핀테크 산업 응용

엄현상 교수님 강의 실습 2차시
반지훈, 서윤형
2022/05/16

Contents

1. GO DataBase 연동
2. 인터넷 보안
3. 해쉬 함수
4. 실습

1 Go Database 연동

- **MySQL 기기 설치**
  - > https://goddaehee.tistory.com/277 (MAC)
  - > https://goddaehee.tistory.com/277 (Window)

1 Go Database 연동

- **MySQL에 DB 만들기**

  > Create database ‘(명칭)’;

  > Show databases;

  > Use (database 이름);

  > MySQL 문법 참조
  http://www.tcpschool.com/mysql/mysql_basic_syntax

1 Go Database 연동

- **테이블 생성**

> Create문

```
CREATE TABLE 테이블이름
(
    필드이름1 필드타입1,
    필드이름2 필드타입2,
    ...
)
```

> 제약 조건(Constraint)

1. NOT NULL : 해당 필드는 NULL 값을 저장할 수 없게 됩니다.
2. UNIQUE : 해당 필드는 서로 다른 값을 가져야만 합니다.
3. PRIMARY KEY : 해당 필드가 NOT NULL과 UNIQUE 제약 조건의 특징을 모두 가지게 됩니다.
4. FOREIGN KEY : 하나의 테이블을 다른 테이블에 의존하게 만듭니다.
5. DEFAULT : 해당 필드의 기본값을 설정합니다.

- **테이블 생성 Example**

```
Create Table IF NOT EXISTS test
(id_key int not null auto_increment primary key,
user_id varchar(255) unique,
user_pw varchar(255))
```

- **테이블의 Record 선택**
  - > **SELECT문**

문법

```
SELECT 필드이름
FROM 테이블이름
[WHERE 조건]
```

- **테이블에 새로운 레코드 추가**
  - > **INSERT문**

문법

```
1. INSERT INTO 테이블이름(필드이름1, 필드이름2, 필드이름3, ...)
   VALUES (데이터값1, 데이터값2, 데이터값3, ...)
2. INSERT INTO 테이블이름
   VALUES (데이터값1, 데이터값2, 데이터값3, ...)
```

1 Go Database 연동

- **Go에서 MySQL Import**
  - > Installation / 설치
    - Go get –u github.com/go-sql-driver/mysql
    - Go get -u github.com/denisenkom/go-mssqldb
  - > 필요한 package import 하기

```
import (
    "database/sql"
    "fmt"
    _ "github.com/go-sql-driver/mysql"
)
```

1 Go Database 연동

- **Database에 연동할 구조체 만들고 선언하기**

  type [타입명] struct {
  필드명 타입
  …
  필드명 타입]

```
type dbInfo struct {
    user     string
    pwd      string
    url      string
    engine   string
    database string
}
```

```
var db1 = dbInfo{"root", "mypassword", "localhost:3306", "mysql", "dbtest"}
```

1 Go Database 연동

- **DB 연결**
  - > Data Source Name Format 맞추기

```
username:password@protocol(address)/dbname?param=value
```

```
db := dbInfo{"root", "qkswl110!@k43nuw8", "localhost:3306", "mysql", "testdb"}
dataSoruce := db.user + ":" + db.pwd + "@tcp(" + db.url + ")/" + db.database
```

  - > Connection: sql.Open[드라이버명, DataSource Format]

```
conn, err := sql.Open(db.engine, dataSoruce)
```

1 Go Database 연동

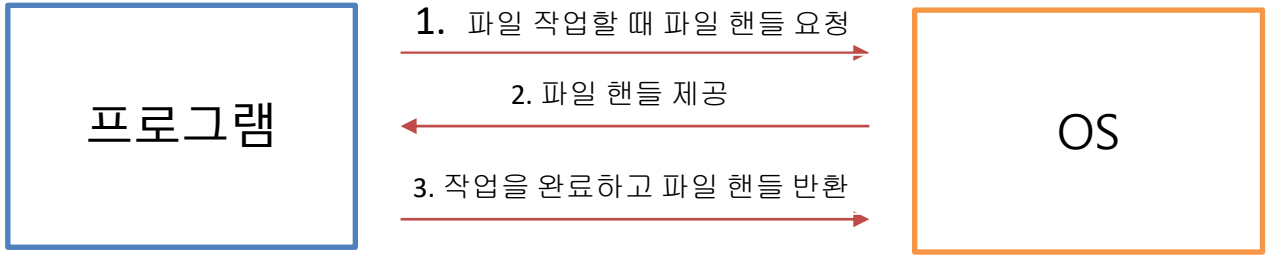
- **DB 연결 확인**
  > **에러 처리하기 & db.Ping()**

```
if err != nil || conn.Ping() != nil {
    log.Fatal(err)
}
```

1 Go Database 연동

- **Defer 지연 실행**

  defer [명령문]




```
defer conn.Close()
```

1 Go Database 연동

```go
package main

import (
    "database/sql"
    "log"

    _ "github.com/go-sql-driver/mysql"
)

type dbInfo struct {
    user     string
    pwd      string
    url      string
    engine   string
    database string
}

var db1 = dbInfo{"root", "mypassword", "localhost:3306", "mysql", "dbtest"}

func main() {

    db := dbInfo{"root", "qkswl110!@k43nuw8", "localhost:3306", "mysql", "testdb"}
    dataSoruce := db.user + ":" + db.pwd + "@tcp(" + db.url + ")/" + db.database
    conn, err := sql.Open(db.engine, dataSoruce)

    if err != nil {
        log.Fatal(err)
    }
    defer conn.Close()

}
```

1 Go Database 연동

- **Table 생성 및 INSERT**

```
_, err = conn.Exec("CREATE TABLE orangeTest(id VARCHAR(255), name VARCHAR(255))")

if err != nil {
    log.Fatal(err)
}

_, err = conn.Exec("INSERT INTO orangeTest(id, name) VALUES ('GREAT', 'GREAT')")
if err != nil {
    log.Fatal(err)
}
```

1 Go Database 연동

- **Table에 Insert 되었는 지 확인 방법**
  - > **RowsAffected() method 사용**

```
result, err := conn.Exec("INSERT INTO orangetest VALUES (?, ?)", "carrot", "orange")
if err != nil {
    log.Fatal(err)
}

n, err := result.RowsAffected()
if n == 1 {
    fmt.Println("1 row inserted")
}
```

1 Go Database 연동

- **Select Query 단일 실행**

```
//Select 확인 법
var name string
err = conn.QueryRow("SELECT id from orangetest where id = 'carrot4'").Scan(&name)
if err != nil {
    log.Fatal(err)
}
fmt.Println(name)
```

1 Go Database 연동

- **Select Query 다수 실행**

```
// 다수의 ROW에서 다음 Row로 이동하기위해 NEXT() 메서드를 사용하는데, 반복문을 사용하여 체크합니다.
var id string
var name1 string

rows, err := conn.Query("SELECT id, name FROM orangetest where id = 'carrot1'")
if err != nil {
    log.Fatal(err)
}
defer rows.Close()

for rows.Next() {
    err := rows.Scan(&id, &name1)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(id, name1)
}
```

Contents

2. 인터넷 보안 개요

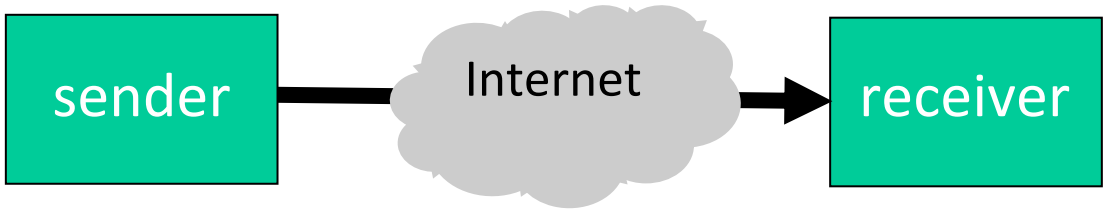
1. 인터넷 보안이란?
2. 보안 서비스
3. 암호 시스템

- **보안의 종류**
  - > Physical security
  - > Resource exhaustion
  - > Computer/System security
  - > Internet/Network security
  - > Cryptography
  - > Social engineering
  - > Emotional security

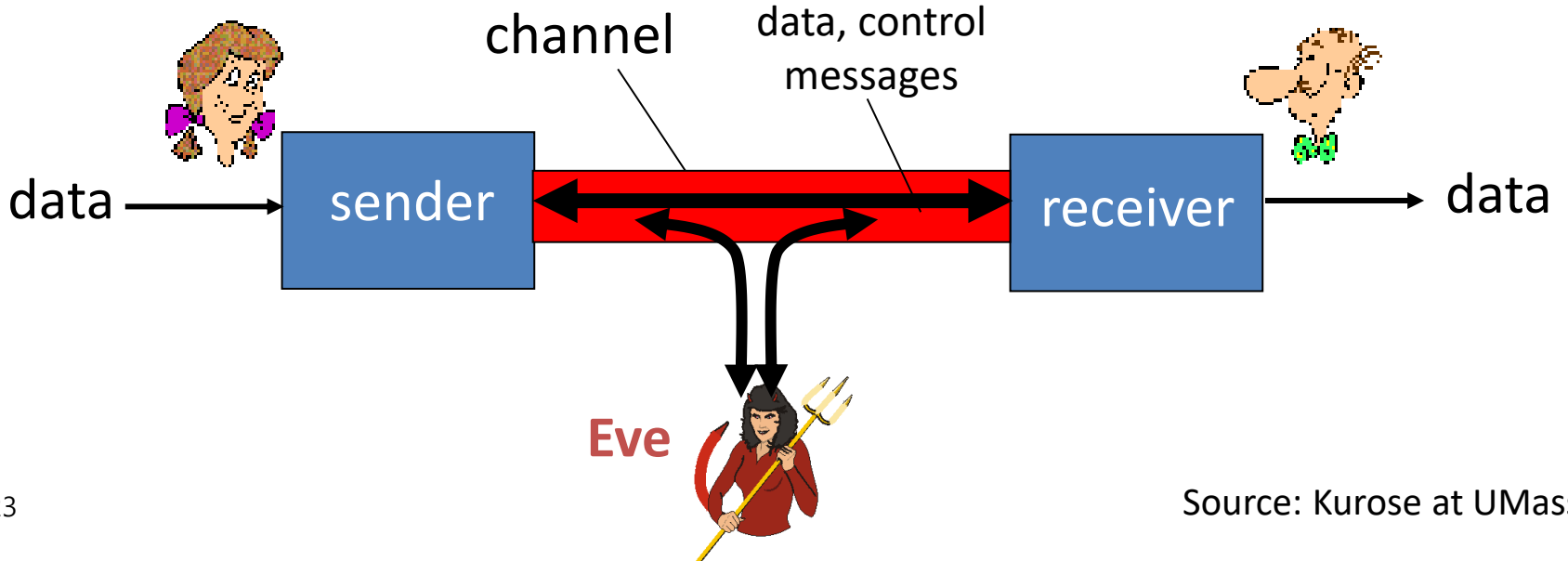
1 인터넷 보안이란?

- **컴퓨터보안 vs 인터넷보안**
  - > **컴퓨터 보안**
    - **컴퓨터가 네트워크에 연결되어 잇더라도 컴퓨터의 데이터를 보호하는 자동화 된 도구 및 메커니즘**
      - **Against hackers (intrusion), malware, …**
      - **Access control, authorization, …**
  - > **인터넷 보안**
    - **네트워크 또는 상호 연결된 네트워크에서 정보 전송과 관련된 보안 위반을 방지, 탐지 및 수정하는 조치를 함**
    - **네트워크의 모든 것이 대상이 될 수 있음**
    - **전송된 모든 비트를 tapped 할 수 있음**

- **Internet Attack Model**



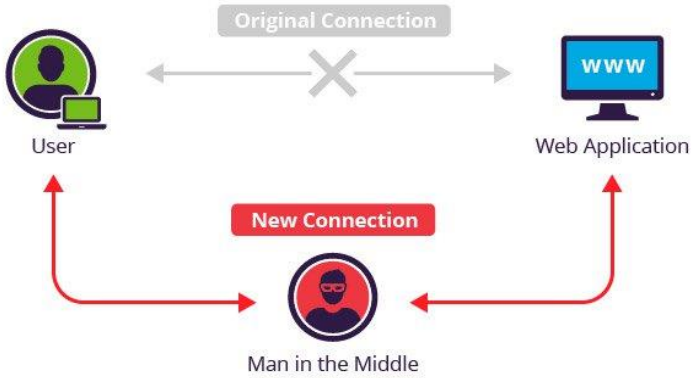
- **Internet Attack Model**
  - > Alice and Bob want to communicate securely
  - > Eve (intruder) map tap, delete, add, modify messages



Source: Kurose at UMass

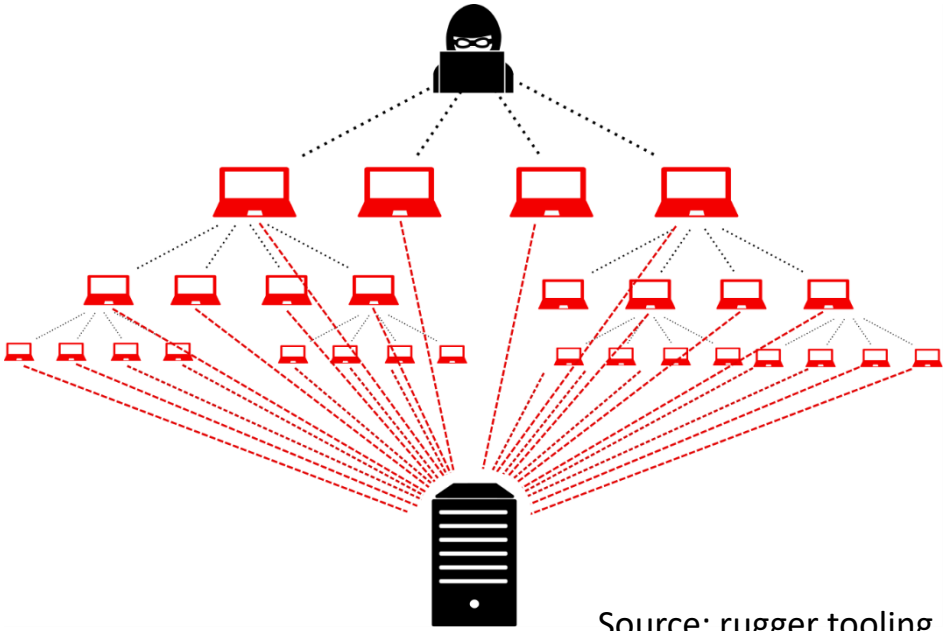
- **인터넷 공격 유형**
  - > Eavesdrop: intercept messages
  - > Insert/modify/delete messages inside the network
  - > Impersonation: fake (spoof) source address in packet or sender
  - > Hijacking: “take over” ongoing connection by removing sender or receiver, inserting himself in place
  - > Denial of service (DoS): prevent service from being used by others (e.g., by overloading resources)

- **실제 인터넷 공격 종류 (1/2)**
  - > **Wiretapping**
  - > **Port scanning**
  - > **Cryptographic attack**
    - **Find the key, find the plaintext for the ciphertext**
  - > **Spoofing**
    - **Phishing, pharming**
    - **Man-in-the-middle**



Source: incapsula

- **실제 인터넷 공격 종류 (2/2)**
  - > Denial of service: host-based vs distributed Dos (DDoS)
  - > Poisoning
  - > Session hijacking
  - > Information gathering attack
    - Phone, Web, SNS,...
  - > Spam

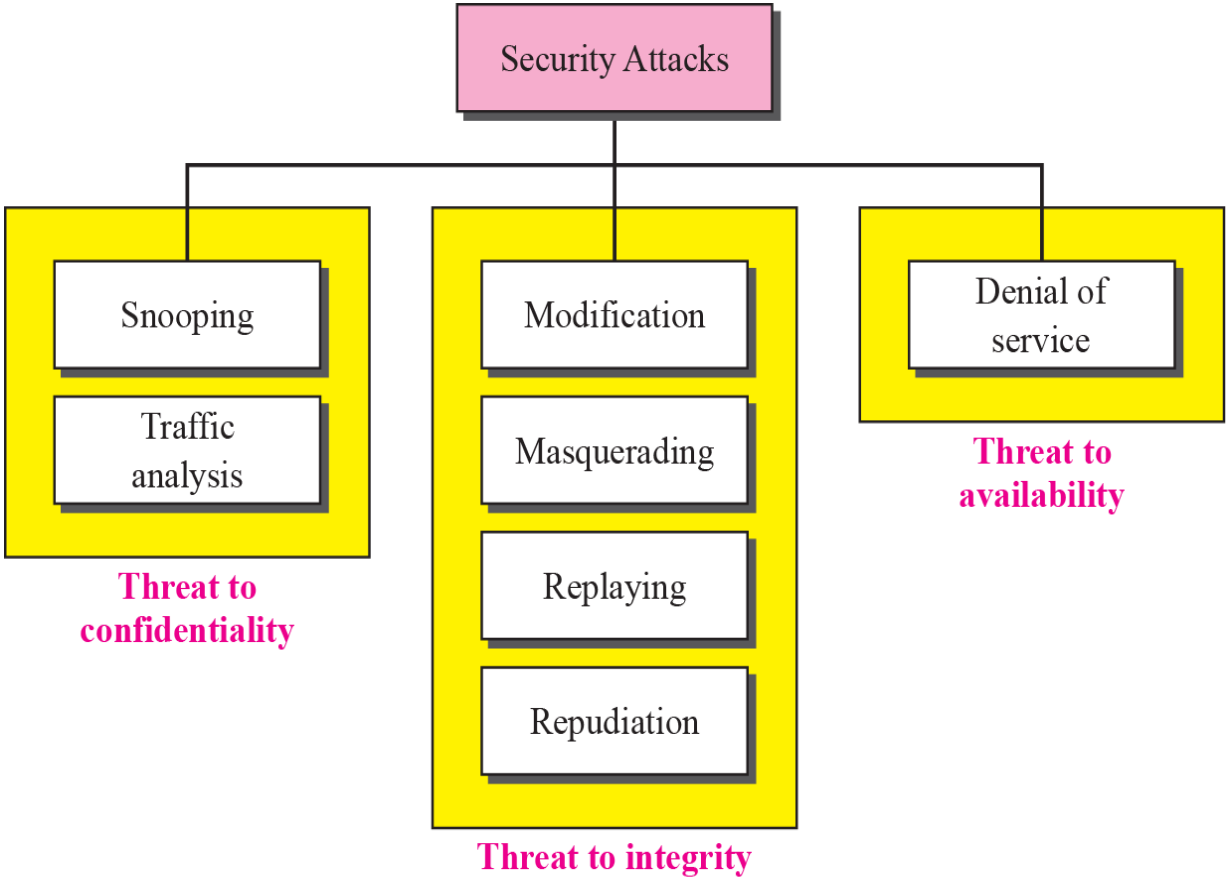


Source: rugger tooling

2 보안 서비스

- 기본 보안 서비스 종류
  - > Confidentiality: 기밀성
  - > Integrity: 무결성
    - Data (or message) integrity
  - > Availability: 가용성
  - > Authentication: 인증
  - > Non-repudiation: 부인 방지

- **Attacks against CIA**



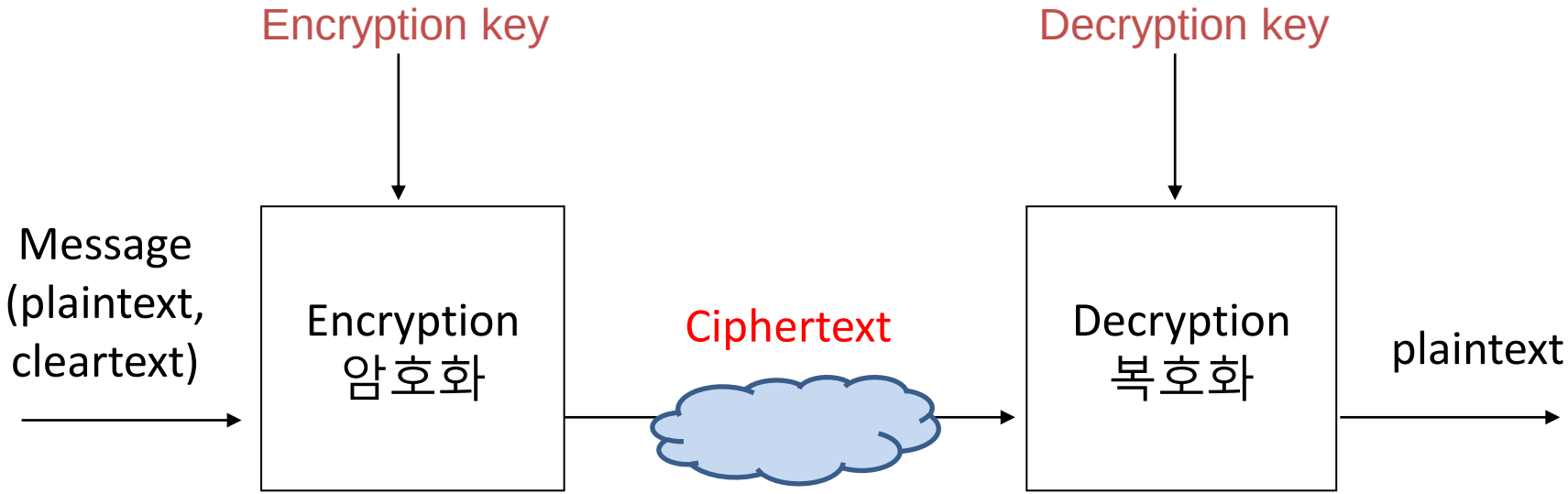
Source: Forouzan at DeAnza

- **추가 보안 서비스 종류**
  - > **Access control: 접근 제어**
    - **Identification (식별)**
    - **Authentication**
    - **Authorization (권한부여, 인가)**
  - > **Anonymity: 익명화**
  - > **Accountability: 책임성**
  - > **Privacy**
  - > **Digital forensics**

2 보안 서비스

- 보안 매커니즘
  - > Encryption(암호화) and decryption(복호화)
    - Key
    - Confidentiality, authentication
  - > Message digest
    - Hash function
    - Integrity
  - > Digital Signatures (디지털 서명)
    - Demonstrating the authenticity of a message
    - Authentication, integrity, non-repudiation
  - > Append-only public server (e.g., Blockchain)
    - Integrity

3 Cryptosystem (암호 시스템)



cipher - algorithm for performing encryption or decryption
key - info used in cipher known only to sender/receiver
encrypt - converting plaintext to ciphertext
decrypt - recovering ciphertext from plaintext

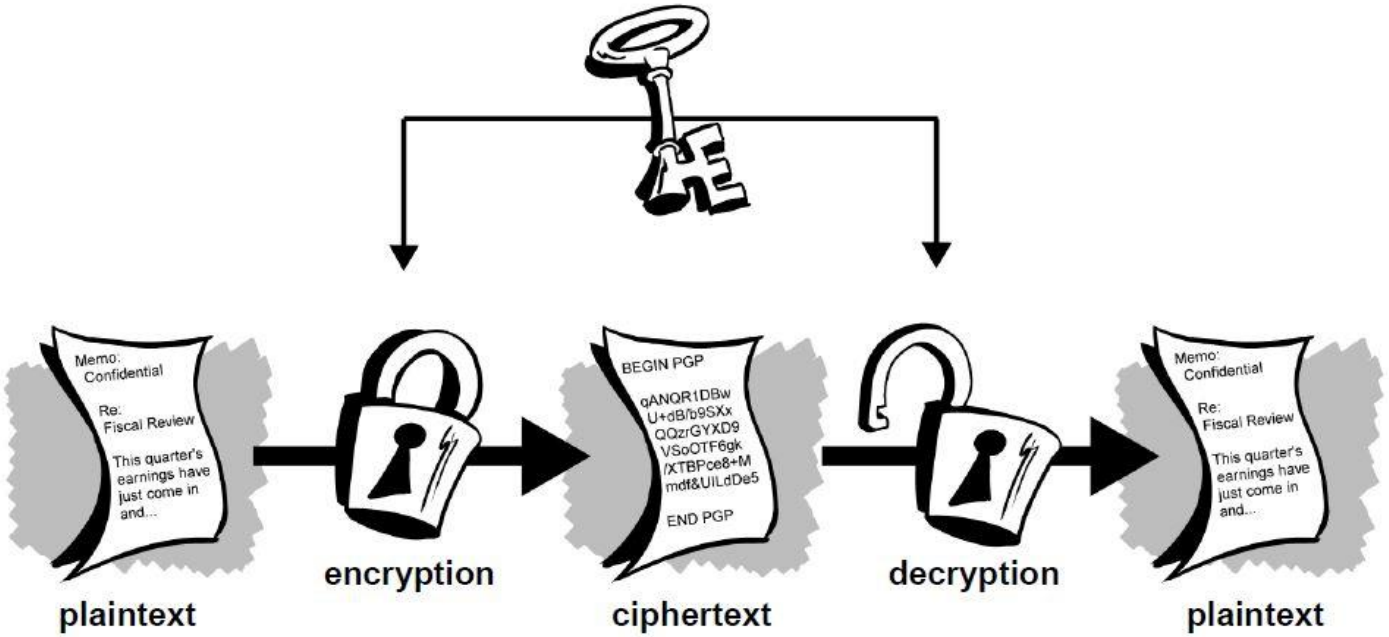
- **암호화 기법의 분류**
  - > **Key가 사용되는 방식에 따라**
    - **대칭키 암호화: Single Key**
    - **공개키 암호화: Public Key와 Private Key**
  - > **평문이 처리되는 방식에 따라**
    - **Block cipher**
    - **Stream cipher**

- **Kerckhoffs's Principle**
  - > **two choices for security of a cryptosystem**
    1. **encryption/decryption algorithm can be hidden**
       - security by/through obscurity
    2. **key can be hidden**
  - > **A cryptosystem should be secure even if everything about the system, except the key, is public knowledge**

3 암호 시스템: 대칭키 암호화

- **Symmetric key cryptography**
  - > **Single Key, Shared Key**
  - > **A cryptosystem should be secure even if everything about the system, except the key, is public knowledge**

3 암호 시스템: 대칭키 암호화



Source: Chris Pacia @ Escape Velocity

3 암호 시스템: 대칭키 암호화

- **대칭키 암호화 모델의 요구사항**
  - > **Two requirements for secure use of symmetric encryption:**
    - **A strong encryption algorithm**
      - Cryptanalysis
    - **A secret key is known only to sender and receiver**
      - » $Y = E_K(X) = E(K,X)$
      - » $X = D_K(Y) = D(K,Y)$
  - > **Assume encryption/decryption algorithm is known**
    - **Kerckhoffs's Principle**
  - > **Imply a secure channel to distribute the key**

X: plaintext
Y: ciphertext
K: shared key
E: encryption cipher
D: decryption cipher

3 암호 시스템: 대칭키 암호화

- **암호화 알고리즘의 요구사항**
  - 암호학에서 Confusion과 Diffusion은 보안 암호의 작동의 두가지 특성이 있음 **[Claude Shannon]**
  - **Diffusion** 이란 만약 우리가 일반 텍스트의 문자를 바꾼다면, 암호 텍스트의 몇 개의 문자가 바뀌어야 하고, 마찬가지로, 우리가 암호 텍스트의 문자를 바꾼다면, 일반 텍스트의 몇 개의 문자가 바뀌어야 한다는 것을 의미 함
  - **Confusion** 은 키가 암호문과 간단한 방식으로 관련되지 않다는 것을 의미한다. 특히, 암호문의 각 문자는 키의 여러 부분에 의존해야 함

3 암호화 시스템: 공개키 암호화

1) 공개키 암호화

> 정의

- **암호 방식의 한 종류로 사전에 비밀 키를 나눠 가지지 않은 사용자들이 안전하게 통신할 수 있게 함**

- **공개 키와 비밀 키가 존재하며, 공개 키는 누구나 알 수 있지만 그에 대응하는 비밀 키는 키의 소유자만이 알 수 있어야 함**

- **일방향 함수(One-way function)를 기반으로 함**
  - 일방향 함수 : 정방향은 계산하기는 쉽지만 역방향을 구하는 것은 어려운 함수

3 암호화 시스템: 공개키 암호화

2) 종류

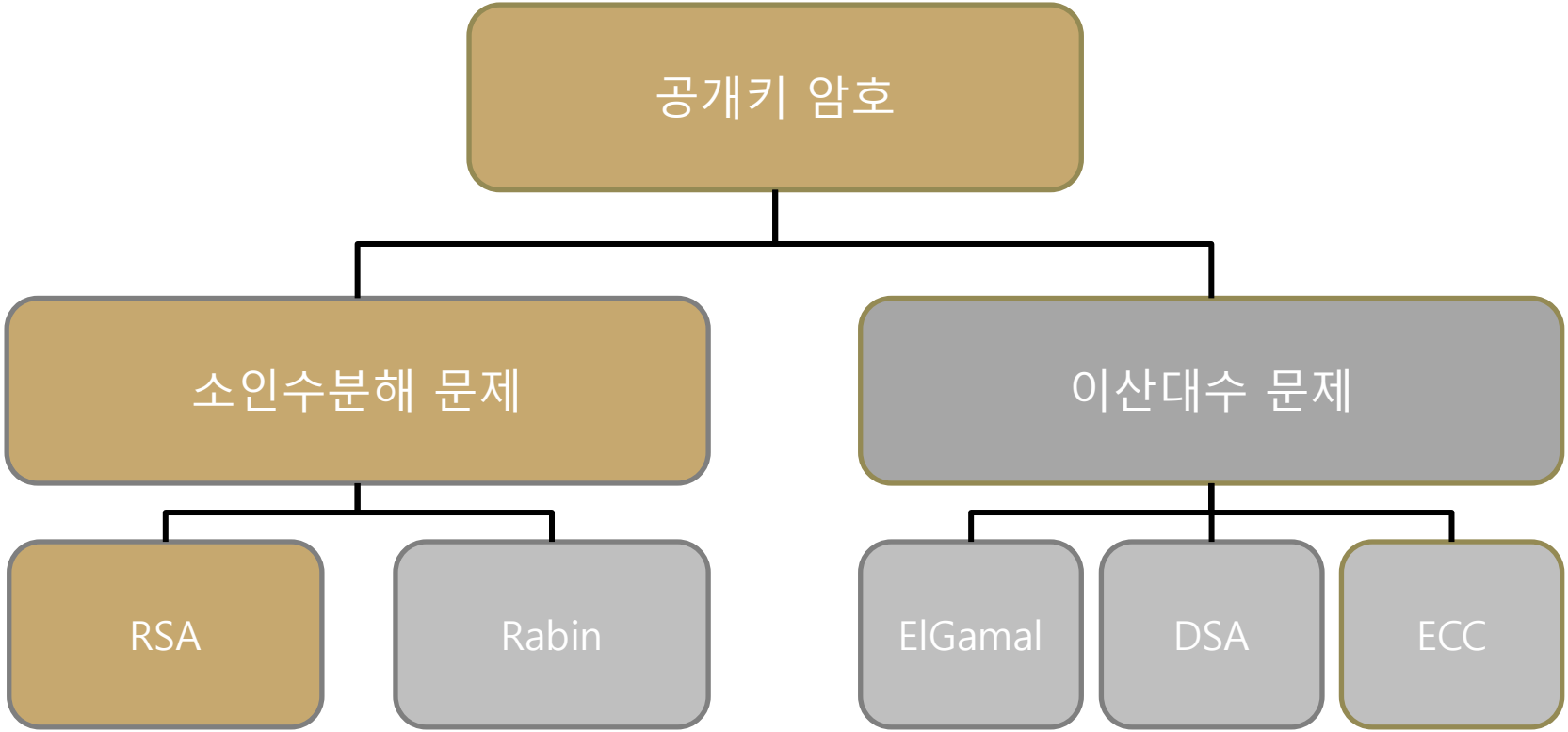
> 소인수 분해

- **2개의 큰 소수들을 곱하는 것은 간단하지만 두 소인수들을 찾는 것은 어려움**
- **소인수 하나를 제시하면 나머지 소인수를 쉽게 찾을 수 있음**
  - 트랩도어 함수 (Trapdoor functions) :
    - » 특수한 종류의 일방향 함수로 일반적으로 역을 구하는 것은 어렵지만, 트랩도어(trapdoor)라고 하는 비밀정보를 얻으면 역을 쉽게 구할 수 있음
    - » 대표적인 예
      - ➢ RSA (Rivest–Shamir–Adleman)

> 이산로그, 타원 곡선 암호화

- **소수로 나눈 나머지를 곱하는 것은 간단하지만, 나눗셈은 사실상 불가능 (현재까지 알려진 트랩도어 없음)**

3 암호화 시스템: 공개키 암호화



공개키 암호화 알고리즘 구성도

3 암호화 시스템: 공개키 암호화

3) 구성요소

> 공개키 (은행 계좌 번호) : 계정 식별

> 개인키 (계좌 비밀 번호) : 계정에 대한 제어권 제공

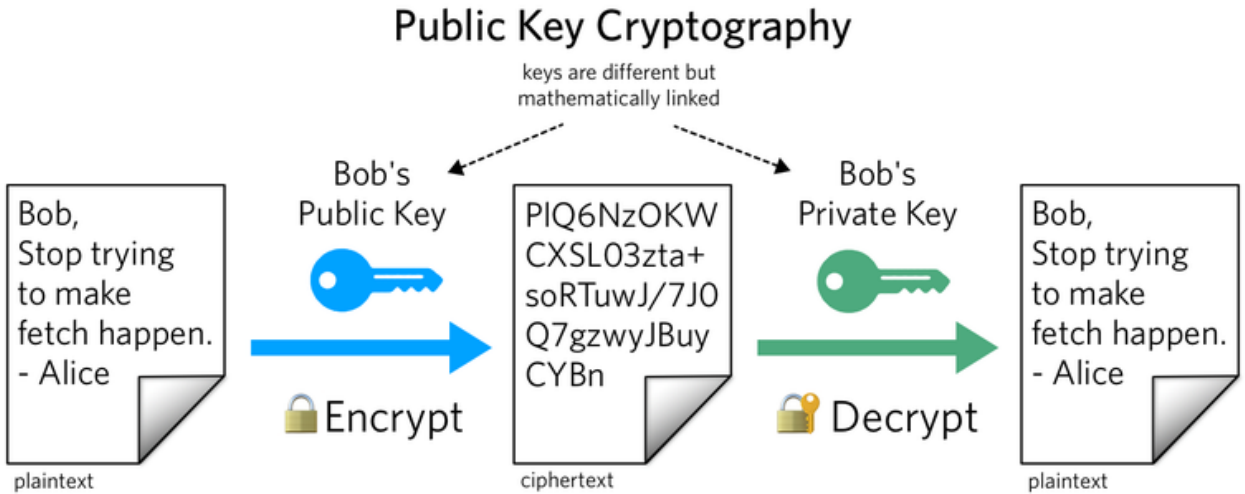


그림 출처: https://www.twilio.com/blog/what-is-public-key-cryptography

Contents

4. 해쉬 함수

1.암호화 해시 함수

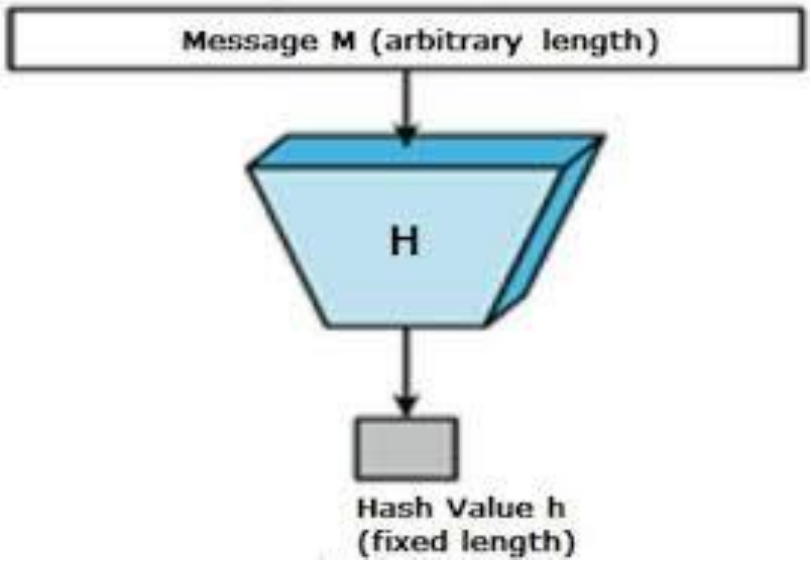
- 개요
- random oracle
- 해시 함수 구조

2. SHA

- SHA-1
- SHA-2
- SHA-3

1 What is a hash function?

- 임의의 크기의 데이터를 고정 크기의 데이터에 매핑하는데 사용할 수 있는 함수임
  - > Output: hash, digest, tag, fingerprint, …
- 해시 함수는 큰 파일에서 중복된 Record를 탐지하여 테이블 또는 데이터베이스 검색을 가속화 할 수 있음.
  1) H: $\{0,1\}^* \rightarrow \{0,1\}^n$



Source: tutorialspoint.com

1 Classifications of hash functions

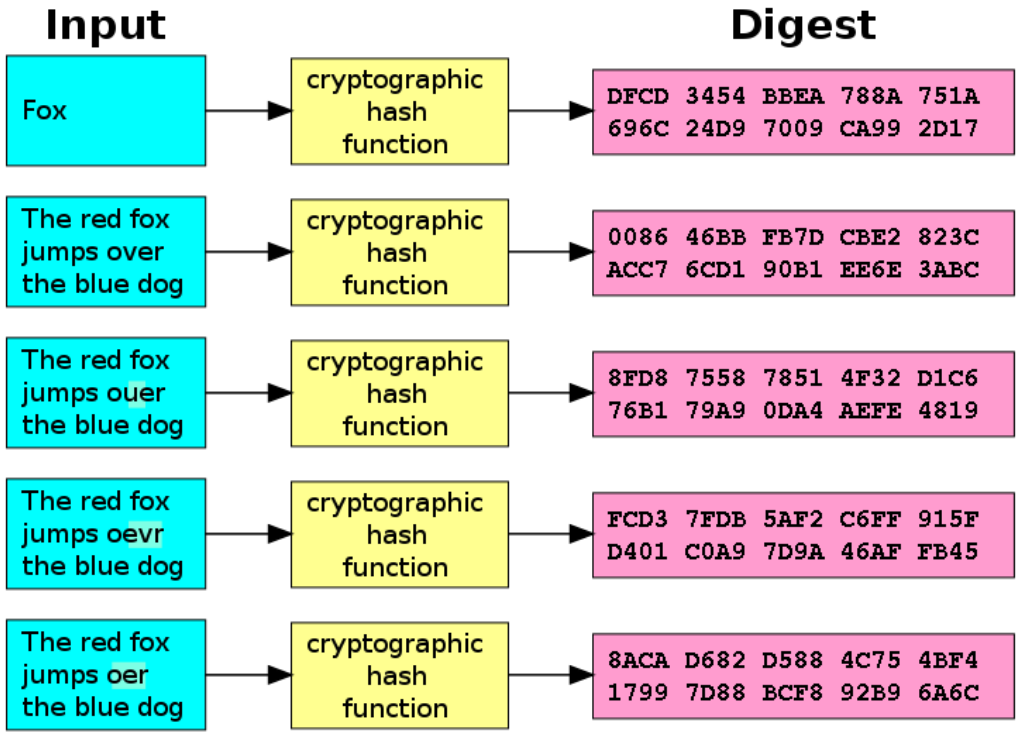
- **Cryptographic**
- **Non-cryptographic**
  - > **Try to avoid collisions for non malicious input**
  - > **Weaker guarantee on collision avoidance**
  - > **E.g., data corruption check in communications**

1 Classifications of cryptographic hash function

- **A cryptographic hash function (CHF) is a special class of hash functions that has certain properties which make it suitable for use in cryptography**

- **What properties?**
  - > **Maybe, output should be sufficiently long**
  - > **What else?**

1 An Example of CHF



"avalanche effect"

Source: wikipedia

1 Another classification of hash functions

- **Unkeyed hash**
- **Keyed hash**
  - > **Two inputs: message and key**

1 Desirable properties of CHF

- **One-Wayness:**
  - Given y, it is infeasible to find x such that $h(x) = y$
- **Collision-resistance:**
  - It is infeasible to find x_1 and x_2, such that $x_1 \neq x_2$ and yet $h(x_1) = h(x_2)$
- **Second pre-image resistance:**
  - Given x, it is infeasible to find $x' \neq x$ such that $h(x') = h(x)$
  - Target collision resistance

- **Pseudo-randomness**
- **Non-malleability**
  - Given $h(x_1)$, It is infeasible to derive $h(x_2)$ where x_1 and x_2 are related,
  - E.g., $x_2 = x_1 + 1$

1 Applications of CHF

- **Password storage**
- **File modification detector**
- **Digital signature**
- **Commitment**
  - **E.g., auction**

1 Random oracle (model)

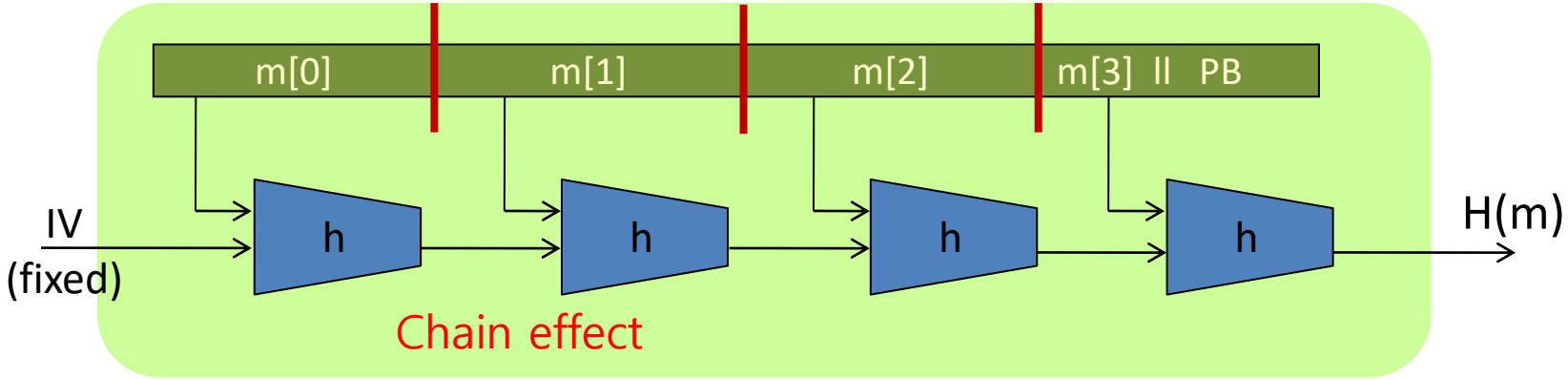
- **A random oracle (RO) is an oracle (a theoretical black box) that responds to every unique query with a (truly) random response chosen uniformly from its output domain**
- **If a query is repeated, it responds the same result**
- **It is an ideal cryptographic hash function**

- **How the random oracle behaves:**

1. 게임 시작 시 오라클의 테이블은 비어 있음
2. 상대방이 Oracle에게 메시지를 해시 하도록 요청할 때마다 오라클은 먼저 해당 입력 값이 테이블에 이미 저장되어 있는 지 확인
3. 그렇지 않으면 임의 출력 문자열을 생성하고 입력 메시지와 새 출력 문자열을 테이블에 저장한 다음 출력 문자열을 호출자에게 반환
4. 오라클이 테이블에서 입력 값을 찾으면 이미 저장된 출력 값을 반환

How to build a hash function

1 Merkle-Damgärd iterated construction



Thm: h collision resistant ⇒ H collision resistant

Goal: construct compression function $h: T \times X \longrightarrow T$

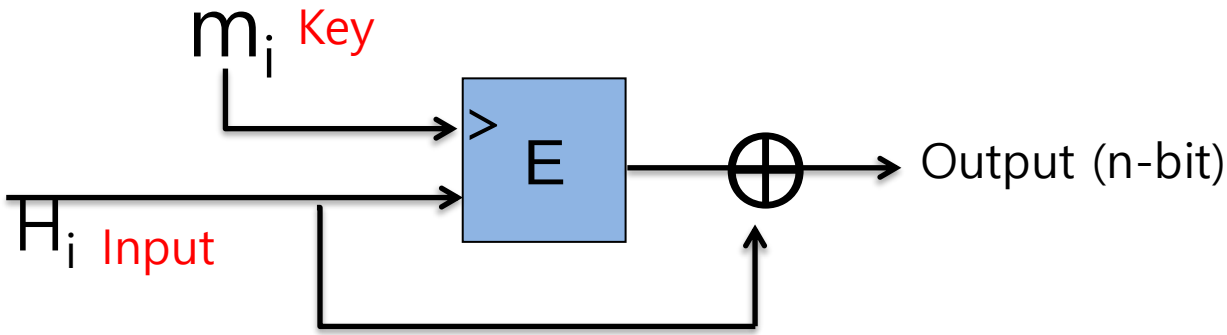
padding block (PB): 1000…0 ll msg len

64 bits

1 Compression function h from a block cipher

- **E: K× {0,1}^{n} ⟶ {0,1}^{n} a block cipher.**
- **Davies-Meyer (D-M)** compression function

  : $h(H_i, m) = E(m, H_i) \oplus H$



- **Thm: Suppose E is an ideal cipher (|K| > 2^n!)**
- **Finding a collision h(H,m)=h(H',m') takes O(2$^{n/2}$) evaluations of (E,D)**

1 Why is there XOR in D-M fn.?

- **For example, let's say you don't XOR, you only encrypt the IV with the block of text you are hashing. Let's call the result H1**

$$H1=E(IV1,text1)$$

We can now create a random block of the same size as the IV and decrypt

$$text2 = D(IV2,H1)$$

And now we know that: E(IV2,text2) = H1

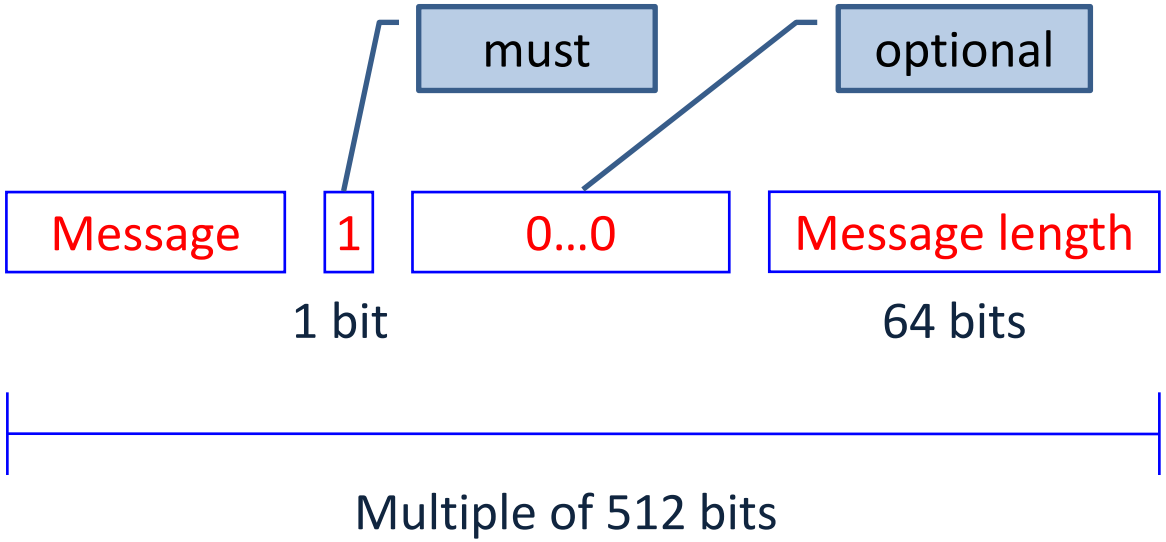
Then E(IV1,text1) = E(IV2,text2)

Thus we have created a collision

2 SHA-1

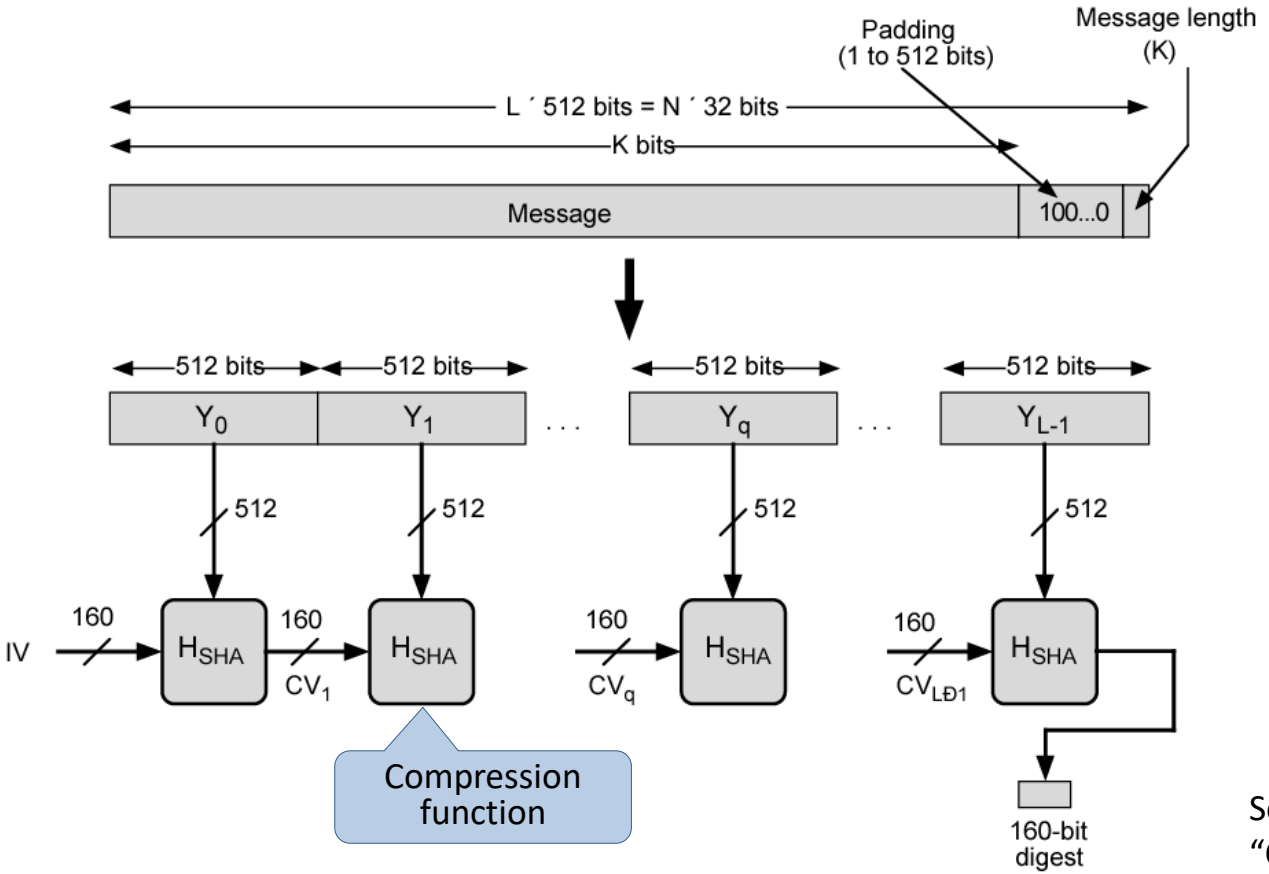
- Secure hash algorithm 1
- It was published as FIPS PUB 180-1 by NSA in 1995
- produces 160-bit hash values
- Merkle-Damgård + Davies-Meyer structure
- It is not recommended for use since 2005
  - collision attacks found
- Microsoft, Google, Apple and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017

2 Padding first



If message is 512k+447 bits: k+1 blocks
If 512k+448 bits: k+2 blocks

2 Overall structure



Source: W. Stallings
"Cryptography and Network Security"

2 Some notation and terminology

- Digest Length = 160 bit
- Message Block = 512 bit
- Sub-block (or word) size = 32 bit
- 512/32 = 16 total Sub-blocks (or words)
- No. of Rounds = 4
- 80 iterations (t): (No. of Rounds = 4) X (Iterations per round = 20)
- Chaining Variable (CV) = 5*32 = 160 bits = [A,B,C,D,E]
- K[t] = constants per round (32 bits each, where t=0 to 79)
- Output → five sub-blocks (5*32 = 160 bits)

2 SHA-1 algorithm (1/2)

1. **Padding: Length of the message is 64 bits short of multiple of 512 after padding (bit sequence 100…0)**
2. **Append: a 64-bit length value of original message is taken**
3. **Divide the input into 512-bit blocks**
4. **Initialize CV (i.e. CV[0]): 5-word (160-bit) buffer (A,B,C,D,E) to:**

(A=01 23 45 67,
B=89 AB CD EF,
C=FE DC BA 98,
D=76 54 32 10,
E=C3 D2 E1 F0)

Nothing Up My Sleeve numbers

4A. Constants in a compression fn

$K_0 – K_{19}$ = 5A827999
$K_{20} – K_{39}$ = 6ED9EBA1
$K_{40} – K_{49}$ = 8F1BBCDC
$K_{60} – K_{79}$ = CA62C1D6

2 SHA-1 algorithm (2/2)

5. **Process Blocks: now the actual algorithm begins**
   message in 16-word (512-bit) chunks:
   - CV를 단일 버퍼에 복사하여 임시 중간 및 최종결과를 저장
   - 현재 512bit block을 16개의 하위 블록으로 나누고 (Word[0]..Word[15]), 각각 32 비트로 구성
   - 각 라운드는 메시지 블록 및 버퍼에서 20비트/단계 반복 작업으로 구성된 라운드 수 = 4
   - Mixing & shiftin을 통해 16개 단어를 80개 단어(Word[0…79])로 확장
   - K[t]는 반복 t 범위(0…79) 따라 4개의 상수 중 하나
   - 입력에 출력을 추가하여 새 버퍼 값 형성
6. **Output hash value is the final buffer value**

2 SHA-1 algorithm (2/2)

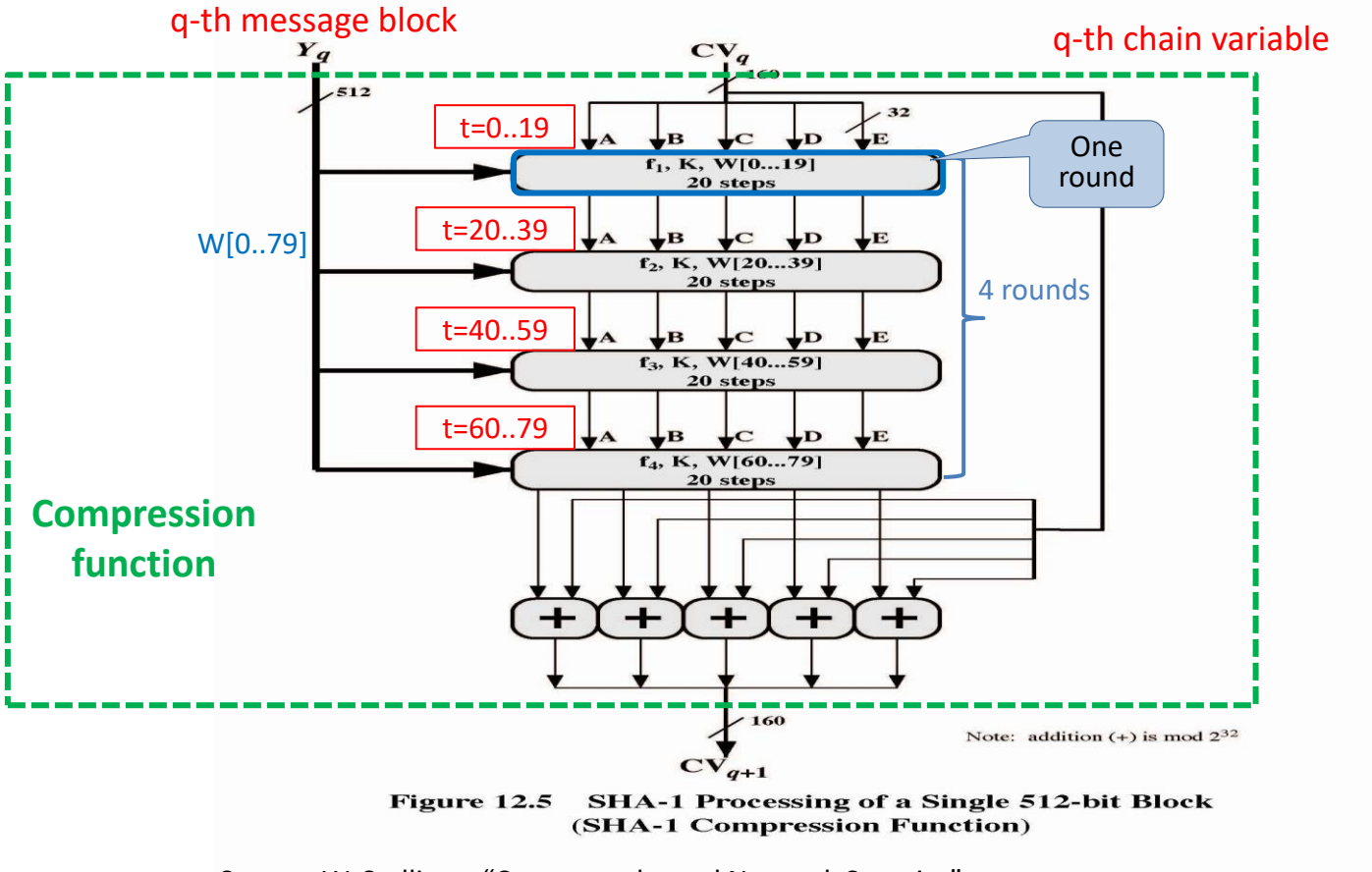
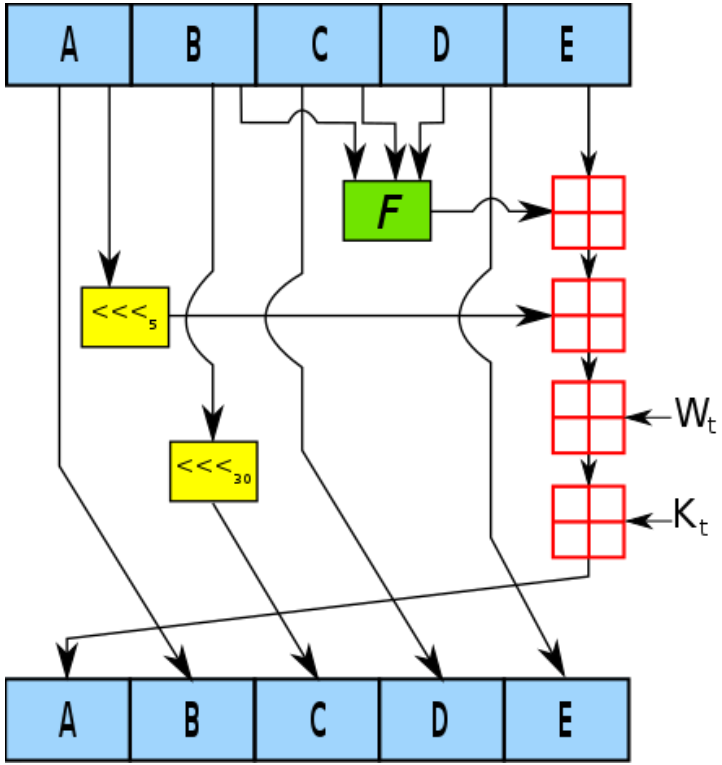


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

Source: W. Stallings, "Cryptography and Network Security"

2 one round in a Compression Function (SHA-1)



Source: wikipedia

A, B, C, D, E (of CV): each 32-bit word of the state;
F (= f_t) is a nonlinear fn. that varies at each round;
$<<<_n$ denotes a left bit rotation by n bits;
W_t is the expanded message word of round/step t;
W_t is the incoming msg block when t<16;
K_t is the constant that varies at each round;
$\boxplus$ denotes addition modulo 2^{32}

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lhd 1 \qquad 16 \le t \le 79$$

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{if } 0 \le t \le 19 \\ B \oplus C \oplus D & \text{if } 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \le t \le 59 \\ B \oplus C \oplus D & \text{if } 60 \le t \le 79 \end{cases}$$

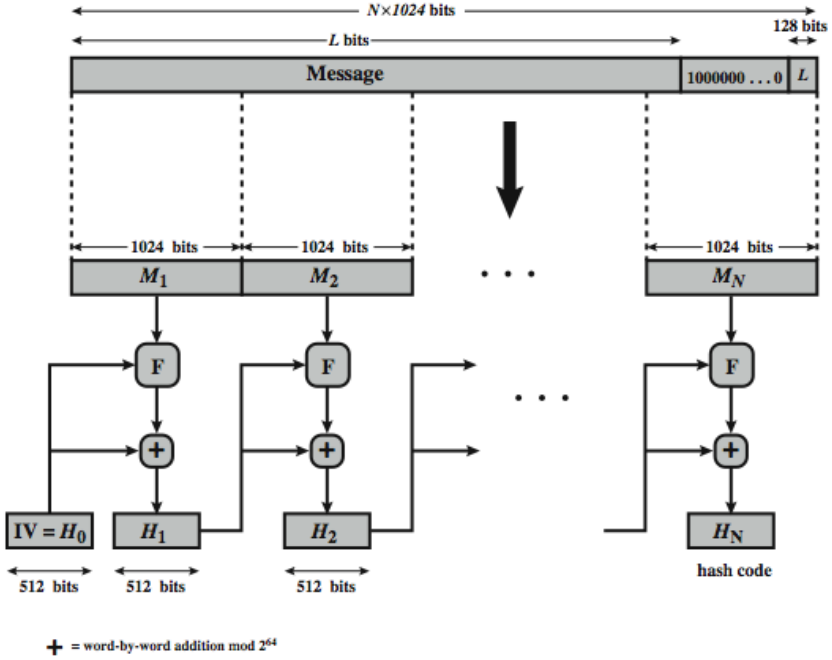
2 SHA-2

- 2001년 NSA에 의해 출판
- SHA2 계열은 digests(hash 값)가 224, 256, 384 또는 512비트인 6개의 해시 함수로 구성:

  SHA-224, SHA-256, SHA-384 SHA-512,

  SHA-512/224, SHA-512/256
- SHA-1에 비해 단어의 bit 간 mixing이 크게 증가함
- 여전히 Merkle-Damgård 구조이고, 길이 연장 공격의 영향을 받음
- F 는 compression 함수

SHA-512



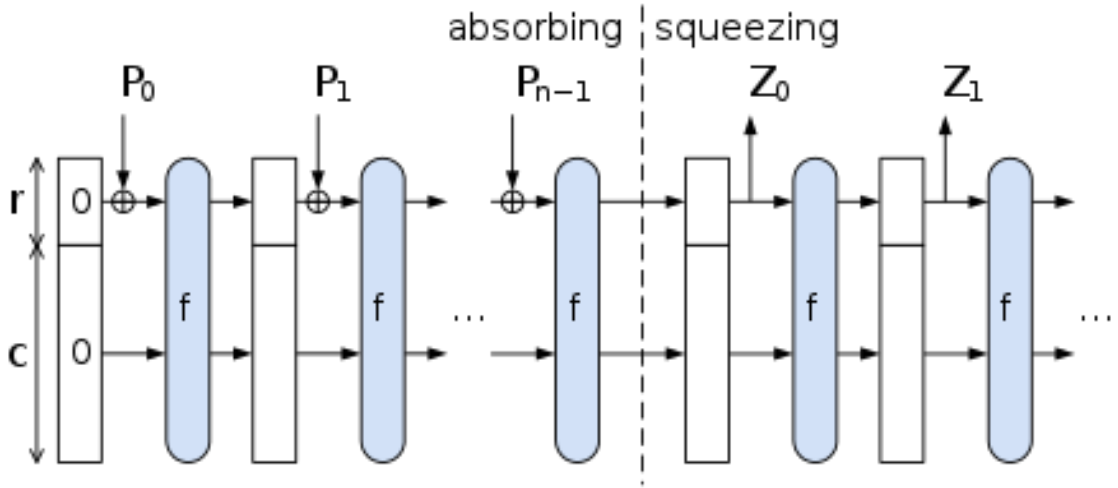
Source: W. Stallings, “Cryptography and Network Security”

2 SHA-512 (compression fn.) in SHA-2

- **Heart of the algorithm**
- **Processing message blocks (each 1024 bits long)**
- **Each CV (and digest) is 512 bits: 8 words each 64 bits long**
- **Consists of 80 rounds (t)**
  - > updating a 512-bit buffer
  - > using a **64-bit** value W_t derived from the current message block
  - > K_t : round constants based on cube root of first 80 prime numbers

2 SHA-3

- **NIST, 2015.**
- **No Merkle-Damgård structure**
- **Different structure from SHA-0,1,2: sponge construction**



Source: wikipedia

f: permutation fn. that uses xor, and, not operations
r: size of the part of the state that is combined with message blocks
c: size of the part of the state that is not combined with message blocks
P_i: input
Z_i: output

Contents

4. 실습

1. RSA Encryption / Decryption 실습

- 실습 목표
- 간단한 crypto package 설명 및 연습
- 암호화/ 복호화 실습

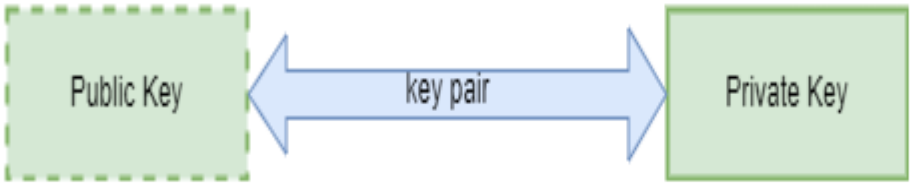
1 실습 목표

- RSA 키 쌍(개인키와 공개키) 생성 실습
  - Func GenerateKey을 만든 후 PrivateKey and Public Key Return 하기
- RSA 암호화/복호화 수행
  - “우리 은행“ 단어를 가지고 sha256을 맞추어서 encrypt 한 후 return 하기
  - Encryption 한 것을 가지고 Decrypt 한 후 return 해보기

2 Crypto package

- ## Concept of the Generate Key

RSA works by generating a public and a private key. The public and private keys are generated together and form a key pair.



func **GenerateKey**

```
func GenerateKey(random io.Reader, bits int) (*PrivateKey, error)
```

GenerateKey generates an RSA keypair of the given bit size using the random source random (for example, crypto/rand.Reader).

Source: https://pkg.go.dev/crypto/rsa

2 Crypto package

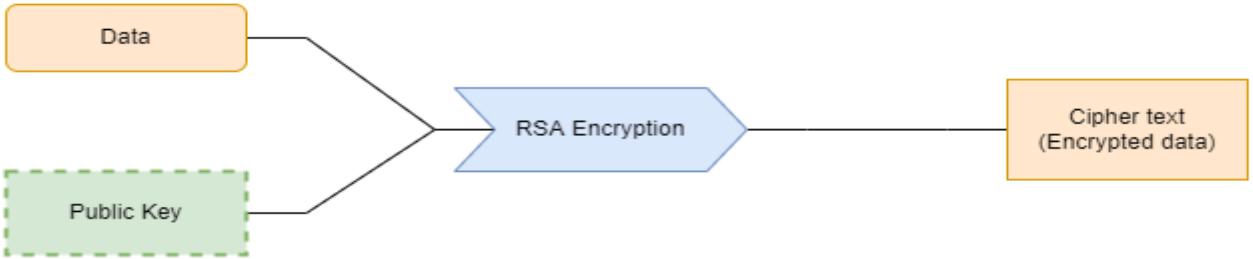
- Func GenerateKeyPair(bits int) (*rsa.PrivateKey, *rsa.PublicKey)

```go
//GenerateKeyPair generates a new key pair
func GenerateKeyPair(bits int) (*rsa.PrivateKey, *rsa.PublicKey) {

    privkey, err := rsa.GenerateKey(rand.Reader, bits)

    if err != nil {
        fmt.Println(err)
    }

    return privkey, &privkey.PublicKey

}
```

2 Crypto package

- ## Concept of RSA Encryption

The public key can be used to encrypt any arbitrary piece of data, but cannot decrypt it.



- Func EncryptOAEP
- Func EncryptPKCS1v15

2 Crypto package

- ## Func EncryptOAEP

func EncryptOAEP ¶

```
func EncryptOAEP(hash hash.Hash, random io.Reader, pub *PublicKey, msg []byte, label []byte) ([]byte, error)
```

EncryptOAEP encrypts the given message with RSA-OAEP.

OAEP is parameterised by a hash function that is used as a random oracle. Encryption and decryption of a given message must use the same hash function and sha256.New() is a reasonable choice.

The random parameter is used as a source of entropy to ensure that encrypting the same message twice doesn't result in the same ciphertext.

The label parameter may contain arbitrary data that will not be encrypted, but which gives important context to the message. For example, if a given public key is used to encrypt two types of messages then distinct label values could be used to ensure that a ciphertext for one purpose cannot be used for another by an attacker. If not required it can be empty.

The message must be no longer than the length of the public modulus minus twice the hash length, minus a further 2.

Source: https://pkg.go.dev/crypto/rsa

2 Crypto package

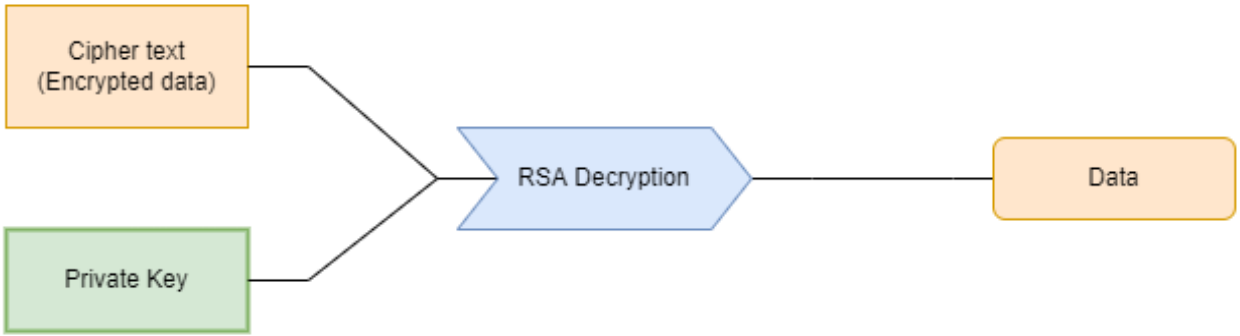
- Example of EncryptOAEP

```go
//EncryptWithPublicKey encrypts data with public key
func EncryptWithPublicKey(msg []byte, pub *rsa.PublicKey) []byte {
    hash := sha256.New()
    ciphertext, err := rsa.EncryptOAEP(hash, rand.Reader, pub, msg, nil)
    if err != nil {
        fmt.Print(err)
    }

    return ciphertext
}
```

2 Crypto package

- **RSA Decryption:**

The private key can be used to decrypt any piece of data that was encrypted by it's corresponding public key.



- DecryptOAEP
- DecryptPKCS1v15
- DecryptPCKS1v15SessionKey

2 Crypto package

- ## func DecryptOAEP

func DecryptOAEP

```
func DecryptOAEP(hash hash.Hash, random io.Reader, priv *PrivateKey, ciphertext []byte, label []byte) ([]byte, error)
```

DecryptOAEP decrypts ciphertext using RSA-OAEP.

OAEP is parameterised by a hash function that is used as a random oracle. Encryption and decryption of a given message must use the same hash function and sha256.New() is a reasonable choice.

The random parameter, if not nil, is used to blind the private-key operation and avoid timing side-channel attacks. Blinding is purely internal to this function – the random data need not match that used when encrypting.

The label parameter must match the value given when encrypting. See EncryptOAEP for details.

Source: https://pkg.go.dev/crypto/rsa

2 Crypto package

- Example of DecryptOAEP

```go
// DecryptWithPrivateKey decrypts data with private key
func DecryptWithPrivateKey(ciphertext []byte, priv *rsa.PrivateKey) []byte {
    hash := sha256.New()
    plaintext, err := rsa.DecryptOAEP(hash, rand.Reader, priv, ciphertext, nil)
    if err != nil {
        fmt.Print(err)
    }
    return plaintext
}
```

3 실습

- Crypto Package을 인용해 Generate Key 생성 함수, RSA Encrypt 생성 함수, Decrypt 생성 함수를 만들어 main으로 실행 해서 return 하기.

- 나만의 Password storage 만들어 보기.

2 Crypto package

- Go파일: https://github.com/iujiban/Go-lang
- SQL EXAMPLE: https://github.com/iujiban/ASP.NET/tree/main/FinalUpload/DB

Thank You