Security

21 겨울 휠세미나

jungnoh (노정훈)

목차

- Cryptography
  - Symmetric Encryption: Block ciphers, Modes of operation
  - Asymmetric encryption: RSA,
  - Hash functions, MAC, Signatures
- TLS
- Linux Security
- 웹 서비스에서 주의해야 할 점들
  - 비밀번호 저장
  - Input sanitization
  - Secret 관리

보안의 3요소 - CIA

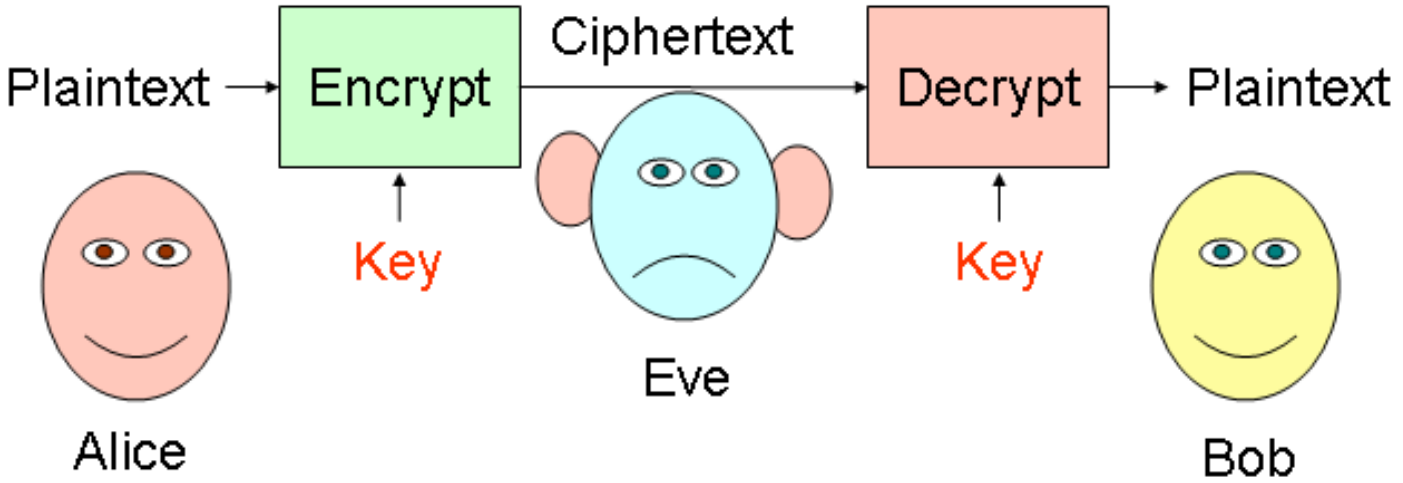
- **Confidentiality** (기밀성)
  권한이 있는 사람에게만 접근이 가능하게 하는 것
  ex) 아이디+비번, 생체인증 (지문)

- **Integrity** (무결성)
  데이터가 옮겨지는 중에 바뀌어서는 안되고, 권한이 없는 사람에 의해 바뀌어서도 안됨
  ex) 파일 퍼미션, TLS

- **Availability** (가용성)
  접근성이 용이해야 함.
  ex) 업데이트, 백업

암호학

- 암호학 (Cryptography): **제 3자가 존재하는 상황**에서 안전한 (제 3자가 들을 수 없는) 통신을 하는 방법을 연구하는 학문



- 대칭키 암호화: 암호문을 읽고 쓸 때 쓰는 비밀 정보가 같다.
- 비대칭키 암호화: 암호문을 읽고 쓸 때 쓰는 비밀 정보가 다르다.

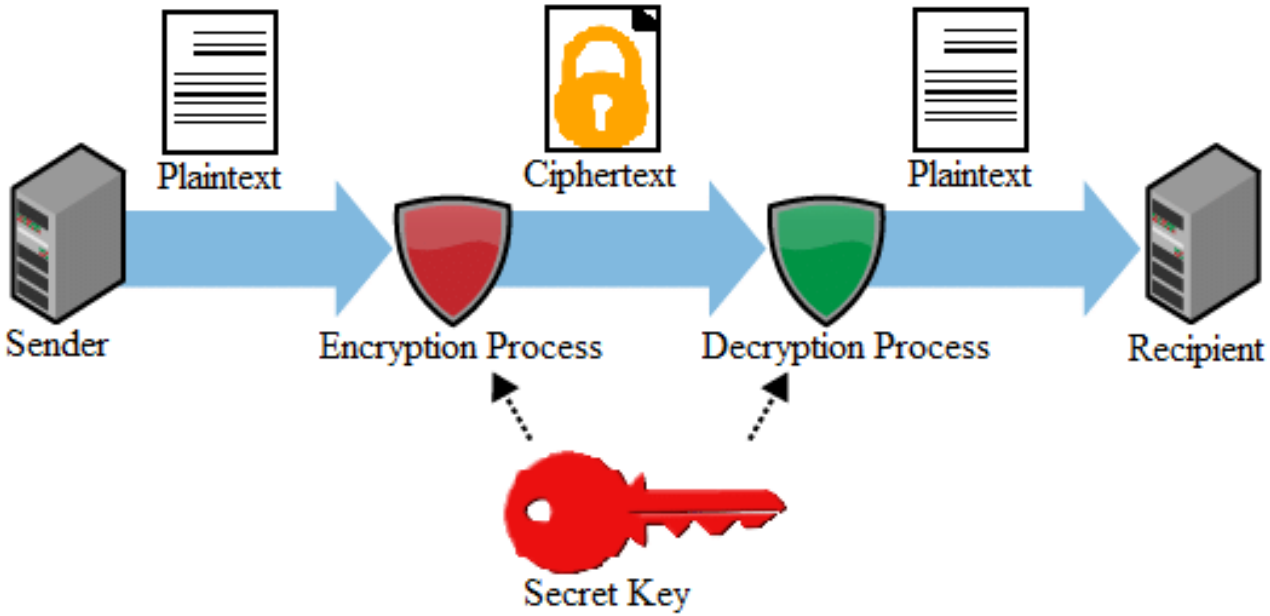
블럭 암호 (Block Ciphers)

용어

- 평문 (plaintext): 원본 데이터, 암호화하고 싶은 것
- 암호문 (ciphertext): 암호 알고리즘으로 평문을 암호화한 것
- 키 (key): 암호 알고리즘의 동작을 결정짓는 매개변수 (=비밀)

블럭 암호: 평문의 길이가 고정된 대칭 키 알고리즘

- 사전에 Secret Key를 공유
- 암호 알고리즘의 보안성은 key의 기밀성에 의존한다.



Block Ciphers

수학적인 정의: 블럭 암호 알고리즘 $\{E, D\}$

$$E_K(P) = \{0,1\}^k \times \{0,1\}^p \rightarrow \{0,1\}^p, D_K(C) = \{0,1\}^k \times \{0,1\}^p \rightarrow \{0,1\}^p$$

- 길이 k의 비밀키 K, 길이 p의 암호화할 데이터 (평문) P
- $C = E_K(P)$가 주어졌을 때, K를 모른다면 P를 구하기가 매우 어렵다.
- K를 안다면 $D_K\big(E_K(P)\big) = P$

한마디로 $E_K(P)$는 $\{0,1\}^p$에 대한 permutation이다. 근데 이제 예상을 하기 힘든..

가장 기초적인 블럭 암호

- XOR 연산: 평문과 비밀키를 XOR해서 암호문을 만든다.

$$m = 0101\ 0101,\ p = 1111\ 0000 \rightarrow m \oplus p = 1010\ 0101$$

- p를 모르면 $m \oplus p$에서 m을 구할 수 없다. p를 안다면? $m \oplus p \oplus p = m$

- 과연 안전할까?
- $c_1 = m_1 \oplus p, c_2 = m_2 \oplus p$를 알고 있다면,

$$c_1 \oplus c_2 = m_1 \oplus p \oplus m_2 \oplus p = m_1 \oplus m_2$$

(XOR은 교환법칙이 성립, $p \oplus p = 0$)

- 우리가 m_1을 선택할 수 있다면 m_2를 알아낼 수 있다.

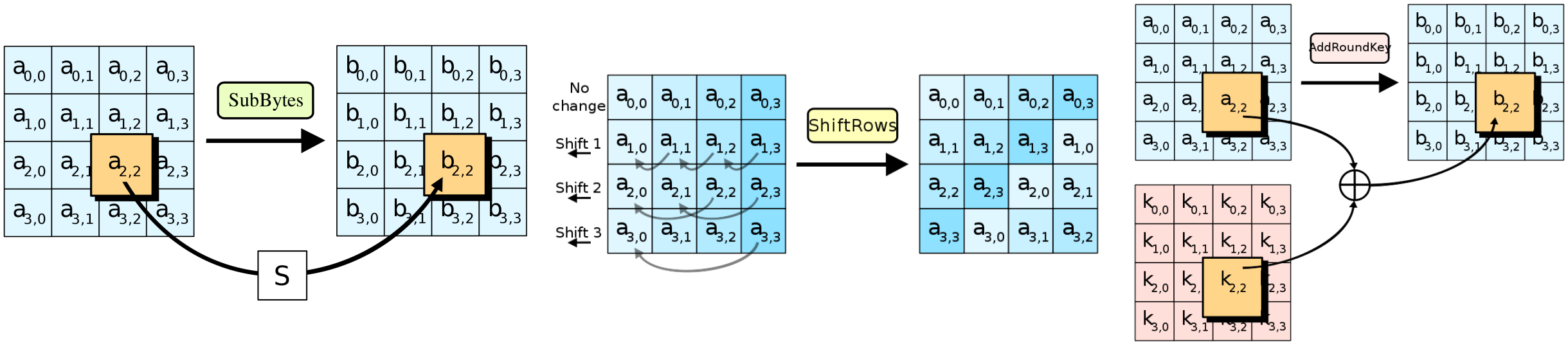
블럭 암호

- DES: 64비트씩 암호화, 키 길이 56비트
  - 3-DES (Triple DES): DES를 세번 실행, $c = DES(DES(DES(m, p), p), p)$
- AES: 128비트씩 암호화, 키 길이 128, 192, 256비트

- 국내 블록 암호: SEED, ARIA 등 (AES와 유사 구조)

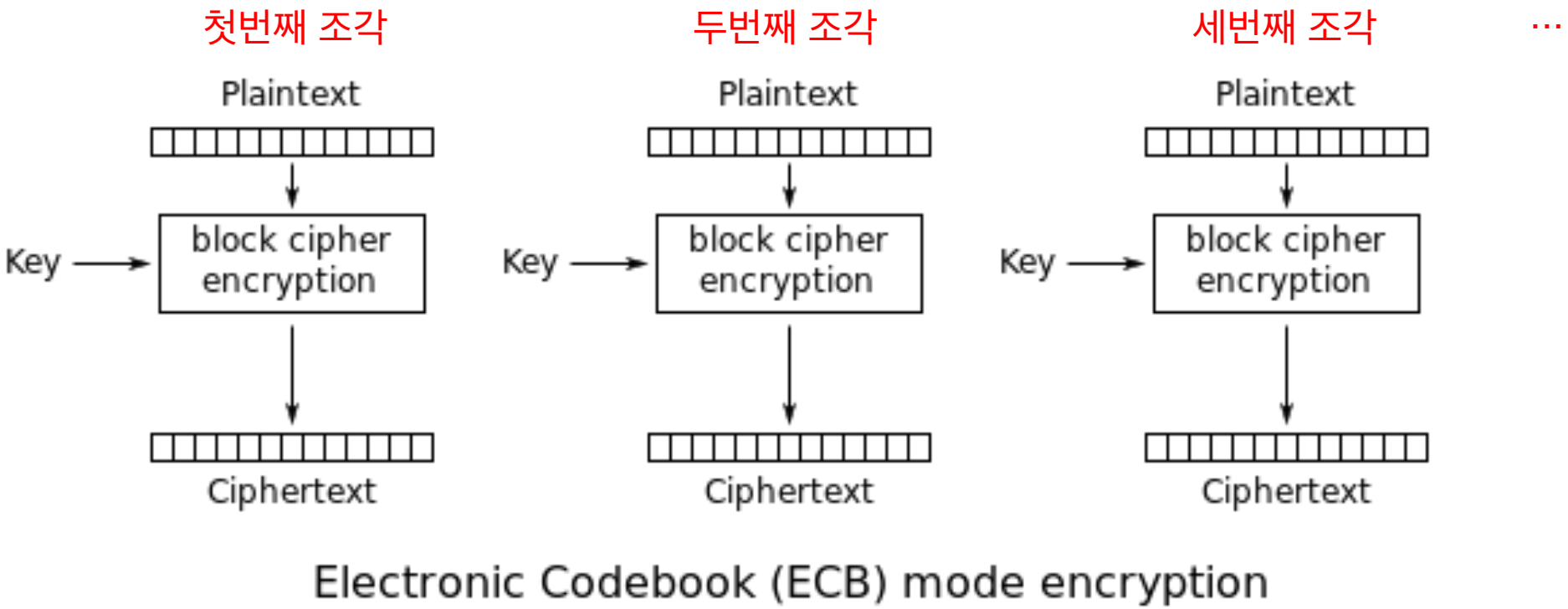
AES

- 1998년 미국 NIST 공모에서 우승한 알고리즘
- 현재 가장 안전하고, 널리 사용되고 있는 블럭 암호
- 계속되고 있는 공격
  - AES-256, AES-512: 각각 $2^{99.5}$, $2^{254.4}$의 시간으로 암호문을 풀 수 있다 (2009)



운영 모드

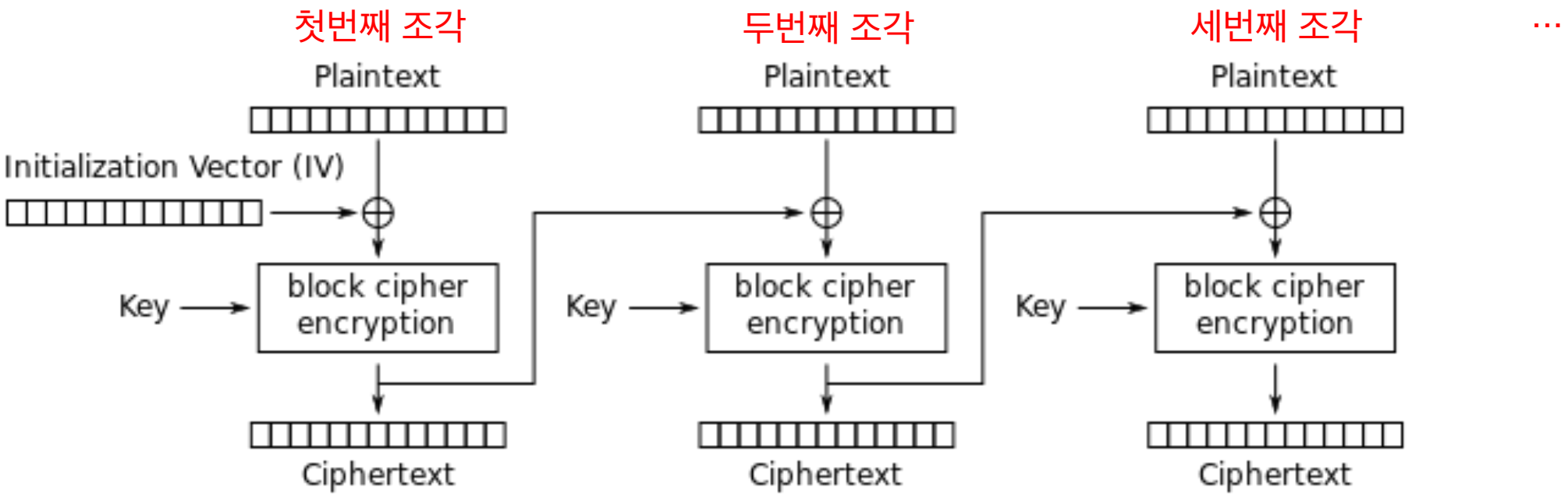
- 긴 파일을 암호화하고 싶다면? 파일을 쪼개 암호화한다.
- ECB: 단순히 쪼개 암호화



Electronic Codebook (ECB) mode encryption

운영 모드

- CBC: 앞 조각을 섞어서 암호화한다.
- Initialization Vector (IV): 맨 첫번째 조각에 섞일 내용
  - 기억하고 있어야 하고, 비밀키처럼 유출되면 안되는 정보는 아님



Cipher Block Chaining (CBC) mode encryption

운영 모드

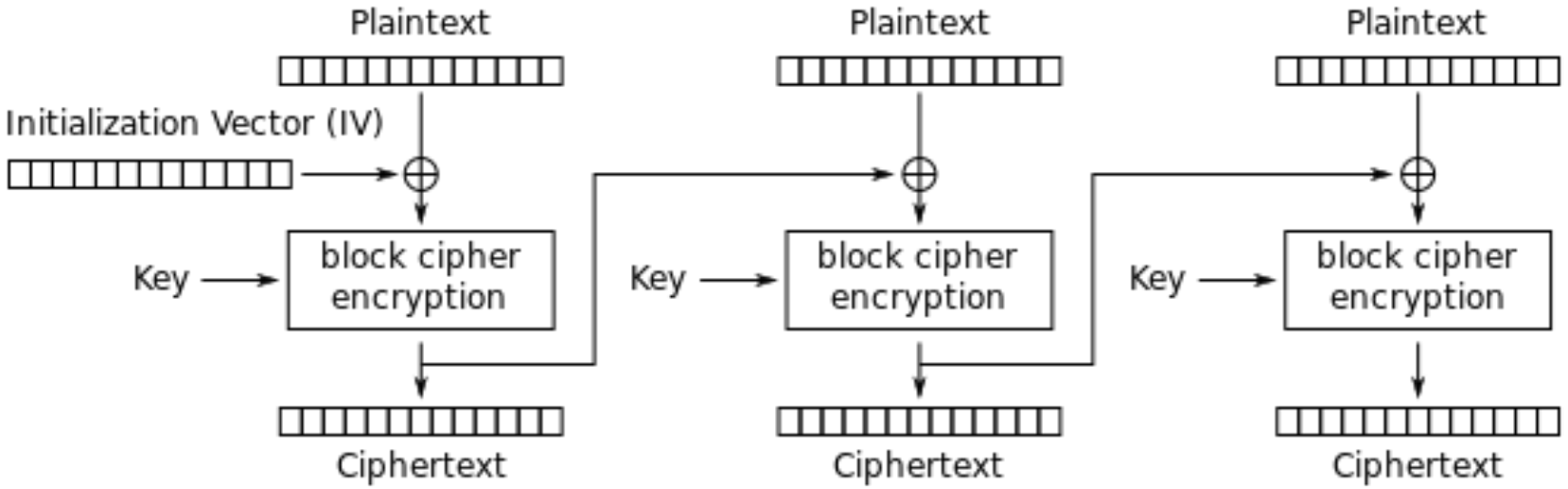
- 왜 여러가지 모드를 사용할까?

Original

ECB

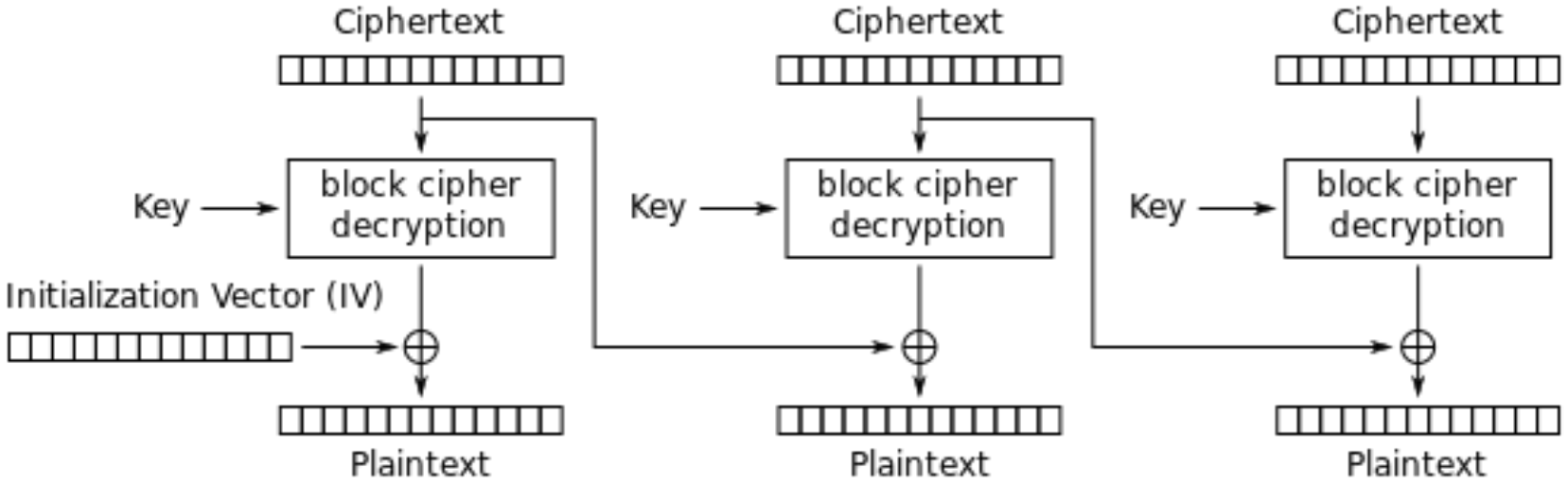
CBC

안전한 운영 모드를 사용하는 것도 매우 중요하다



Cipher Block Chaining (CBC) mode encryption

$$C_1 = E_K(P_1 \oplus IV)$$
$$P_1 = D_K(C_1) \oplus IV$$



Cipher Block Chaining (CBC) mode decryption

생각해볼 것

(1) 병렬화가 가능할까?

(2)암호문이 고장나면?

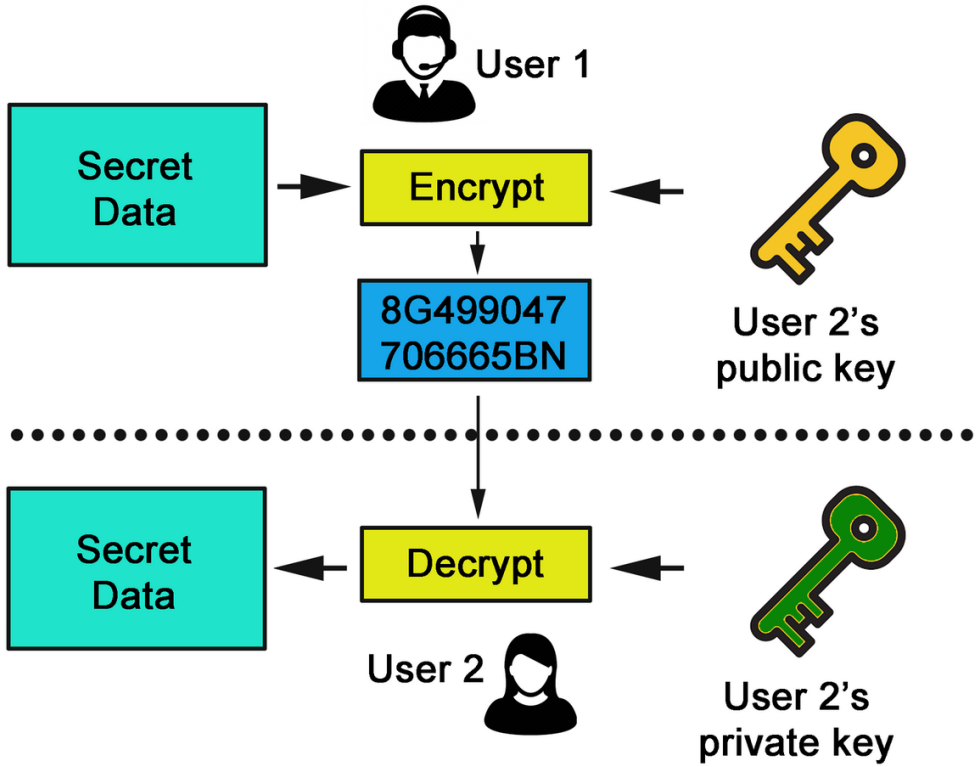
운영 모드

- 프로덕션에서는 GCM (Galois/Counter Mode)를 사용하자.
  - https://en.wikipedia.org/wiki/Galois/Counter_Mode

- 안전한 운영 모드를 사용하는 것은 매우 중요하다!!
- 블럭 암호 알고리즘 + 운영 모드가 한 세트를 이룬다.
  - ex) AES-256-GCM: 256비트 AES 암호 알고리즘, GCM 운영 모드

비대칭키 암호화

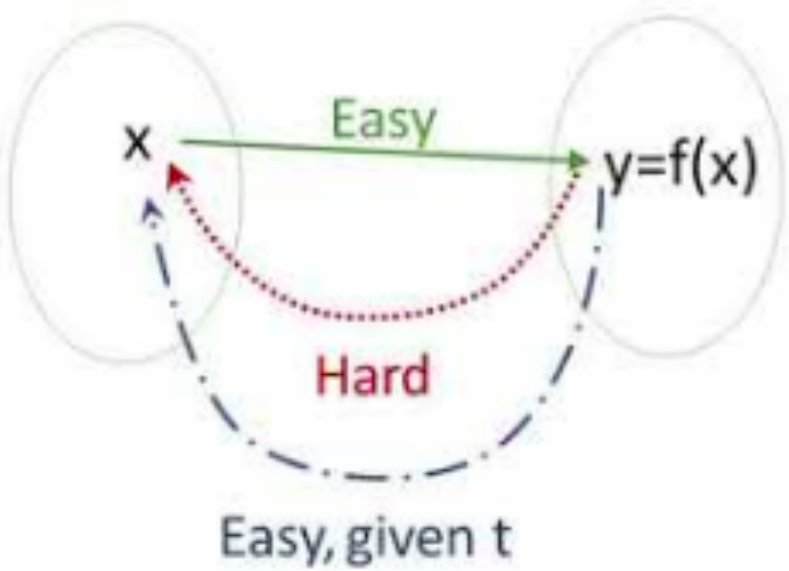
- 암호화하는 사람: 여러 명 (누구나)
  복호화하는 사람: 한 명
- (암호화 키) != (복호화 키)여야 한다!
- 암호화: 공개키 (public key) / 복호화: 비밀키 (private key)



비대칭키 암호화

- Trapdoor function: 한 방향으로 가는 것은 쉽지만, 정보가 없으면 역방향은 매우 어려운 함수

안에서 문을 닫으면 키 없이 들어가기 매우 힘들다
안에서는 문을 열기만 하면 쉽게 나갈 수 있다



공개키: 문을 안에서 열고 나가기 위한 정보
비밀키: 밖에서 문을 열기 위한 키
공개키 + 비밀키가 키 쌍이 된다.

RSA

- 아주 큰 수의 인수분해가 어렵다는 점을 Trapdoor function으로 활용!

함수 $\phi(n)$: n보다 작은 수 중 n과 서로소인 수의 개수. 소수 p, q에 대해 $\phi(pq) = (p-1)(q-1)$이다.

오일러 정리: 자연수 n과 서로소인 a에 대해 $a^{\phi(n)} \equiv 1 \ (mod\ n)$

잉여 역원: 정수 $e, 0 \leq d < n$에 대해 $e \cdot d \equiv 1 \ (mod\ n)$ 이면 d 를 n에 대한 e의 잉여 역원이라고 한다.

a^e를 알고 있다면, $a \equiv ? \ (mod\ n) \rightarrow e, \phi(n)$을 안다면 쉽게 구할 수 있다. → $\phi(n)$를 구하기 매우 어렵게 하고, $\phi(n)$에 대한 정보를 비밀키로 쓰자! → $e \cdot d \equiv 1 \ (mod\ \phi(n))$에서 d가 암호키.

결론) 공개키: (e, n) 쌍, 비밀키: (d, n) 쌍 $\quad a^{ed} \equiv \left(a^{\phi(n)}\right)^q \cdot a \equiv 1^q \cdot a \equiv a \ (mod\ n)$

$\phi(n)$를 n에서 구해버리면 어떡하지? → 아주 큰 소수 p, q를 고르고, $n = pq$로 하자.

→ p, q를 알고 있다면 $\phi(n)$을 구하기 매우 쉽지만, n에서 바로 구하기에는 아주아주 어렵다 (연산시간상)

RSA

- **키 생성 과정**
  (1) 아주 큰 소수 p, q를 미리 찾아둔다. $n = p \cdot q$
  (2) $1 < e < \phi(n) = (p-1)(q-1)$이고 n과 서로소인 e를 고른다.
  (3) $\phi(n)$에 대한 e의 잉여 역원 d를 구한다.
  (4) (e, n)은 공개키 쌍으로 뿌리고, (d, n)는 비밀키 쌍으로 내가 갖고 있는다.
- **암호화**
  평문 m에 대해 $c = m^e \ (mod\ n)$을 구한다. c가 암호문이다.
- **복호화**
  암호문 c에 대해 $m = c^d \ (mod\ n)$을 구한다. m이 암호문이다.

안전하고, 암호화+복호화 연산은 로그 시간 안에 가능해 매우 빠르다.

RSA + OAEP

- RSA 혼자서는 충분히 안전하지 못하다

  평문 m_1, m_2가 동일하다면, 이들의 암호문 $C_1 \equiv m_1^e = m_2^e \equiv C_2 \ (mod\ n)$

  → 메세지 내용은 모르더라도, 두 메세지가 같은지는 판단할 수 있다.

- Deterministic한 단점을 보완하는 것이 OAEP (Optimal Asymmetric Encryption Padding)

$OAEP - pre(m)$:
r =랜덤 값
$X = (m||000 \ldots 0) \oplus H(r)$
$Y = r \oplus H(X)$
결과는 $X||Y$

$OAEP - post(c)$:
c를 X, Y로 쪼갬
$r = Y \oplus H(X)$
$m|000 \ldots 0 = X \oplus H(r)$
결과는 m

$Encrypt_K(m) = E_K(OAEP - pre(m))$
$Decrypt_K(m) = OAEP - post(D_K(m))$

랜덤값 (r, nonce)가 들어가기 때문에
문제 해결!

타원 곡선 공개키 알고리즘 (Elliptic Curve ElGamal)

- 이산 로그 문제 (Discrete Logarithm Problem)를 Trapdoor function으로 이용

이산 로그 문제: $a = g^x \ (mod\ n)$일 때, x를 찾는 것이 매우 어렵다.

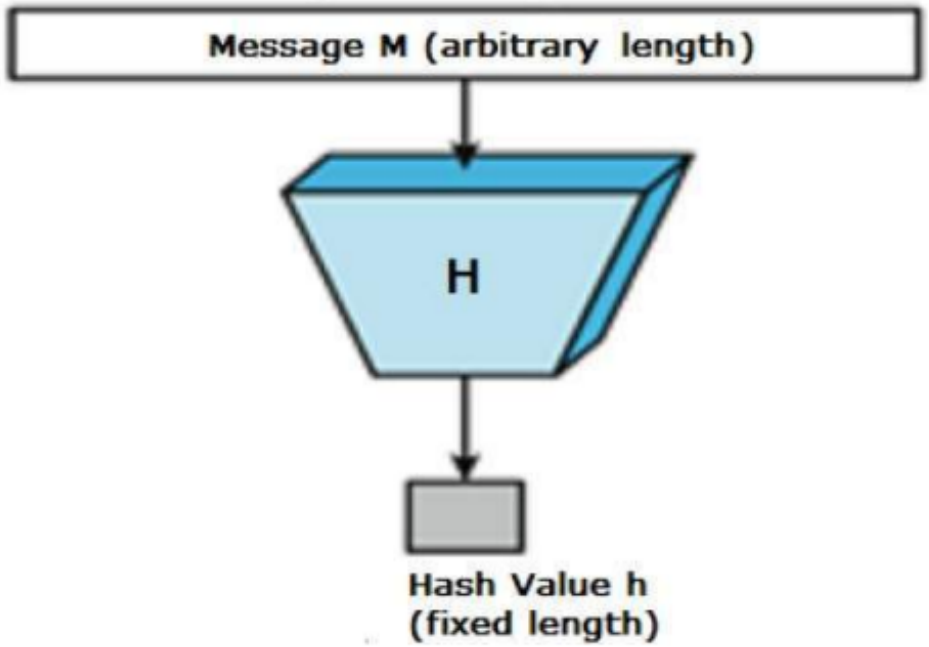
- 타원 곡선과의 연결... 생략
- 이 문제를 이용한 것이 타원 곡선 암호화 알고리즘 (ElGamel)
- RSA 대비 연산이 빠르고, 같은 키 길이에서도 더 안전하다.

Table 1. Comparable Key Size (in bits) [3]

Symmetric Algorithms	ECC	RSA
80	163	1024
112	233	2240
128	283	3072
192	409	7680
256	571	15360

해쉬 함수

- DS에서 배우는 것
- 가변적인 크기의 데이터에 mapping 되는 고정된 크기의 값

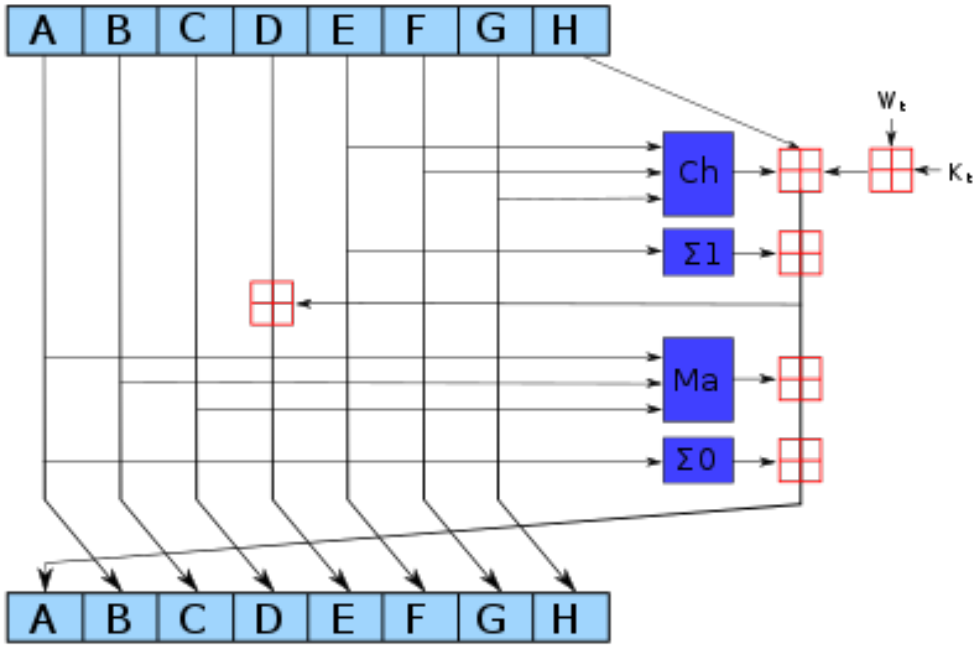


암호학적 해쉬 함수

- 암호학적 해쉬 함수 H는 다음 조건을 만족해야 한다.
  - 해시값 h이 주어졌을 때, $\mathrm{h} = H(m)$인 m을 찾는 것이 매우 어렵다.
  - 메세지 m_1이 주어졌을 때, $H(m_1) = H(m_2)$인 m_2를 찾는 것이 매우 어렵다.
  - $H(m_1) = H(m_2)$인 쌍 m_1, m_2를 찾는 것이 매우 어렵다.

- 이런 효과를 만들기 위해 구조가 복잡함
  - Merkle-Damgard 구조 (압축)
  - Diffusion

- 용도: 비밀번호 저장, 전자 서명 등



SHA-2 알고리즘의 한 조각

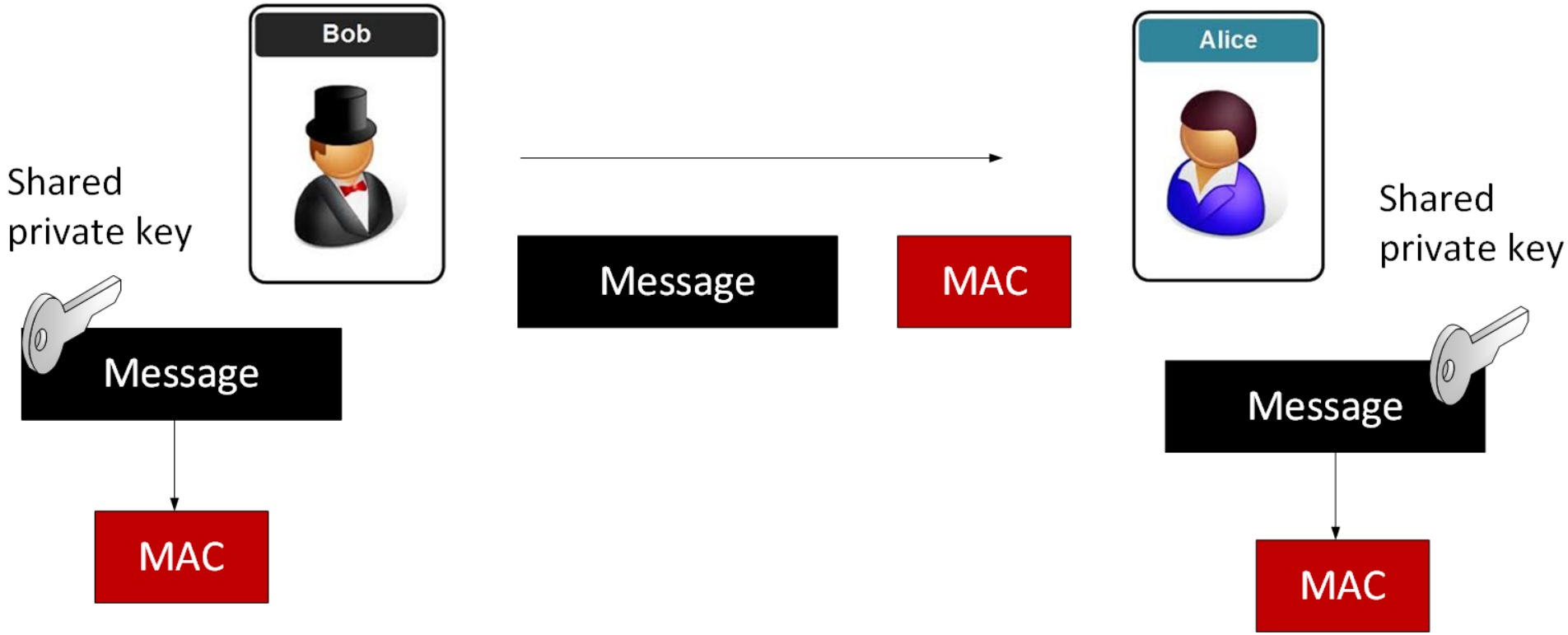
해쉬 함수

- 많이 쓰이는/쓰이던 알고리즘

- MD5 → 뚫림
- SHA-1 → 뚫림
- SHA-2 (SHA-256, SHA-384, SHA-512)
- SHA-3
  - 아직 CPU 하드웨어 가속 지원이 없음 (armv8만 지원)

MAC

- 메세지 인증 코드 (Message Authentication Code)
- (메세지 내용 + 비밀키)로 만든 인증코드를 메세지와 함께 전송
- 수신자는 MAC을 다시 계산해 메세지가 변조되었는지 확인할 수 있다.
  - 비밀키를 알고 있어야만 MAC을 만들 수 있음 → 상대방이 안전한 사람인지 (위조 여부를) 확인 가능



HMAC

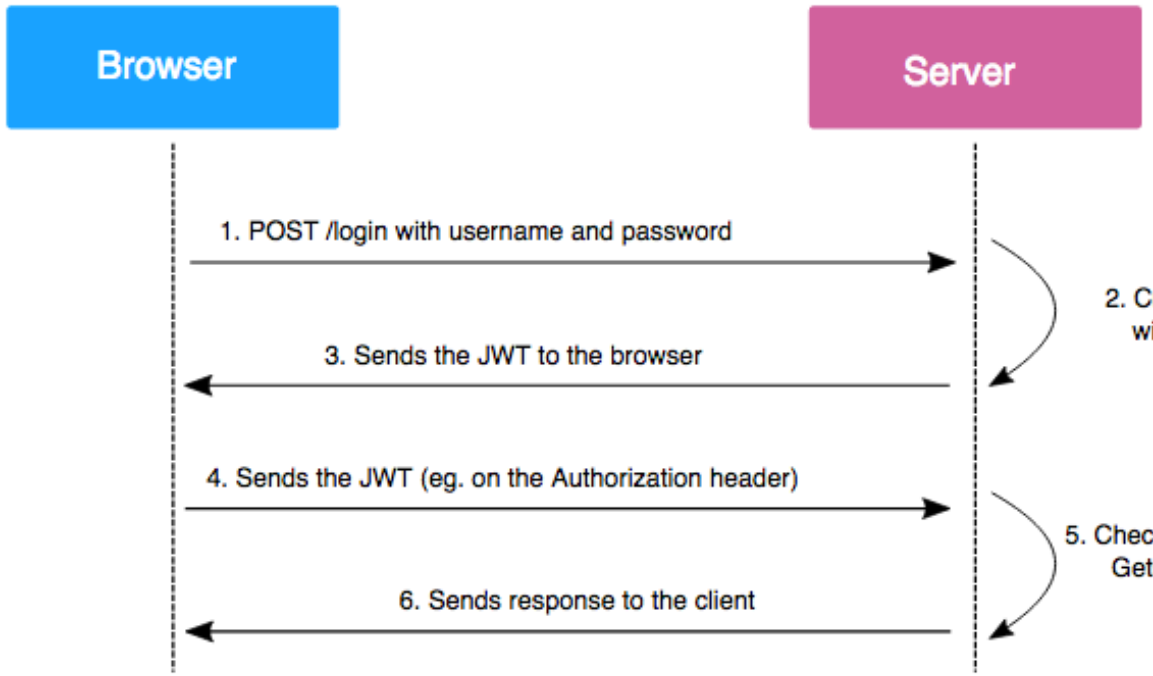
- Hash-based MAC
- 해쉬함수를 이용해 만드는 MAC. 메세지 m, 비밀키 k가 있다면

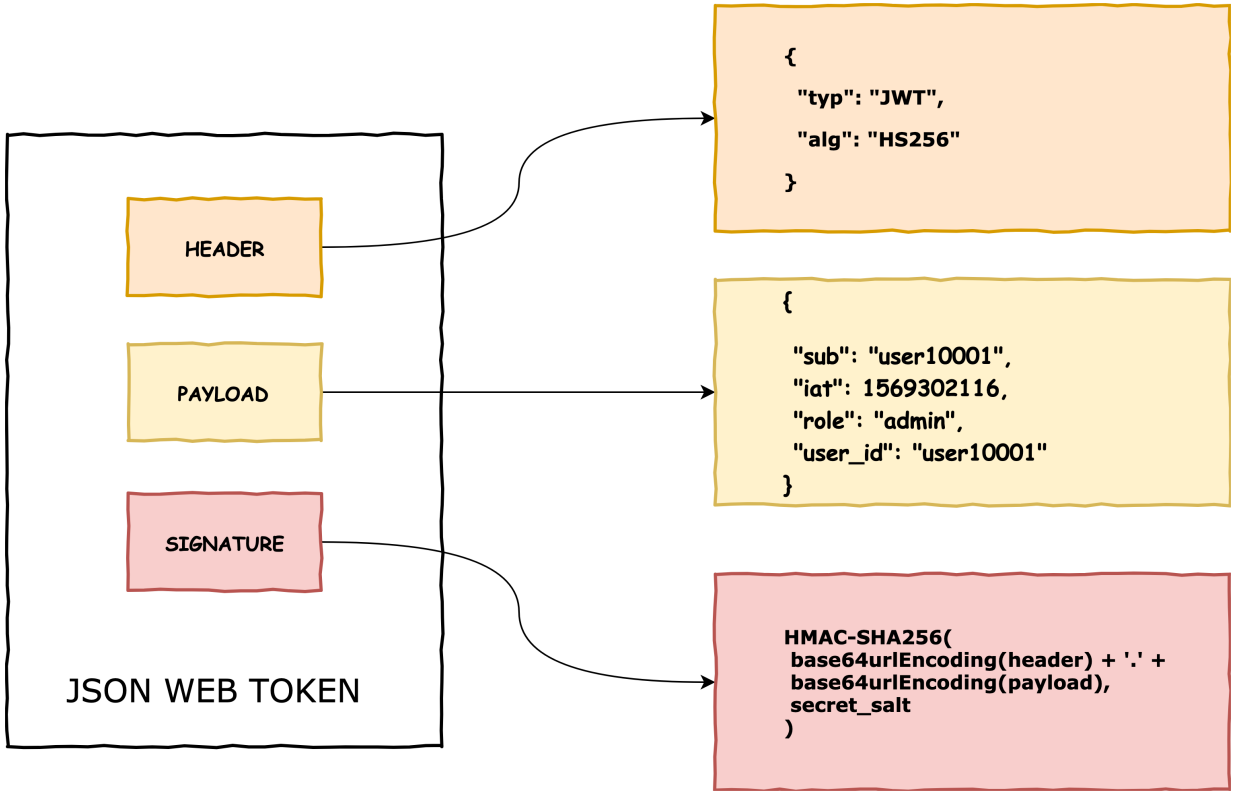
$$MAC = H(H(m||k)||k)$$

- 왜 해쉬를 두번 할까?
  - Extension Attacks

- 개발에서 쓰이는 HMAC: JWT (JSON Web Token)

JWT JSON Web Token



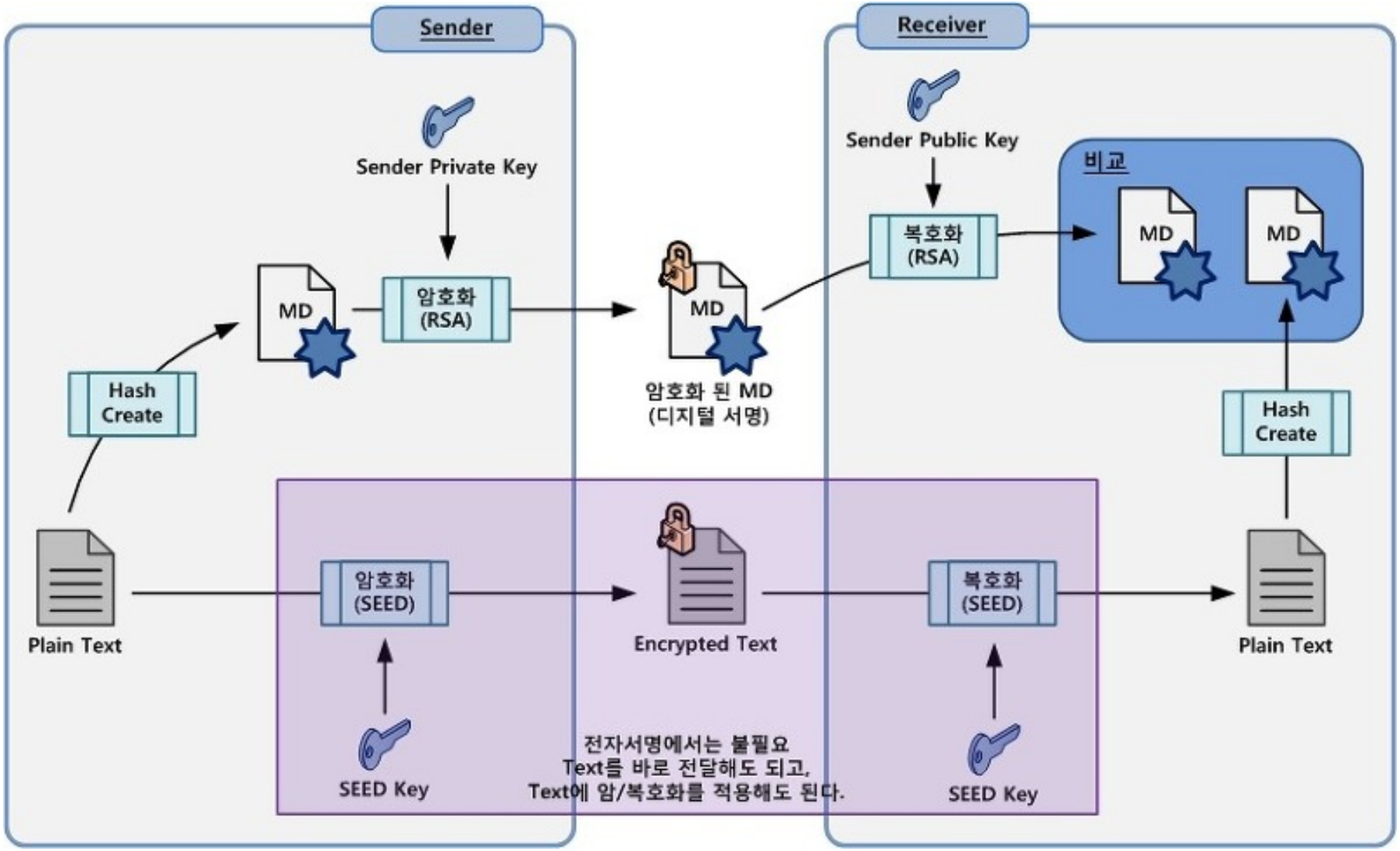


전자 서명

- MAC의 단점
  - 비밀키가 있다면 누구나 가능 → 누가 만든 지 모름, 안 만들었다 우길 수 있음
  - 사용자마다 키를 하나씩 줘야 한다. N명의 사용자 → N개의 키
  - 키를 가지고 있어야만 MAC을 검증할 수 있다.
- 전자 서명은 이런 문제를 해결한다.

	MAC	전자 서명
기반 시스템	대칭키 시스템	공개키 시스템
키 관리 (N명의 사용자)	N개 키 관리	서명자의 (공개키, 비밀키) 한 쌍만 관리
공개 검증 여부	비밀키가 있어야 가능	누구나 검증 가능
부인 방지	불가능	가능

전자 서명



ex) 공인인증서

비대칭 암호화 vs 전자서명

- 비대칭 암호화 알고리즘을 그대로 전자 서명에 쓴다.

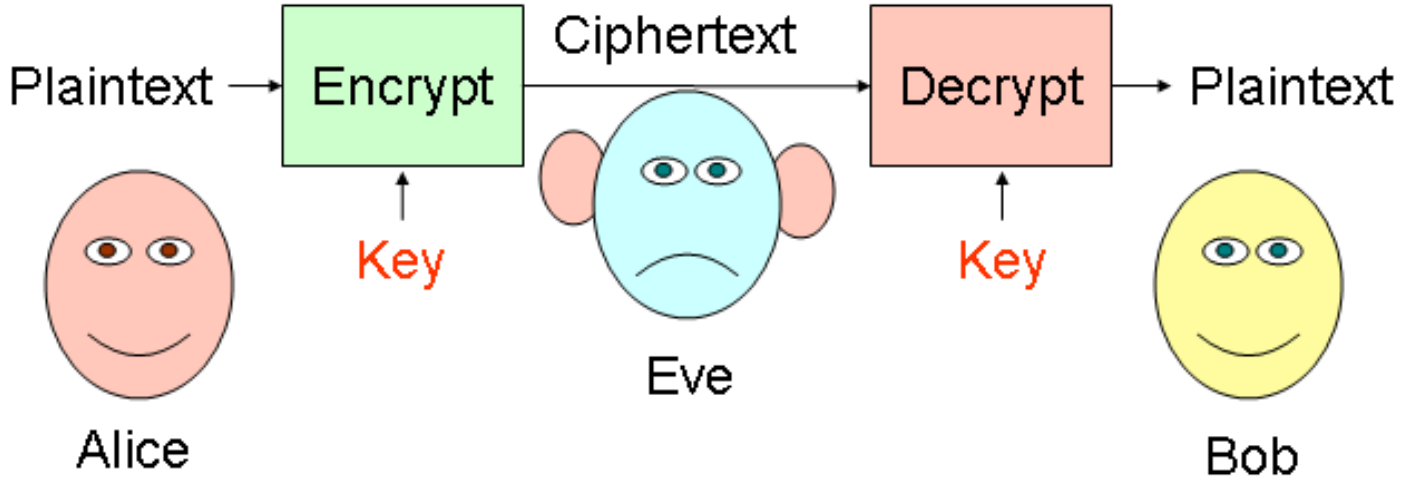
  비대칭 암호화: 공개키를 암호화에 사용 → 전자서명: 공개키를 서명 검증에 사용

  비대칭 암호화: 비밀키를 복호화에 사용 → 전자서명: 비밀키를 서명 작성에 사용

문제	전자서명의 해결방식
키 관리 (N명의 사용자)	내가 서명한 키만 가지고 있으면 됨
공개 검증 여부	누구나 공개키를 받고, 검증할 수 있음
부인 방지	비밀키는 한 사람만 가지고 있기 때문에 부인 방지

TLS

- Transport Layer Security: 네트워크 상에서 통신을 위한 보안 프로토콜
  - Transport Layer: Application Layer (HTTP)보다 아래 있는 계층 (see OSI-7)
  - FTP, SMTP 등 HTTP가 아닌 프로토콜에서도 쓰일 수 있음
- SSL은 TLS의 과거 버전 → 취약점이 발견돼 더 이상 쓰이지 않음
- 중간에 제 3자에 의해 통신 내용에 대해 감청되거나, 변경되는 것을 막음



TLS Handshake

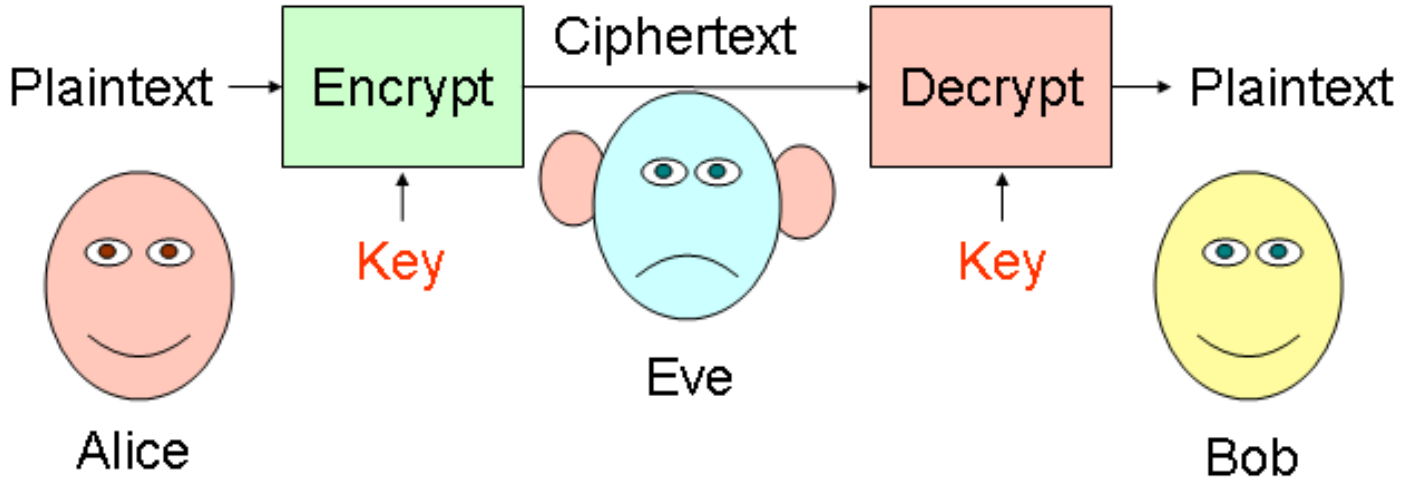
- 실제 Application Layer (HTTP 등)에서 통신은 대칭키 암호로 이루어짐
- 이전에 비대칭키로 서로를 확인하고, 비대칭키로 대칭키를 교환하는 과정

SPARCS

TLS의 인증서

- Handshake과정에서 서버가 보내는 인증서
- 대칭키 전송을 위한 암호키가 인증서에 포함

- 인증서는 “검증된 기관”에 의해 발급된 인증서여야 함
  - 중간에 새로운 인증서로 가로채지 못하게 하기 위함 (인증서에 도메인 정보가 포함)
  - 검증된 기관의 정보는 브라우저에 내장되어 있음



TLS와 HTTPS

- HTTPS: HTTP + TLS
- HTTP/1.1에서는 단순히 TLS 위에서 돌아가는 HTTP이지만, HTTP/2에서는 TLS를 더 빠르게 쓸 수 있게 지원함

Random Values

- 랜덤값은 굉장히 많이 쓰인다.
- 의사 난수 생성기Psuedo-random Number Generator; PRNG 를 사용
- 실제 PRNG: 랜덤값 표를 이용하거나, 적당한 값들을 섞어 계산을 이용

```
def random_generator() -> int:
  return 5
```

- CSPRNG (Cryptographically-Secure PRNG)
  조건: 처음 k비트가 주어졌을 때, 50%보다 높은 확률로 k+1번째 비트를 다항 시간 안에 구할 수 없어야 한다.
- RNG (Random Number Generator): 진짜 랜덤 값을 구함
  마우스 움직임, CPU 잡음, 외부 소음 같은 것을 무작위성 (entropy) 의 출처로 이용

https://blog.cloudflare.com/randomness-101-lavarand-in-production/

Random Values

리눅스의 RNG 디바이스

- **/dev/random** : 진짜 RNG
  암호 키 (TLS, SSH 등) 생성을 위해 이용. 엔트로피가 비면 채우는데 오래 걸린다.

- **/dev/urandom** : 초기값을 받아 PRNG 스트림 생성
  하드디스크 암호화 같은 용도에 적합

- https://wiki.archlinux.org/index.php/Random_number_generation

Linux Security

중요한 파일들

- /etc/passwd, /etc/group: 사용자, 그룹들의 목록과 관련된 정보가 저장
- /etc/shadow: 모든 사용자들의 비밀번호 해쉬 값
- /usr/lib: 오브젝트 파일과 내부 바이너리, 동적 linking되는 의존성들 (ex. libc)
- /usr/bin: 바이너리 파일들
- /var: 프로그램들에 의해 변하는 파일들 (로그, DB, 웹서버 데이터, 이메일 보관, …)

Linux Security

- fail2ban
  - SSH/FTP 등 로그인을 여러 번 실패할 경우, IP를 일정 시간 차단
  - 커널 방화벽에 IP를 등록
- iptables
  - 오고가는 패킷들을 필터링 할 수 있다.
  - 어디서 패킷이 오는지, 얼마나 자주 오는지, 어떤 패킷이 오는지로 차단 가능
- 로그 확인
  - /var/log/auth.log: 보안 관련 로그 (로그인 기록)
  - /var/log/kern.log: 커널 이벤트나 오류 로그
  - /var/log/syslog: 시스템 통합 로그 - 웹서버 or Docker에서 여기로 로그를 보낼 수 있음
  - 등 많은 것들이 /var/log에 있다.

Password Storage

- 비밀번호를 저장하는 것은 아주 중요한 문제

- 평문으로 저장: 당연히 안됨
- 대칭키 암호화: 좀 그럼 (복호화를 굳이 해야할까?)

- 해시 함수를 사용한다.
- 비밀번호의 해시값을 저장하고, 비밀번호가 입력되면 해시를 계산해서 저장된 값과 비교

Password Storage

- **Rainbow Table Attacks**
  (평문, 해시값) 쌍을 모두 미리 계산하여 저장 → 나중에 해시값만 보고 평문을 찾는다.
- **방지하는 방법**: 평문에 랜덤값을 섞어 해시하고 (랜덤값, 해시)를 DB에 저장
  Rainbow Table을 사용하더라도 평문과 랜덤값이 섞인 것만 얻을 수 있다.
  이 랜덤값을 **Salt**라고 한다.

User	Password
Stephen	auhsoJ
Lisa	hsifdrowS
James	1010NO1Z
Harry	sinocarD tupaC
Sarah	auhsoJ

User	Password Hash
Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	f567c40623df407ba980bfad6dff5982
James	711f1f88006a48859616c3a5cbcc0377
Harry	fb74376102a049b9a7c5529784763c53
Sarah	39e717cd3f5c4be78d97090c69f4e655

User	Random Salt	Password Hash
Stephen	06917d7ed65c466fa180a6fb62313ab9	b65578786e544b6da70c3a9856cdb750
Lisa	51f2e43105164729bb46e7f20091adf8	2964e639aa7d457c8ec0358756cbffd9
James	fea659115b7541479c1f956a59f7ad2f	dd9e4cd20f134dda87f6ac771c48616f
Harry	30ebf72072134f1bb40faa8949db6e85	204767673a8d4fa9a7542ebc3eceb3a2
Sarah	711f51082ea84d949f6e3efecf29f270	e3afb27d59a34782b6b4baa0c37e2958

Password Storage

- 어떤 해쉬함수를 써야 할까?
  암호학적 해쉬 함수 (SHA 등)의 단점: **빠름**
- 의도적으로 느리게 설계된 해시 함수를 사용한다.

- PBKDF2 Password Based Key Derivation Function
  비밀번호로부터 키를 만드는 해쉬 함수, 오래 걸리게 하기 위해 아주 많이 (10~20만 번) 반복
- bcrypt + scrypt: 시간, 메모리 사용량을 많게 해 GPU, ASIC로 뚫는 것을 어렵게 함
- Argon2: GPU 연산과 메모리 공격을 어렵게 함. argon2i,argon2d, argon2id 버전이 존재

1Password: PBKDF2 (10만번)

SPARCS SSO (Django): PBKDF2 (12만번)

Input Sanitization

SQL Injection: 악의적인 입력을 보내 SQL의 내용물을 바꿔버리는 것

1) 로그인창에 아이디를 ‘ OR 1==1; DROP TABLE users; --라고 입력
2) 서버는 sql.run(“SELECT * FROM users where id=‘%s’ and pw=‘%s’”)

실제로 DB에 들어간 쿼리

SELECT * FROM users where id=‘’ OR 1==1; DROP TABLE users; --’ and pw=‘%s’

→ SELECT 문은 대충 해석되고, **DROP TABLE users;** 가 실행됨!

→ **파멸적인 결과**

Input Sanitization

XSS (Cross-Site Scripting)

게시판에 <script>alert(document.cookie)</script>를 적고 올리면?

→ HTML 태그가 그대로 올라감

→ 다른 사용자가 게시물을 클릭하게 되면..

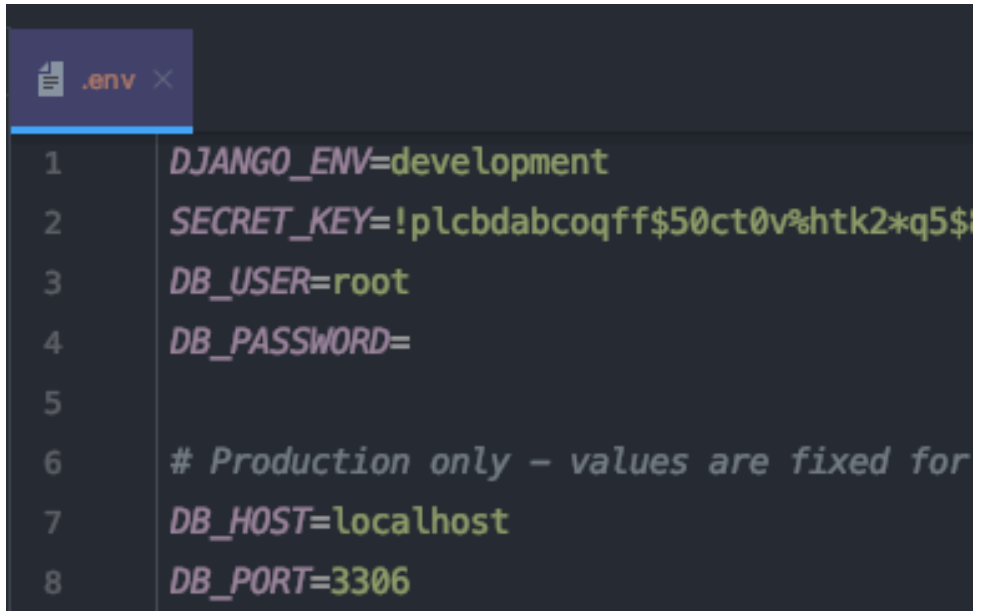
Input Sanitization

- 모든 사용자의 정보는 **악의적이라고 가정**하고, 이에 대응해야 한다.
- 특정 키워드들을 escape하거나 제거
  - XSS: <, > 같은 키워드들을 < >로 escape
  - SQLi: ’가 입력되면 ‘’로 escape
- 정규식 필터를 이용하여 제거
  - 아이디/비번을 입력받는 경우, [a-zA-Z0-9_\-]으로 제한
  - script, input 등 위험한 키워드를 제거
- 직접 필터를 만들기보다, 잘 준비된 라이브러리를 사용하자
  - Node: npm install xss
  - Django: django.utils.html에 escape 있음

Secret Management

- 안전한 Secret 관리: Secret이 털리면 다 털린다. ex) 세션 MAC의 시크릿 값, DB 비밀번호

- 시크릿 값은 코드로 저장하지 말고, 환경 변수에 넣어서 사용한다.
- 환경 변수를 저장하는 파일 (.env)는 반드시 .gitignore 한다.
- Secret 값을 찾아주는 도구 (ex. git-secrets)를 사용한다.
- 이상한 곳에 secret을 저장하지 않는다 (ex. 카톡)



Secret Management

- Secret 보관을 해주는 서비스들
  - .env로는 충분하지 않기 때문에..

AWS KMS
(Key Management Service)

Cloud Key Management Service

HashCorp Vault

키 길이 선택하기

- 암호 알고리즘을 쓸 때, 키 길이는 어떻게 골라야 할까?
- 시간이 지날수록 연산 성능이 발전 → 더 긴 키를 골라야 함

Method	Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash
[1] Lenstra / Verheul	2021	86	1937 1536	153	1937	163	172
[2] Lenstra Updated	2021	82	1416 1614	164	1416	164	164
[3] ECRYPT	2018 - 2028	128	3072	256	3072	256	256
[4] NIST	2019 - 2030	112	2048	224	2048	224	224
[5] ANSSI	2021 - 2030	128	2048	200	2048	256	256
[6] NSA	-	256	3072	-	-	384	384
[7] RFC3766	-	-	-	-	-	-	-
[8] BSI	2020 - 2022	128	2000	250	2000	250	256

https://www.keylength.com/

감사합니다

⭐ 봄학기 SSO 팀원 모집중 ⭐

문의: jungnoh (노정훈) 슬랙 or 카톡