Crypto I

Kuruwa

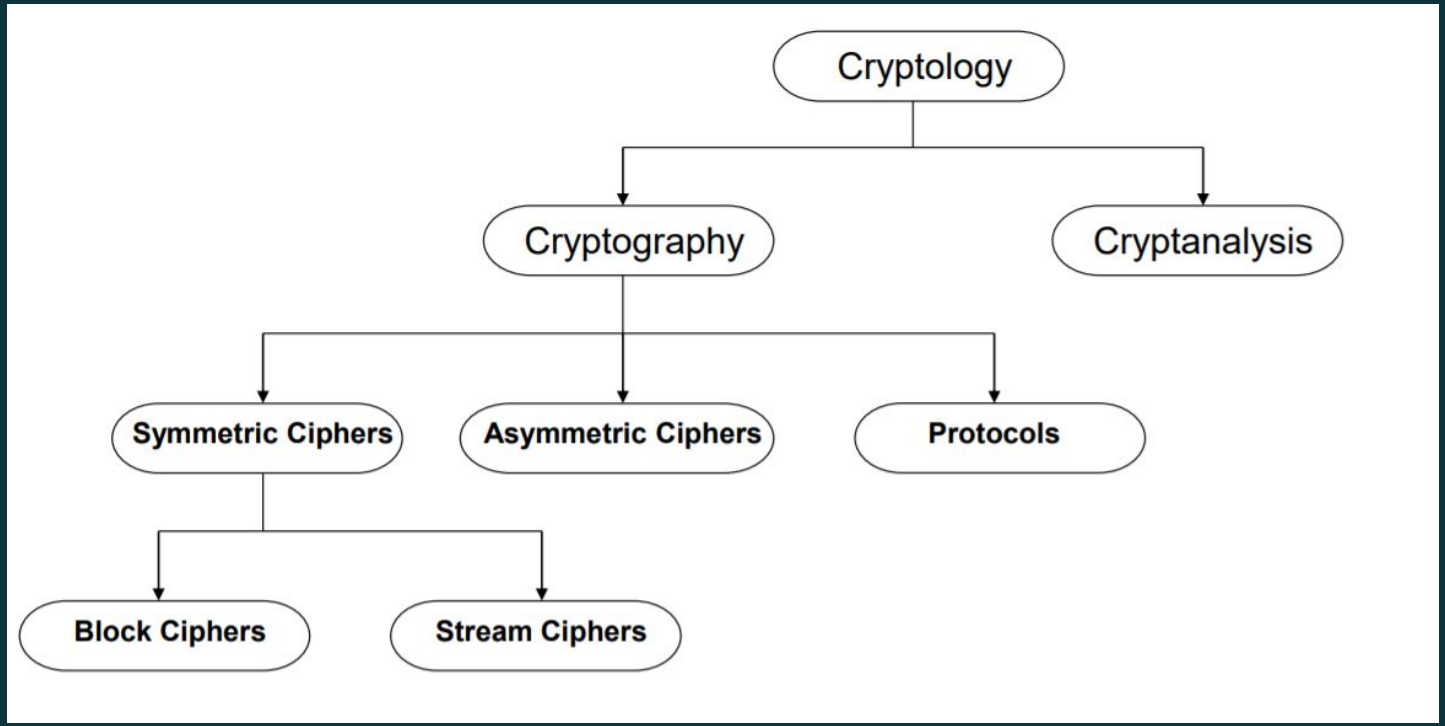
Useful Tools

- pyCryptodome
  - pip install pycryptodome
- Sage
  - apt-get install sage
  - https://cocalc.com

ToC

Introduction

Overview


Cryptology
Cryptography
Cryptanalysis
Symmetric Ciphers
Asymmetric Ciphers
Protocols
Block Ciphers
Stream Ciphers

Cryptanalysis

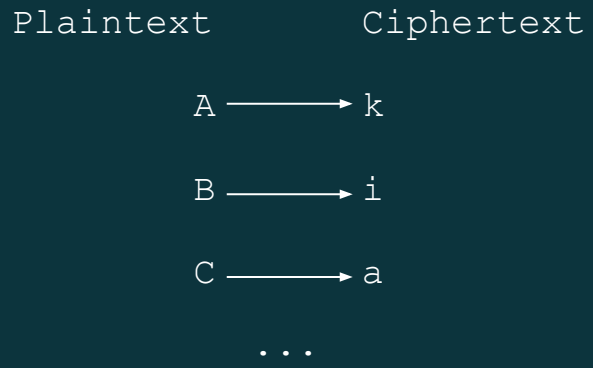
- Kerckhoff's Principle
  - "A cryptosystem should be secure even if the attacker knows all details about the system, with the exception of the secret key"

- Classical Cryptanalysis
  - Mathmatical Analysis
  - Brute-Force Attacks
- Implementation Attacks
- Social Engineering

Brute-Force Attack

Key length in bit	Key space	Security life time (assuming brute-force as best possible attack)
64	2^{64}	**Short term** (few days or less)
128	2^{128}	**Long-term** (several decades in the absence of quantum computers)
256	2^{256}	**Long-term** (also resistant against quantum computers – note that QC do not exist at the moment)

Substitution Cipher

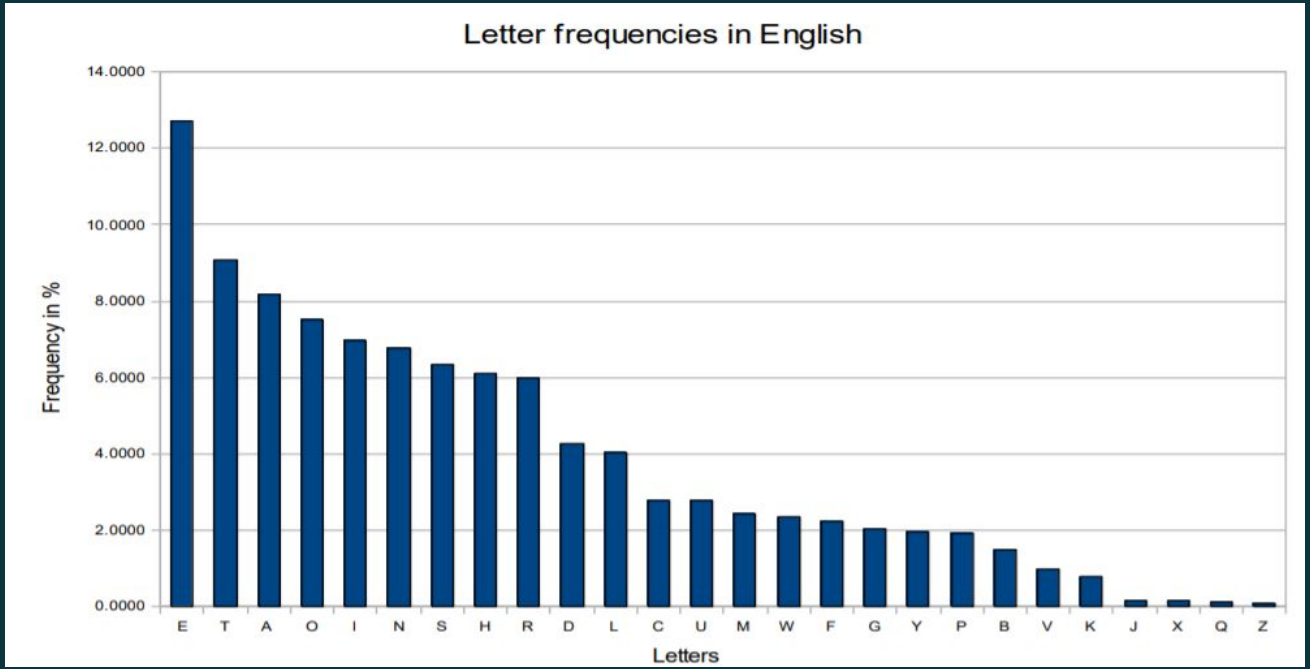


iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc hwwhbsqvqbre hwq vhlq

Attack 1: Exhaustive Key Search

- How many substitution tables (= keys) are there?
  - 26 × 25 × 24 × … × 1 ≈ 2^88

Attack 2: Letter Frequency Analysis



Attack 2: Letter Frequency Analysis

iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc hwwhbsqvqbre hwq vhlq

Attack 2: Letter Frequency Analysis

iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc hwwhbsqvqbre hwq vhlq

iE ifcc vEEr fb rdE vfllcE na rdE cfjwhwz hr bnnb hcc hwwhbsEvEbre hwE vhlE

Attack 2: Letter Frequency Analysis

iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc
hwwhbsqvqbre hwq vhlq

iE ifcc vEEr fb rdE vfllcE na rdE cfjwhwz hr bnnb hcc
hwwhbsEvEbre hwE vhlE

iE ifcc vEET fb THE vfllcE na THE cfjwhwz hT bnnb hcc
hwwhbsEvEbTe hwE vhlE

Attack 2: Letter Frequency Analysis

iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc
hwwhbsqvqbre hwq vhlq

iE ifcc vEEr fb rdE vfllcE na rdE cfjwhwz hr bnnb hcc
hwwhbsEvEbre hwE vhlE

iE ifcc vEET fb THE vfllcE na THE cfjwhwz hT bnnb hcc
hwwhbsEvEbTe hwE vhlE

WE WILL MEET IN THE MIDDLE OF THE LIBRARY AT NOON ALL
ARRANGEMENTS ARE MADE

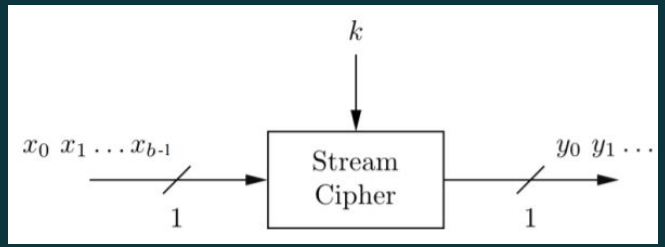
Lab 1:
N-gram attack

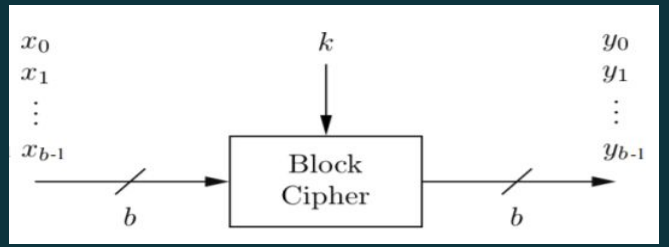
Stream Cipher

- Intro
- Random Number Generators
  - LCG
  - MT19937
  - Linear Feadback Shift Register

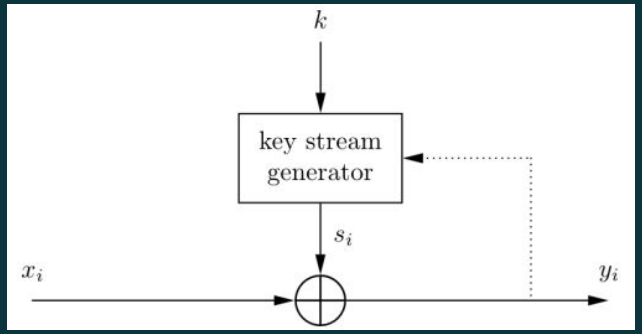
Stream Cipher v.s. Block Cipher

- Stream Ciphers
  - Encrypt bits individually
  - Usually small and fast
- Block Ciphers
  - Always encrypt a full block (several bits)
  - Common for Internet applications





Stream Cipher



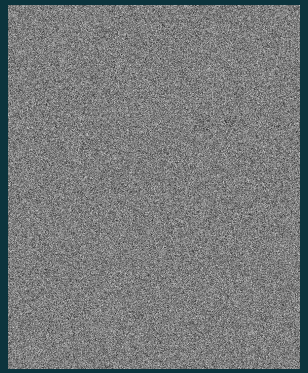
- Encryption
  - $y_i = e_{s_i}(x_i) = x_i \oplus s_i$
- Decryption
  - $x_i = d_{s_i}(y_i) = y_i \oplus s_i$
- Security depends entirely on key stream
  - random
  - reproducible
- Synchronous v.s. Asynchronous

Reused Key Attack

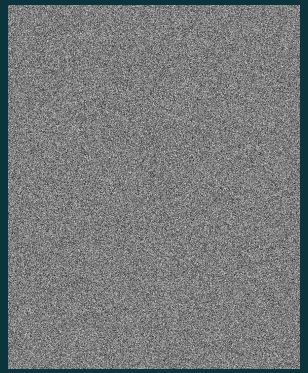
- Stream ciphers are vulnerable if the same key is used twice or more.
  - E(A) = A xor C
  - E(B) = B xor C
  - E(A) xor E(B) = A xor B
- Even if neither message is known, as long as both messages are in a natural language, such a cipher can often be broken by paper-and-pencil methods.

Reused Key Attack

⊕

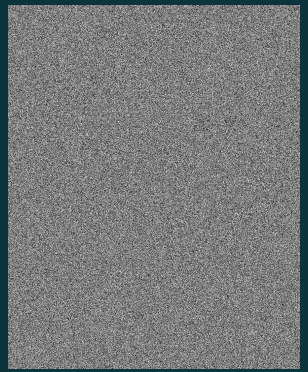


=

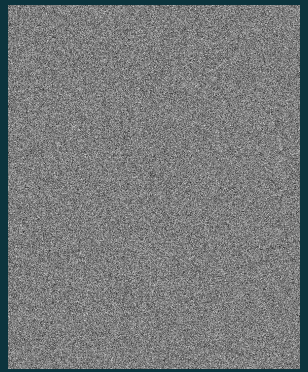


FLAG{Do_not_teLL_Golem_i_use_his_photo}

⊕



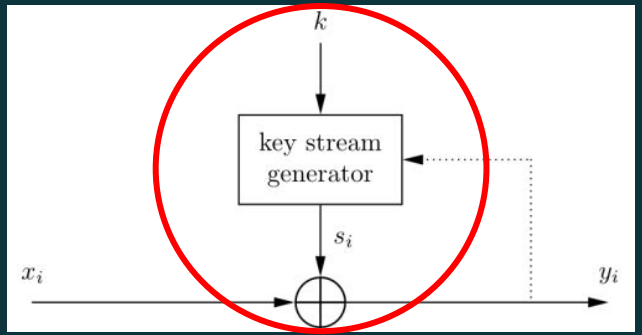
=



Reused Key Attack

Pseudorandom Number Generator

- The key stream generator works like a RNG
- Generate sequences from initial seed (key) value
- Computed recursively
  - s_0 = seed
  - $s_{i+1} = f(s_i, s_{i-1}, \ldots, s_{i-t})$



Linear Congruential Generator

$$S_0 = seed$$
$$S_{i+1} = AS_i + B \mod m$$

Assume

- unknown A,B and S_0 as key
- m=2^32
- 96 bits of output are known, i.e. S_1, S_2, S_3

Solving

- $S_2 = AS_1 + B \pmod m$
- $S_3 = AS_2 + B \pmod m$

MT19937

- Python’s default RNG
- A very long period of 2^19937-1
- Can be recovered by 32×624 consecutive bits

```
class MT19937:
    def __init__(self, seed):
        self.mt = [0] * 624
        self.mt[0] = seed
        self.mti = 0
        for i in range(1, 624):
            self.mt[i] = _int32(1812433253 * (self.mt[i - 1] ^ self.mt[i - 1] >> 30) + i)

...
```

MT19937

output is function of mt[mti]

```
class MT19937:
...
    def extract_number(self):
        if self.mti == 0:
            self.twist()
        y = self.mt[self.mti]
        y = y ^ y >> 11
        y = y ^ y << 7 & 2636928640
        y = y ^ y << 15 & 4022730752
        y = y ^ y >> 18
        self.mti = (self.mti + 1) % 624
        return _int32(y)
...
```

invertible!

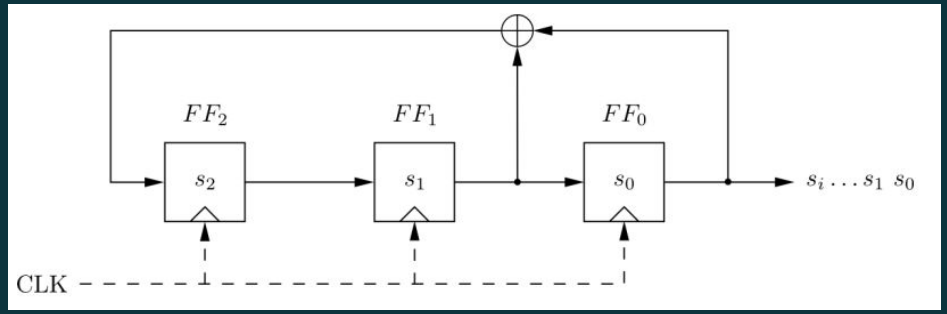
MT19937

mt[i+624] = f(mt[i], mt[i+1], mt[i+397])

```
class MT19937:
...
    def twist(self):
        for i in range(0, 624):
            y = _int32((self.mt[i] & 0x80000000) + (self.mt[(i + 1) % 624] & 0x7fffffff))
            self.mt[i] = (y >> 1) ^ self.mt[(i + 397) % 624]

            if y % 2 != 0:
                self.mt[i] = self.mt[i] ^ 0x9908b0df
```

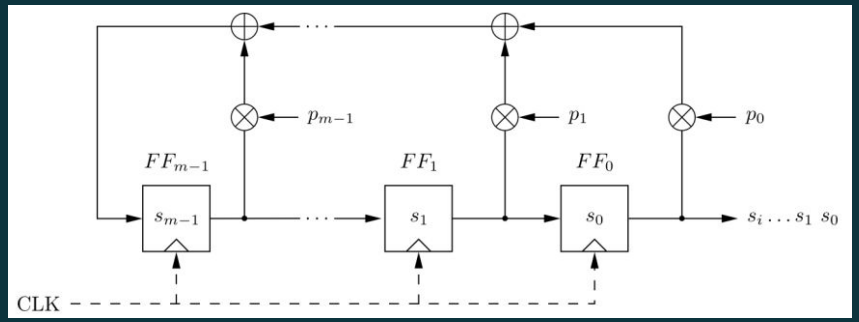
Linear Feedback Shift Register



- $S_{i+3} = S_{i+1} \oplus S_i$
- Maximum output length acchieved

clk	FF_2	FF_1	$FF_0=s_i$
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0

Linear Feedback Shift Register



- Characteristic Polynoimal
  - $P(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + p_0$
- LFSR has maximum output period, iff P(x) is primitive polynomial
  - State of LFSR <-> (x, x^2, x^3, …) mod P(x)
  - Output is the coefficient of the term of degree m-1

Companion Matrix

$$C(P) = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ p_0 & p_1 & p_2 & \dots & p_{m-1} \end{bmatrix} \qquad Cs = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ p_0 & p_1 & p_2 & \dots & p_{m-1} \end{bmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{m-1} \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_m \end{pmatrix}$$

$$s_m = p_0 s_0 + p_1 s_1 + \dots + p_{m-1} s_{m-1}$$

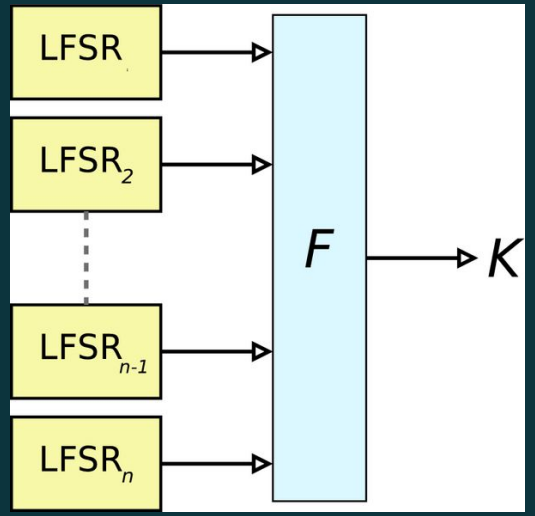
Companion Matrix

$$C(P) = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ p_0 & p_1 & p_2 & \dots & p_{m-1} \end{bmatrix} \qquad C^i s = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ p_0 & p_1 & p_2 & \dots & p_{m-1} \end{bmatrix}^i \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{m-1} \end{pmatrix} = \begin{pmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ \vdots \\ s_{i+m-1} \end{pmatrix}$$

One can recover seed by gathering m-bit outputs

Improving Security of LFSR

- Non-linear combining functions
  - boolean function of outputs of multiple LFSR
- Filter Generator
  - boolean function of states of one LFSR



Correlation Attack

- 3×32 bits = 96 bits safety?

```
class myLFSR:
    def getbit(self):
        x1 = LFSR1.getbit()
        x2 = LFSR2.getbit()
        x3 = LFSR3.getbit()
        return x2 if x1 else x3
```

Correlation Attack

- output = x_2 if x_1 else x_3
  - 75% of output = x_2
  - 75% of output = x_3
- 2^96 -> 3×2^32

x_1	x_2	x_3	output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

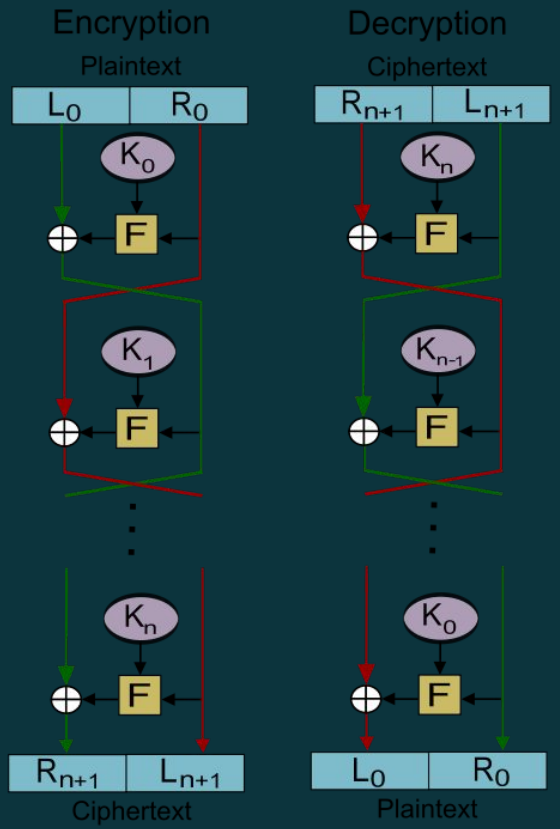
Lab 2:
Correlation Attacks

Block Cipher

- Design
  - Feistel
  - SPN
- Block of Operation
  - ECB
  - CBC
  - CTR
  - GCM

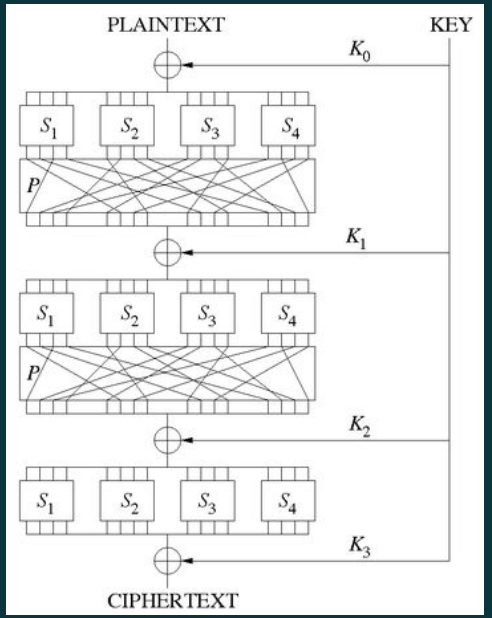
Feistel Cipher

- Round function can be arbitrarily complicated
  - even not invertible
- Encryption and decryption are similar
  - halve the code size
- Example
  - DES
  - Blowfish



Substitution-permutation Network

- Confusion
  - each bit of the ciphertext depend on several parts of the key
- Diffusion
  - change a single bit of the plaintext, then about half of the bits in the ciphertext should change
- Example
  - AES

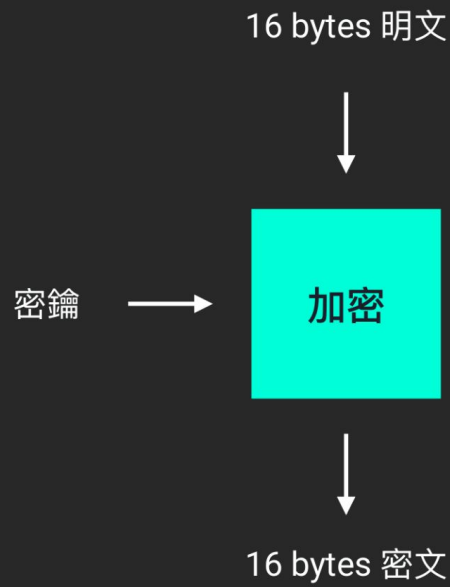


Block Cipher Mode

by oalieno

https://github.com/oalieno/Crypto-Course

Block Cipher



- **輸入輸出長度固定**
- 以 AES 為例
- 輸入明文 16 bytes
- 輸出密文 16 bytes
- 密鑰長度有 16, 24, 32 bytes 三種

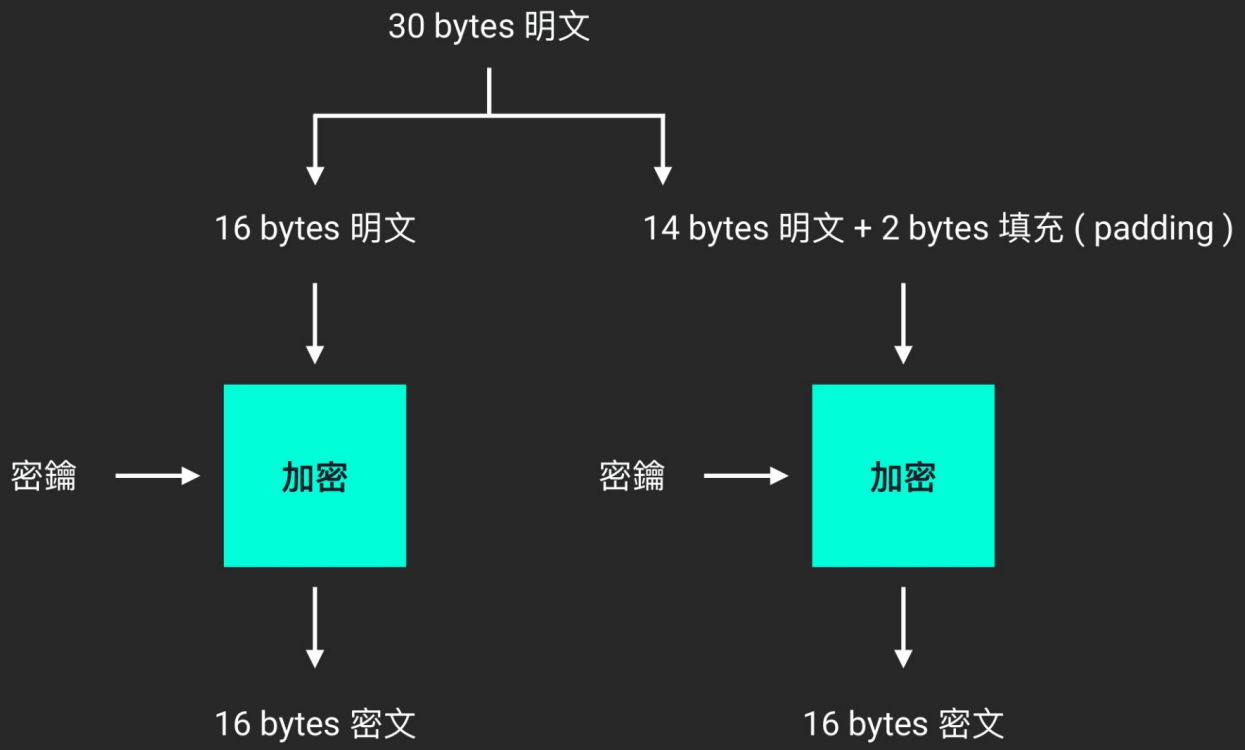
最常見

Block Cipher

我要加密的明文不是 16 bytes 怎麼辦？

切成很多個 16 bytes

Block Cipher



Block Cipher Mode

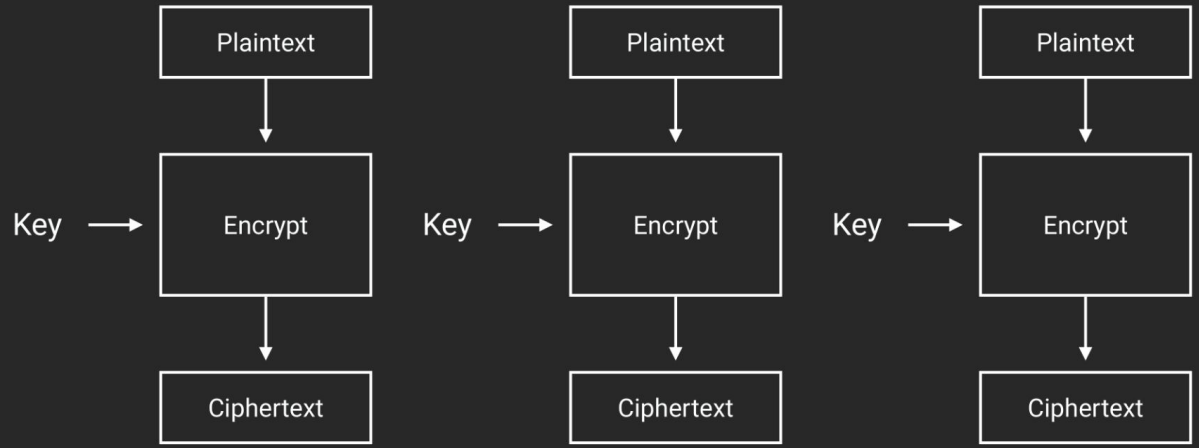
- AES 是 Block Cipher，輸入輸出長度固定
- 所以要加密**任意長度**的明文，需要一些額外加工
- 一些常見的加工模式：ECB, CBC, CFB, OFB, CTR...
- 有 AEAD 的模式：CCM, GCM, OCB...

Authenticated Encryption with Associated Data

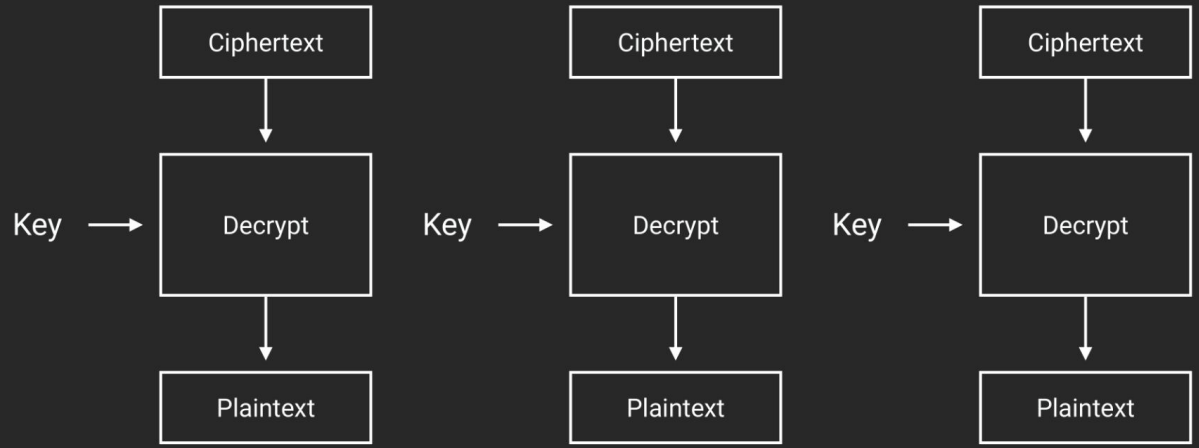
- Authenticated Encryption (AE)
  - 會多計算一個 MAC (Message Authentication Code)
- Associated Data (AD)
  - 可以連同一些沒加密的資料一起做 Authentication
  - 比如一些不需要加密的 Header Information

ECB Mode

ECB Mode Encryption



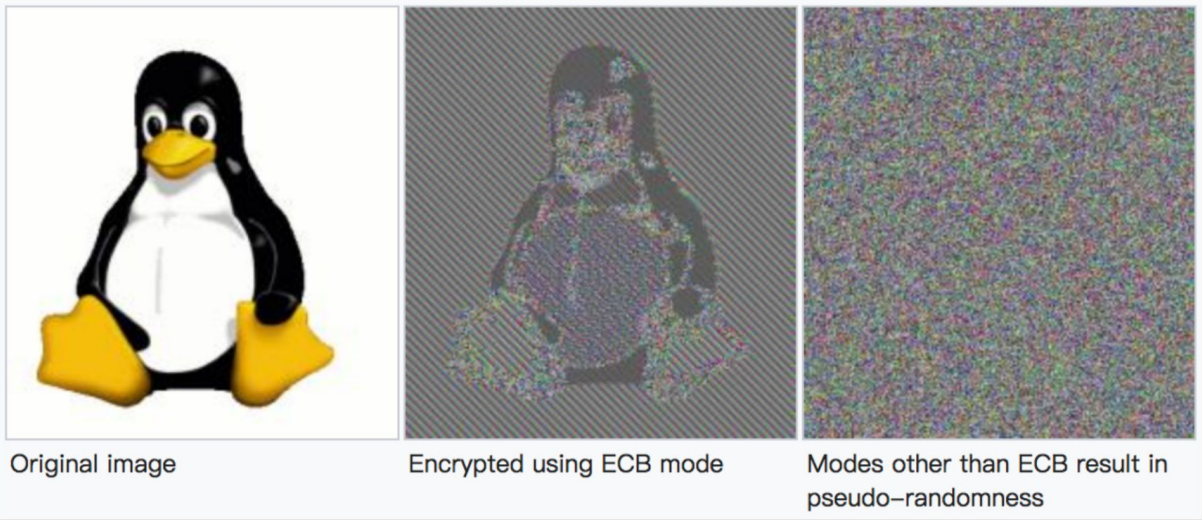
ECB Mode Decryption



ECB Mode

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

ECB 模式的缺點：相同的明文區塊會加密出相同的密文區塊



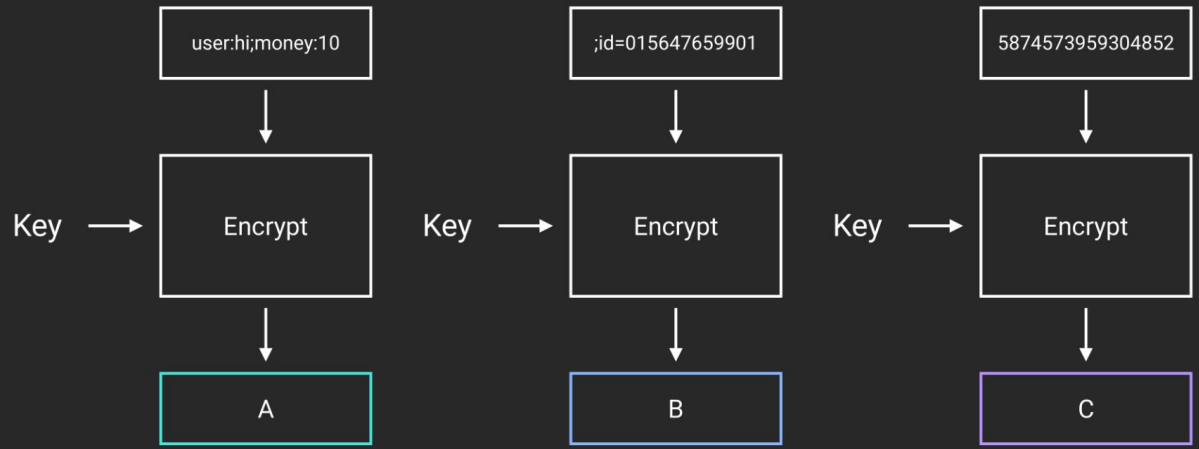
Original image

Encrypted using ECB mode

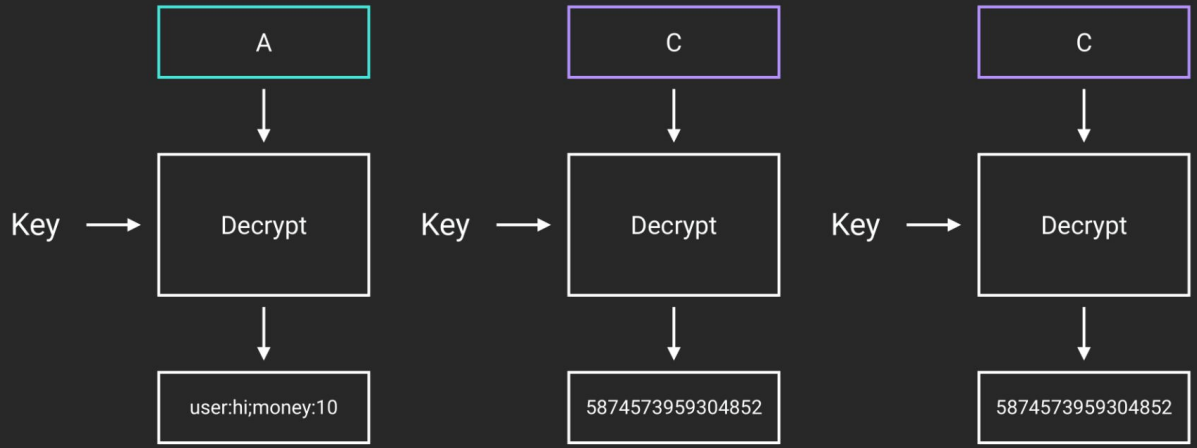
Modes other than ECB result in pseudo–randomness

ECB Mode / Cut & Paste

ECB Mode / Cut & Paste



ECB Mode / Cut & Paste



錢錢變多了xD

ECB Mode / Prepend Oracle Attack

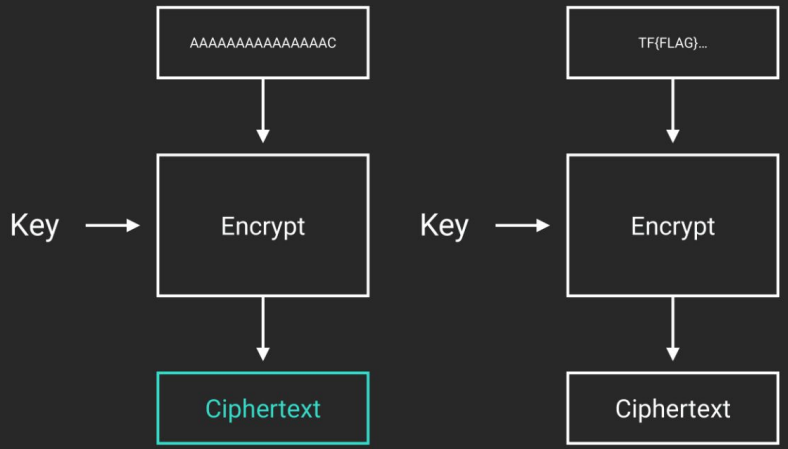
The Oracle

- 使用 AES ECB Mode
- 伺服器將我們的明文 prepend 在 flag 前面作加密

```
def oracle(plain):
    aes = AES.new(KEY, AES.MODE_ECB)
    return aes.encrypt(plain + flag)
```

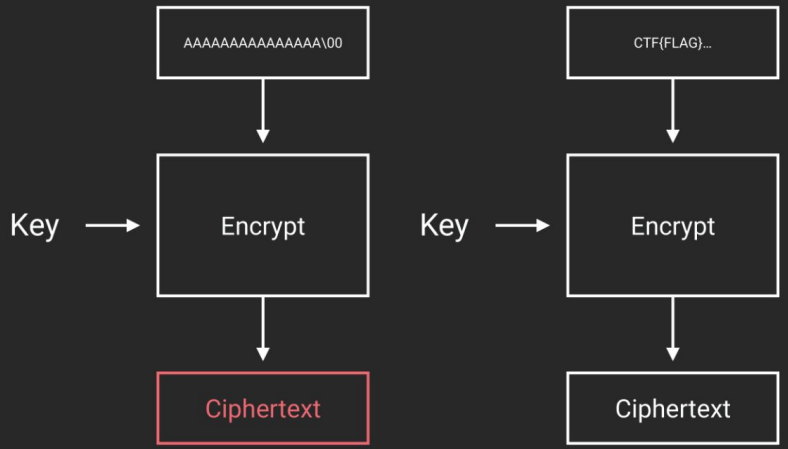
找 flag[0]

先塞 15 個垃圾
讓 flag 的第一個字元掉進來



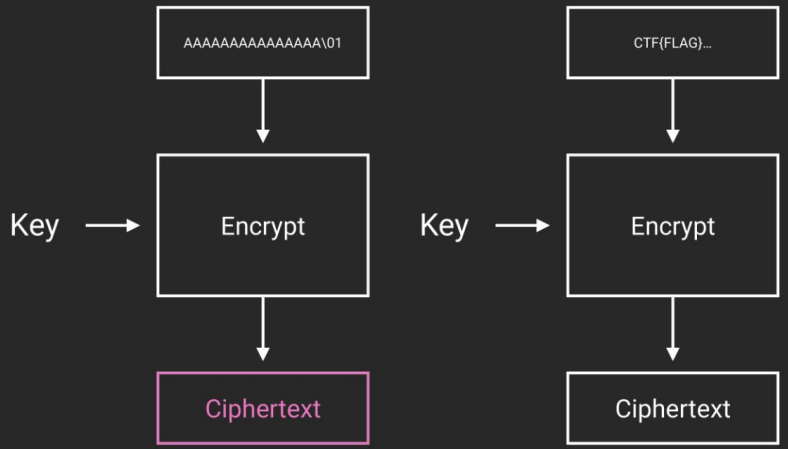
找 flag[0]

塞 15 個一樣的垃圾 +
爆搜最後一個 byte



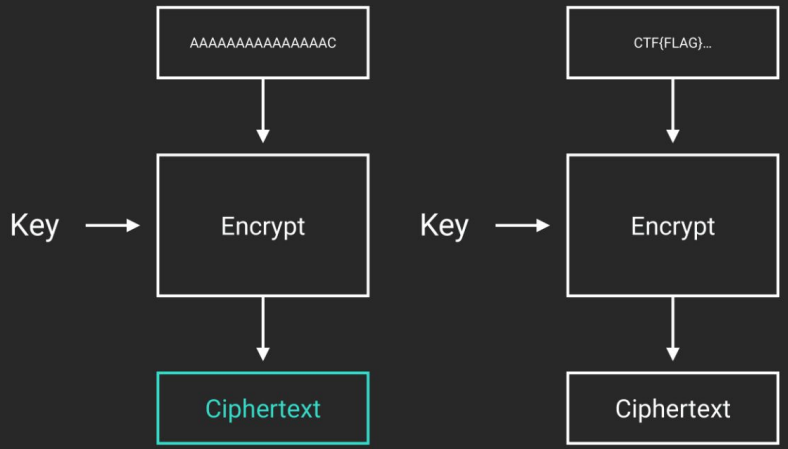
找 flag[0]

塞 15 個一樣的垃圾 +
爆搜最後一個 byte



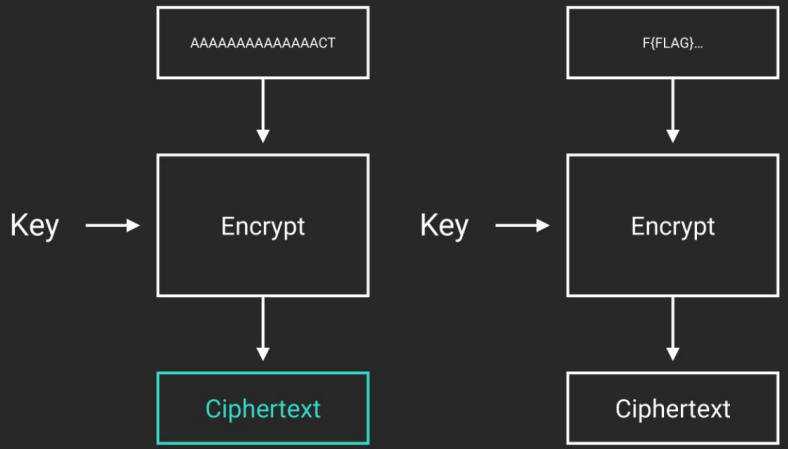
找 flag[0]

找到了 flag 的第一個 byte



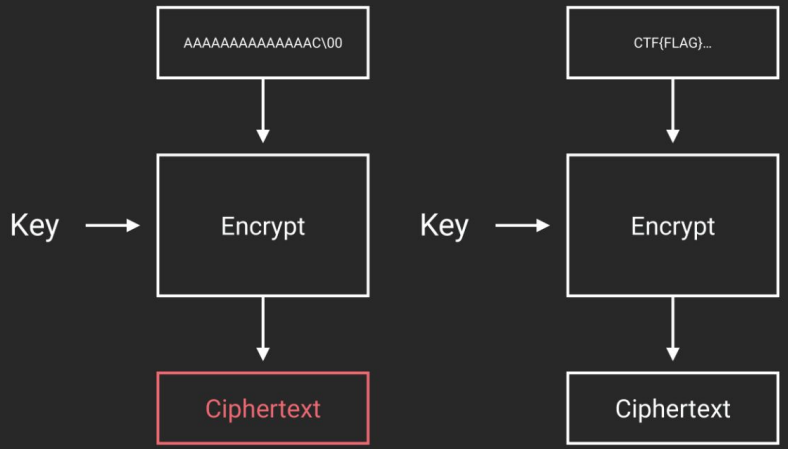
找 flag[1]

先塞 14 個垃圾
讓 flag 的前兩個字元掉進來



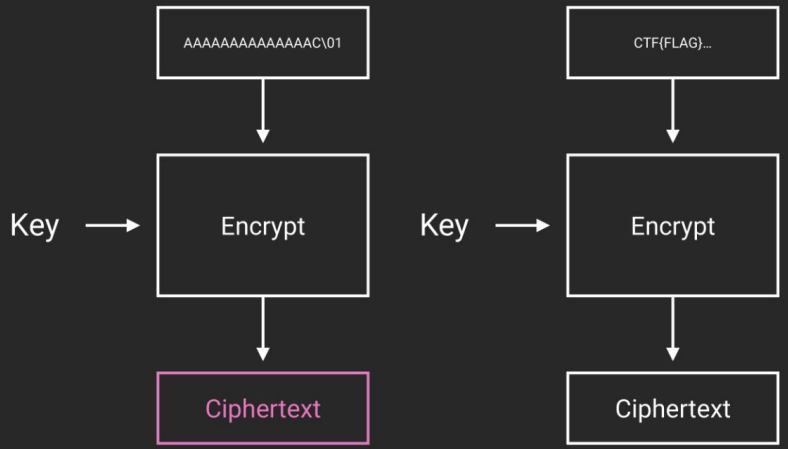
找 flag[1]

塞 14 個一樣的垃圾 +
已知的 flag 第一個 byte +
爆搜最後一個 byte



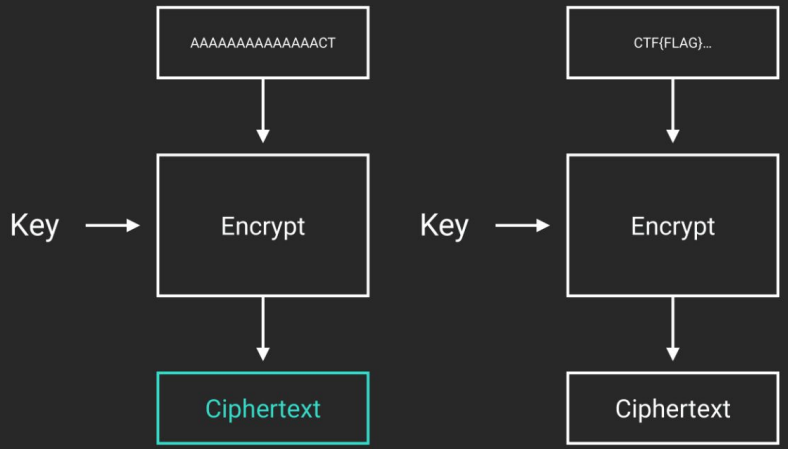
找 flag[1]

塞 14 個一樣的垃圾 +
已知的 flag 第一個 byte +
爆搜最後一個 byte



找 flag[1]

找到了 flag 的第二個 byte



Finally

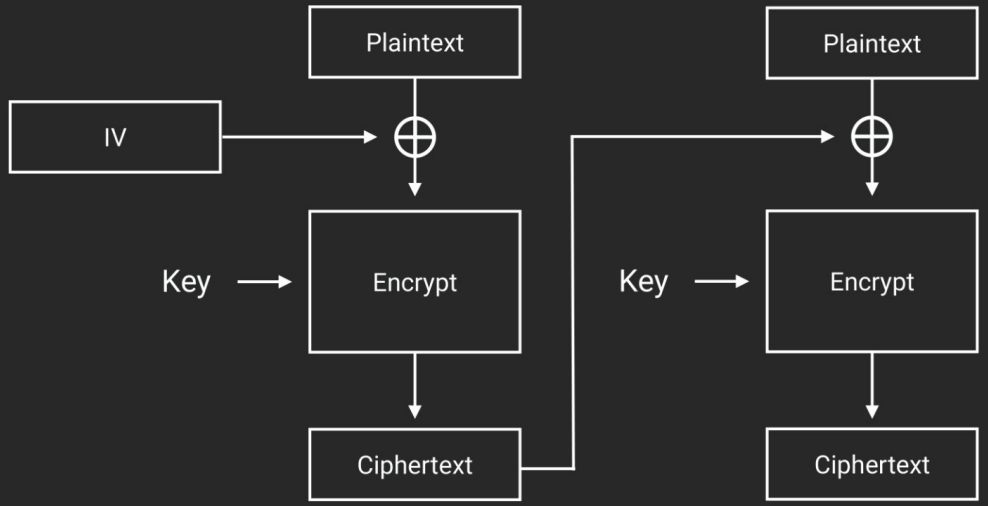
- 這樣一直做下去就能解密出 flag 的所有字元了
- 最多需要做 len(flag) * 257 次

CTF Challenges

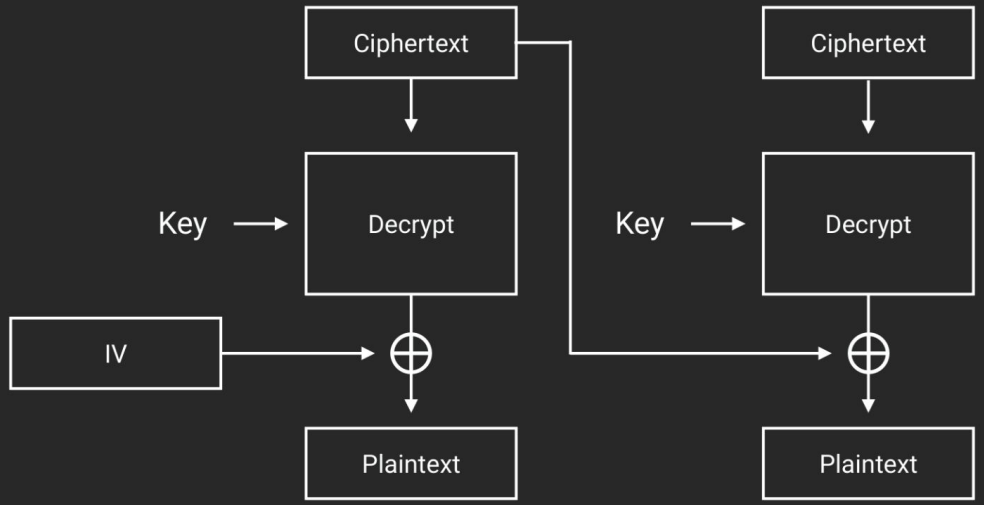
- **UTCTF 2020 - Random ECB**
- **TUCTF 2018 - AESential Lesson**
- **cryptohack.org - ECB Oracle**

CBC Mode

CBC Mode Encryption



CBC Mode Decryption

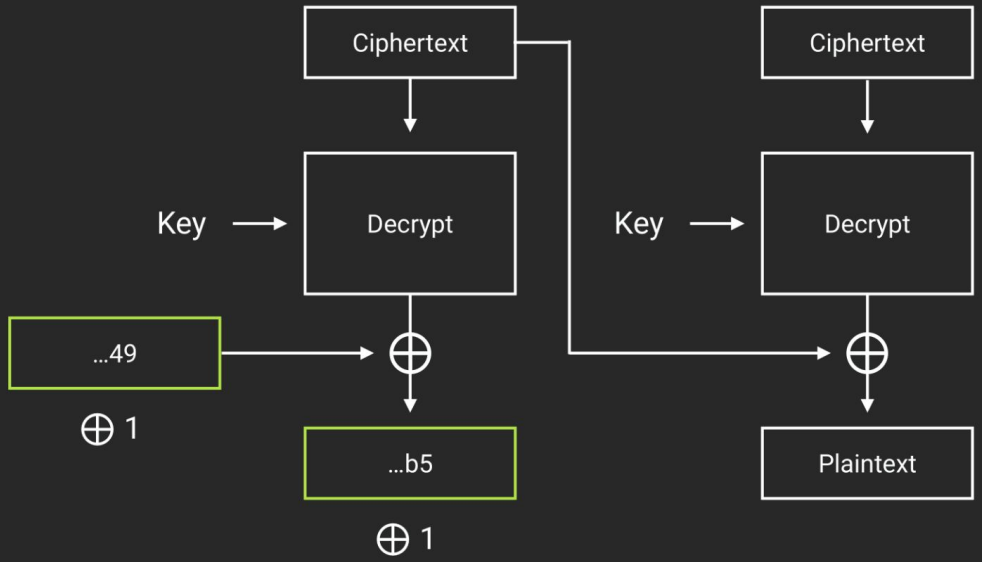


CBC Mode

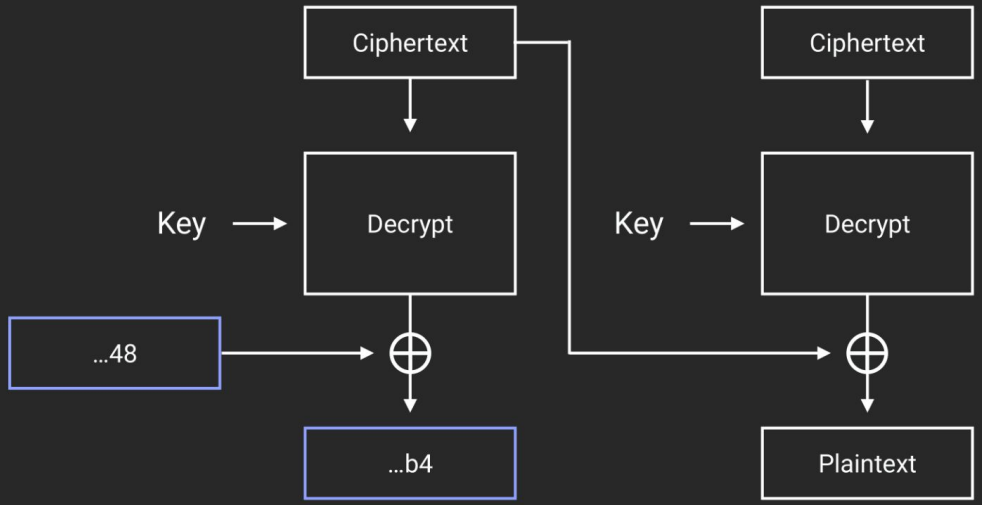
- 每塊密文都依賴前面所有的明文
- 相同的區塊明文會加密出不同的區塊密文
- 有一個初始化向量 (Initial Vector)，簡稱 IV

CBC Mode / Bit-Flipping Attack

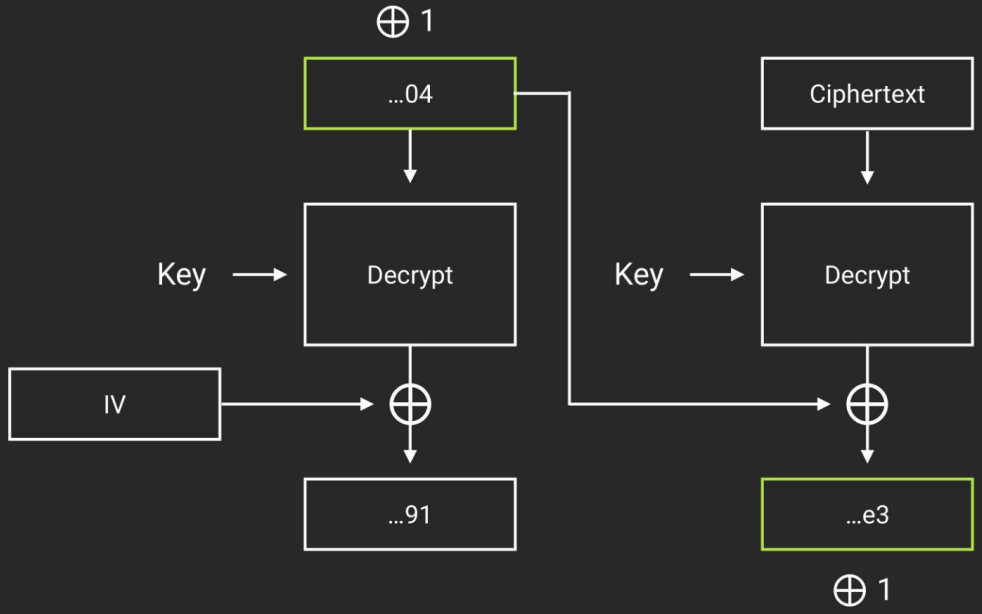
CBC Mode / Bit-Flipping Attack



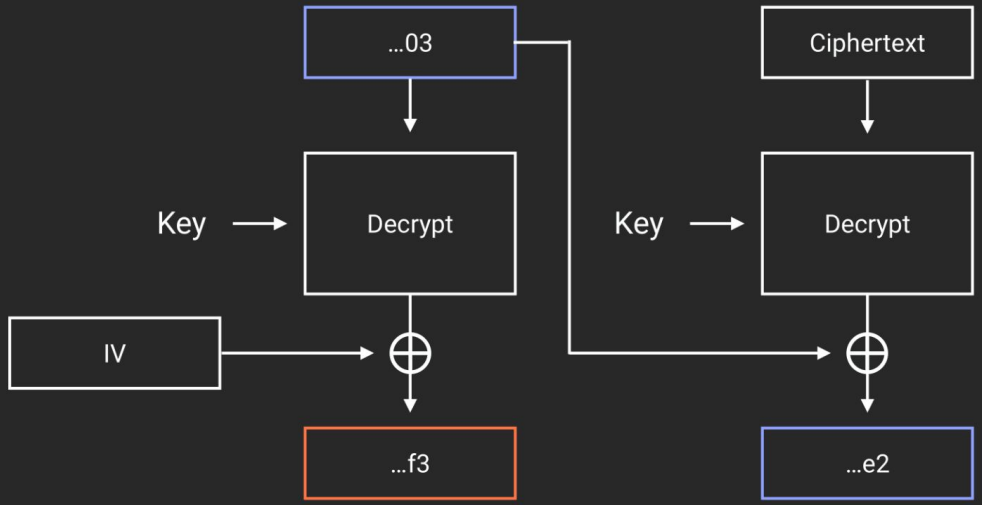
CBC Mode / Bit-Flipping Attack



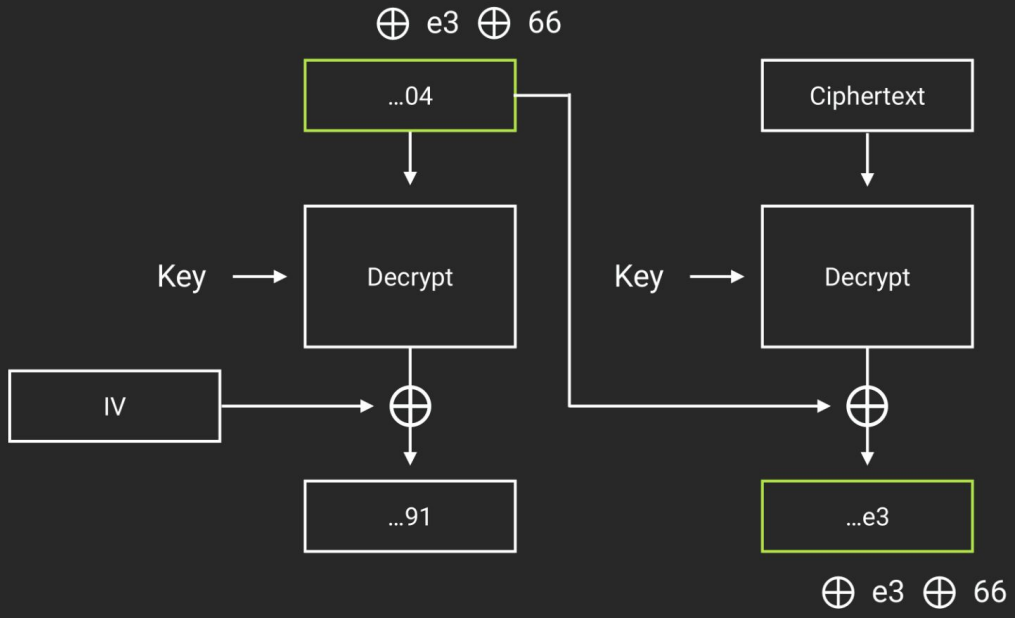
CBC Mode / Bit-Flipping Attack



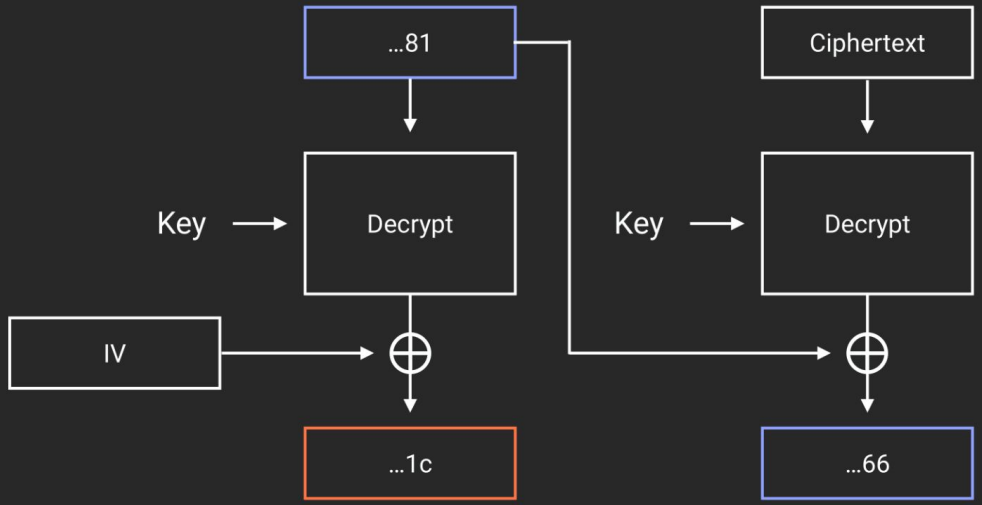
CBC Mode / Bit-Flipping Attack



CBC Mode / Bit-Flipping Attack



CBC Mode / Bit-Flipping Attack



將解密的明文改成我們指定的 0x66

CBC Mode / Bit-Flipping Attack

- 透過修改 IV 和 Ciphertext 來控制 Plaintext
- 其他 CFB, OFB, CTR 也存在相同的攻擊方式

CTF Challenges

- AIS3 preexam 2018 - EFAIL
- AIS3 preexam 2018 - BLIND
- AceBear CTF - CNVService
- Teaser Dragon CTF 2018 - AES-128-TSB

CBC Mode / Padding Oracle Attack

The Oracle

- 使用 AES CBC Mode 配上 PKCS#7 Padding Scheme
- 伺服器能幫我們解密訊息
- 如果解密出來的訊息 Padding 錯誤會噴錯

```
def unpad(data):
    if not all([x == data[-1] for x in data[-data[-1]:]]):
        raise ValueError
    return data[:-data[-1]]

def oracle(cipher):
    aes = AES.new(KEY, AES.MODE_CBC)
    try:
        plain = unpad(aes.decrypt(cipher))
    except ValueError:
        return False
    return True
```

PKCS#7

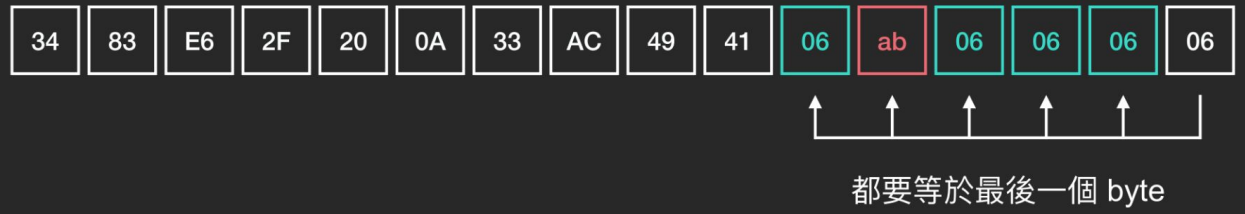
PKCS#7 : Cryptographic Message Syntax

https://tools.ietf.org/html/rfc2315

- 這個標準裡面定義了一種 Padding 的格式
  - 要填充 5 個 bytes 就填充 5 個 0x05
  - 要填充 2 個 bytes 就填充 2 個 0x02

Padding 錯誤

- 怎麼樣會 Padding 錯誤？
- 抓最後一個 byte 就可以知道 Padding 長度



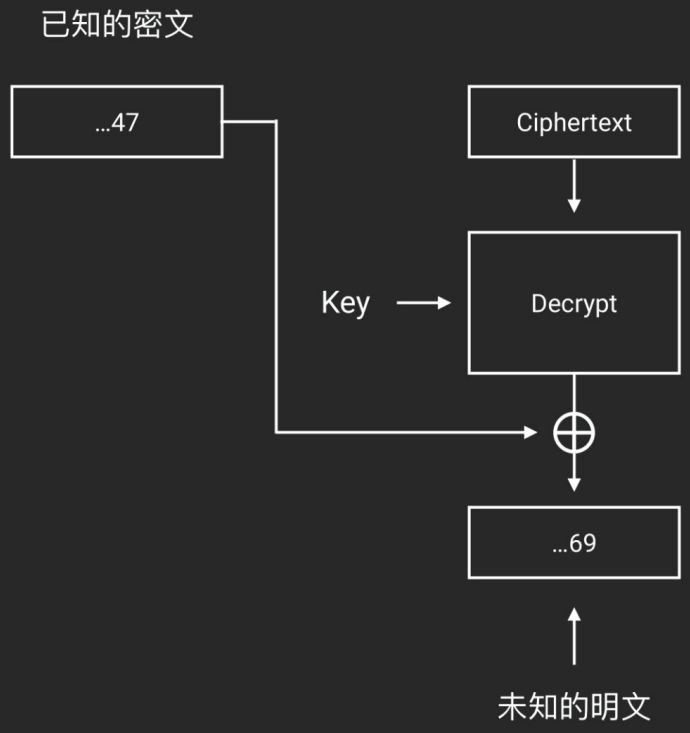
Python Implementation

```
def pad(data):
    p = 16 - len(data) % 16
    return data + bytes([p]) * p

def unpad(data):
    if not all([x == data[-1] for x in data[-data[-1]:]]):
        raise ValueError
    return data[:-data[-1]]
```

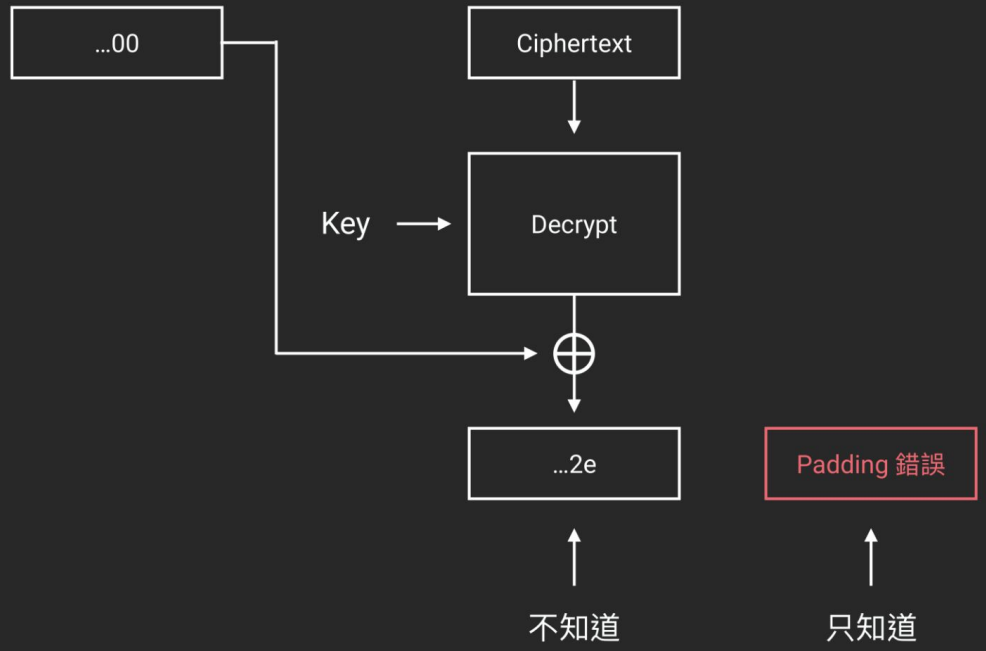
解 plaintext[-1]

Padding Oracle Attack



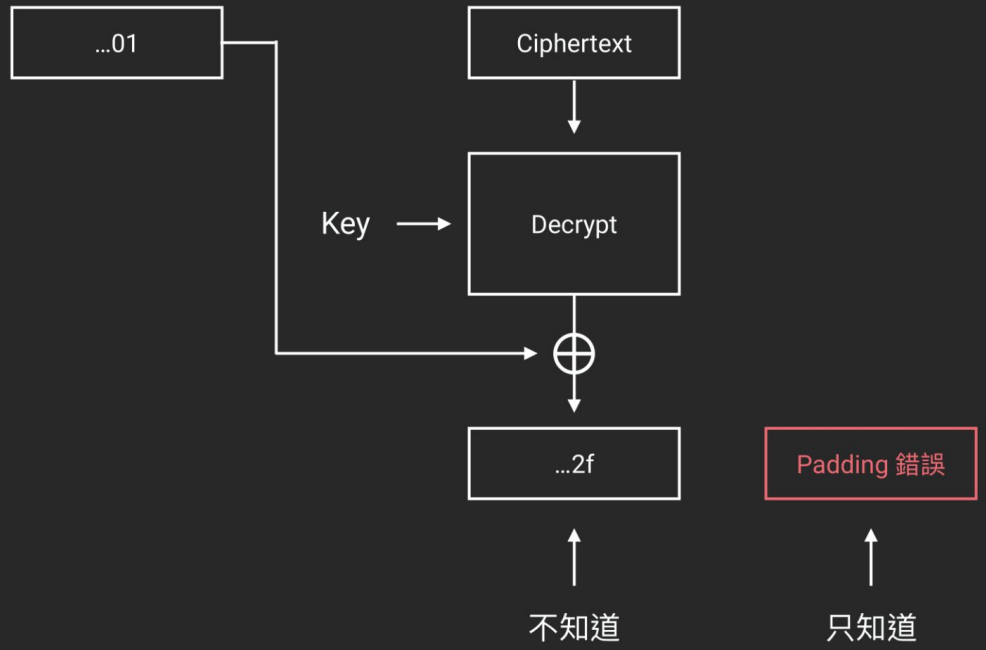
Padding Oracle Attack

暴力嘗試最後一個 byte



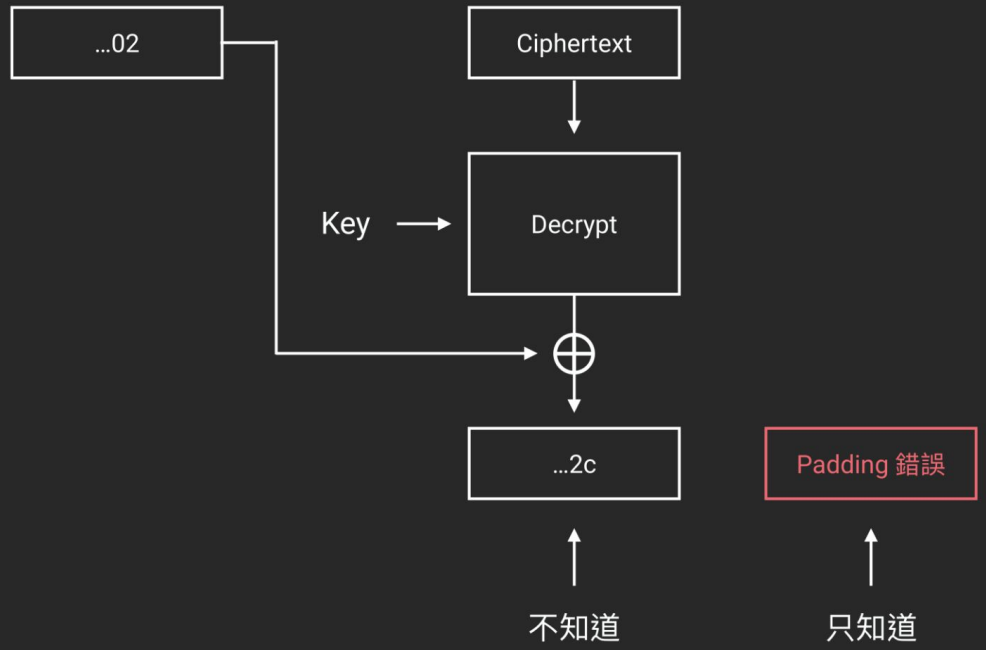
Padding Oracle Attack

暴力嘗試最後一個 byte



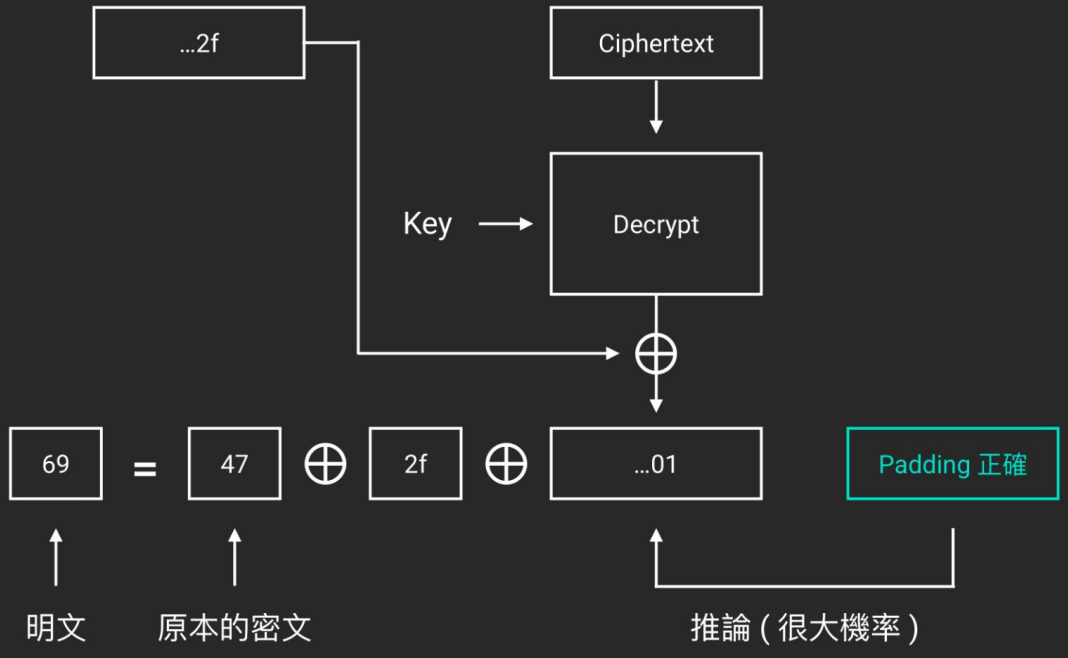
Padding Oracle Attack

暴力嘗試最後一個 byte



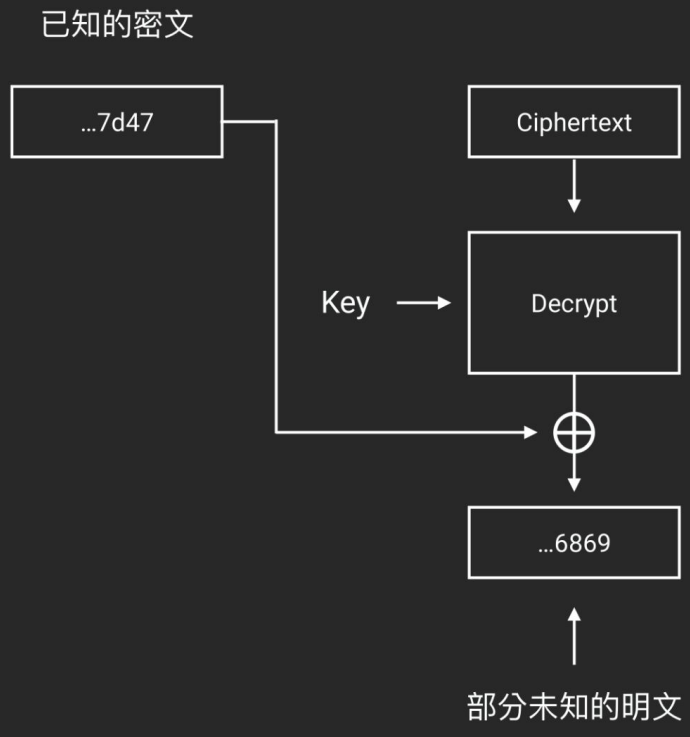
Padding Oracle Attack

暴力嘗試最後一個 byte

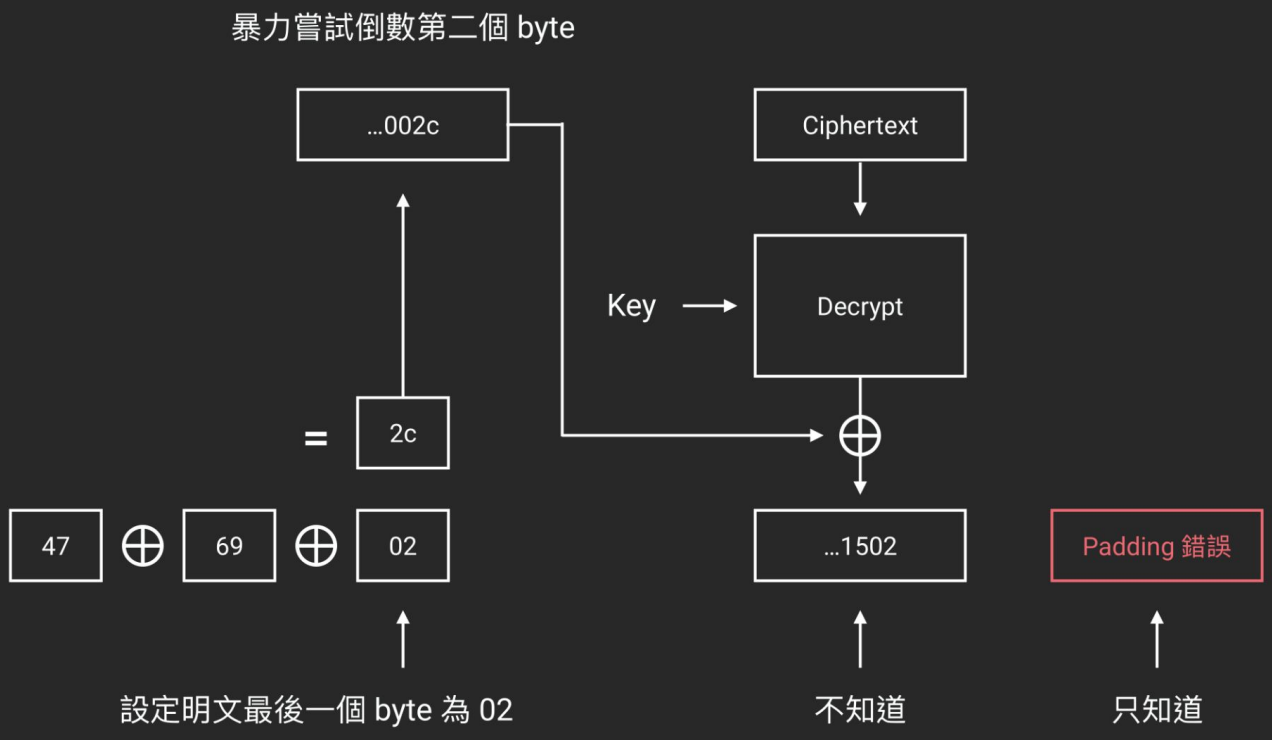


解 plaintext[-2]

Padding Oracle Attack

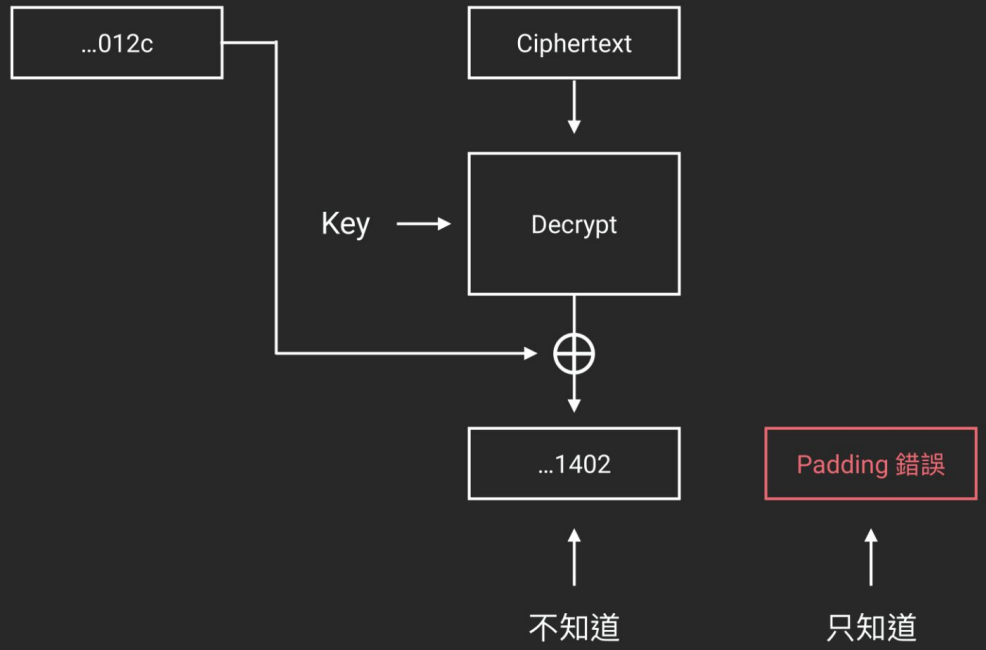


Padding Oracle Attack



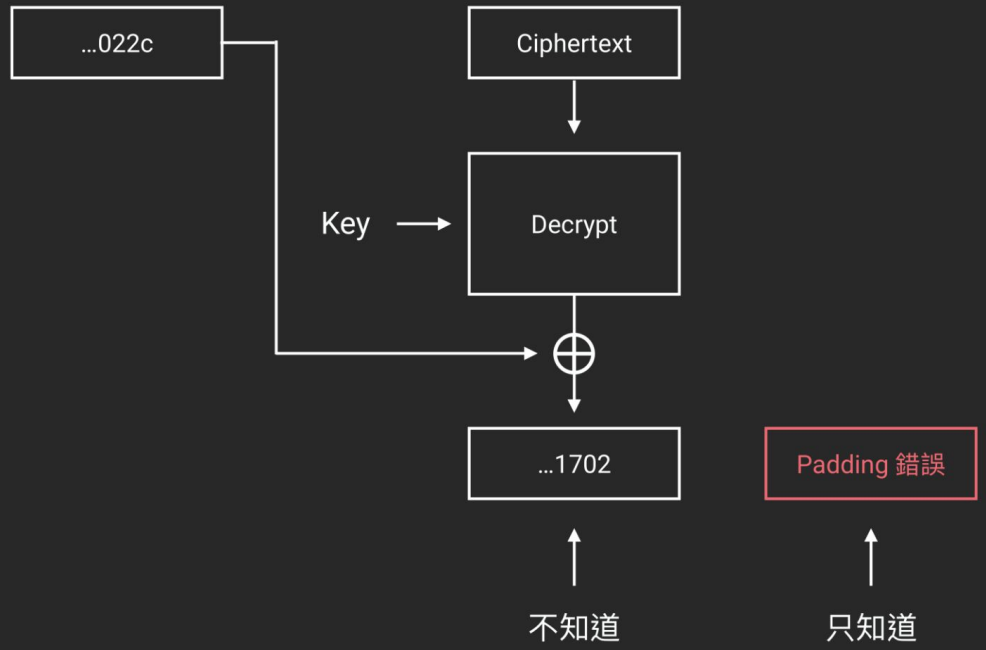
Padding Oracle Attack

暴力嘗試倒數第二個 byte



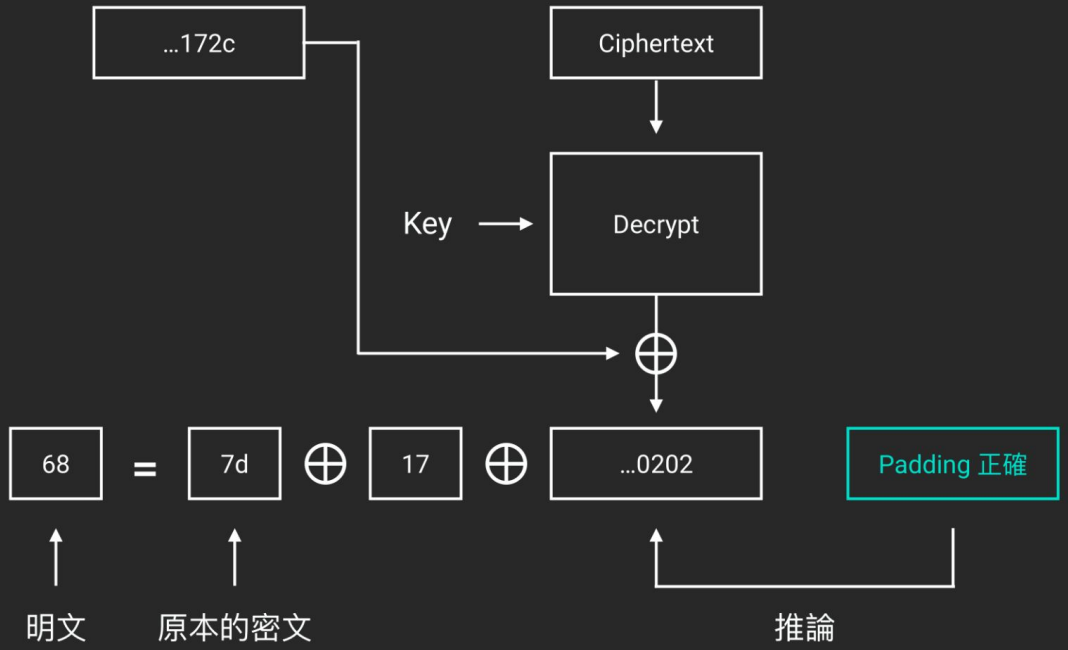
Padding Oracle Attack

暴力嘗試倒數第二個 byte



Padding Oracle Attack

暴力嘗試倒數第二個 byte



Summary

- 總共有三層迴圈
  - 猜 0 - 255 直到猜出一個 byte
  - 一次解出一個 byte 直到解完一個 block
  - 一次解出一個 block 直到解完所有 blocks
- 解出一個 block 最多需要 4096 次嘗試

CTF Challenges

原汁原味 padding oracle attack :

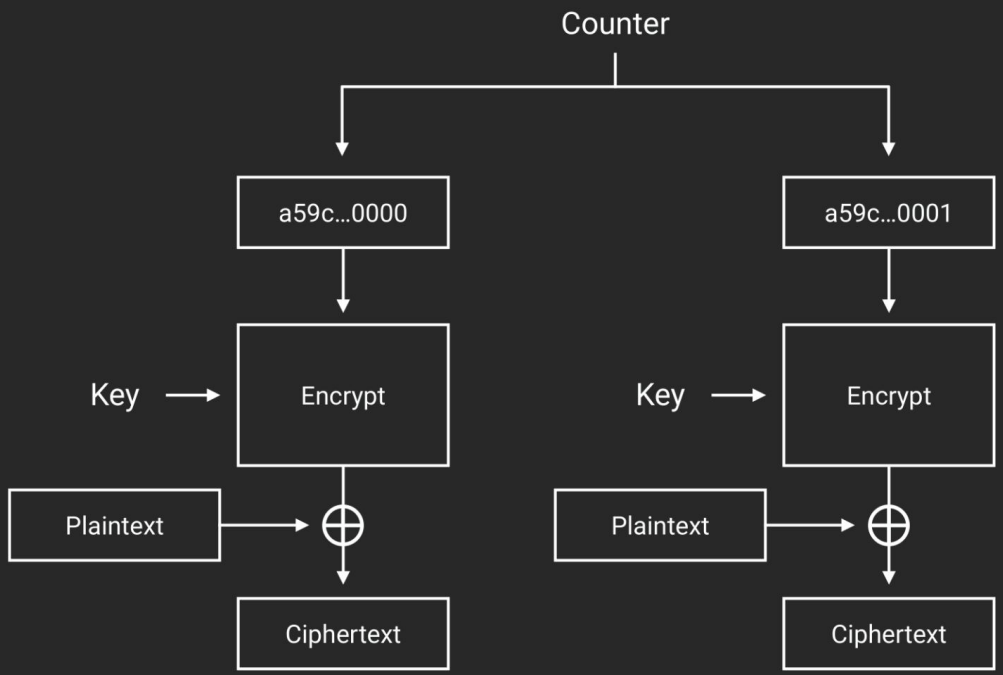
- CSAW CTF 2016 Quals - Neo
- HITCON CTF 2016 Quals - Hackpad
- BAMBOOFOX CTF 2018 - mini-padding

padding 相關攻擊技巧 :

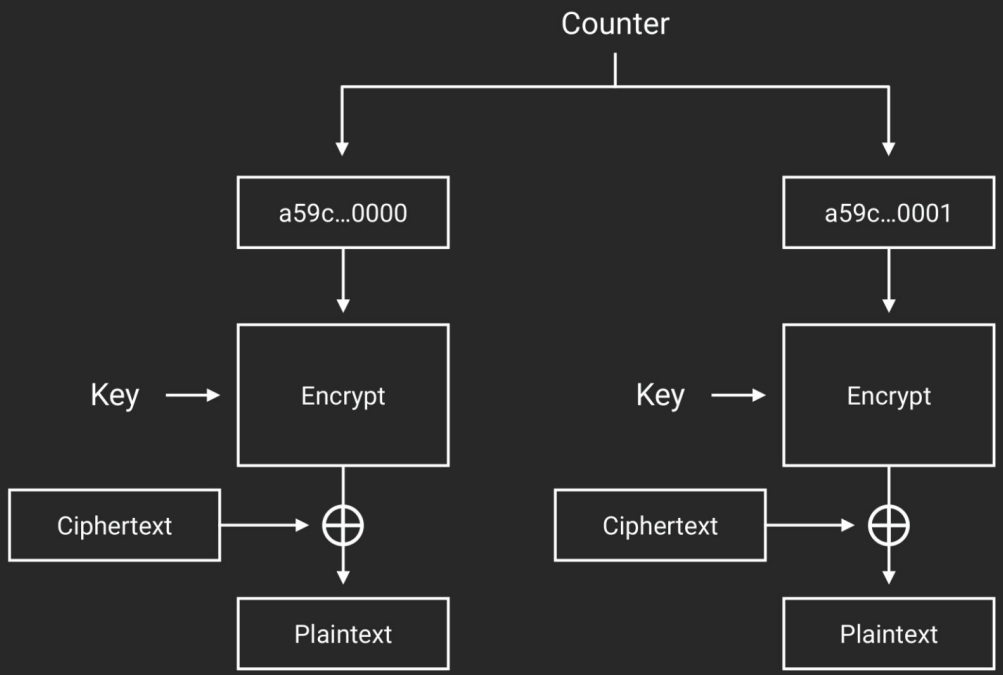
- HITCON CTF 2017 Quals - Secret Server
- HITCON CTF 2017 Quals - Secret Server Revenge
- BAMBOOFOX CTF 2018 - baby-lea-revenge
- BAMBOOFOX CTF 2018 - baby-lea-impossible

CTR Mode

CTR Mode Encryption



CTR Mode Decryption



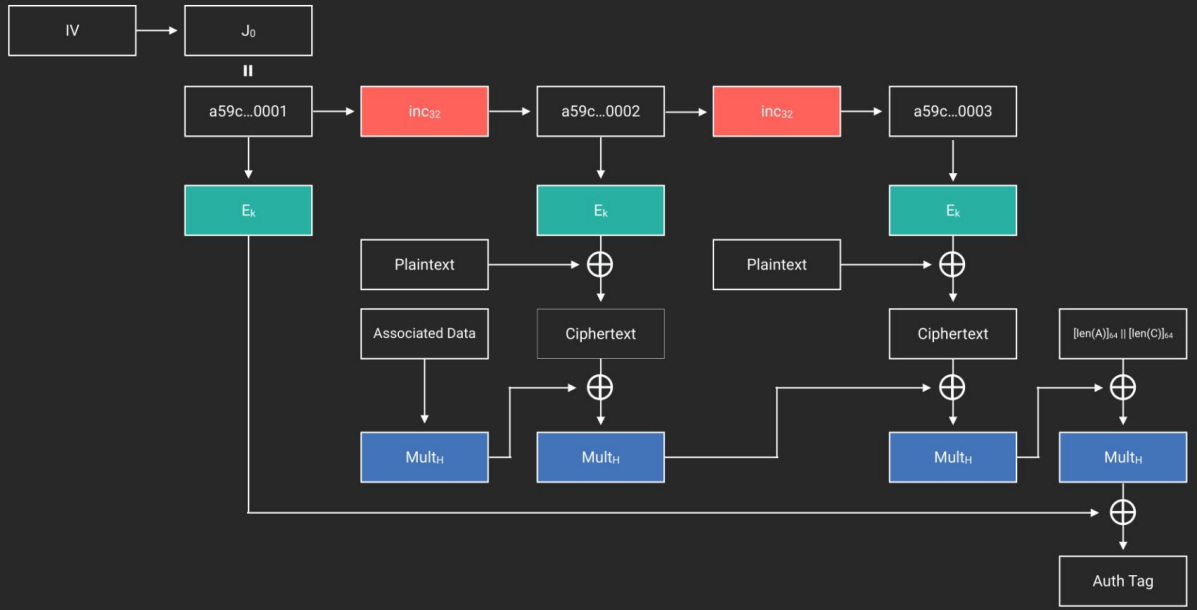
CTR Mode

- 利用 AES 去產生 xor key 然後做 xor cipher
- 加密和解密都是用 AES Encryption，因為要產生相同的 xor key
- Counter 會初始一個隨機數字，每次加一
- Block 之間沒有互相依賴，可平行運算

GCM Mode

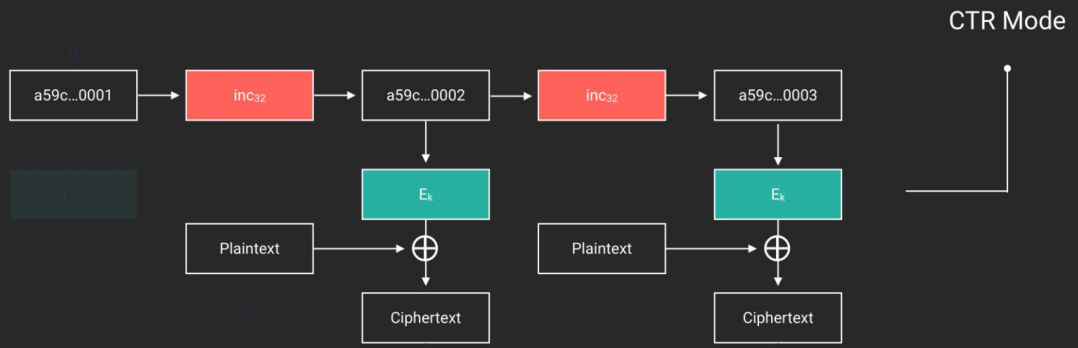
GCM Mode Encryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



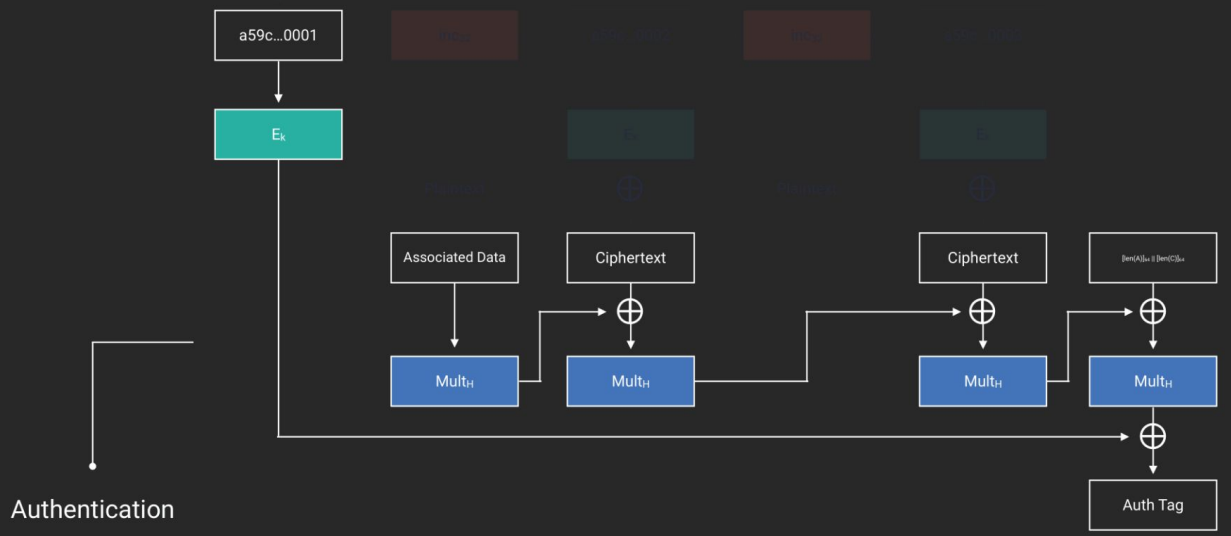
GCM Mode Encryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



GCM Mode Encryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



GCM Mode Encryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

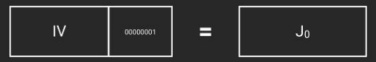
- len 是 bitlen
- $[10]_{64}$ → 000000000000000a

不是 16 bytes 的倍數會在後面補 \x00

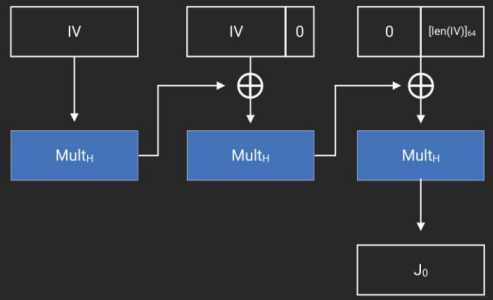
GCM Mode Encryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

如果 len(IV) = 96

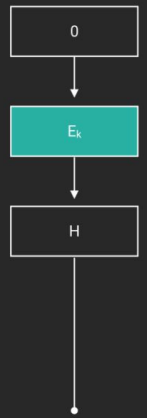


如果 len(IV) ≠ 96



GCM Mode Encryption

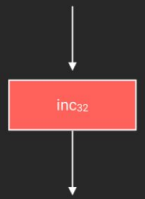
https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



$H = E_k(0)$

GCM Mode Encryption

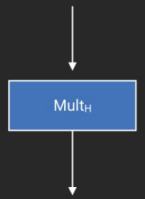
https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



- 低位 32 bits 加一 modulo 2^{32}
- 剩下的高位不動

GCM Mode Encryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



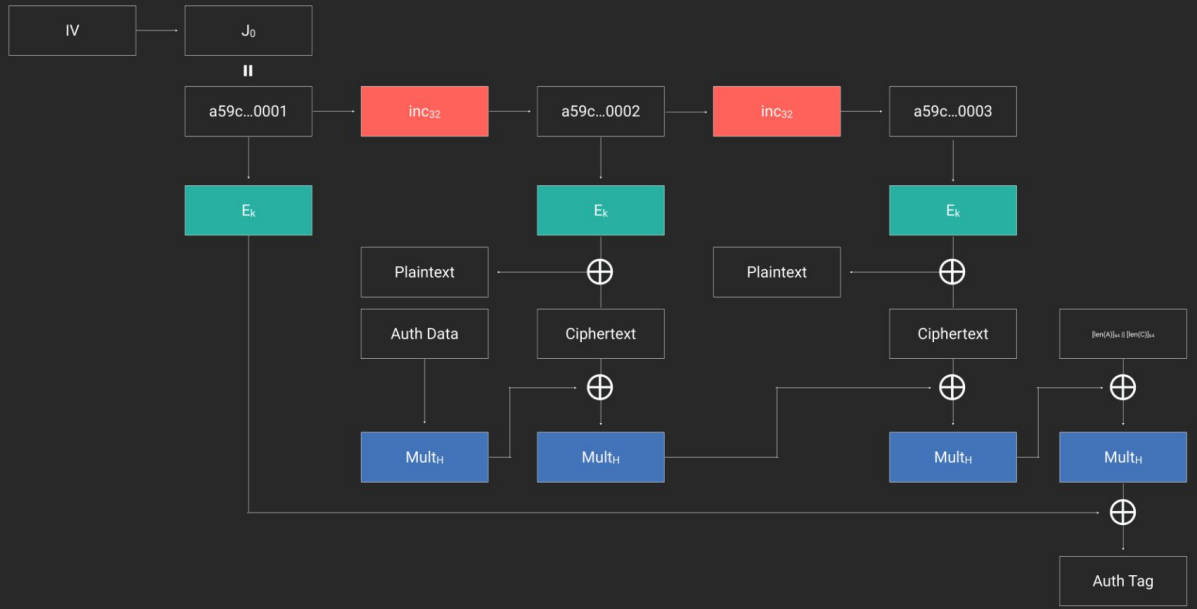
- $Mult_H$ 是在 GF(2^{128}) 下乘上 H
- GF(2^{128}) 的 reduction modulus 是 $x^{128} + x^7 + x^2 + x + 1$
- 從 bytes 轉成 GF(2^{128}) 下的元素
  - u is the variable of the polynomial
  - $x_0x_1...x_{127} \rightarrow x_0 + x_1u + ... + x_{127}u^{127}$
  - '\x00...\x00\x01\x02' → $x^{119}+x^{126}$

Galois Field

- Galois Field (GF) 或叫 Finite Field
- GF(2^{128}) 裡面的元素可以表示成一個 degree = 127 的多項式
- 加法和乘法就是做多項式的加法和除法 modulo 一個 degree = 128 的 irreducible polynomial
  - GCM Mode 用的 GF(2^{128}) 是 modulo $x^{128} + x^7 + x^2 + x + 1$

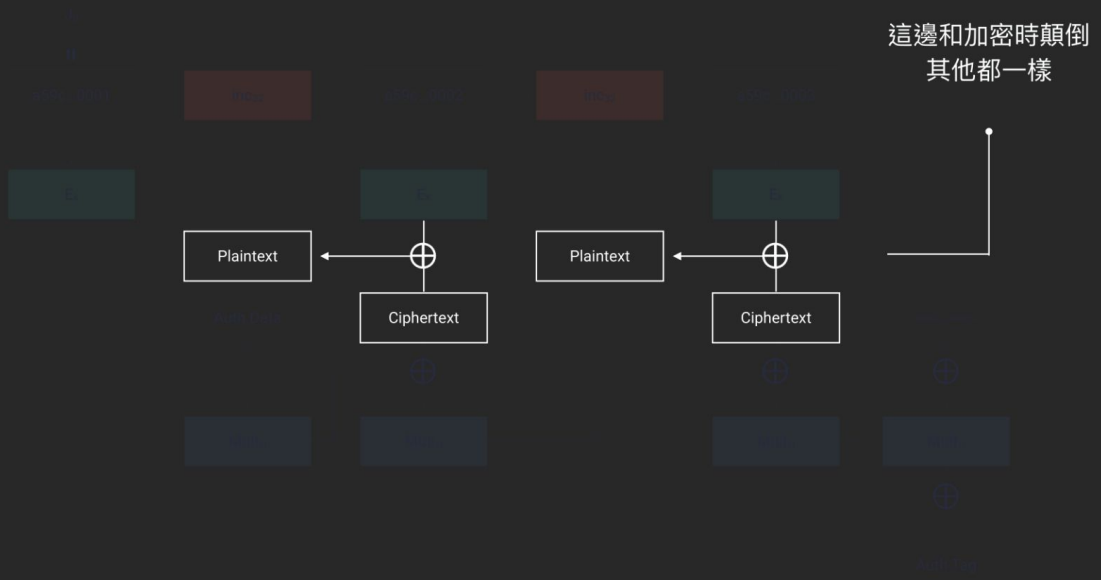
GCM Mode Decryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



GCM Mode Decryption

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf



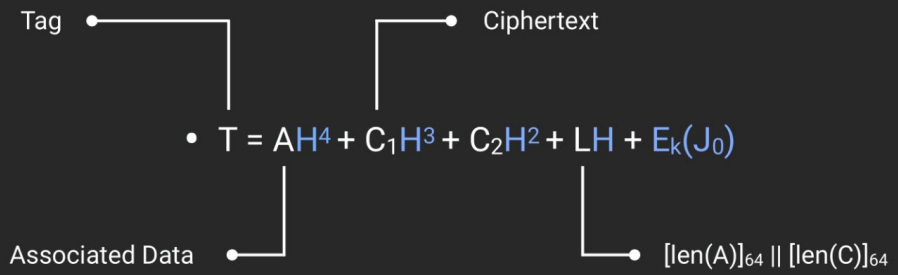
GCM Mode / Forbidden Attack

GCM Mode / Forbidden Attack

- 又叫 Nonce Repeating Attack
- 也就是當 IV 重複用的情況
- 假設可以拿到任意明文加密的結果
- 這個攻擊可以做到**偽造簽章**

GCM Mode / Forbidden Attack

- 在 Authentication 中做的 xor 就是 GF(2^{128}) 中的加法
- Tag 可以表示成在 GF(2^{128}) 中的式子如下



未知的只有藍色的部分

GCM Mode / Forbidden Attack

- 做兩次加密拿到兩個 Tag
- 兩個相減後只剩 H 未知，解方程式求根可得 H
- 有 H 帶回原式可得 $E_k(J_0)$ 就可以偽造 Tag 了

- $T_1 = A_1H^4 + C_{11}H^3 + C_{12}H^2 + L_1H + E_k(J_0)$
- $T_2 = A_2H^4 + C_{21}H^3 + C_{22}H^2 + L_2H + E_k(J_0)$
- $T_1 - T_2 = (A_1 - A_2)H^4 + (C_{11} - C_{21})H^3 + (C_{12} - C_{22})H^2 + (L_1 - L_2)H$

CTF Challenges

- **UTCTF 2020 - Galois**
- **VolgaCTF 2018 Quals - Forbidden**

Lab 3: Padding Oracle Attack