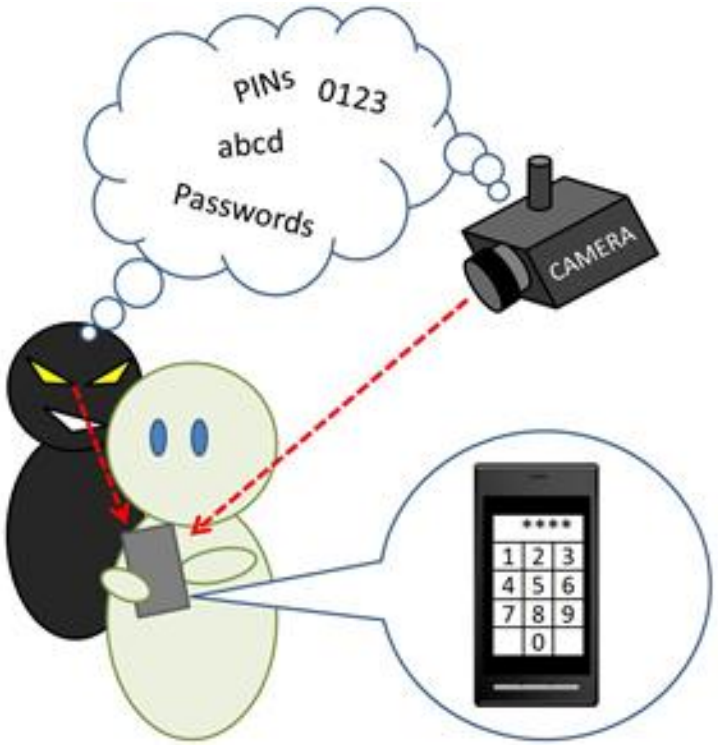
แก้ไขครั้งล่าสุดเมื่อวันที่ 24 เมษายน 2568

Chapter 12
Mobile Security

Shoulder Surfing Attack



Smudge Attack

(a) Input picture

(b) Reference device

(c) Extracted screen

(d) Edge detection

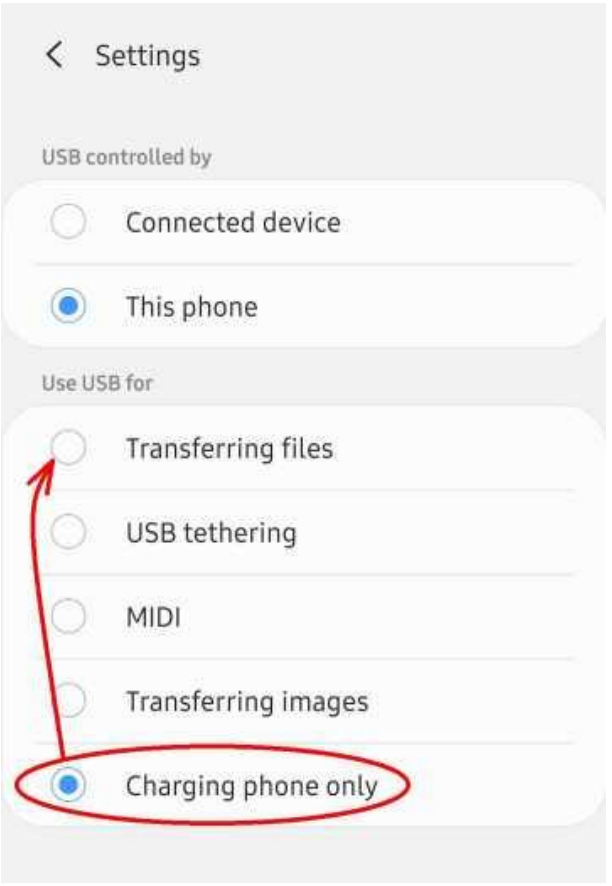
(e) Probabilistic transform

(f) Template grid

(g) Adding the grid

(h) Pattern detection

Juice Jacking



XcodeGhost

XcodeGhost (and variant XcodeGhost S) are modified versions of Apple's Xcode development environment that are considered malware.[1] The software first gained widespread attention in September 2015, when a number of apps originating from China harbored the malicious code.[2] It was thought to be the "first large-scale attack on Apple's App Store",[3] according to the BBC. The problems were first identified by researchers at Alibaba, a leading e-commerce firm in China.[3] Over 4000 apps are infected, according to FireEye, far more than the 25 initially acknowledged by Apple,[4] including apps from authors outside China.

Security firm Palo Alto Networks surmised that because network speeds were slower in China, developers in the country looked for local copies of the Apple Xcode development environment, and encountered altered versions that had been posted on domestic web sites. This opened the door for the malware to be inserted into high profile apps used on iOS devices.[5][6]

Even two months after the initial reports, security firm FireEye reported that hundreds of enterprises were still using infected apps and that XcodeGhost remained "a persistent security risk".[7][8] The firm also identified a new variant of the malware and dubbed it XcodeGhost S; among the apps that were infected were the popular messaging app WeChat and a Netease app Music 163.[9]

https://en.wikipedia.org/wiki/XcodeGhost

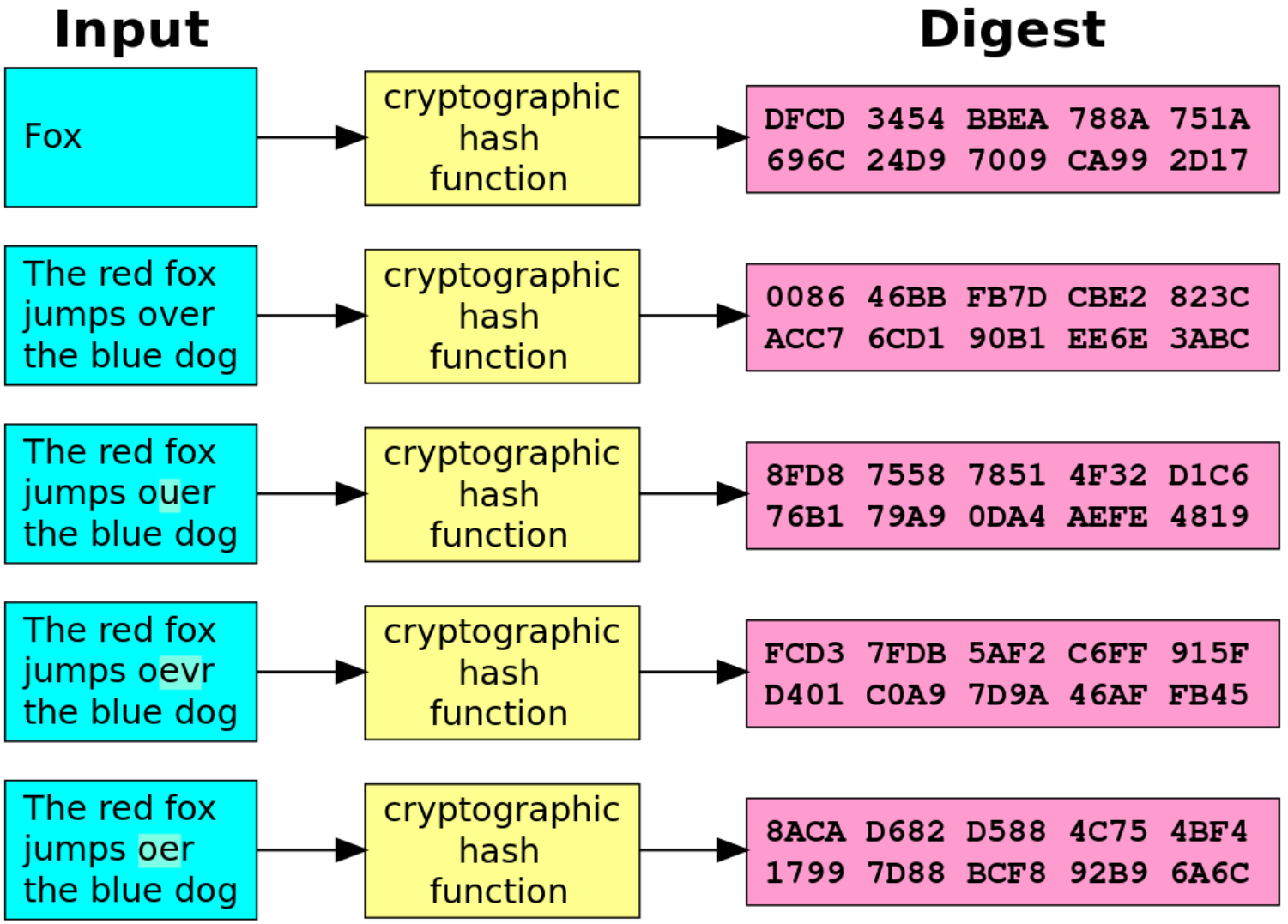
IMSI Catcher

- Mobile station international subscriber number (MSISDN) เบอร์โทร เช่น (66) 081-123-4567
- International mobile subscriber identity (IMSI) เบอร์ค่าย เช่น (520) 03-1234567890

https://www.thaipbs.or.th/news/content/328296

Malware	ซอฟต์แวร์ที่ตั้งใจออกแบบมาให้ทำอันตราย
Trojan horse	ไม่ได้ทำงานตามที่โฆษณา แต่ทำอันตรายอย่างอื่นด้วย
Spyware	แสดงโฆษณา ขโมยข้อมูล ส่ง spam mail (90% in 2010)
Ransomware	เข้ารหัสข้อมูล เรียกค่าไถ่ (ransom)
Trap door	ผู้พัฒนาระบบ/แอป ทำทางเข้าสำหรับตัวเองไว้ใช้ภายหลัง เช่น แอบเข้ามาปัดเศษสตางค์จากบัญชีอื่น XCodeGhost 2015 เติม machine code ลงไปในทุกแอปที่คอมไพล์ (ควรติดตั้งแอปจากแหล่งที่เชื่อถือได้เท่านั้น)
Logic bomb	ตั้งเวลา เงื่อนไข ให้โจมตี เช่น เมื่อไม่ได้เป็นพนักงานแล้ว
Virus	มีการแพร่พันธุ์ (self-replicating) โดยใช้ human activity เช่น การก็อปปี้ไฟล์ข้ามเครื่อง ไวรัสมีหลายประเภท file, boot, macro (spreadsheet), rootkit (easy root access), source code, polymorphic, encrypted, stealth, multipartite, armored
Worms	ไม่ใช้ human activity แพร่ไปได้เองใน network
Monoculture	ระบบเดียวทั้งโลก เช่น ปี 2004 ระบบปฏิบัติการ Windows Server ติดตั้ง Internet Information Services (IIS) ผู้ท่องเว็บใช้โปรแกรม Internet Explorer ไวรัสติด IIS ก่อน แล้วผู้ท่องเว็บทุกคนที่มาดาวน์โหลดไฟล์จากเว็บนี้จะติดไวรัสไปด้วยทุกครั้ง ไวรัสจะติดตั้ง back door เช่น keystroke logger จำทุกอย่างที่ผู้ใช้พิมพ์ผ่านคีย์บอร์ด

Hash Functions



คุณสมบัติที่สำคัญคือ หา inverse function ยากมาก และโอกาสเกิด collision น้อย

ในทางโปรแกรมมิ่งการเก็บ password ของผู้ใช้ แบบไหนถึงจะปลอดภัย ?

1.

Password

1234

เก็บ password

ตัวอย่างฐานข้อมูล

User	Password
Manee	1234

Hacker ชอบมองเห็นพาสเวิร์ด

2.

Password

1234

เก็บ hash value

Hash function

ตัวอย่างฐานข้อมูล

User	Hash values (password)
Manee	a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9

Hacker อ่านพาสเวิร์ดไม่ออก ก็จริง แต่เดาสุ่มได้ไม่ยาก

3.

Password

1234

+

Salt (random)

bv5PehSMfV11Cd

เก็บ hash value และ salt value

Hash function

ตัวอย่างฐานข้อมูล

User	Hash values (password)	Salt values
Manee	f7b007b41cf9e48d0414]81bb12dbdd88d6d5<aafe8odbd68bja1gGb	bv5PehSMfV11Cd

Hacker อ่านพาสเวิร์ดไม่ออก แถมเดาสุ่มก็ยากด้วย

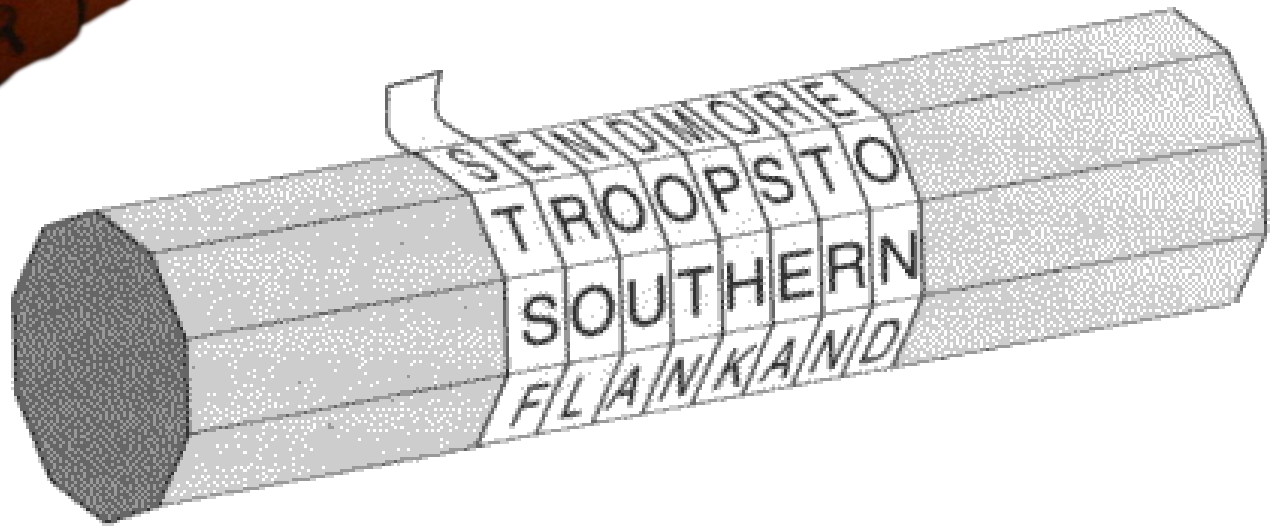
```
using System;
using System.Security.Cryptography;
using Microsoft.AspNetCore.Cryptography.KeyDerivation;
```

```
Console.Write("Enter a password: ");
string password = Console.ReadLine();

// generate a 128-bit salt using a secure PRNG
byte[] salt = new byte[128 / 8];
using (var rng = RandomNumberGenerator.Create())
{
    rng.GetBytes(salt);
}
Console.WriteLine($"Salt: {Convert.ToBase64String(salt)}");

// derive a 256-bit subkey (use HMACSHA1 with 10,000 iterations)
string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
    password: password,
    salt: salt,
    prf: KeyDerivationPrf.HMACSHA1,
    iterationCount: 10000,
    numBytesRequested: 256 / 8));
Console.WriteLine($"Hashed: {hashed}");
```

```
Enter a password: helloworld
Salt: intE98uKsFx7H9Tpk/tTEA==
Hashed: xit0xfNlIfAK2c3T0uogUm/617EoAxUVl4iUSNYpDy0=
```

Cryptography (hidden/secret to write)

Scytale ไม่ต้องใช้ key ก็ได้
ขอแค่ให้รู้วิธีอ่านที่ถูกต้อง



https://en.wikipedia.org/wiki/Cryptography

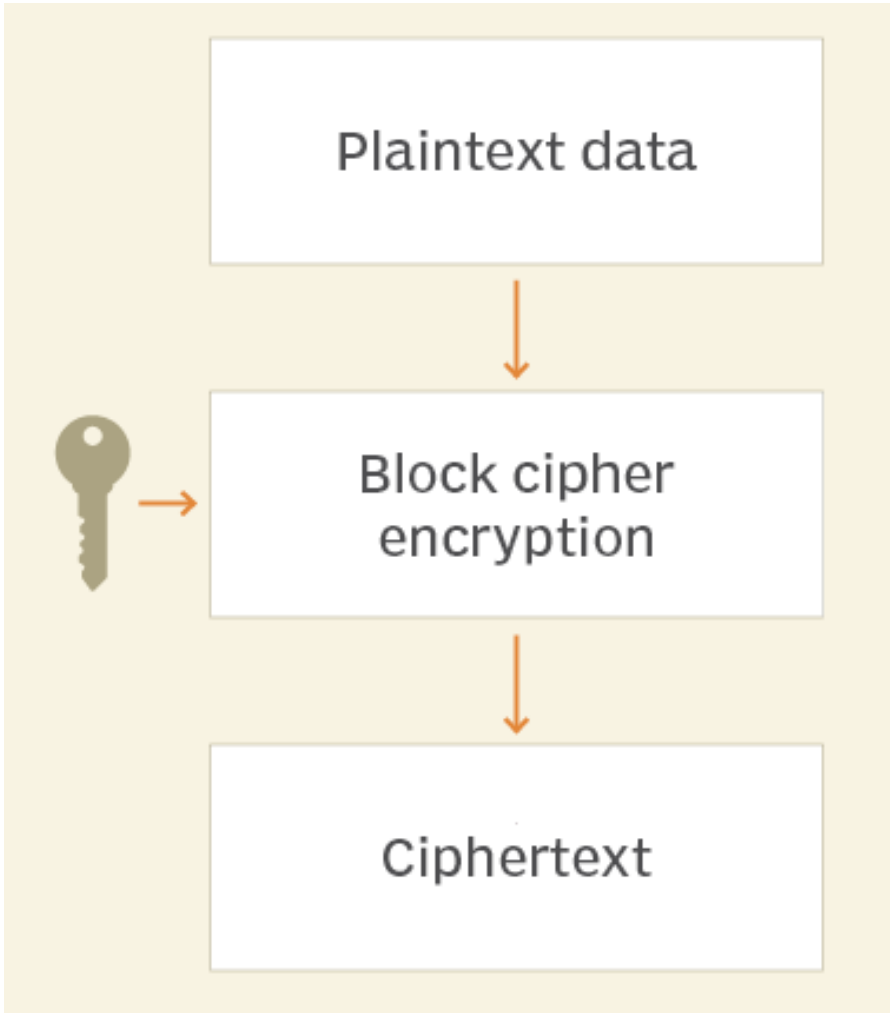
Encryption

An encryption algorithm consists of the following components:

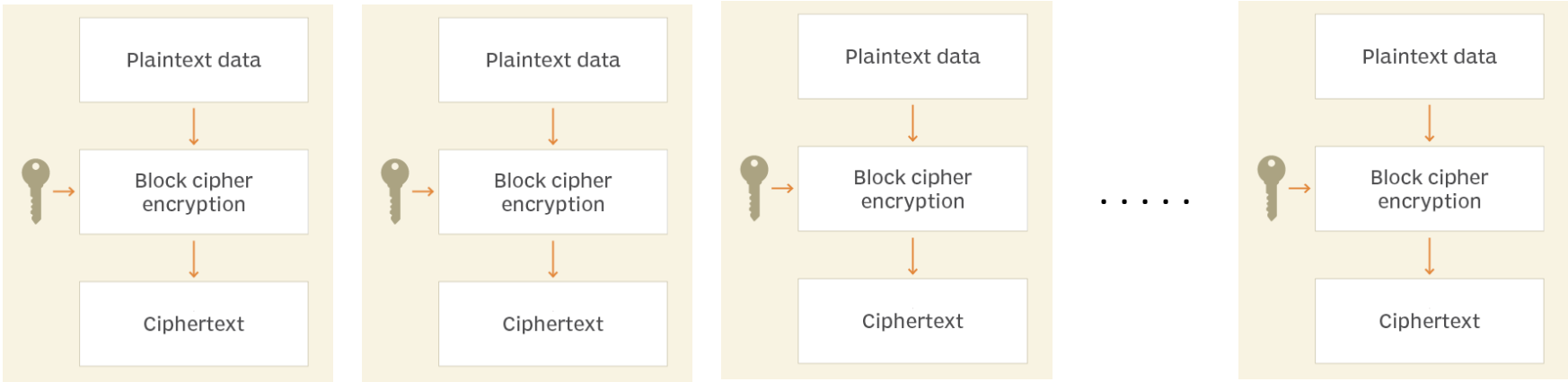
- A set K of keys. ต้องใช้ key
- A set M of messages.
- A set C of ciphertexts.
- An encrypting function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, E_k is a function for generating ciphertexts from messages. Both E and E_k for any k should be efficiently computable functions. Generally, E_k is a randomized mapping from messages to ciphertexts.
- A decrypting function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, D_k is a function for generating messages from ciphertexts. Both D and D_k for any k should be efficiently computable functions.

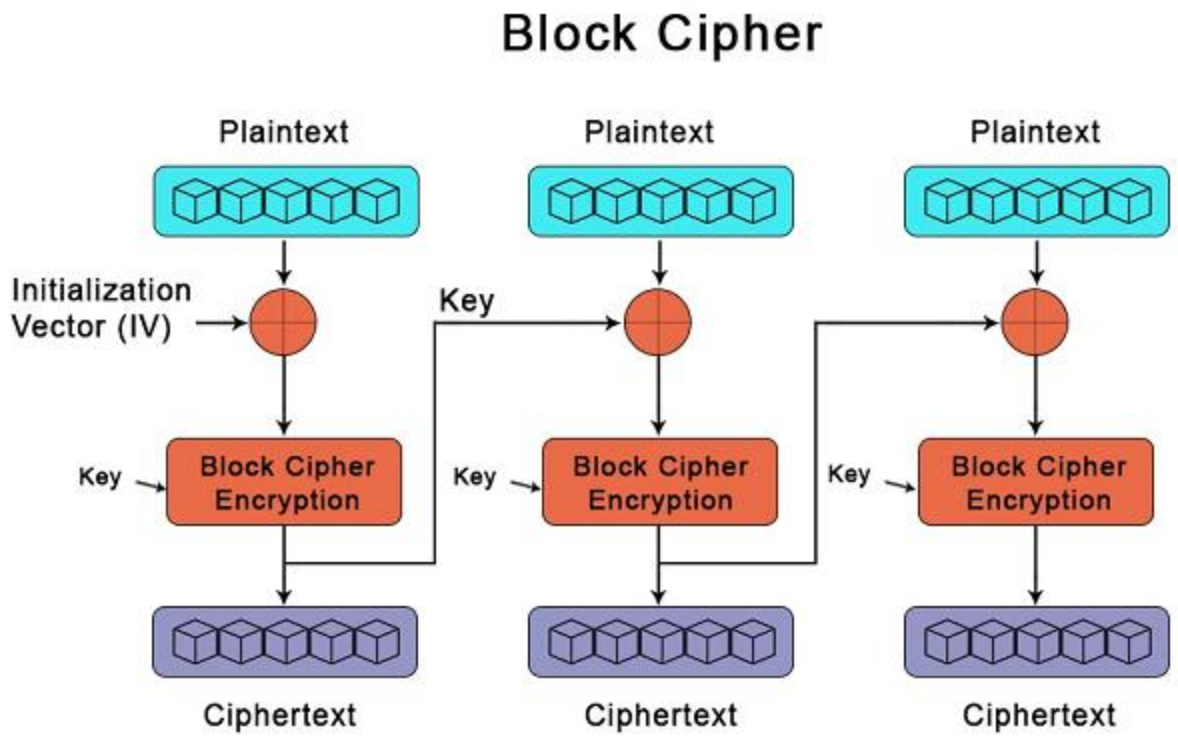
Symmetric Encryption: Block Cipher

- Data Encryption Standard (DES) 64-bit block, 56-bit key
- Advanced Encryption Standard (AES) 128-bit block, 128/192/256-bit key

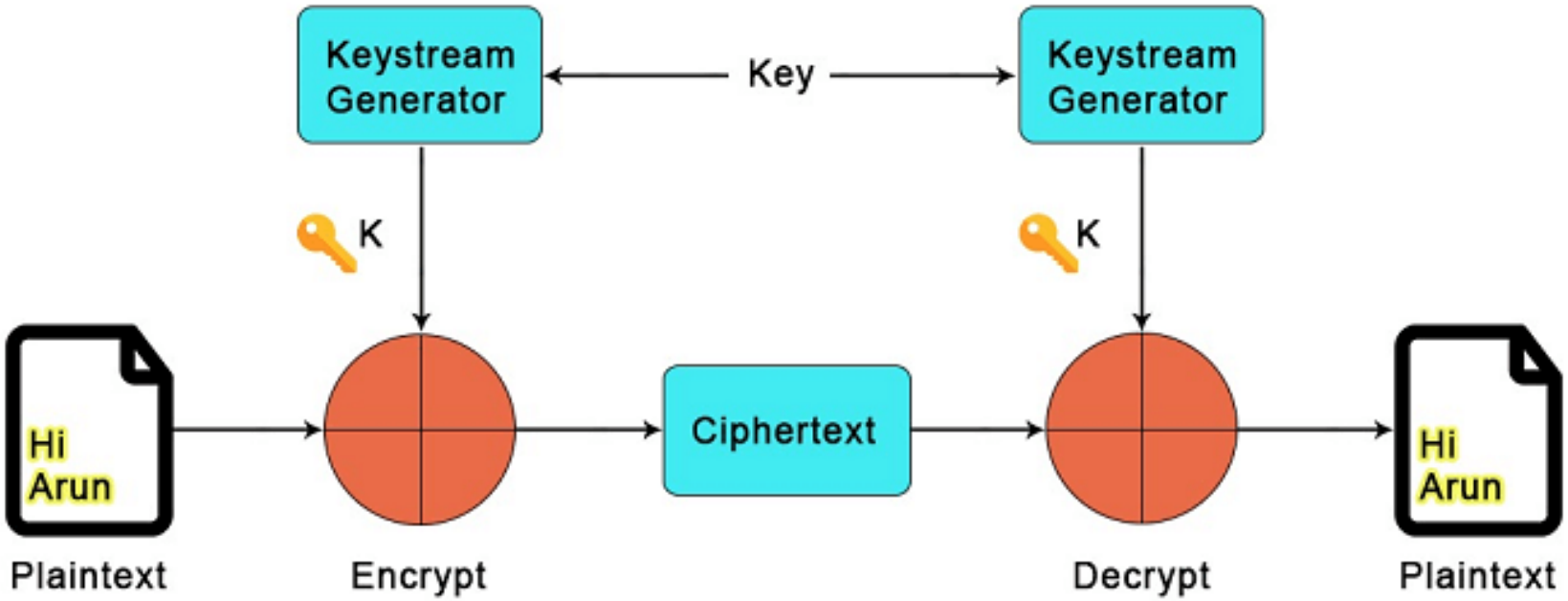


- ใช้ block cipher แบบ parallel ไม่ปลอดภัย





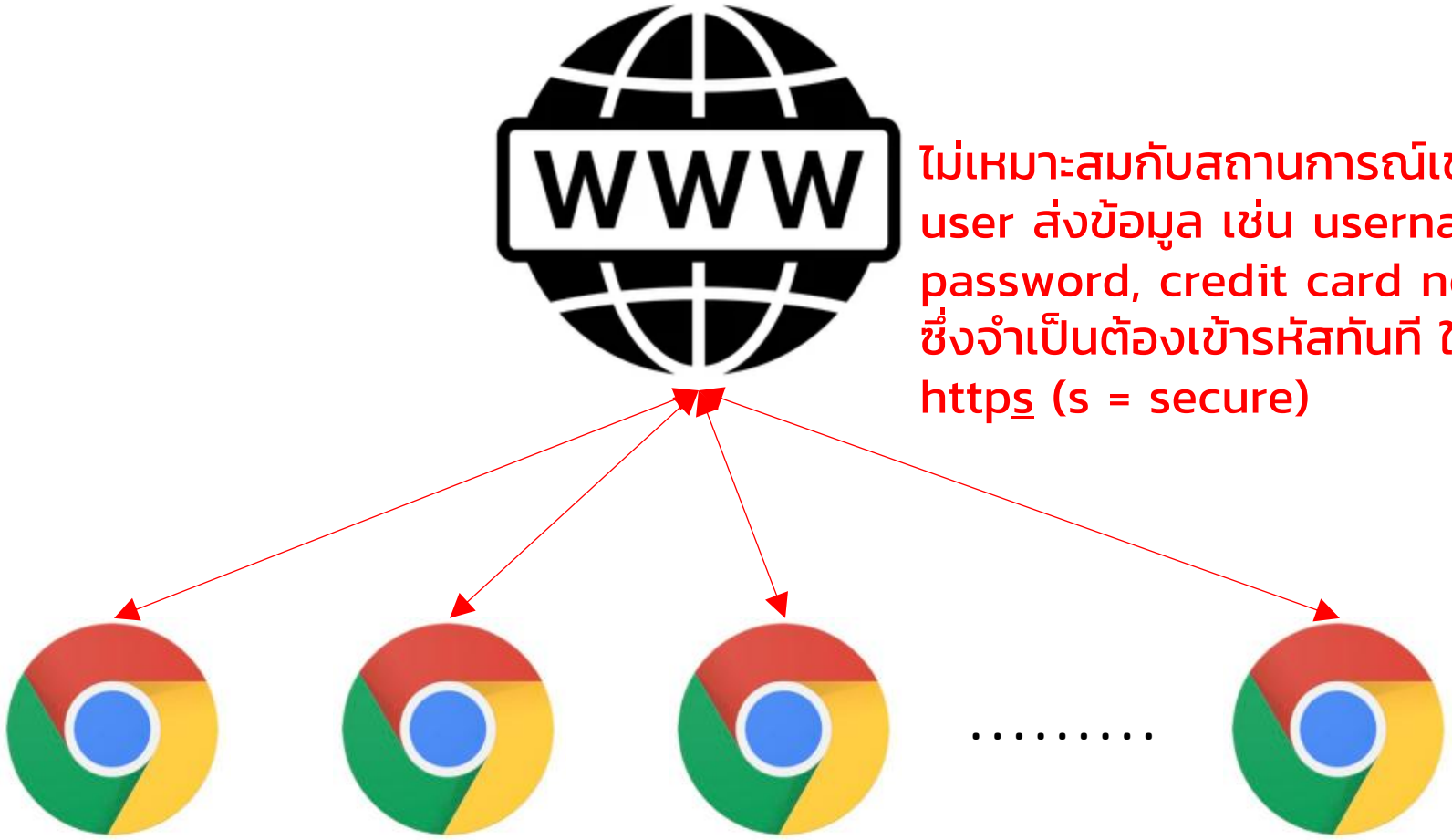
Stream Cipher



Stream ciphers typically execute at a higher speed than block ciphers and have lower hardware complexity. However, stream ciphers can be susceptible to security breaches; for example, when the same starting state (seed) is used twice.

Asymmetric Encryption

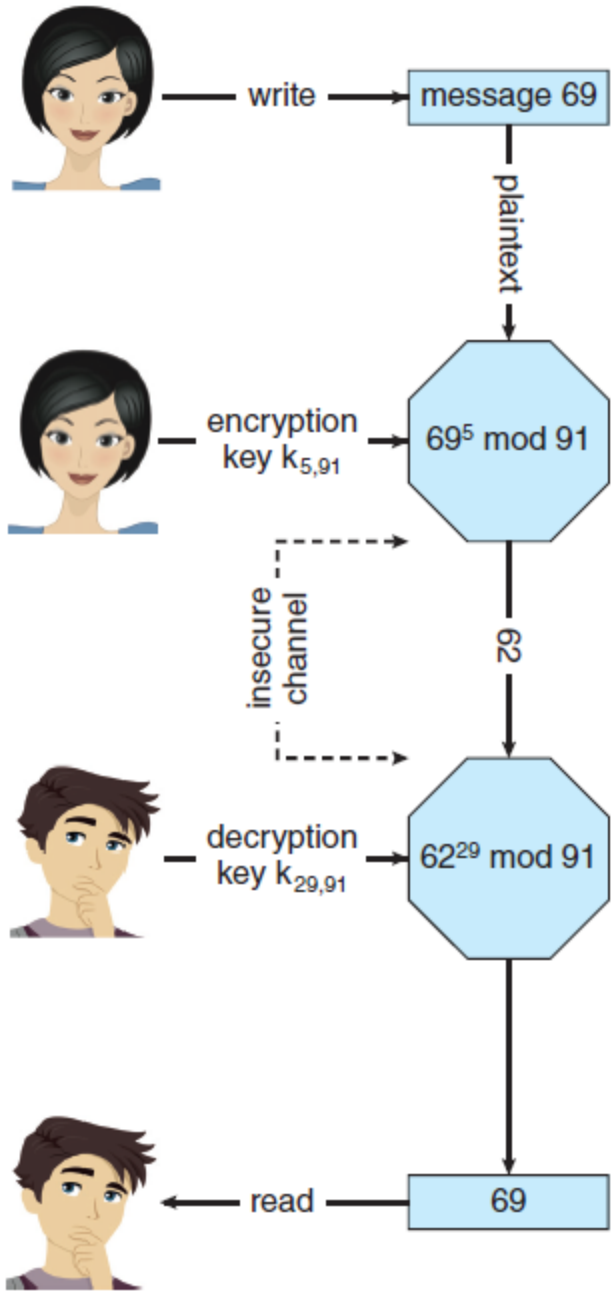
- ข้อเสียของ symmetric encryption คือ sender/receiver ต้องมี secret key ก่อนที่จะเริ่มการสื่อสาร (ต้องไปหาที่ลับ ๆ เพื่อตกลงว่าจะใช้ key อะไร)



RSA – Rivest, Shamir, and Adleman

In RSA, k_e is the **public key**, and k_d is the **private key**. N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 2048 bits each). It must be computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, so that k_e need not be kept secret and can be widely disseminated. The encryption algorithm is $E_{ke,N}(m) = m^{k_e} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$. The decryption algorithm is then $D_{k_d,N}(c) = c^{k_d} \bmod N$.

An example using small values is shown in Figure 16.8. In this example, we make $p = 7$ and $q = 13$. We then calculate $N = 7 * 13 = 91$ and $(p-1)(q-1) = 72$. We next select k_e relatively prime to 72 and < 72, yielding 5. Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29. We now have our keys: the public key, $k_{e,N} = 5, 91$, and the private key, $k_{d,N} = 29, 91$. Encrypting the message 69 with the public key results in the message 62, which is then decoded by the receiver via the private key.



ป้องกันข้อมูลสำคัญ เช่น
- password
- credit card no

public key = 5 (ทุกคนบน www ทราบ)
private key = 29 (เป็นความลับ)

Authentication (proving that a message has not been modified)

เงินกระดาษของราชวงศ์ชิง

ในสมัยโบราณใช้ตราประทับที่ทำจากหยกหรือไม้ไผ่ ผู้อ่านต้องใช้ความชำนาญในการจดจำรอยต่าง ๆ (ต้องเรียนรู้และฝึกฝนก่อน)

Authentication (proving that a message has not been modified)

An authentication algorithm using symmetric keys consists of the following components:

- A set K of keys.
- A set M of messages.
- A set A of authenticators.
- A function $S : K \rightarrow (M \rightarrow A)$. That is, for each $k \in K$, S_k is a function for generating authenticators from messages. Both S and S_k for any k should be efficiently computable functions.

JWT.IO

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

ข้อมูลนี้ไม่ได้เข้ารหัส (encrypted)
แค่เขียนในรูป base-64 string

มี 3 ส่วน คั่นด้วย . (fullstop)

header . payload . signature

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

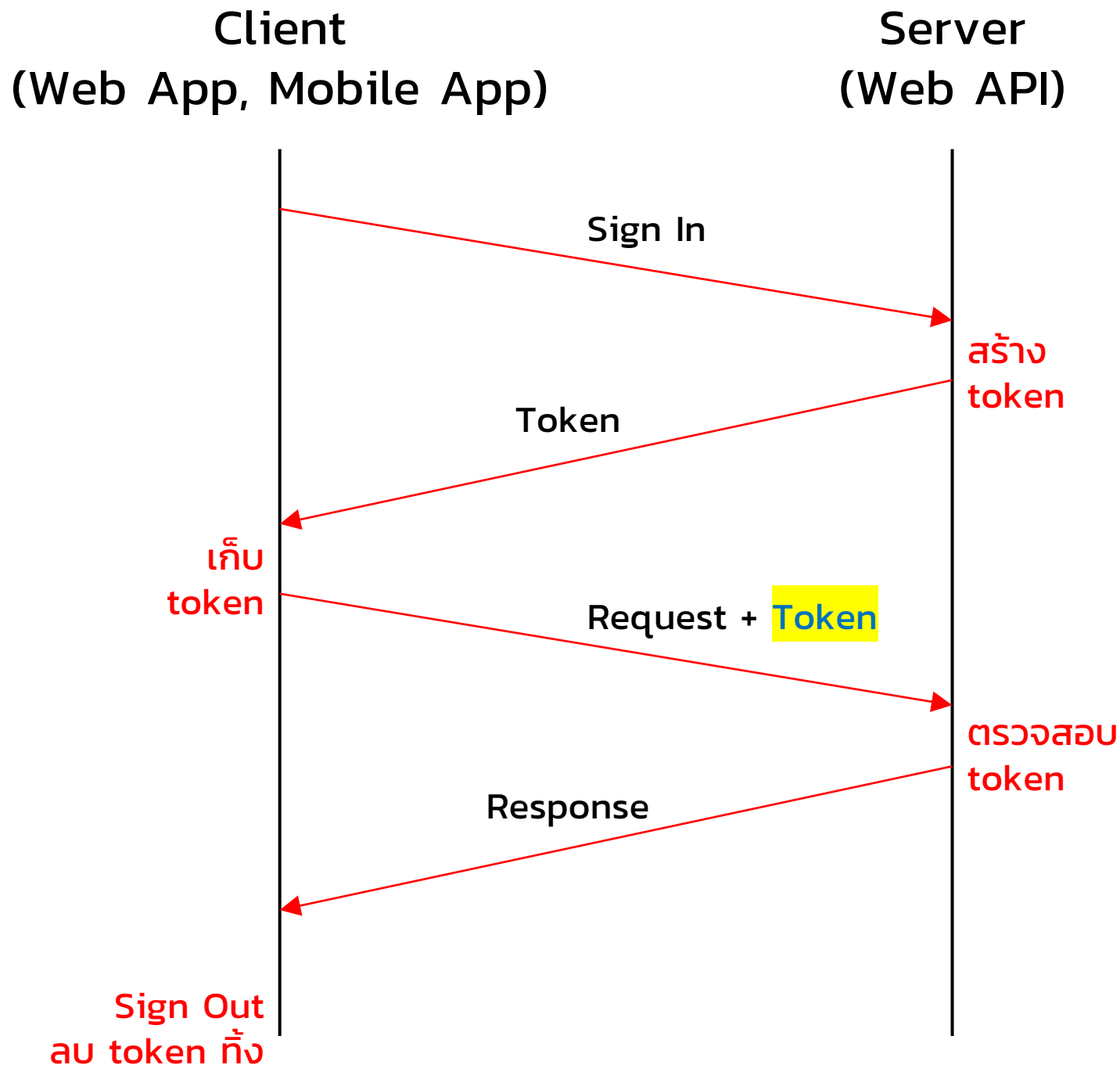
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

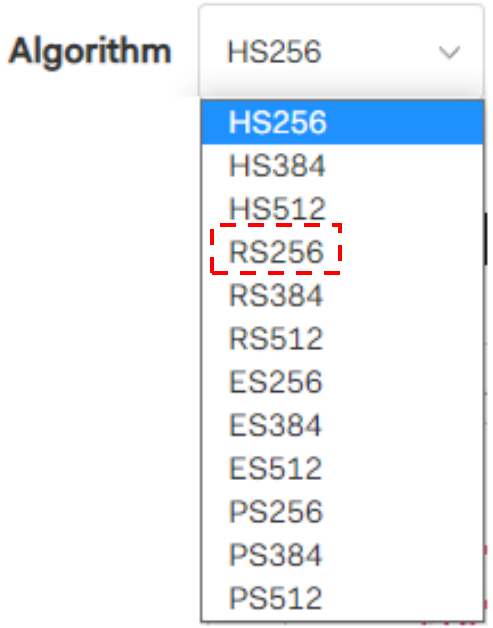
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

เมื่อล็อกอินสำเร็จ จะได้ Jason Web Token (JWT) ข้อมูลใน payload จะบอกว่า user คือใคร และวันหมดอายุของ token เมื่อจะโหลด web app หน้าต่อไป ก็ต้องส่ง token มาด้วยทุกครั้ง เพื่อยืนยันว่าเป็น user คนนี้จริง ๆ เช่น คลิกส่งอีเมล อัปโหลดไฟล์ เป็นต้น


Client
(Web App, Mobile App)
Server
(Web API)
Sign In
สร้าง
token
Token
เก็บ
token
Request + Token
ตรวจสอบ
token
Response
Sign Out
ลบ token ทิ้ง

HS256 vs RS256 (HMAC using SHA256, RSA using SHA256)



Symmetric key

Asymmetric key

private key ใช้ sign

public key ใช้ verify

วาง public key ไว้บนอินเทอร์เน็ต

เก็บ private key ไว้เป็นความลับ

token ที่ sign ด้วย private key นำ token นี้ไปใช้กับ web/mobile app อื่น ๆ ได้

เช่น token ออกโดย google บอกว่าเรามีสิทธิ์ใช้งาน service อะไรได้บ้าง

หรือจะใช้พิสูจน์ว่าเราเป็นบุคคลนี้โดยอนุโลม (เพราะอาจเป็น token ที่ขโมยมา)

24 เม.ย. 2568 (โมบายแอป - บัญญวัฒ)

https://chula.zoom.us/rec/share/c7TftGheRKbOD-qEoiM2rOwlJdVPsnrBfZBYrdAziFDInkaqT93_lv7v8I_D7vaf.p8aJNEVLVAs4WiGO?startTime=1745432170000
Passcode: &#q0I37D\

เอกสารประกอบ
https://docs.google.com/document/d/14QySI_BCYKDDI8kfeLGopZfQKQy4nttfwgVndbz75ts/edit?usp=sharing