이것만 알면 되는

데이터 암호화 (Data Encryption)

정광섭(https://lesstif.com)

서비스 보안

- 개발자들은 바빠서 보안은 고려 대상이 아니거나 우선순위가 9,999 위
- 하지만 이제는 보안도 중요한 기능
- 보안을 비용이라고만 생각하고 나중에 아주 비싼 비용을 치를 수 있음

  사용자 이탈, 비즈니스 평판 저하, 법적 책임 등등....

- 경영자가 보안의 중요성을 이해하고 지원해야 개발자와 운영자도 관심 가질 수 있음

완벽한 보안은 없지만...

- 보안은 실패할 수 있지만 대비에 따라 결과가 달라짐 (신문에 회사 이름이 나오거나 사과를 할 확률이 적어짐)
- 암호화는 이런 상황에 대비한 최후의 보루
- 알고리즘에 대해 깊이 몰라도 암호화에 필요한 지식만 이해하기
- 데이터 특성에 따라 적합한 알고리즘 및 암호화 방식 선택하기

암호화(Encryption)

- 허가된 이 외에는 읽을 수 없도록 정보를 부호화 하는 것
- 부호화하는 방식을 암호화 알고리즘이라 함

암호 알고리즘의 구분

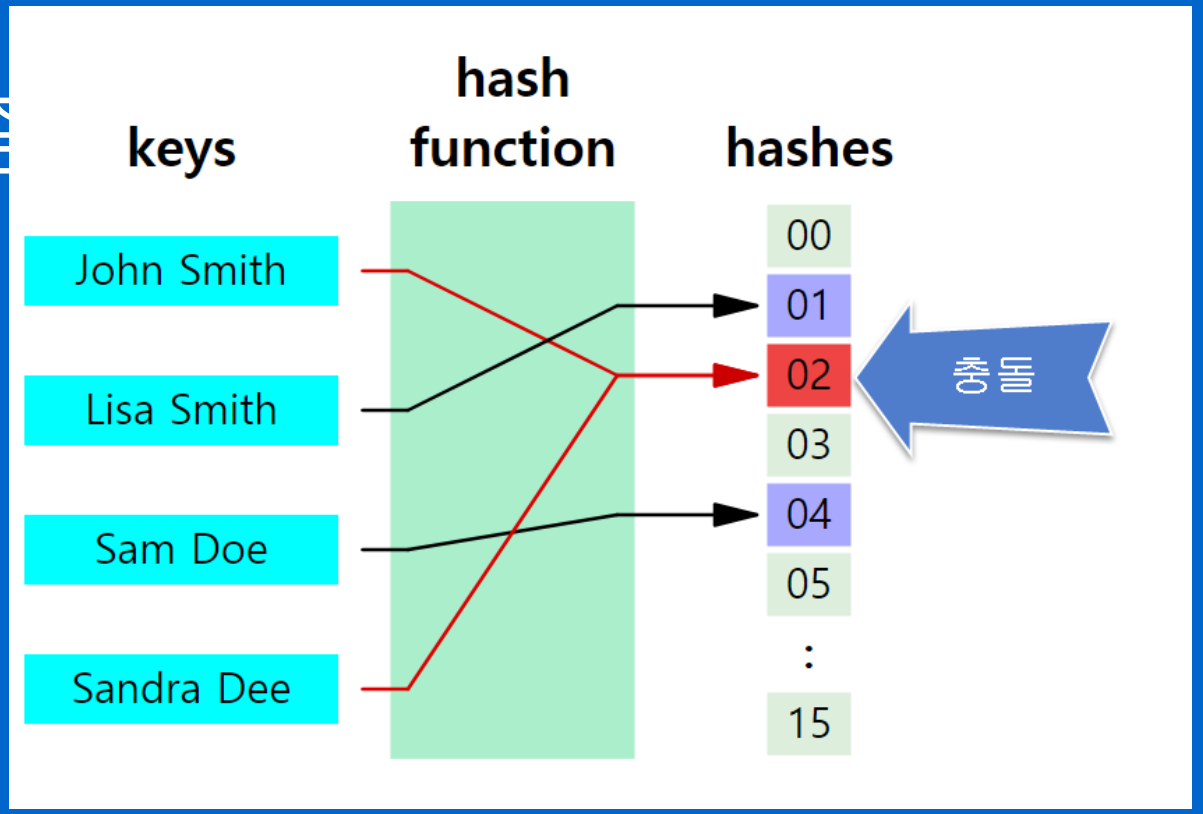
- 키의 갯수 :
  - 1 key: 비밀키(대칭키)
  - 2 key: 공개키(비대칭키)
- 데이터의 처리 단위: 스트림(Stream), 블록(Block)
- 원본 복호화 여부: 단방향, 양방향

해시 함수(Hash Function)

- 임의의 길이의 입력을 고정된 길이의 출력으로 바꾸는 함수
- 동일한 입력에 대해서는 동일한 출력이 보장
- Hash Table 기반의 빠른 검색이 필요한 경우 등에 많이 사용
- 해시 함수는 빠른 연산속도를 특징으로 하며 출력값은 digest 라고 부름.

해시 함수(Hash Function)

- h(M) = H 로 표현
- h() 는 해시 함수, M 는 입력 값, H = 해시 값을 의미
- 다른 입력이지만 동일한 출력이 발생할 수 있으며 이를 충돌 (Collision) 이라고 함
- 충돌이 적어야 좋은 알고리즘



출처: https://en.wikipedia.org/wiki/Hash_function

암호화 해시(Hash) 함수

- 일반 해시 함수에 아래 3가지 특성을 만족하는 함수
- 복호화가 안되므로 단방향(One way) 암호화 함수라고도 함

역상 저항성 (Preimage Resistance)

- 주어진 해시 값에 대해, 그 해시 값을 생성하는 입력 값 (M) 을 찾는 것이 계산상 불가능
- H(M) = H 에서 H 값인 'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d' 를 알아도 입력인 M 값 'hello' 를 계산하기는 어려움

Rainbow Attack

- 역상 저항성을 우회해서 입력 값을 찾기 위한 Brute-force attack 의 일종
- 다양한 문자열을 조합하여 해시값을 계산한 rainbow table 작성
- 획득한 해시값을 table 과 비교하여 원본 메시지 추출
- 입력 값에 임의의 값(Salt) 를 추가하여 hash 를 계산하면 방지

2 역상 저항성(second preimage resistance)

- 주어진 입력 값(M)에 대해, 동일한 해시 값을 갖는 입력값(M`)을 찾는 것이 계산상 불가능
- h(M) = H 에서 H 값인 'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d'
- M 인 'hello'를 알 경우 동일한 출력을 내는 M1 을 찾는 문제

충돌 저항성(collision resistance)

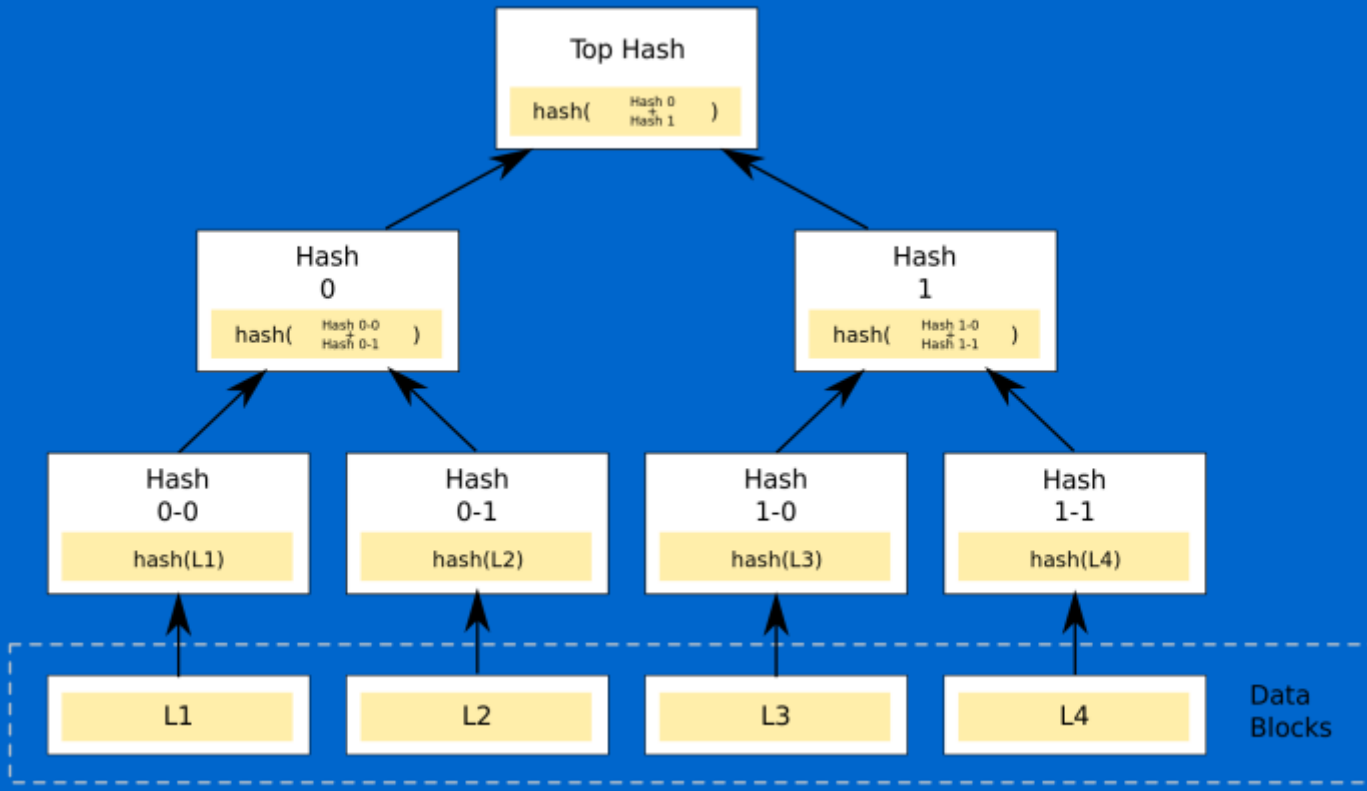
- 같은 해시 값을 생성하는 두 개의 입력 값을 찾는 것이 계산상 어려움
- 동일한 출력 값을 내는 임의의 M1, M2 을 찾는 문제
- 충돌 저항성에 따라 Hash 의 안정성은 길이의 반정도로 판단 (SHA1 = 80bit, SHA256 = 128bit)

암호화 해시 알고리즘 용도

- 원본 파일의 Checksum 확인
- 메시지 위변조 검출 및 인증을 위한 HMAC(keyed-Hash Message Authentication Code)
- 전자서명시 속도 향상을 위해 원본에 hash 함수를 적용한 digest 값에 대해 서명
- 결론은 메시지의 무결성 확인에 사용

Merkle Tree(Hash Tree)

- 대량의 데이터를 빠르게 검증하고 손상된 데이터를 찾아낼 수 있는 자료 구조
- BitTorrent 등의 P2P, git 의 내부 저장소, Bitcoin 등의 BlockChain 구현물에 사용

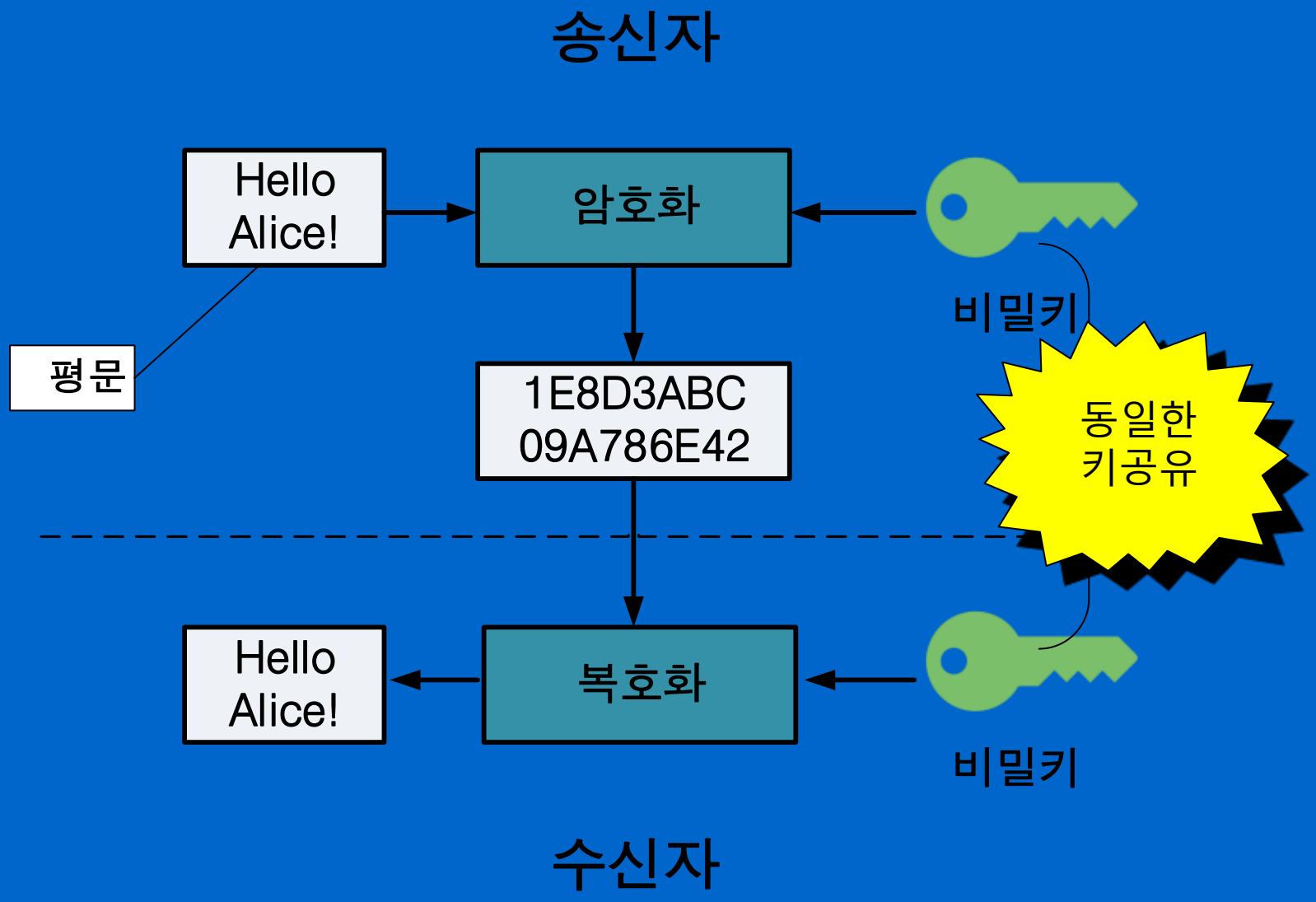


출처: https://en.wikipedia.org/wiki/Merkle_tree

해시 알고리즘 종류

- MD5, SHA1, SHA2(SHA256, SHA384, SHA512)
- MD5(128bit)는 매우 취약하고 SHA1(160bit) 은 사용하지 않는 것을 권고
- SHA1 이 당장 문제되는 것은 아니고 전자 서명등 장기 보존하는 데이터 사용에만 권고하지 않음
- Session에만 사용하거나 파일 checksum, git commit hash 등에 사용은 아무 문제없음

대칭키 알고리즘(Symmetric Algorithm)



대칭키 알고리즘(Symmetric Algorithm)

- 하나의 키를 사용하여 데이터를 암/복호화
- 암/복호화 Key는 대칭키 또는 비밀키(Secret Key) 라고 부름
- 암/복호화 속도가 빠르고 구현이 용이
- 블록(Block) 암호 알고리즘과 스트림(Stream) 암호 알고리즘으로 나눠짐

스트림 암호 알고리즘

- 대칭키를 만든 후에 Bit 단위로 XOR 연산으로 암호화
- RC4, AS/2 등의 알고리즘이 있음
- 속도는 빠르지만 실무에서는 거의 사용하지 않으므로 몰라도 됨.!

블록 암호 알고리즘

- 대칭키 알고리즘 중 암/복호화시 데이터를 블록 단위로 처리하는 알고리즘
- 이제는 쓰면 안 되는 DES, 현재 미국 표준인 AES, 국내 표준 SEED등이 있음
- 블록 크기와 Key 길이는 알고리즘마다 다름
- SEED 는 128 bit(16 byte), AES 는 128, 192, 256 bit Key 길이 지원

블록 암호 알고리즘

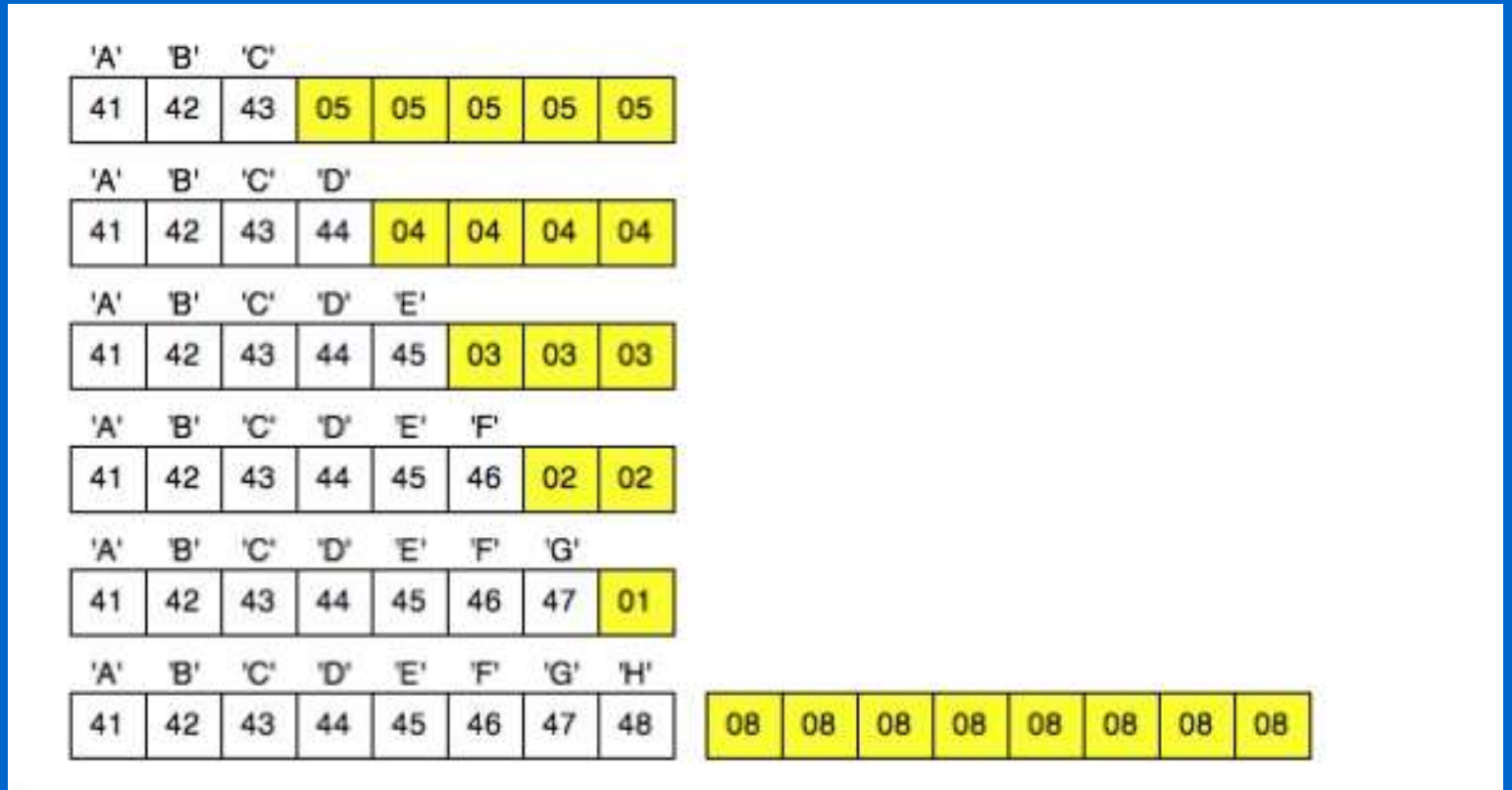
- 일본이 만든 Camellia 는 SSL/TLS에서 많이 사용됨
- AES 는 전 세계 공모를 통해 선정 - 벨기에의 암호학자 2명이 제출한 Rijndael 이 채택
- 국가 표준 암호를 공모하는 개방성이 미국의 경쟁력

DES: Data Encryption Standard

AES: Advanced Encryption Standard

Padding

- 만약에 입력 데이터가 블록 사이즈의 배수가 아니라면?
- 블록 사이즈에 맞추기 위한 방식이 Padding
- 부족한 size 만큼 바이트 값을 추가하는 PKCS7 Padding 을 많이 사용
- 3byte 가 부족할 경우 03 을 3 개 패딩



출처:
https://stackoverflow.com/questions/34865313/bouncy-castle-pkcs7-padding

운영 모드(Operation Mode)

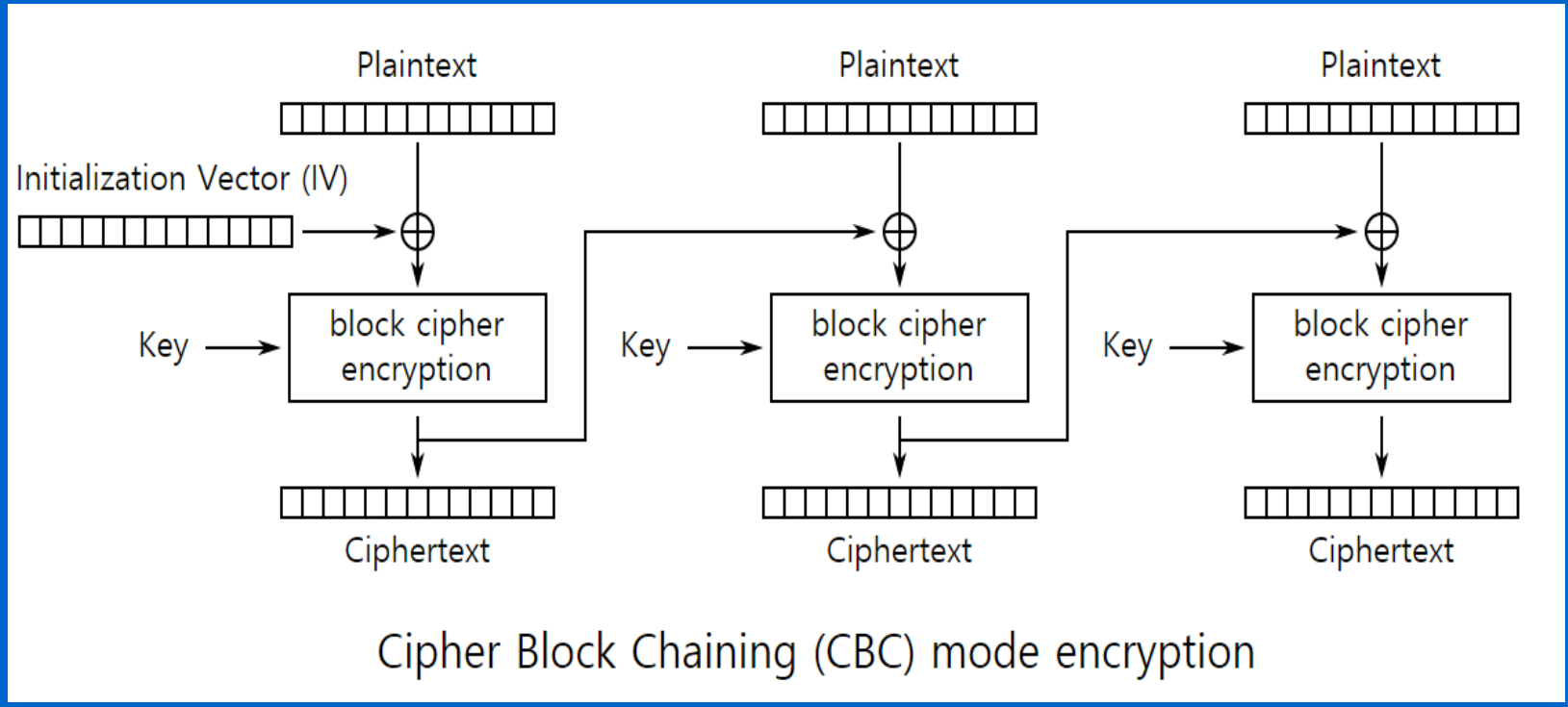
- 보통 입력 데이터가 블록보다 크므로 암/복호화시 여러 개의 블록이 생성됨
- 각 블록간의 관계를 처리하는게 운영 모드
- ECB, CBC, CFR, GCM 등의 모드가 있음

운영 모드 - ECB

- Electronic Code Book
- 개별 블록은 각각 암복호화 수행(운영 모드 없음)
- 입력 값을 유추할 수 있는 치명적인 문제가 있음
- 대칭키 암호화시 ECB 를 사용하면 안 됨!

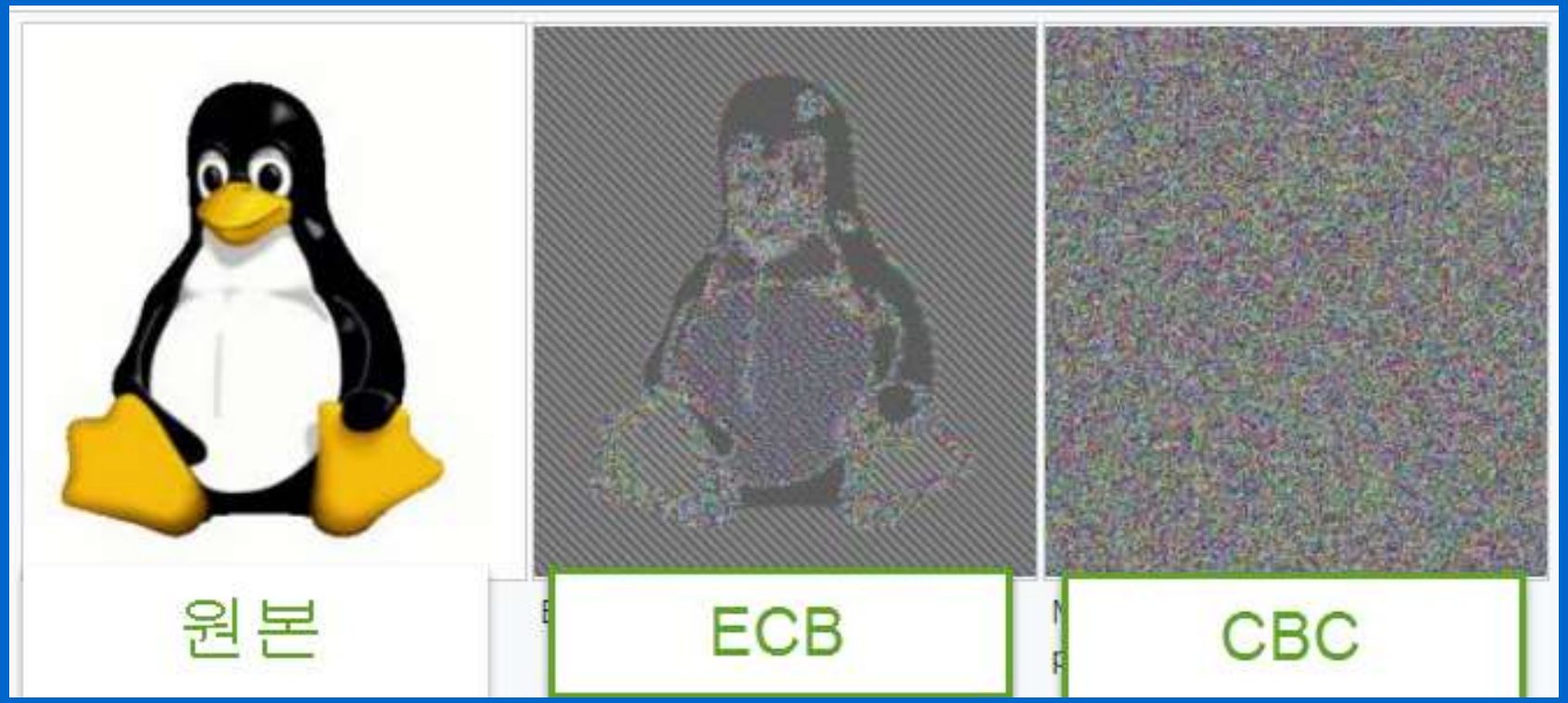
운영 모드 – CBC(Cipher Block Chaining)

- 직전 블록은 다음 블록의 입력으로 사용하여 안정성 증대
- 초기 블록 유추 어렵도록 Key 이외에 IV(Initial Vector) 사용



출처:
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

운영 모드별 결과물



출처:
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

CBC 의 단점

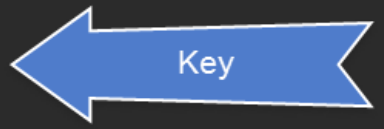
- 이전 블록 암호화의 결과가 다음 블록의 입력이므로 병렬 처리 불가능
- 병렬 처리 불가에 따라 현대의 멀티 프로세서의 장점을 살릴 수 없음(CBC 는 70년대말 개발)
- 용량이 클 경우 시간이 오래 걸리고 복호화시 특정 블록만 복호화 불가
- Padding 이 필수
- 메시지 인증(MAC) 에 사용할 수 없음

GCM(Galois/Counter mode)

- CBC 의 단점을 해결한 운영 모드
- 패딩 불필요
- 인증 기능 제공
- 병렬 처리가 가능해서 암/복호화 속도가 매우 빠름
- 브라우저와 서버가 지원할 경우 SSL/TLS 에서 많이 사용
- 많은 장점이 있지만 아직 활성화되지는 않음(예제가 모두 CBC...)

대칭키 암호화 예(Java)

```java
Security.addProvider(new BouncyCastleProvider());

String input = "Hello World";

KeyGenerator gen = KeyGenerator.getInstance("AES");
gen.init( i: 256);

// 대칭키 생성
SecretKey key = gen.generateKey();

// Initial Vector 생성
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
byte[] iv = new byte[16];
sr.nextBytes(iv);
IvParameterSpec spec = new IvParameterSpec(iv);

// 암호화
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");

c.init(Cipher.ENCRYPT_MODE, key, spec);

// 암호화된 데이타
byte[] encData = c.doFinal(input.getBytes());

// 복호화
Cipher d = Cipher.getInstance("AES/CBC/PKCS5Padding");

d.init(Cipher.DECRYPT_MODE, key, spec);
byte[] decData = d.doFinal(encData);
```



대칭키 암호화 예(PHP)

```php
$key = openssl_random_pseudo_bytes(32);

$iv = openssl_random_pseudo_bytes(16);

$data = 'Hello World';

$enc = openssl_encrypt($data, 'AES-256-CBC', $key, 0, $iv);

echo base64_encode($enc) . "\n";

$dec =  openssl_decrypt($enc, 'aes-256-cbc', $key, 0, $iv);

echo $dec . "\n";
```

Key 와 Initial Vector 생성

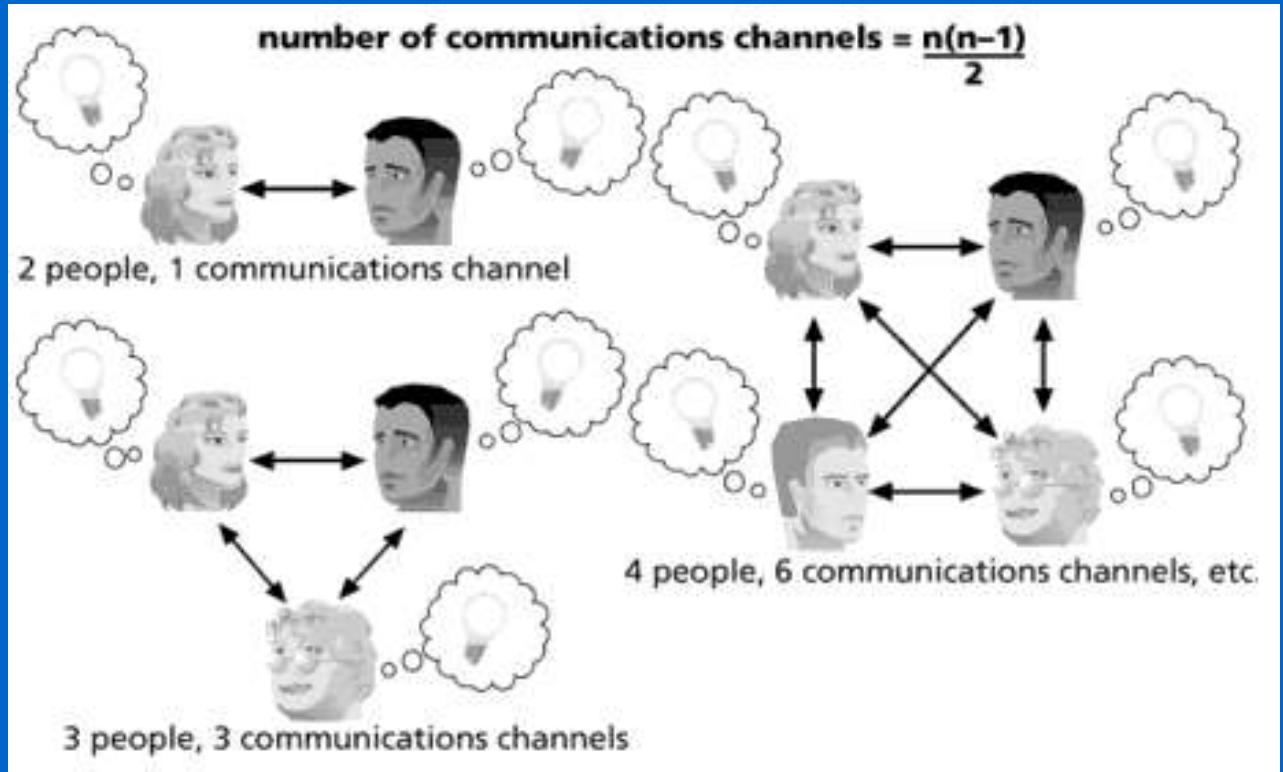
- 대칭키와 IV 는 random 값을 생성하여 사용하는 경우가 많음
- 암복호화시 Key 와 IV 를 입력해야 함
- Random 값이므로 사람이 기억하고 직접 입력은 불가능

PBKDF2 (Password-Based Key Derivation Function 2)

- 사람이 기억할 수 있도록 암호 기반으로 random key 생성
- 입력한 암호를 기반으로 Salt 를 추가하고 정해진 횟수 (Iteration)만큼 HASH 수행
- Hash 함수가 빠르므로 Brute-force attack 을 방지하기 위해서는 충분한 Iteration 을 실행 => Key stretching
- 암복호화시 사용자에게 암호를 입력 받는 SW들은 거의 대부분 PBKDF2 를 사용

대칭키 알고리즘 단점

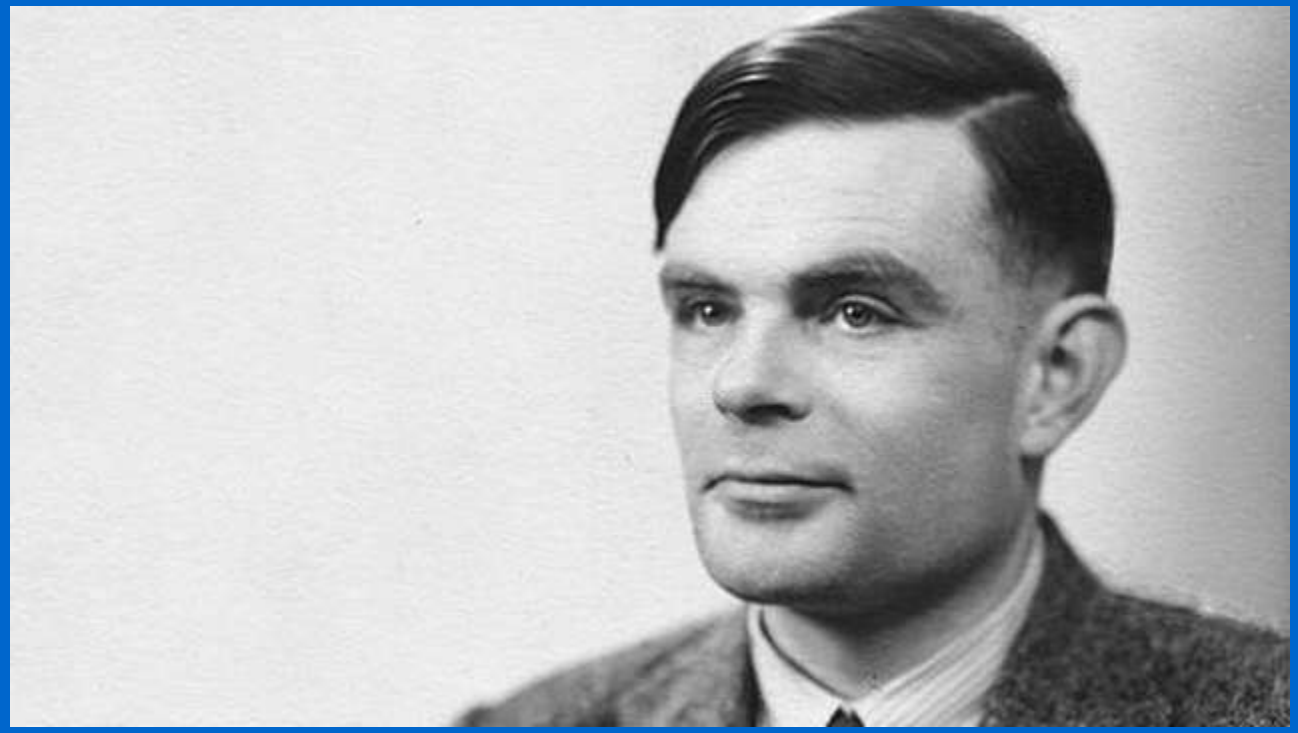
- 상대방과 Key 공유 필요하나 안전하게 키 공유가 어려움
  Ex: 전시에 적진에 둘러싸인 아군에게 키 전달
- 통신 상대방이 많아질수록 키 관리가 어려움 - n(n-1)/2 개의 키 필요



출처:
http://learnline.cdu.edu.au/units/hit241/applied/communications.html

암호화의 중요성

- 튜링과 에니그마 해독

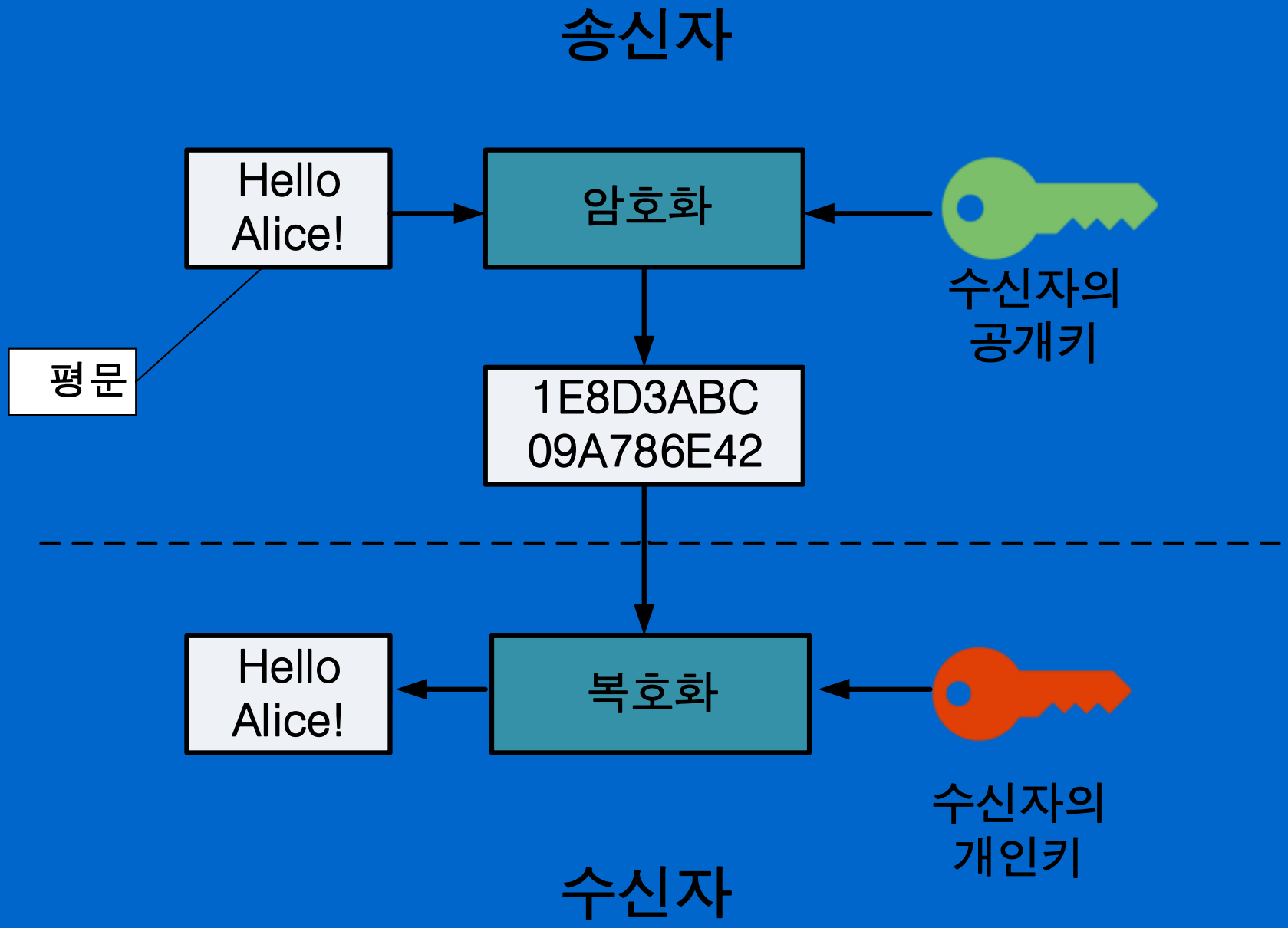


출처: https://en.wikipedia.org/wiki/Alan_Turing

공개키 암호화(Public Key Encryption)

- 2개의 키를 생성해서 하나(Public Key)는 공개하고 하나(Private Key)는 안전하게 보관
- Private Key 로 암호화한 내용은 Public Key 로만 복호화 가능
- 대칭키의 단점인 암호 통신시 Key 공유 문제 해결
- Asymmetric encryption 이라고 함

공개키 알고리즘


송신자
Hello
Alice!
암호화
수신자의
공개키
평문
1E8D3ABC
09A786E42
Hello
Alice!
복호화
수신자의
개인키
수신자

공개키 알고리즘 - RSA

- 391 = A * B 일 때 A 와 B 는

공개키 알고리즘 - RSA

- 17 * 23 = ?
- RSA 는 위와 같은 인수 분해의 어려움을 기반으로 한 알고리즘
- 설계자인 Rivest, Shamir, Adleman 3 명의 약자를 따서 명명

공개키 알고리즘 - 기타

- 이외 RSA 와 비슷한 Rabin, 이산 대수의 어려움을 기반으로 한 DSA, 속도가 빠른 타원 곡선(ECDSA) 알고리즘이 있음
- 타원 곡선 알고리즘은 bitcoin에서 사용

DSA: Digital Signature Algorithm

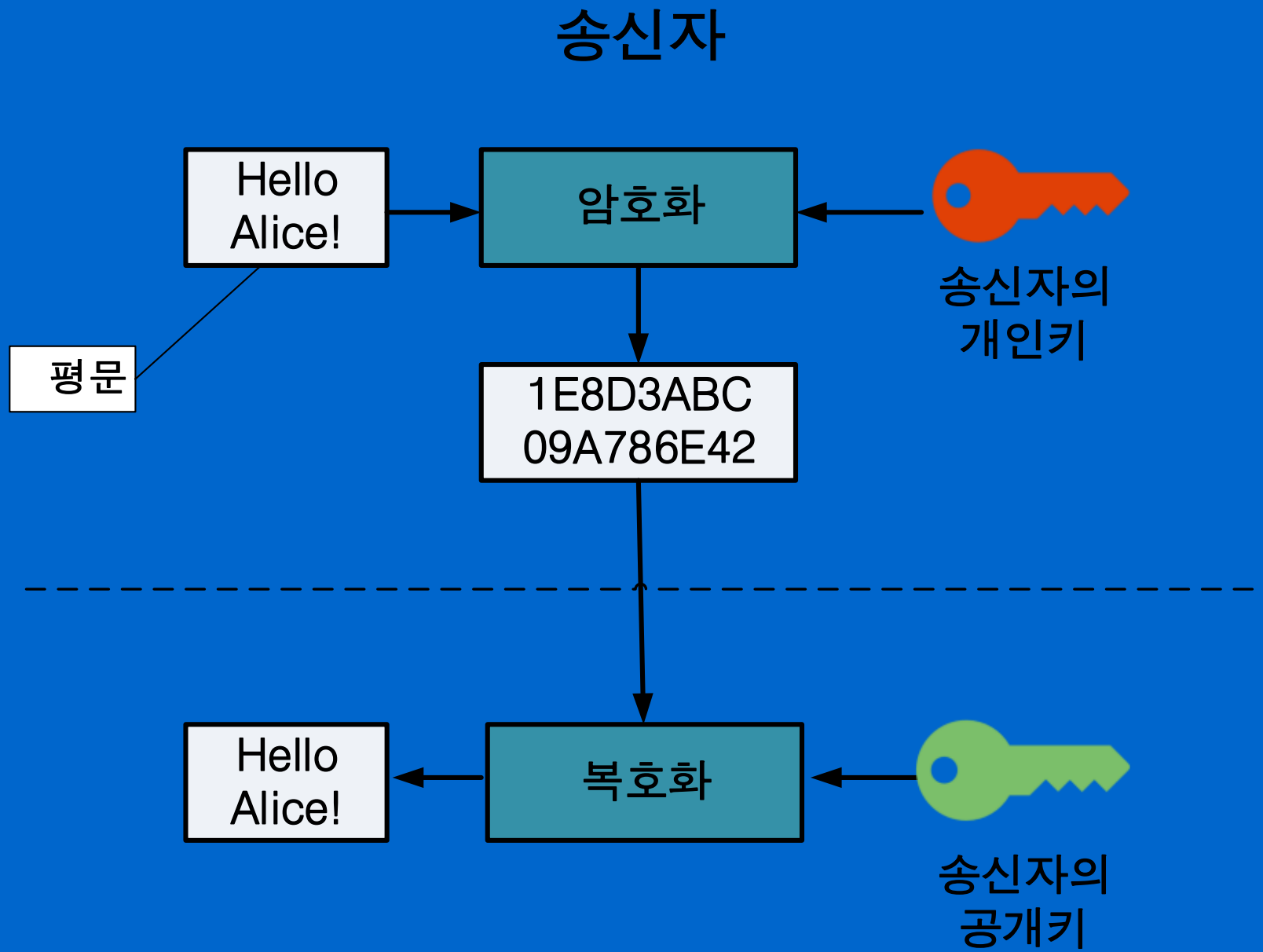
ECDSA: Elliptic Curve Digital Signature Algorithm

공개키 알고리즘 – 개인키 암호화

- 개인키로 암호화한 내용은 공개키로만 검증 가능
- 개인키 소유자가 만든 데이터임을 알 수 있으므로 송신자를 식별(Authentication)할 수 있음
- 송신자가 메시지 작성 사실을 부인하지 못하므로 "*부인 방지*"(Non-Repudiation) 라고도 함
- 개인키 암호화를 전자서명(Digital Signature) 라고 부름

부인 방지: 부인 봉쇄라고도 함.

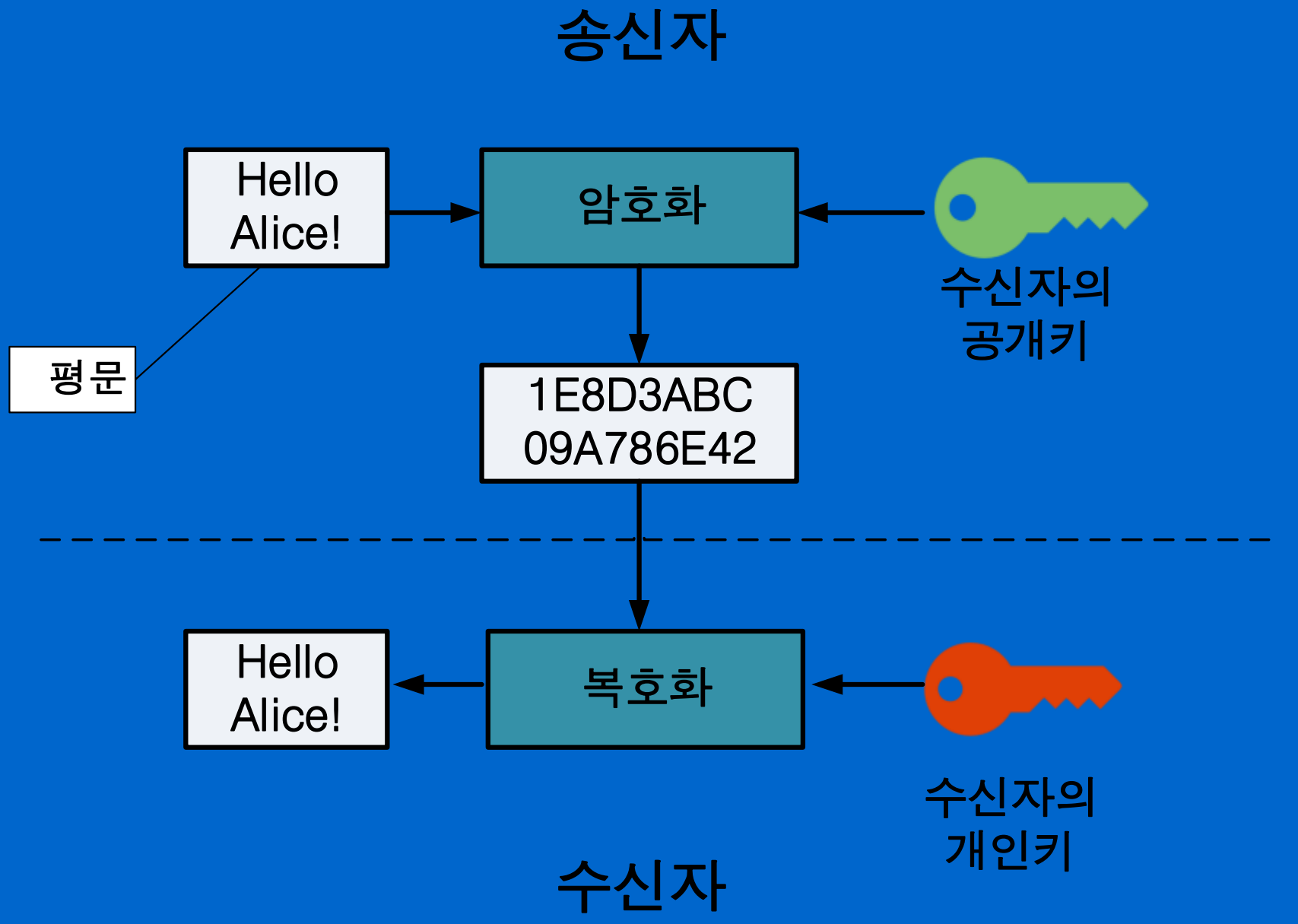
공개키 알고리즘 – 개인키 암호화



공개키 알고리즘 – 공개키 암호화

- 수신자의 공개키로 데이터 암호화
- 개인키 소유자만 해독할 수 있으므로 안전하게 데이터 전달 가능
- RSA,Rabin, Elgamal 알고리즘은 공개키 암호화 기능 제공
- DSA 는 전자 서명 기능만 제공

공개키 알고리즘 – 공개키 암호화



공개키 알고리즘 Key 유형별 결과물

출처:
https://twitter.com/JZdziarski/status/753223642297892864

(현재는 삭제된 트윗)

공개키 알고리즘 단점

- 복잡한 수학적 연산때문에 구현이 매우 어려움(H/W, S/W 모두)
- 대칭키에 비해서 속도가 매우 매우 매우 느림(몇 백배 이상)

```
$ openssl speed aes-128-cbc rsa1024


Doing aes-128 cbc for 3s on 16 size blocks: 22920078 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 6343026 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 256 size blocks: 1621301 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 408313 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 51219 aes-128 cbc's in 3.00s

Doing 1024 bit private rsa's for 10s: 52898 1024 bit private RSA's in 10.00s
Doing 1024 bit public rsa's for 10s: 907416 1024 bit public RSA's in 9.99s

OpenSSL 1.0.1e-fips 11 Feb 2013
```

키 교환(Key Exchange)

- 도/감청이 가능한 채널에서 암호화 통신을 하기 위해 안전하게 Key 를 교환하려면?
- 이를 해결하기 위한 방법이 키 교환이며 크게 “키 동의“ 와 “키 암호화“ 이 있음

키 동의(Key Agreement)

- 통신 당사자들이 사전에 준비한 공개키 쌍이 없이도 정해진 알고리즘에 의해 안전하게 키를 합의
- Key를 합의해 가는 과정이므로 키 동의라고 함
- 디피-헬만 (Diffie-Hellman - 2명의 수학자 이름) 알고리즘이 유명

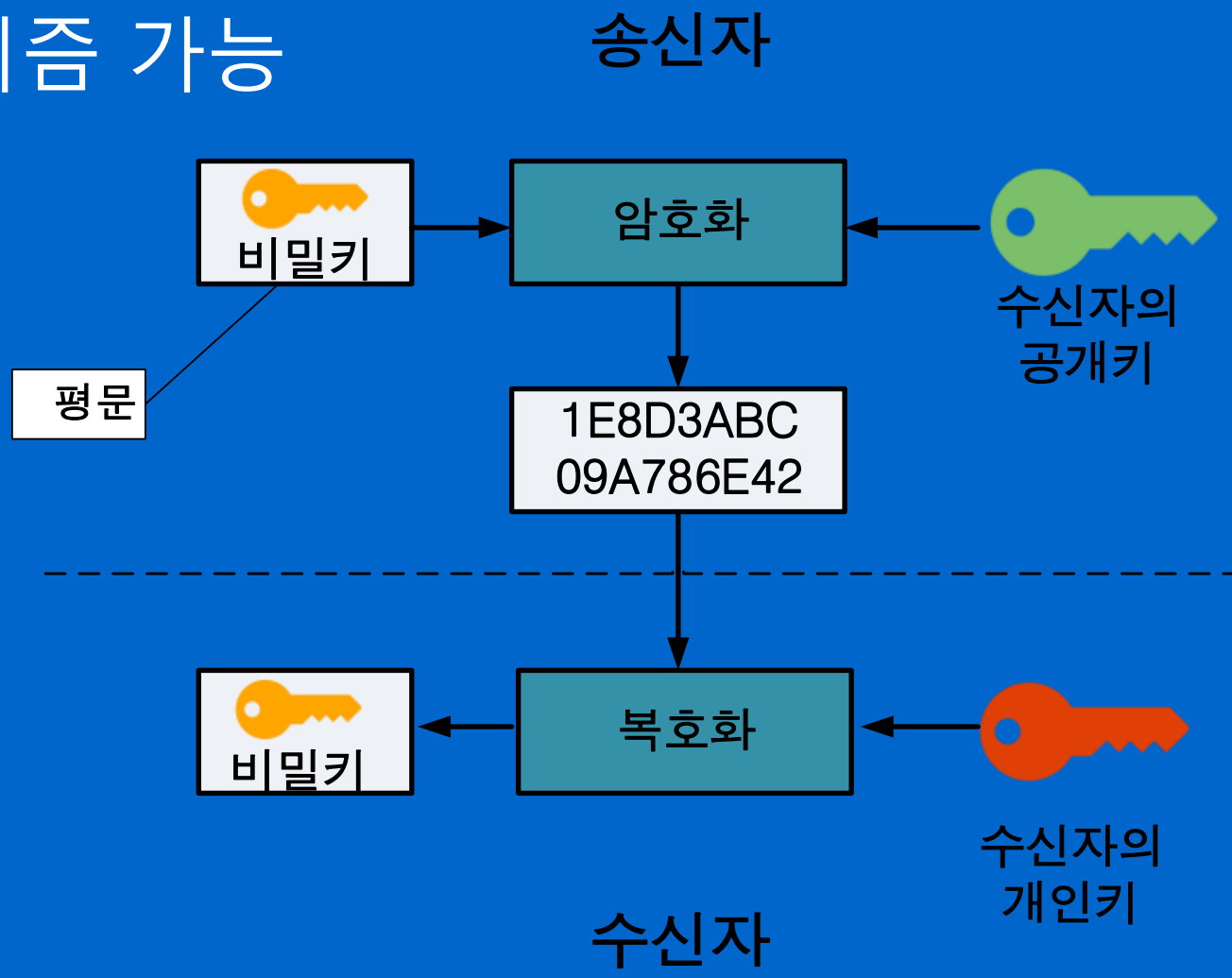
디피-헬만 키 교환

- 이산대수의 어려움에 기반한 혁신적인 키 교환 알고리즘
- 최초로 발표(1976년)된 공개키 기반 알고리즘
- 암호화되지 않은 통신망에서 상대방의 공개키가 없어도 안전하게 Secret Key 공유
- SSH, SSL 등 암호화 통신에 많이 사용

이산 대수 문제: https://ko.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/discrete-logarithm-problem 참고

키 암호화(Key Encipherment)

- 수신자가 대칭키 생성후 상대방 공개키로 암호화하여 전달
- RSA 알고리즘 가능



암호화 통신

- SSH, SSL 등 암호화 통신은 속도때문에 대칭키 방식을 사용
- 대칭키 교환은 공개키 방식의 키 동의나 키 전달을 통해 안전하게 공유
- 2 가지 방식을 혼용하여 대칭키 단점인 키 공유 문제를 해결하고 공개키 단점인 속도 문제를 해결

암호화 통신

- 사용하는 대칭키는 암호화 통신 세션 내에서만 사용하므로 Session Key 라고도 부름
- SSL 은 초기에 키 교환 패킷과 알고리즘 합의 과정이 오래 걸림
- SSL 세션이 구축되면 속도가 그리 느리지 않음(**GCM 운영 모드를 지원**하도록 브라우저와 서버 업그레이드 필요)
- SSL 성능을 높이려면 초기 키 교환 부담을 줄이기 위한 Session Key cache 기능 필수
- Session timeout 이 너무 크면 취약해지므로 적절한 선에서 타협

암호화 기술의 접근성

- 예전에는 암호 기술은 군대/정부등 일부만 사용 가능
- 미국은 40비트 이상의 Key 길이를 사용하는 암호 기술을 전략 물자로 취급하여 무기수출통제법 대상
- 개인이 암호화를 하는 것은 기술적 장벽(응용 SW 없음)과 사상적 장벽(암호화하면 간첩?)이 있었음

PGP(Pretty Good Privacy)

- 1991년 Phil Zimmermann 이 개발한 혁신적인 SW
- PGP의 등장으로 개인도 전자 서명, Email 암호화 사용 가능
- 미국 정부는 무기수출통제법에 의거해 PGP S/W의 해외 반출을 금지시킴
- 필 짐머만은 S/W는 대상이지만 책은 대상이 아닌 것에 착안하여 소스 코드를 출력하여 책으로 출판하여 수출

공개키 획득

- 인터넷과 PGP 의 등장으로 개인도 암호화 된 이메일 가능
- 이를 위해서는 사전에 상대방의 PGP 공개키가 필요
- 없을 경우 공개키를 구할 방법은?

공개키 저장소(Public Key Repository)

- 공용 저장소에 자기의 공개키를 올리는 방법 확산
- MIT 는 PGP 공개키를 올릴 수 있는 웹 서비스 구축 (http://pgp.mit.edu)
- 이제 상대방을 검색한 후 공개키 기반으로 암호화 통신을 하면 끝!

공개키 소유자의 신뢰성

- 공개키를 올린 이가 내가 통신하려는 그 사람이 맞을까?
- 통신을 도청하려는 누군가가 사칭해서 공개키를 올렸다면?
- 그렇다면 암호화를 해도 소용이 없을 텐데...

공개키 소유자 인증 필요성

- 누가 공개키 소유자를 인증해 준다면 이런 문제가 없을 텐데
- 이왕이면 신뢰된 기관이나 회사가 이런 업무를 수행해 주었으면..

인증 기관(CA; Certificate Authority)의 탄생

- CA는 공개키 소유자를 인증(Certification)해 주는 업무를 수행
- 인증된 공개키는 CA 가 소유한 개인키로 전자 서명
- 이 데이터를 공개키 인증서(Public Key Certificate) 이라 함
- 암호화 통신시 상대방 인증서를 CA 의 공개키로 검증하여 위/변조 여부 확인
- 공개키 소유자는 이미 CA 가 인증했으므로 온라인상에서 신뢰

신원 확인 및 공개키 인증

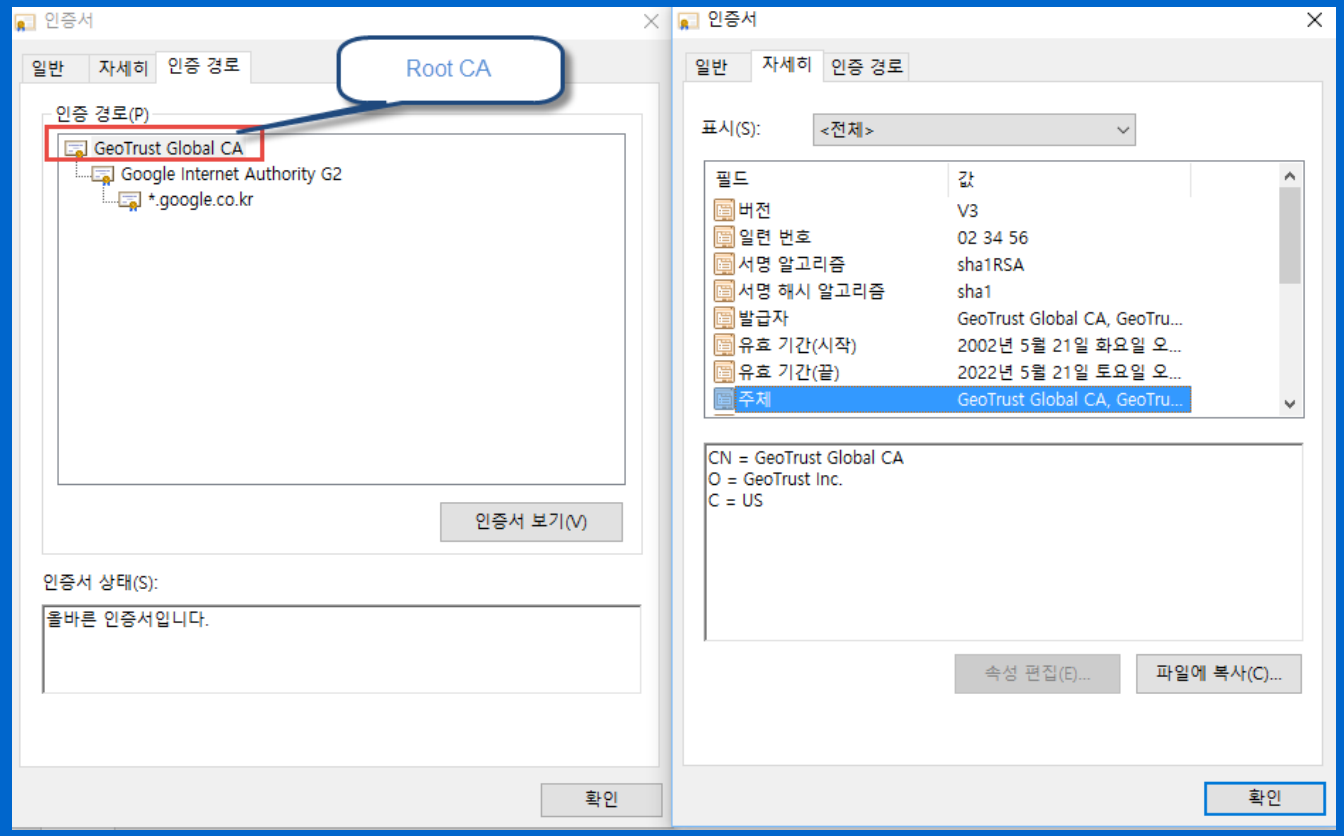
- CA 는 공개키 소유자의 신원을 확인하여 개인키로 서명된 데이터의 신뢰 부여
- 웹 사이트일 경우 SSL 인증서 신청자가 도메인의 소유자가 맞는지 소유권 검증(Verification Domain Ownership)
- 허술하게 할 경우 CA 가 발급한 인증서를 신뢰할 수 없게 됨
- 중국의 몇몇 CA는 소유권 검증을 제대로 하지 않고 SSL 인증서를 발급하여 Firefox, Chrome 에서 퇴출됨

공개키 기반 구조(Public Key Infrastructure)

- 공개키 암호화 기술을 바탕으로 디지털 인증서를 발급하고 폐지하고 응용하는 기반
- 기술뿐만 아니라 법/제도와 정책을 포괄
- CA 가 없을 경우 PKI 라 하지 않고 PK 라고 지칭
- PKI 는 중앙에 신뢰된 CA 가 있으며 하위에 개별 CA 를 두어 업무를 위임하는 중앙 집중형 방식

CA 인증서 체인

- Browser 에는 신뢰하는 여러 CA 인증서가 사전 등록됨
- 계층 구조이므로 체인이라 표현
- 사전 등록된 CA 목록을 Trusted CA list 라고 함.



인증의 2가지 의미

- 우리 말로 인증은 한 가지지만 영어는 2 가지 단어
- Authentication 과 Certification
- 사실상 동음이의어 이며 2 가지의 의미는 미묘하게 다르나 혼용하여 사용하는 경우가 많음

인증(Authentication)

- 어떤 이가 실제하고 그가 맞는지 확인하는 것
- 주로 온라인 상에서 신원을 확인 하는 일을 의미
- 지식기반 : Password, 이미지 선택, 패턴 매칭
- 소지기반 : 보안카드, OTP(One Time Password), 스마트카드
- 생체 기반 : 지문, 홍채, 정맥, 얼굴, 목소리 등.

인증(Certification)

- 특정 행위를 할 자격을 갖췄다는 것을 증명
  (의사자격증, 졸업증명서, 운전면허증 등)
- 흔히 이야기하는 공인인증서는 사용자 공개키를 인증기관이 Certification 했다는 의미
- 공인인증서의 용도는 인증(Authentication) 보다는 전자도장의
  용도이며 부인 방지를 제공

공인인증서와 PKI

- 인증서가 악의 축은 아니고 비판은 2가지에 초점을 맞추는 게 좋음

1. 해당 기술을 사용하는 방식(Active-X, Exe)
2. 사이트 로그인등 중요도에 맞지 않게 무분별하게 인증서 사용

공인인증서와 PKI

- 개인을 인증(Auth)하는 수단으로 사용하는 것은 잘못 (어디 출입시 인감도장 찍는 셈)
- 전자서명 (전자도장)의 기능이 필요한 계약, 입찰 등의 업무에만 공인인증서를 사용하는 게 적합
- 공인인증서의 기반 기술인 PKI 외에 온라인에서 부인 방지(Non-Repudiation)를 제대로 제공하는 기술은 없음
- PKI 는 FIDO(Fast IDentity Online), BlockChain 등 최신 기술의 핵심 인프라

암호화의 마지막 난제 – Key 관리

- 매우 민감한 고객 정보가 저장되어 있는 A 서비스
- 비밀 번호는 SHA256 과 Salt 로 hash 하여 보관중
- 개인 정보는 AES256-CBC 모드를 사용하여 암호화
- 방화벽과 침입 탐지 시스템 운영중

데이터와 Key 가 유출된다면?

- 침입자에 의해 DB 와 Key 가 유출되었다면?
- 암호화된 Data 라도 Key 가 있다면 무용지물
- 일반적으로 Key 는 Digital data 이므로 무한 복제 가능하며 유출 여부를 판단하기도 쉽지 않음
- 중요한 정보를 암호화할 경우 Key 를 안전하게 관리하는 것은 암호화의 마지막 관문

HSM(Hardware Security Module)

- Key 관리 문제를 해결해 주는 전용 하드웨어
- 장비내에서 Key 를 생성하고 내부에서 암호화 연산을 수행
- 장비내 Key 는 외부로 유출 안 됨(FIPS140-2)
- 보안 레벨이 높은 제품은 임의 분해시 Key 파괴 기능 내장
- 금융 등 Compliance 가 높은 서비스에서 많이 사용
- AWS 에서는 CloudHSM 이라는 전용 서비스 제공
  (매우 비싸고 서울 리전에는 없음)

HSM managed Service

- HSM 은 고가이고 사용하려면 별도의 프로그래밍 필요 (PKCS#11)
- 이 때문에 Cloud 업체에서는 Managed Service 제공
- AWS Key Management Service(KMS)
- Azure Key Vault Service
- Cloud 환경에서 중요한 정보를 암호화하려면 HSM Managed Service 권장

적절한 암호 알고리즘 선택법

알고리즘을 만들어 쓰지 말 것

- 현대의 암호 알고리즘은 암호화 방법이 모두 투명하게 공개됨
- 개발한 암호학자들은 해독시 현상금 지불
- 혹독한 검증을 거쳐 살아 남은 게 현대의 암호 알고리즘
- 스스로 만든 알고리즘은 전혀 검증이 되지 않았으므로 직접 설계 및 구현하지 말 것(그 시간에 서비스 개발...)

많이 쓰는 알고리즘 선택

- 대칭키는 AES, 공개키는 RSA 등 많이 사용되고 구현물이 많은 알고리즘 사용
- 대칭키 운영 모드로 ECB 는 절대 피할 것!
- Hash 함수와 암호화 key 길이는 데이터의 보존 연한을 고려하여 선택(양자 컴퓨터가 나오면 무용지물이라는데 그건 그때 가서 고민...)

사례별 암호화 적용방법

- 암호화는 시스템과 데이터가 털리는 최악의 상황에 대비한 최후의 방어책
- 보안 취약점 패치/업데이트, 방화벽, SELinux, 침입탐지, Secure Coding 등의 보안 대책을 먼저 적용해야 함

단방향 암호화(사용자 암호)

- Hash 함수보다는 *Bcrypt* 등의 password 해싱 전용 함수 사용
- 꼭 Hash 를 직접 사용해야 한다면 그래도 Bcyrpt
- 그래도 고집을 꺽지 못하겠다면 2가지 반드시 반영
- Rainbow attack 을 방지하기 위한 Salt 첨가 및 관리
- Brute-force attack 을 방지하기 위한 적절한 Key stretching

양방향 암호화(사용자만 해독 가능)

- Lastpass 같이 사용자의 데이터를 보관하는 서비스
- 고객의 신뢰 확보를 위해서는 서비스 제공자는 고객 데이터를 풀 수 있으면 안 됨
- 사용자만 풀 수 있도록 key 와 IV 는 PBKDF2 로 생성
- 알고리즘은 AES256-CBC 사용
- Brute-Force Attack 을 방지하기 위해 Key stretching 용 Iteration count 는 최소 몇 천번 이상 수행

양방향 암호화(서비스 제공자만 해독 가능)

- 고객의 신용 카드 정보, 계좌 정보등을 보관할 경우 서비스 제공자가 필요시 해독해서 사용해야 함
- 중요 데이터는 강력한 알고리즘 (AES256등) 을 사용하여 암호화
- 암호화에 사용한 Key 를 파일로 관리하면 위험
- HSM 을 도입하여 HSM 내부에 key 관리 및 암호화 연산 수행
- Cloud 환경일 경우 HSM Managed Service 사용

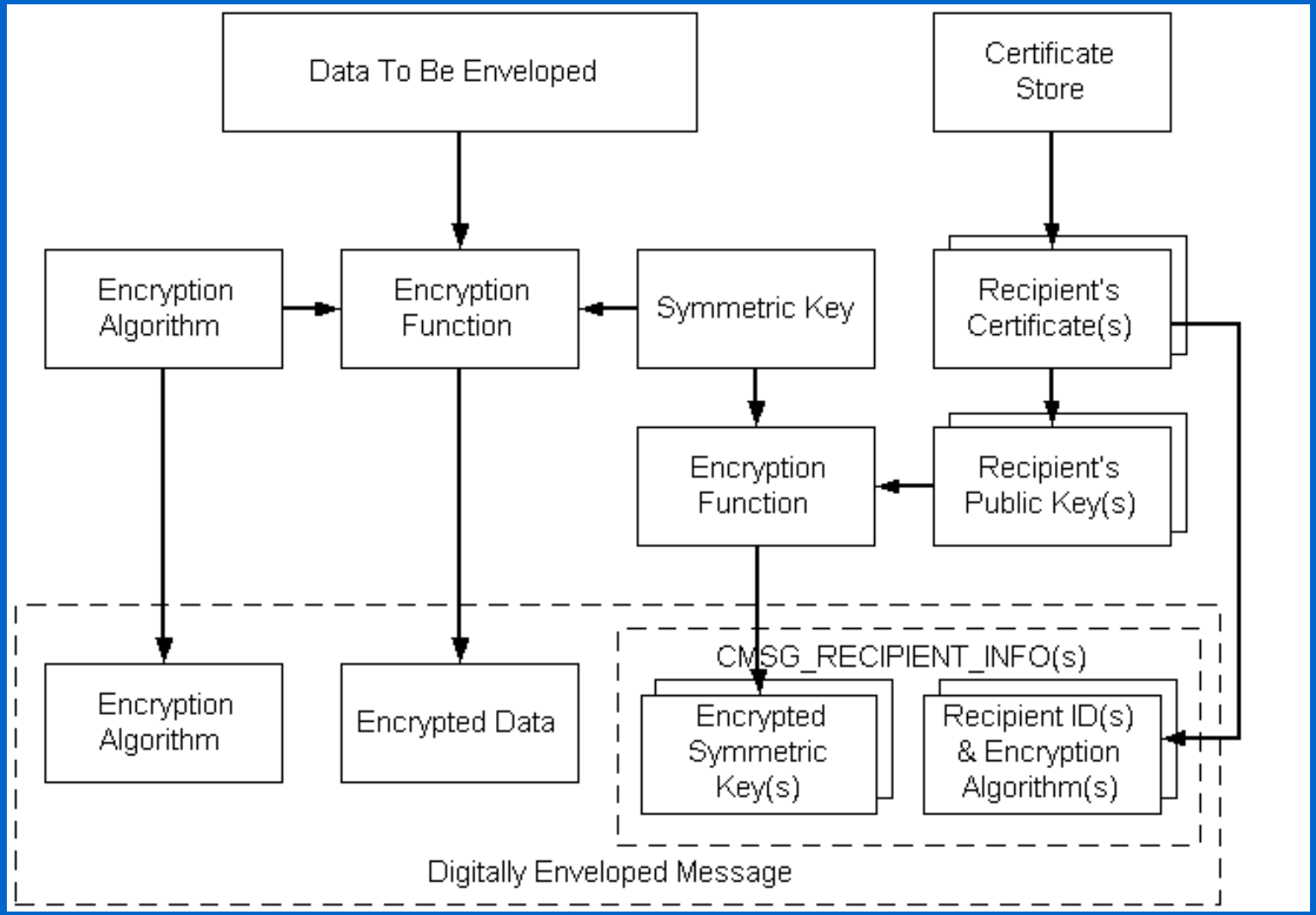
Transport layer 암호화

- Network 을 통해 상대방과 데이터를 주고 받을 경우
- 별도의 전용 프로토콜이 있는 곳(금융권등)이 아니면 무조건 HTTPS 를 사용
- 사용도 쉽고 보안도 견고(Web, WAS 기본 탑재, 방대한 구현 library..)
- 최신 버전의 TLS 와 알고리즘, HSTS(HTTP Strict Transport Security) 를 적용
- 자세한 내용은 https://lesstif.gitbooks.io/web-service-hardening/content/ssl-tls-https.html#hstshttp-strict-transport-security 참고

Application Layer 암호화

- HTTPS 로도 전송 계층의 암호화와 무결성이 보장
- 하지만 특정 업무 도메인의 경우 App layer 의 보안을 요구하는 경우가 있음
- 이런 용도를 위해 암호 메시지 규격(CMS; Cryptographic Message Syntax) 중 Enveloped Data 를 사용

CMS Enveloped Data



출처: https://msdn.microsoft.com/en-us/library/windows/desktop/aa382008(v=vs.85).aspx

CMS Enveloped Data

- 대칭키를 생성하여 데이터 암호화 => Encrypted Data (1)
- 수신자의 인증서에서 공개키 추출
- 암호화에 사용한 대칭키를 수신자 공개키로 암호화 -> Encrypted Symmetric Key (2)
- 수신자의 인증서 정보와 암호화 알고리즘 정보 추출 -> Recipient ID & Algorithm(3)
- 1, 2, 3 을 정해진 포맷에 담아서 수신자에게 전달
- 수신자는 개인키로 암호화된 대칭키 해독
- 대칭키로 Encrypted Data 해독

CMS Enveloped Data

- 키 교환중 키 암호화 기능을 활용한 유용한 표준
- 개인키 소유자만 해독할 수 있으므로 전자 봉투라고도 부름
- 대부분의 암호 라이브러리(BouncyCastle, OpenS니, Windwos Crypto) 에서 기능 제공
- 그러니 HTTPS 와 함께 Application Layer 의 암호화가 필요한 경우 Enveloped Data 사용

CMS Enveloped Data 응용

- Enveloped Data 로 만들면 XML 등 고유 언어는 의미 없는 바이너리 값으로 변경됨(암호화 되니…)
- XML element 만 암호화하여 장점을 살리고 싶다면?
- XML 은 XML Encryption 표준을 사용하면 XML 을 유지하며 암호화 가능
- JSON 도 Web Encryption 표준이 진행중이나 가볍고 빠른 곳에 많이 사용하는 JSON 의 특성상 많이 사용되지는 않는 듯 함.

참고 자료 & QnA

- https://github.com/lesstif/web-service-hardening
- http://d2.naver.com/helloworld/318732
- JAVA 자바 보안과 암호화
- 스토리로 이해하는 암호화 알고리즘
- Cryptographic Right Answers