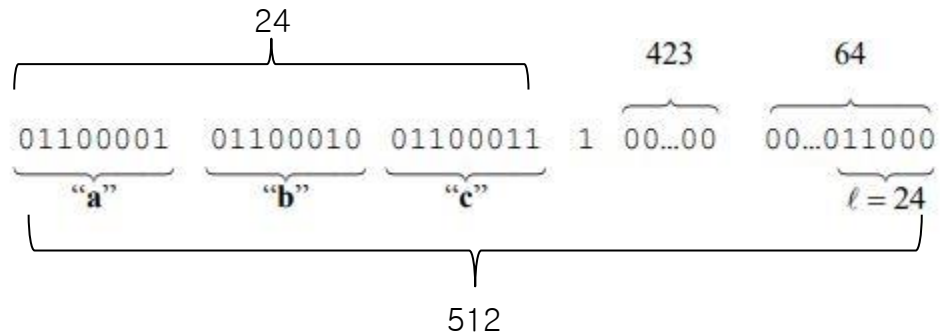
● 설명

- sha-256의 출력값 길이는 256비트(=32 바이트)
다만, 이 32바이트는 바이너리 데이터라서 문자로 표시할 수 없기 때문에 16진수 문자열로 표기
- 1바이트 바이너리 데이터는 두 자리 16진수문자열로 변환됨
- 따라서 256비트 바이너리 데이터는 64자리의 16진수문자열이 됨

Sha-256 구현

메시지 전처리

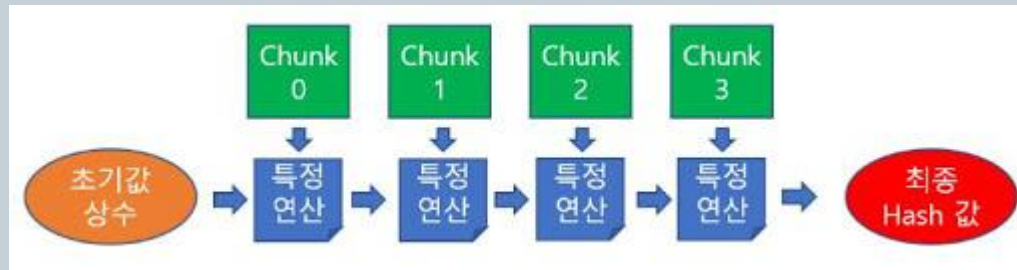
- 메시지 bit 의 길이가 512의 배수가 되도록 padding 을 추가
- 구체적인 규칙은 아래와 같음.
 - 원본 메시지의 바로 뒤에 비트 '1' 을 하나 추가한다.
 - 메시지의 길이가 512의 배수가 되도록 메시지에 0을 추가한다.
 - 메시지의 마지막 64bit에는 원본 메시지의 bit 길이를 적는다.



메시지 처리과정의 개요

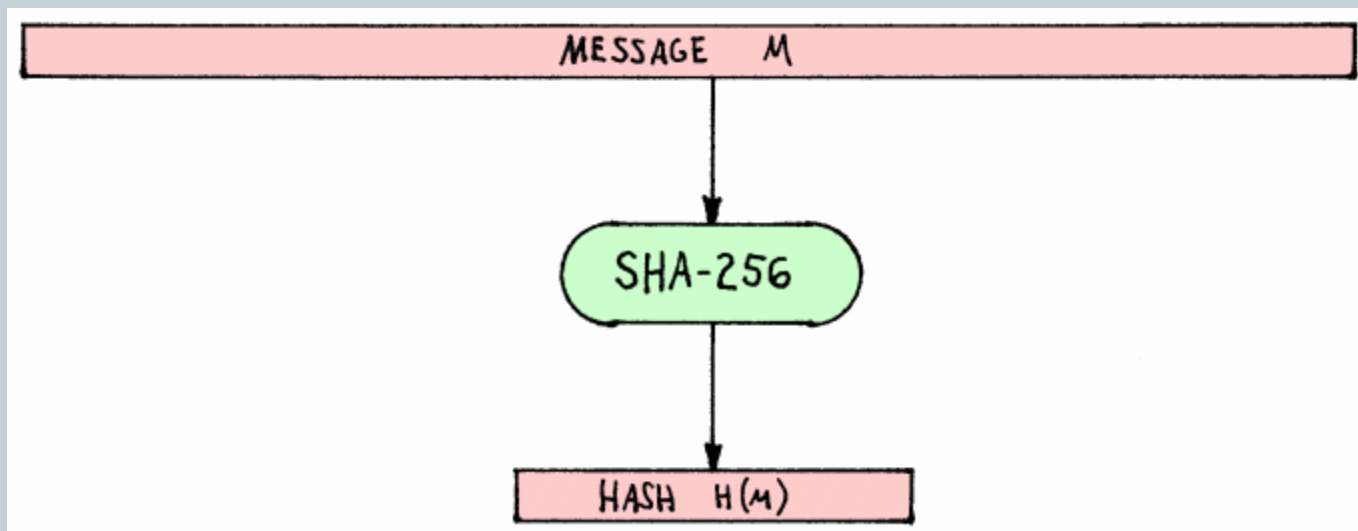


- 메시지 전처리가 끝났다면 메시지의 bit 길이는 512의 배수 형태
- 이러한 전처리된 메시지를 512bit 단위로 쪼개서 여러 개의 chunk 를 생성
- 이러한 chunk 들을 차례대로 순회하면서 특정 연산을 수행하여 최종적인 hash 값을 계산

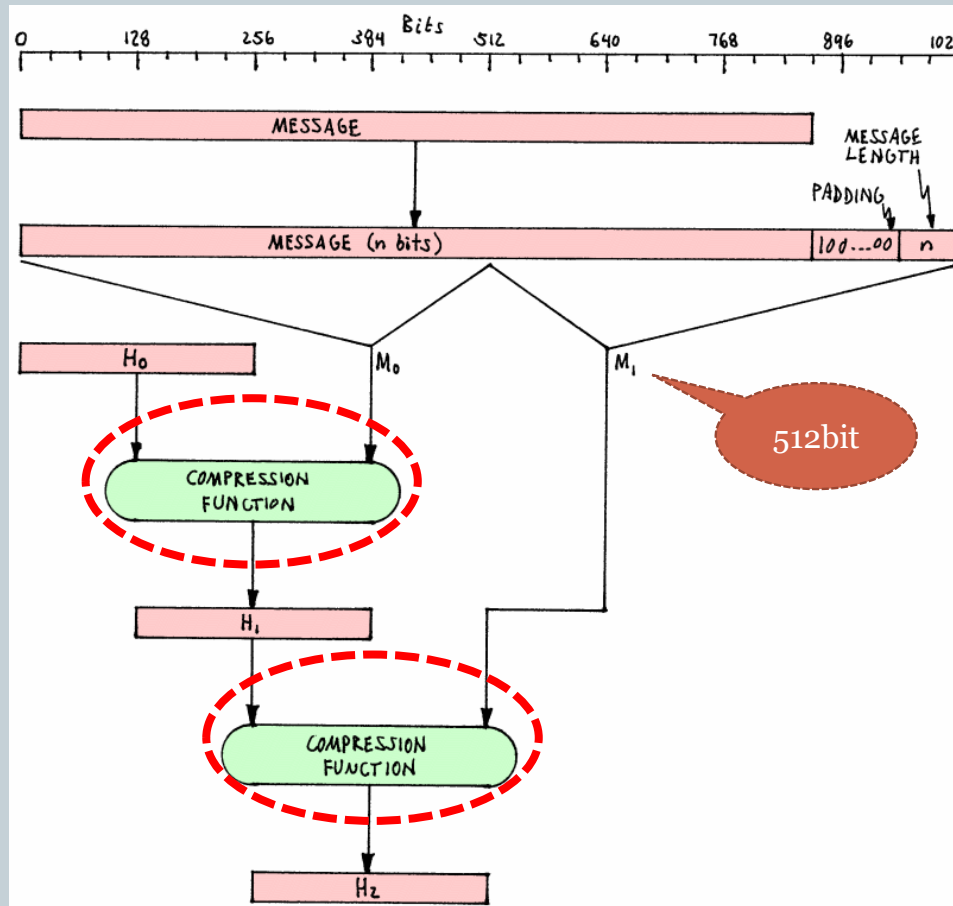


Top-Level Function

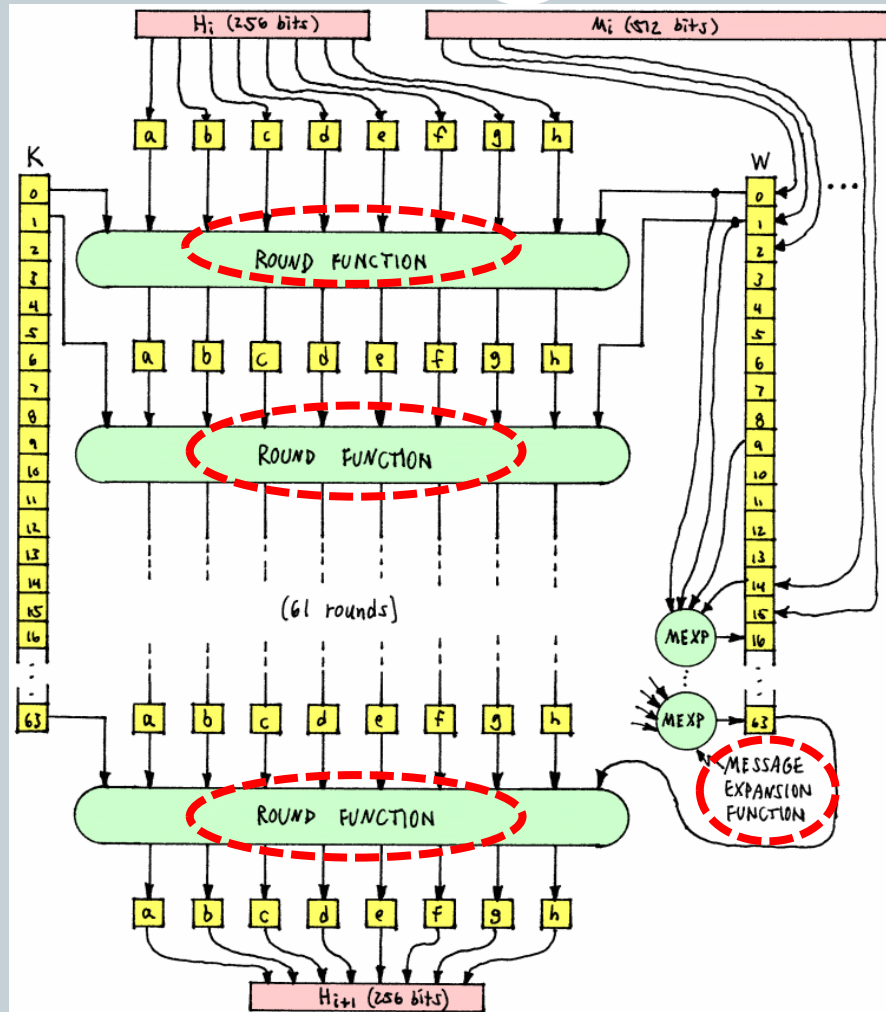
- SHA-256 is a typical **iterated** hash function



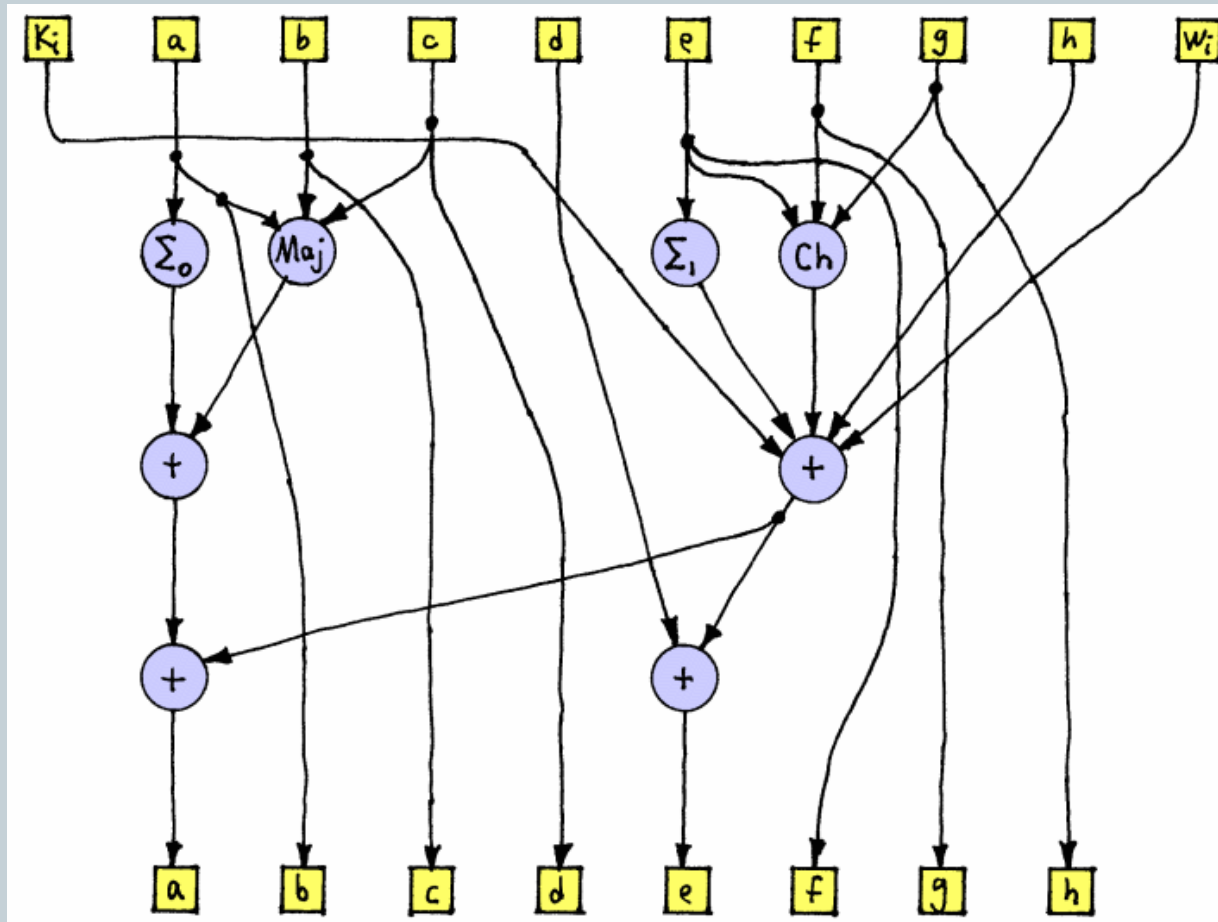
Iterated Hash Function Structure



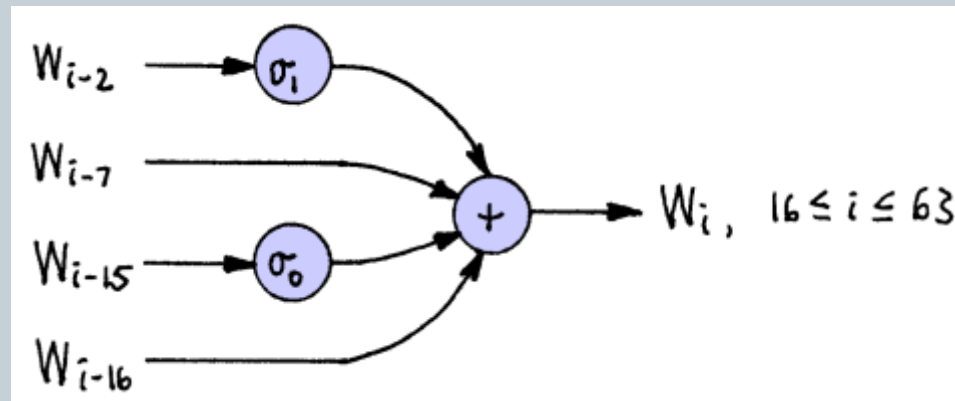
Compression Function



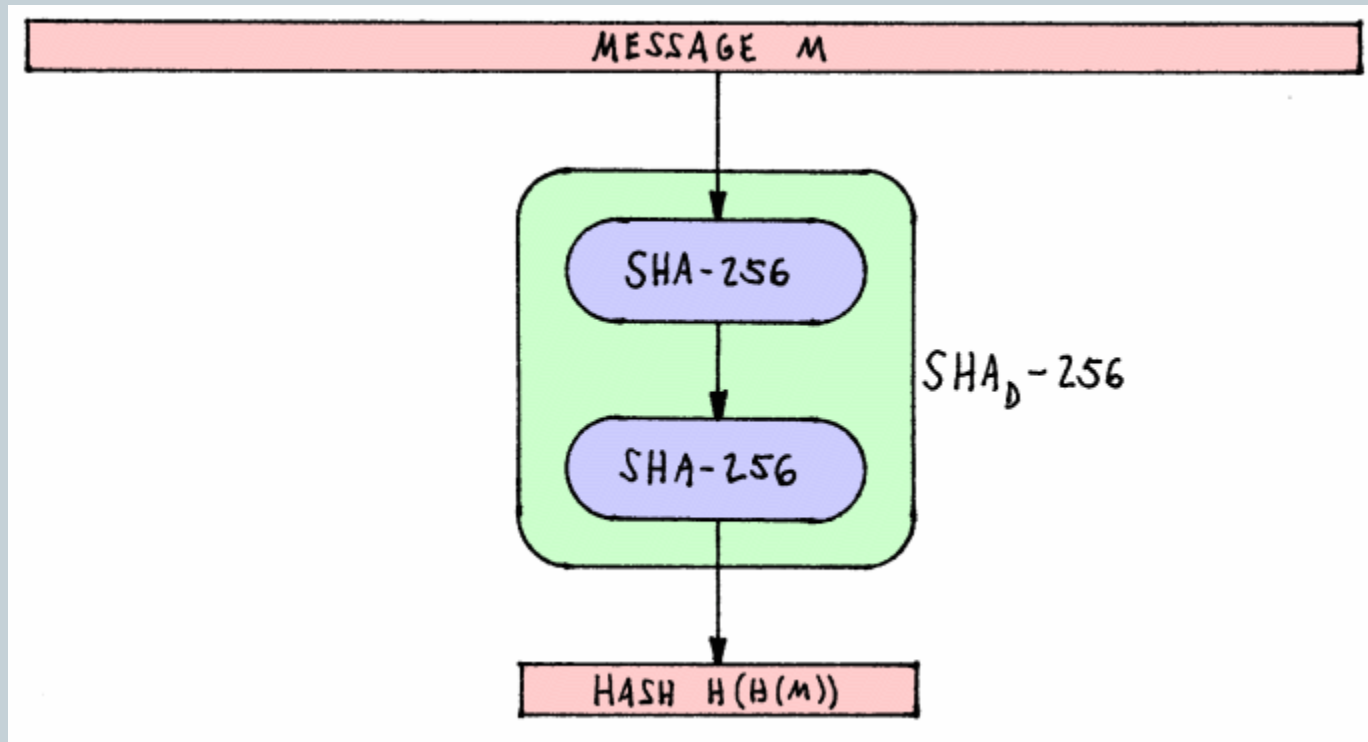
Round Function



Message Expansion Function



Double Hashing



C# 코딩

```
• using System;
• using System.Text;
• using System.Security.Cryptography;

• namespace Backjoon
• {
•     class Program
•     {
•         static void Main()
•         {
•             // 문자열 입력
•             string value = Console.ReadLine();

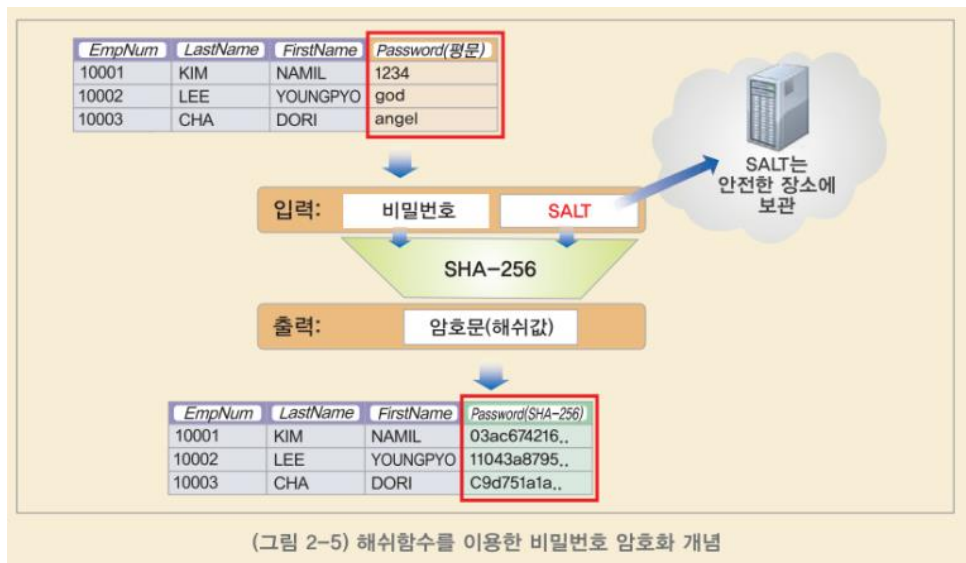
•             // SHA256 해시 생성
•             SHA256 hash = new SHA256Managed();
•             byte[] bytes = hash.ComputeHash(Encoding.UTF8.GetBytes(value));

•             // 16진수 형태로 문자열 결합
•             StringBuilder sb = new StringBuilder();
•             foreach (var item in bytes)
•                 sb.AppendFormat($"{item:x2}");

•             // 문자열 출력
•             Console.WriteLine(sb.ToString());
•         }
•     }
• }
```

안전한 비밀번호 저장

- 비밀번호를 저장하는 두 가지 방법
 - 단순 텍스트(plain text) 저장
 - 단방향 해시 함수의 다이제스트(digest) 저장



- 어느 게 안전할까?

단방향 해시 함수의 문제점(1)



- 레인보우 공격(rainbow attack)
 - 레인보우는 사전처리(pre-computing)된 다이제스트를 말함
 - 이를 가능한 한 많이 확보한 다음 이를 탈취한 다이제스트와 비교해 원본 메시지를 역추적
 - 이와 같은 다이제스트 목록을 레인보우 테이블(rainbow table)

단방향 해시 함수의 문제점(2)



- Brute-Force 공격

- 해시 함수는 암호학에서 널리 사용되지만 원래 패스워드를 저장하기 위해서 설계된 것이 아니라 짧은 시간에 데이터를 검색하기 위해 설계된 것
- 해시 함수의 빠른 처리 속도로 인해 공격자는 매우 빠른 속도로 임의의 문자열의 다이제스트와 해킹할 대상의 다이제스트를 비교할 수 있음
- MD5를 사용한 경우 일반적인 장비를 이용하여 1초당 56억 개의 다이제스트를 대입할 수 있음.

단방향 해시 함수 보완하기(1)



- 대안

- 충분히 길고 복잡한 패스워드 사용

- ✦ 패스워드가 짧거나 복잡하지 않은 경우에는 레인보우 공격에 취약
 - ✦ 충분히 복잡하고 긴 패스워드 사용

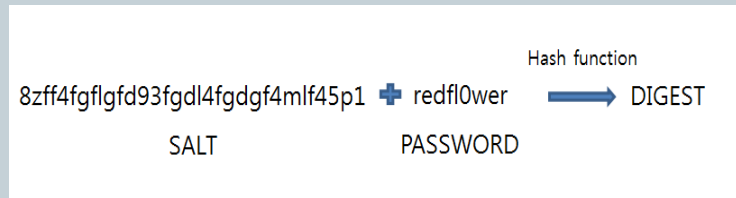
- 연산 시간의 연장

- ✦ 사용자는 패스워드를 인증하는 데 걸리는 시간에는 그리 민감하지 않다.
 - ✦ 사용자가 로그인하기 위해 아이디와 패스워드를 입력하고 확인 버튼을 누르는 과정에 10초가 걸린다고 가정했을 때, 다이제스트를 생성하는 데 0.1초 대신 1초가 소요된다고 해서 크게 신경 쓰는 사람은 많지 않다.
 - ✦ 즉, 해시 함수의 빠른 처리 속도는 사용자들보다 공격자들에게 더 큰 편의성을 제공하게 된다.

단방향 해시 함수 보완하기(2)



- 솔팅(salting)
- 솔트(salt)는 다이제스트를 생성할 때 **추가되는** 바이트 단위의 임의의 문자열
- 원본 메시지에 이 문자열을 추가하여 다이제스트를 생성하는 것을 솔팅(salting)
- 예



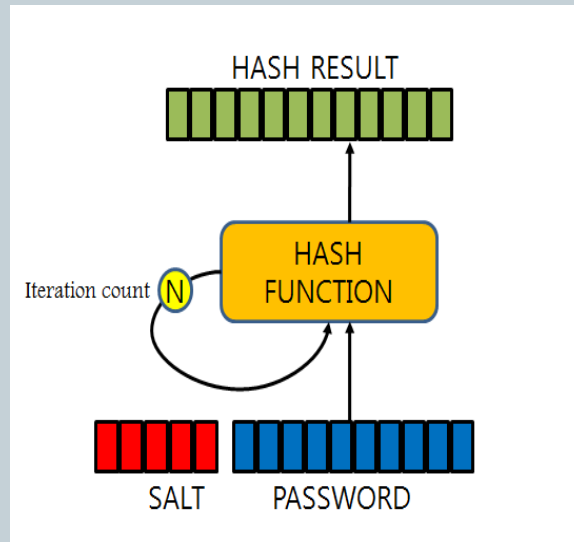
패스워드 "redfl0wer"에 솔트를 추가해 다이제스트 생성

- 공격자가 "redfl0wer"의 다이제스트를 알아내더라도 솔팅된 다이제스트를 대상으로 패스워드 일치 여부를 확인하기 어렵다.
- 또한 사용자별로 다른 솔트를 사용한다면 동일한 패스워드를 사용하는 사용자의 다이제스트가 다르게 생성되어 인식 가능성 문제가 크게 개선된다.
- 솔트와 패스워드의 다이제스트를 데이터베이스에 저장

단방향 해시 함수 보완하기(3)



- 키 스트레칭(key stretching)
- 패스워드의 다이제스트를 생성하고, 이를 다시 입력 값으로 하여 다이제스트를 생성
- 이를 여러 번 반복
- 입력한 패스워드를 동일한 횟수만큼 해시해야만 입력한 패스워드의 일치 여부를 확인할 수 있음
- 하나의 다이제스트를 생성할 때 어느 정도(일반적인 장비에서 0.2초 이상)의 시간이 소요되게 설정
- 이는 brute-force attack으로 패스워드를 추측하는 데 많은 시간이 소요되도록 하기 위한 것



솔팅과 키 스트레칭을 적용하여 다이제스트 생성

권장사항



- 이미 검증된 암호화 시스템을 사용할 것을 권장
 - 널리 알려진 검증된 시스템을 사용하면, 암호화 시스템을 잘못 구현해서 발생하는 위험을 피할 수 있다.
- 자신만의 암호화 시스템을 구현하는 것은 매우 위험
 - 취약점을 확인하기 어렵고, 대부분의 경우 구현된 암호화 시스템을 점검하고 확인하는 사람은 암호화 시스템을 구현한 당사자 한 명이다.
 - 만약 구현한 암호화 시스템에 취약점이 있다면, 많은 사람들이 사용할수록 그만큼 많은 사람들이 피해를 입게 된다.
 - 이런 취약점이 내포된 시스템은 여러 차례 발견되었음

메시지 무결성



- 메시지가 위조되거나 변조되지 않았음을 보장
 - MDC
 - MAC

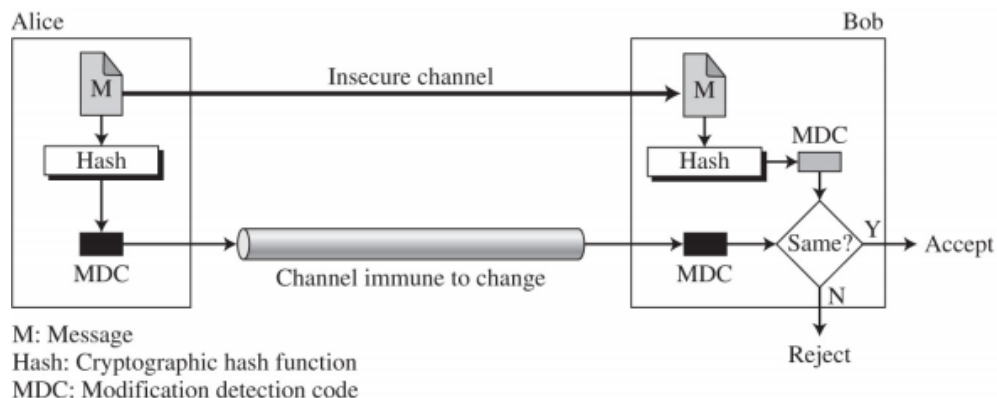
MDC

(Modification Detection Code)



- **MDC(메시지 변경 감지 코드)**
- 메시지의 무결성(메시지가 변하지 않았다는 것)을 보장하는 메시지 다이제스트. 전달된 메시지가 중간에 변조, 위조되었는지를 검사하는 기법
- **MDC절차**
 - 송신자 **A**는 송신할 메시지를 해시함수를 이용하여 메시지 다이제스트를 만듦.
 - 이때 생성된 메시지 다이제스트를 일반적으로 **MDC**라고 부름.
 - 송신자 **A**가 메시지와 **MDC**를 수신자 **B**에게 보냄.
 - 수신자 **B**는 받은 메시지로 **MDC**를 만들고, 송신자 **A**가 보낸 **MDC**와 비교하여 메시지의 무결성을 확인함.

MDC절차



No	절차	설명
1	해시 알고리즘 선정	■ Alice와 Bob은 해시에 사용할 해시 알고리즘 선정
2	Alice MDC	■ Alice는 원문(Message)을 해시
3	Bob에게 전달	■ Alice는 원문과 그 해시 결과(MDC)를 Bob에게 전달
4	Bob MDC	■ Bob은 원문을 Alice와 같은 알고리즘으로 해시한 결과(Bob의 MDC)를 생성
5	MDC값 비교	■ Bob이 만든 MDC와 Alice가 만든 MDC가 같다면 원문은 변경되지 않았고 신뢰 수 있는 상태

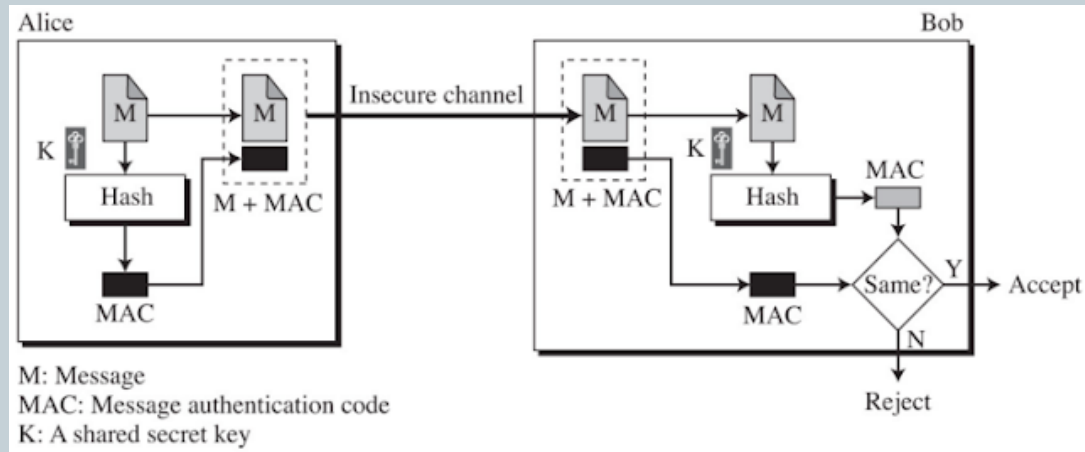
- Bob은 수신한 메시지에서 새로운 MDC를 생성하여 Alice로부터 수신된 MDC와 비교, 두 값 동일 시 무결성 보장
- MDC는 메시지 무결성은 보장하나 인증된 사용자로부터 수신된 메시지라는 것을 보장하지 못함. MAC를 통해 메시지 인증 필요

MAC (Message Authentication Code)



- MAC의 개념
 - MDC는 송신자의 인증여부를 알 수 없다
 - MAC은 해시함수 + 대칭키로 메시지 무결성을 인증하고 거짓행세를 검출
 - MAC
 - 임의 길이의 메시지와 송신자 및 수신자가 **공유하는 키**를 이용하여 고정 비트길이의 출력을 만드는 함수.
- MAC의 안전성
 - MAC의 안전성은 사용되는 해시함수의 안전성과 같음.
- SSL/TLS에서 사용됨
 - 온라인 쇼핑 시 사용되는 통신 프로토콜
 - 인증, 무결성 확인을 위해 MAC을 사용

MAC절차

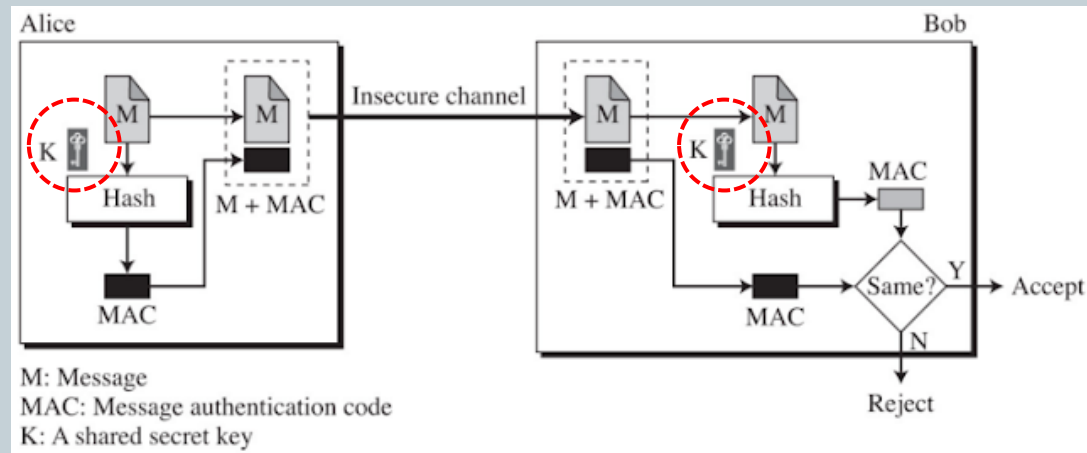
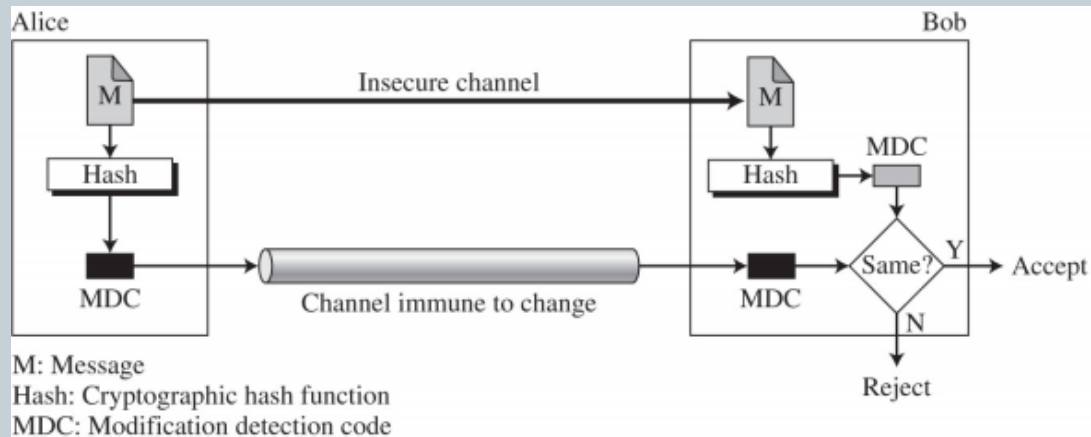


키 K는 오직
송,수신자만 알고 있음

No	절차	설명
1	Shared Key 공유	<ul style="list-style-type: none"> 별도 채널을 이용하여 Alice와 Bob은 해시에 사용할 키(Shared Key)를 공유 해시 알고리즘 선정
2	Alice MAC	<ul style="list-style-type: none"> Alice는 Shared Key를 사용해서 원문(Message)을 해시
3	Bob에게 전달	<ul style="list-style-type: none"> Alice는 원문과 그 해시 결과(MAC)를 Bob에게 전달
4	Bob MAC	<ul style="list-style-type: none"> Bob은 원문을 Shared Key를 사용하여 같은 알고리즘으로 해시한 결과(Bob의 MAC) 생성
5	MAC값 비교	<ul style="list-style-type: none"> Bob이 만든 MAC과 Alice가 만든 MAC이 같다면 원문은 변경되지 않았고 신뢰할 수 있는 상태

- 메시지 인증(Message Authentication) : 메시지가 인증된 사용자인 Alice로 왔다는 것을 보장 가능
- MAC의 목적은 메시지 무결성 뿐만 아니라 메시지가 인증된 사용자로부터 왔다는 것을 보장하는 것에 목적이 있다는 것에 주의 해야 함
- Shared Key와 해싱 기법을 적용하여 메시지 무결성은 물론 Alice가 메시지의 원 전송자임을 보장

MDC/MAC 비교



키 K 는 오직
송,수신자만 알고 있음