Crypto

Week 1

Thanks!

- Over 95% of these slides provided by Kuruwa

whoami

- AnJ
- A bachelor studying computer science in NTU
- A member of Balsn

Useful Tools

- PyCryptodome
  - pip install pycryptodome
- Sage
  - apt-get install sagemath
  - https://sagecell.sagemath.org/
  - If you have any installation issues and do not want to fix it: https://cocalc.com/
  - In sagemath:
    ^^ : xor, ^ : power

Before we start ……
hw0 review!

HW0

- xor
- Multiplicative Inverse
- Chinese Remainder Theorem (a.k.a. CRT)

xor

- Retrieve from xorrrrr

```python
def xorrrrr(nums):
    n = len(nums)
    result = [0] * n
    for i in range(1, n):
        result = [ result[j] ^ nums[(j+i) % n] for j in range(n)]
    return result
```

- a ⊕ a ⊕ a ⊕ … ⊕ a = ?
  - a ⊕ 0 = a
  - a ⊕ a = 0
  - Odd times -> a
  - Even times -> 0

Multiplicative Inverse

```
hint = [secret * muls[i] % mods[i] for i in range(20)]
```

- a * b = c (mod n)
  -> a = c * b^{-1} (mod n)
- b^{-1}: b's Multiplicative Inverse

- Satisfying b * b^{-1} = 1 (mod n)
- For example:
  3 * 6 = 1 (mod 17), 17 * 19 = 1 (mod 23)……

- In Python, you can just use `pow(b, -1, n)`

Chinese Remainder Theorem (CRT)

- $x \% n_1 = a_1$
- $x \% n_2 = a_1$
- …
- $x \% n_k = a_k$

- $N = n_1 * n_2 * \dots * n_k$
- $y_i = \frac{N}{n_i}$, $z_i = y_i^{-1} \pmod{n_i}$
- $x \% N = \sum_{i=1}^{k} a_i y_i z_i$

ToC

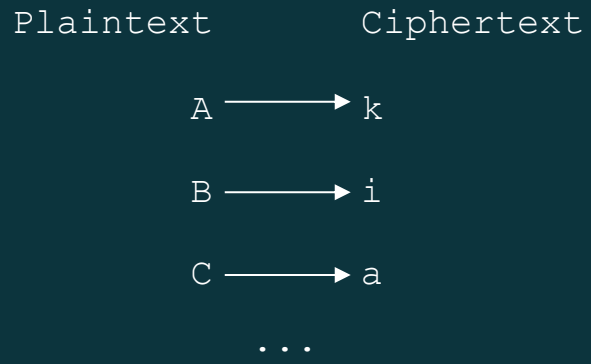
Introduction

Cryptanalysis

- Kerckhoffs'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
  - Brute-Force Attacks
  - Mathematical Analysis
- Implementation Attacks
- Social Engineering
- ……

Substitution Cipher

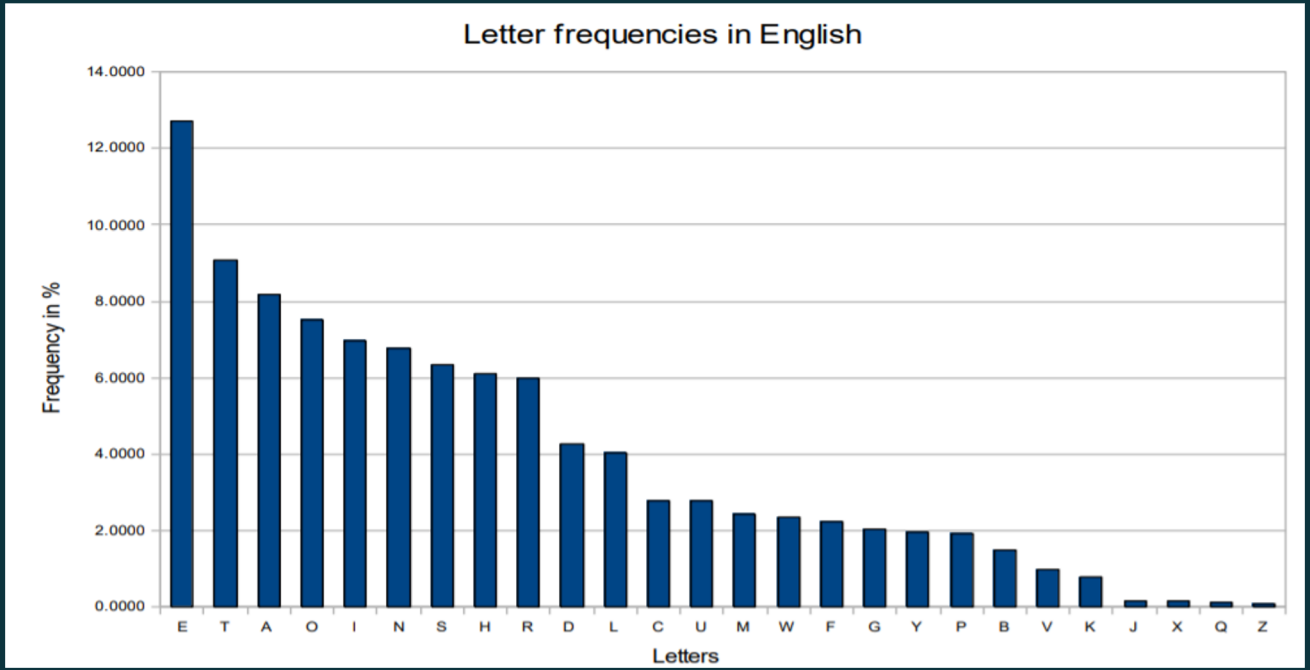


iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc hwwhbsqvqbre hwq vhlq

Attack 1: Exhaustive Key Search

- How many substitution tables (= keys) are there?
  - $26 \times 25 \times 24 \times \ldots \times 1 \approx 2^{88}$

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

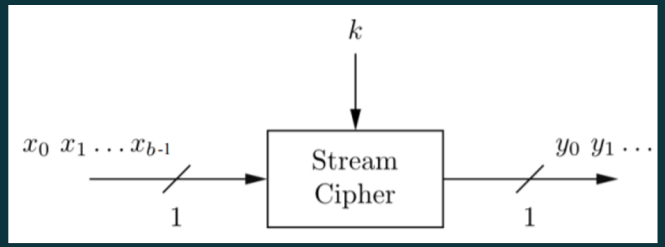
Attack 2: Letter Frequency Analysis

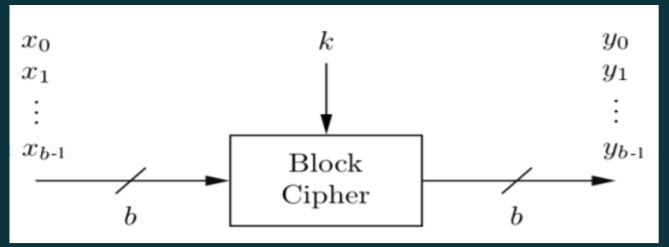
- In practice, not only frequencies of individual letters can be used for an attack, but also the frequency of letter pairs (i.e., “th” is very common in English), letter triples, etc

Symmetric cryptography

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

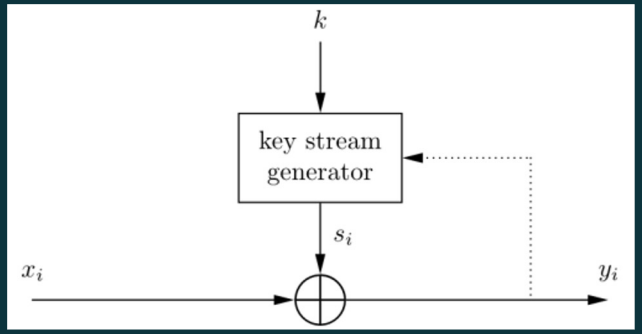




Stream Cipher

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

Stream Cipher



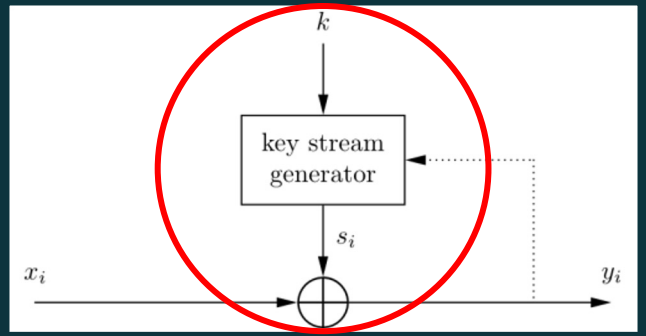
- Encryption
  - $y_i = e_{si}(x_i) = x_i \oplus s_i$
- Decryption
  - $x_i = d_{si}(y_i) = y_i \oplus s_i$
- Security depends entirely on key stream
  - random
  - reproducible
- Synchronous v.s. Asynchronous

Reused Key Attack

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

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

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

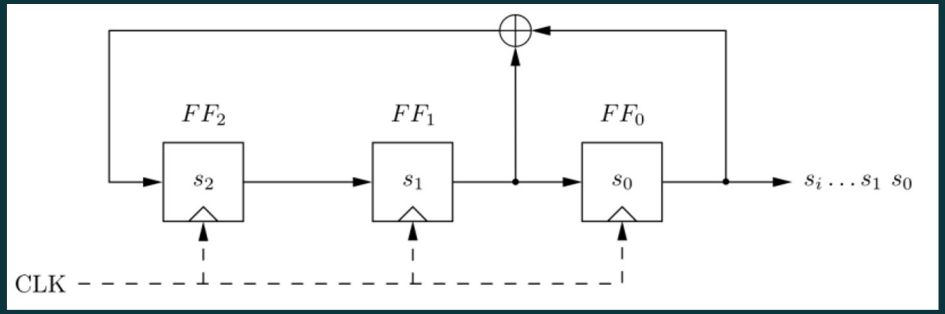
Assume

- 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

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

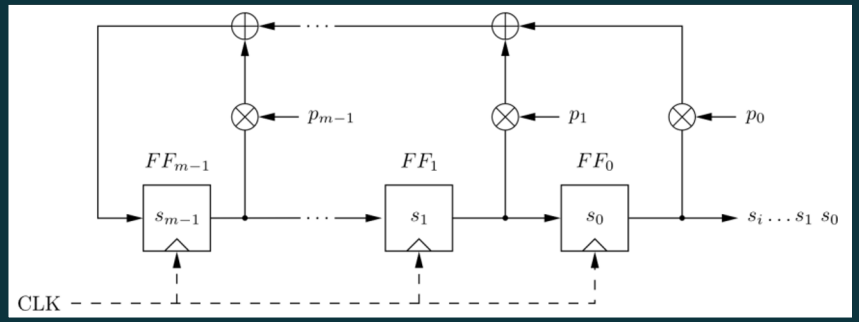
Linear Feedback Shift Register



- $S_{i+3} = S_{i+1} \oplus S_i$
- 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
  - $P(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1x + p_0$
  - State of LFSR <-> $(1, x, x^2, x^3, \ldots) \bmod P(x)$
  - Output is the coefficient of the term of degree m-1
- LFSR has maximum output period, iff P(x) is primitive polynomial

Companion Matrix

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

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

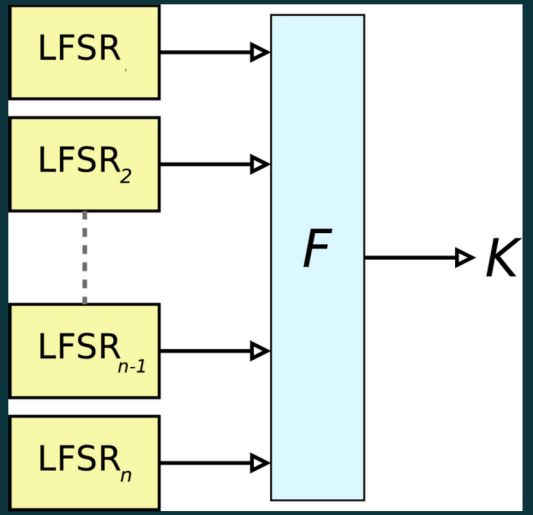
Companion Matrix

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

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 1:
Correlation Attacks

Block Cipher

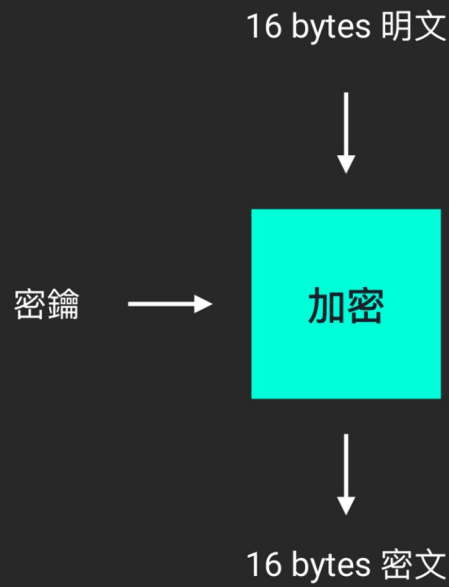
- Block of Operation
  - ECB
  - CBC
  - CTR
  - GCM

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

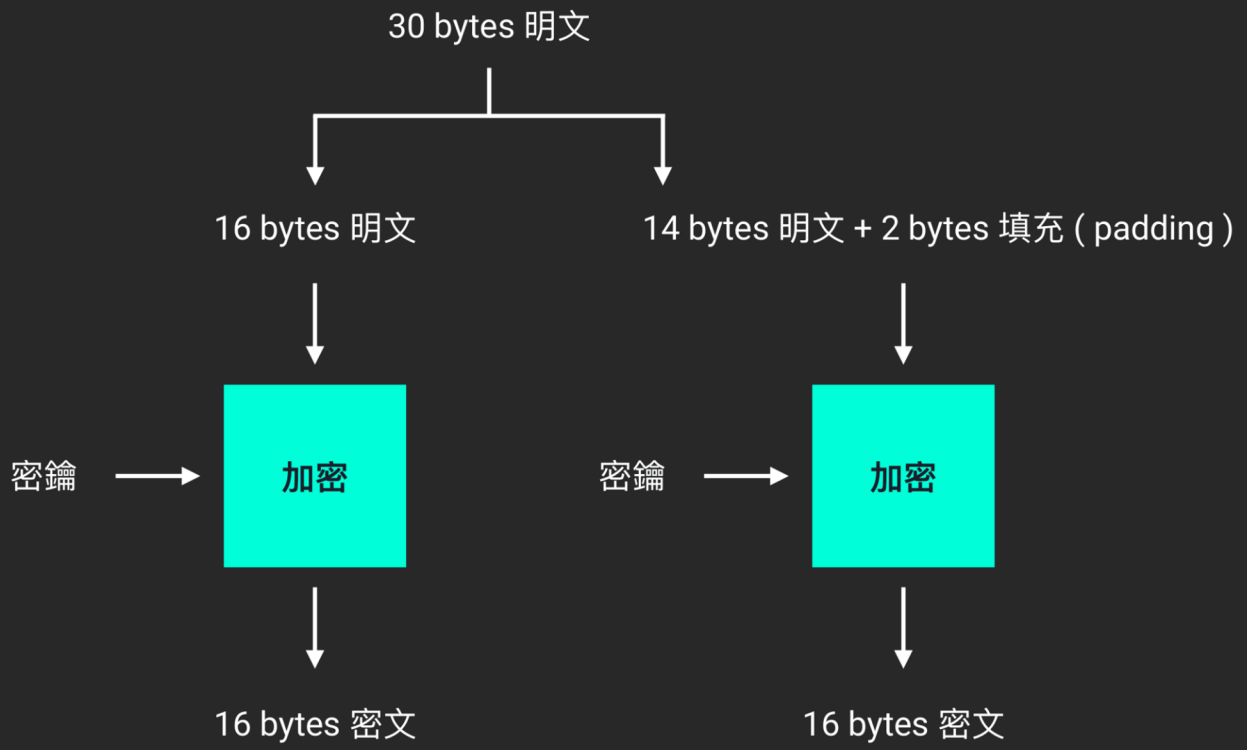
最常見

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

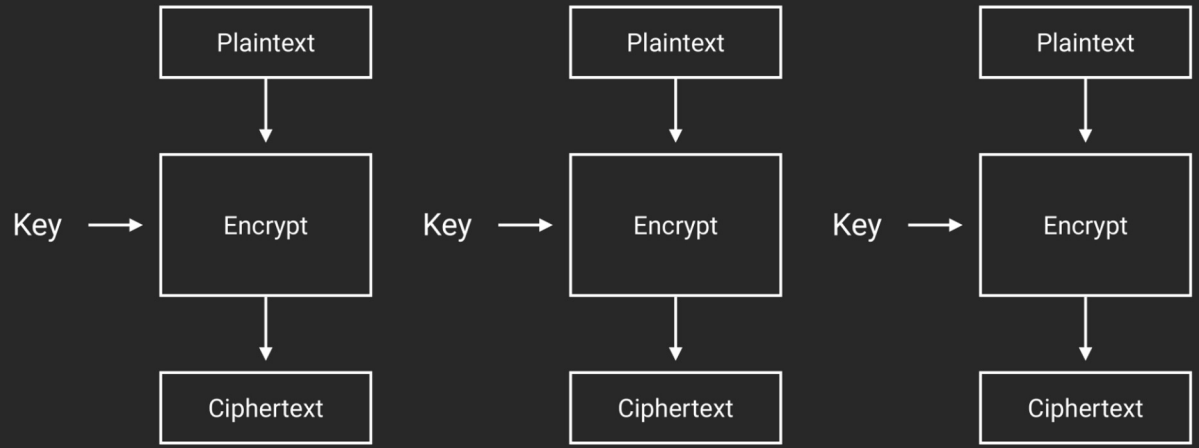


Block Cipher Mode

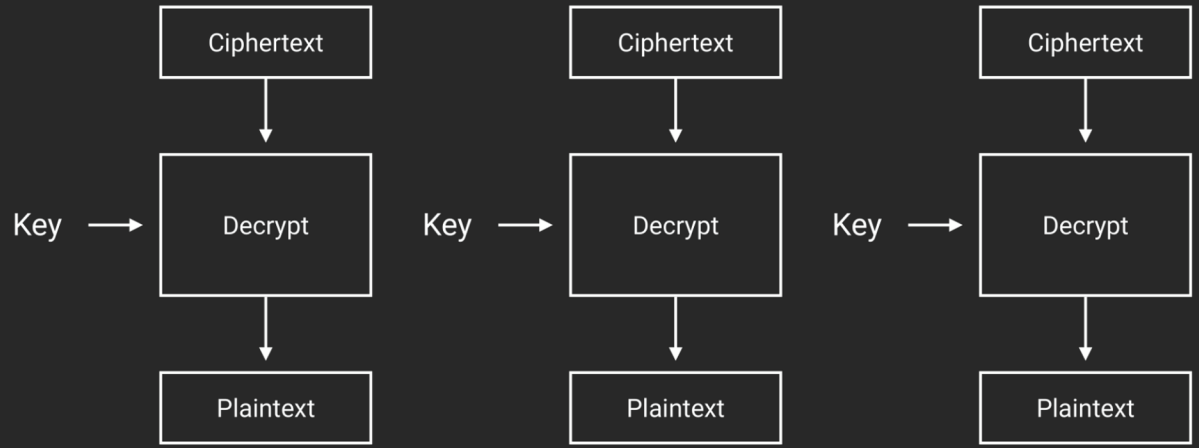
- AES 是 Block Cipher，輸入輸出長度固定
- 所以要加密**任意長度**的明文，需要一些額外加工
- 一些常見的加工模式：ECB, CBC, CFB, OFB, CTR...
- 有 AEAD 的模式：CCM, GCM, OCB...

ECB Mode

ECB Mode Encryption



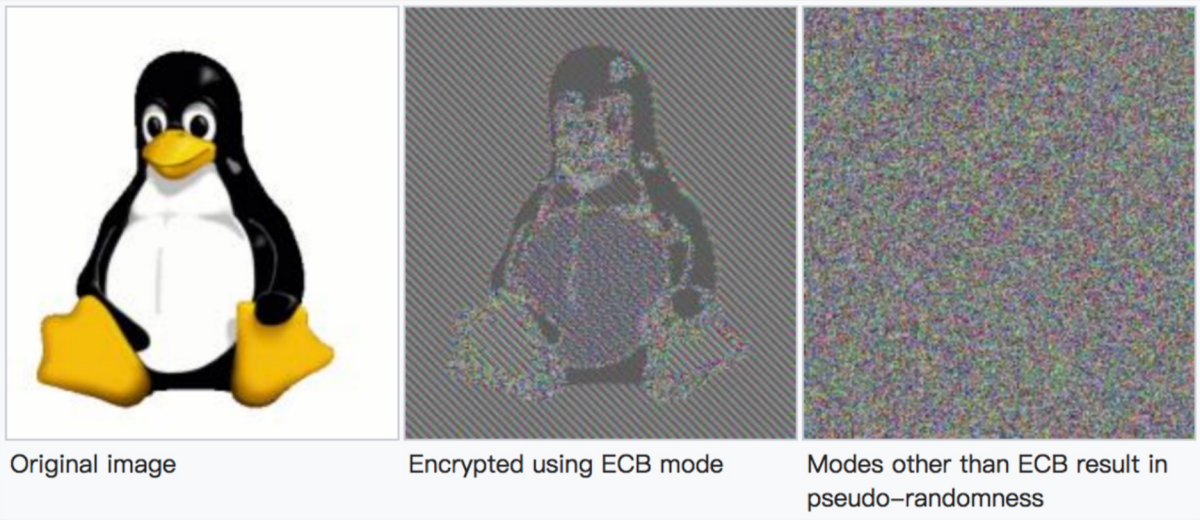
ECB Mode Decryption



ECB Mode

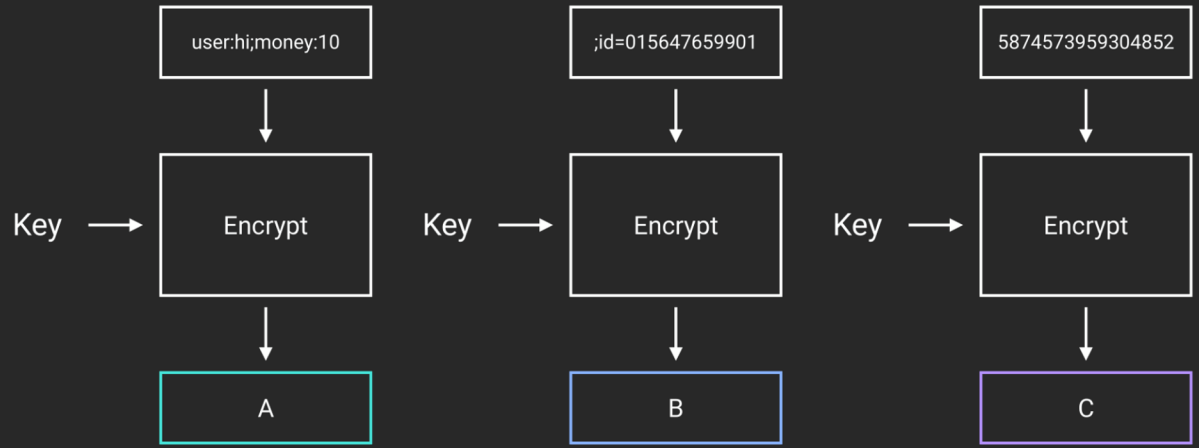
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

ECB 模式的缺點：相同的明文區塊會加密出相同的密文區塊

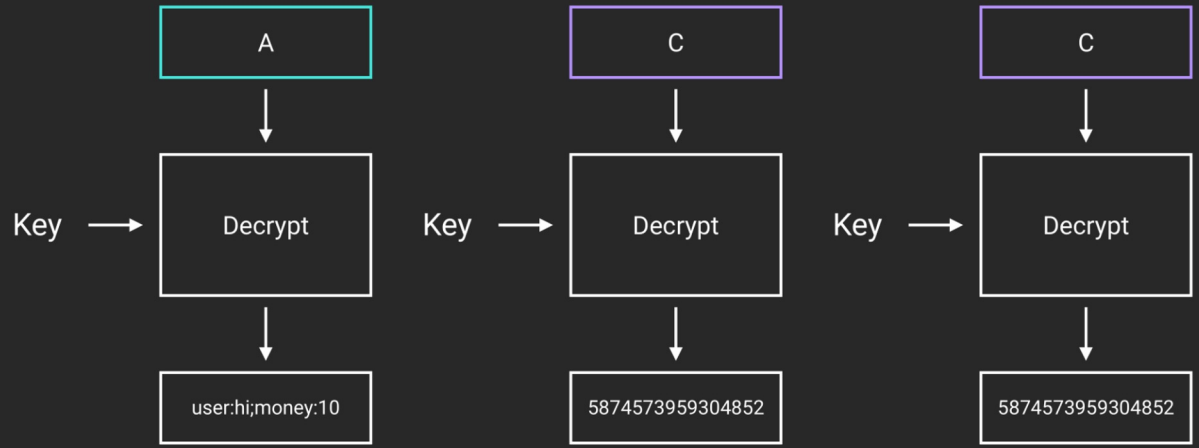


ECB Mode / Cut & Paste

ECB Mode / Cut & Paste



ECB Mode / Cut & Paste



錢錢變多了xD

ECB Mode / Prepend Oracle Attack

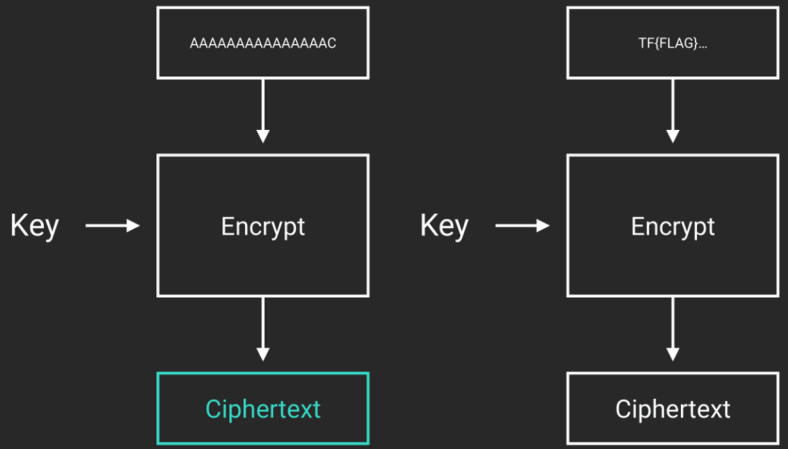
The Oracle

- 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)
```

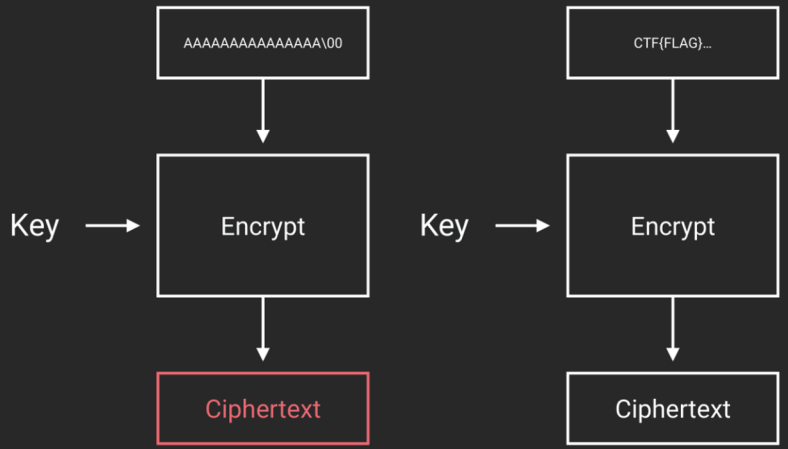
找 flag[0]

先塞 15 個垃圾
讓 flag 的第一個字元掉進來



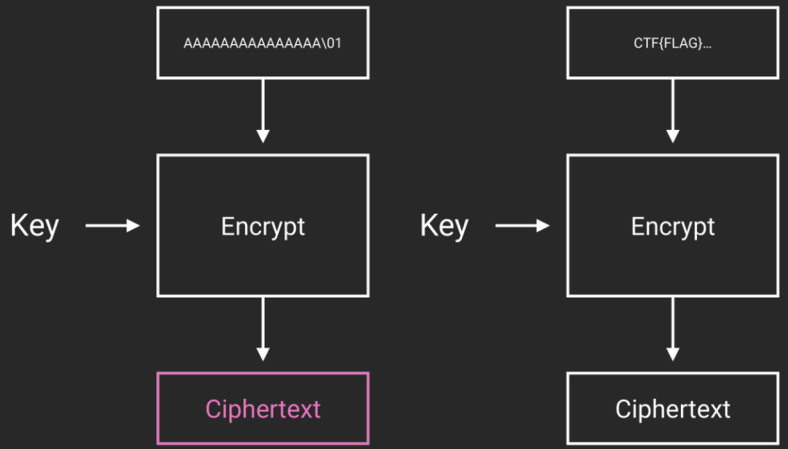
找 flag[0]

塞 15 個一樣的垃圾 +
爆搜最後一個 byte



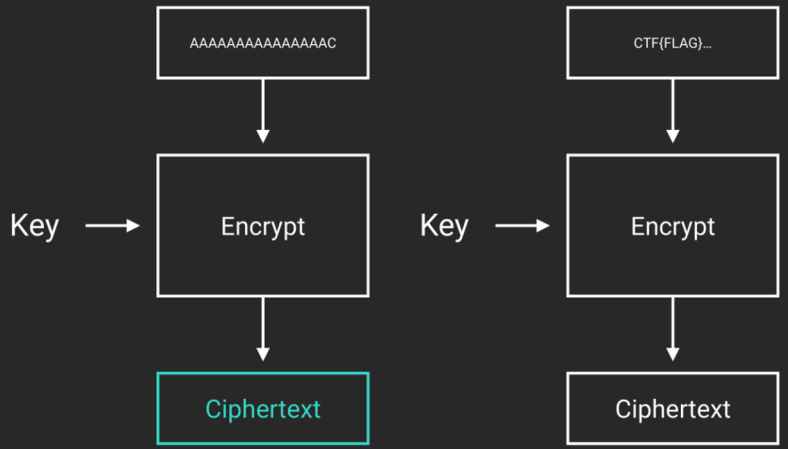
找 flag[0]

塞 15 個一樣的垃圾 +
爆搜最後一個 byte



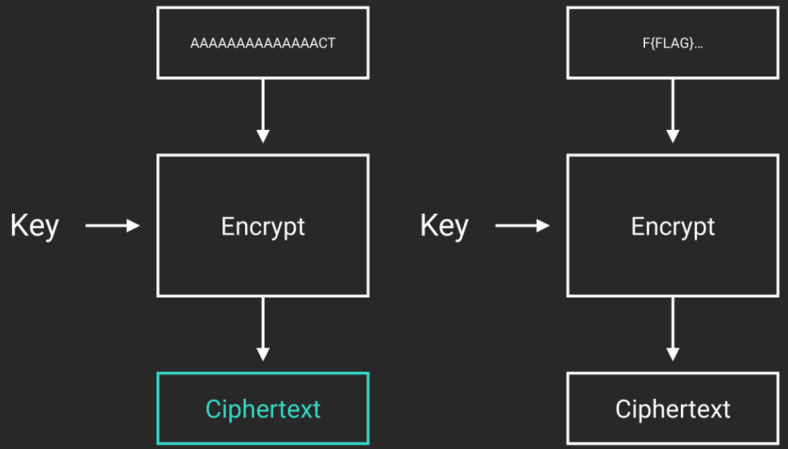
找 flag[0]

找到了 flag 的第一個 byte



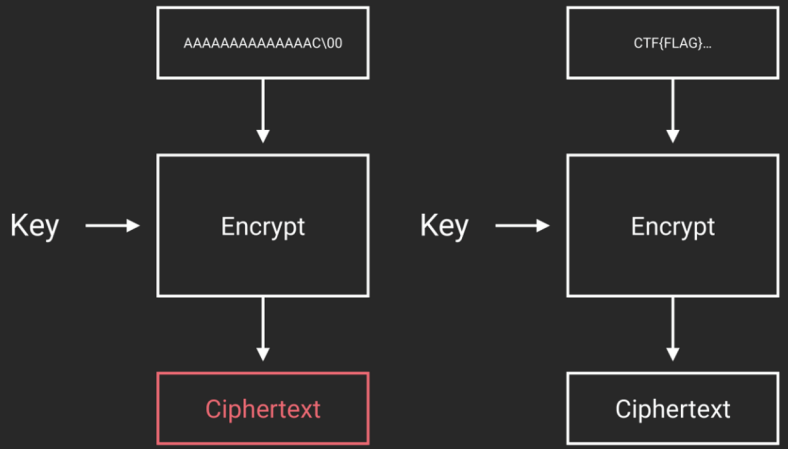
找 flag[1]

先塞 14 個垃圾
讓 flag 的前兩個字元掉進來



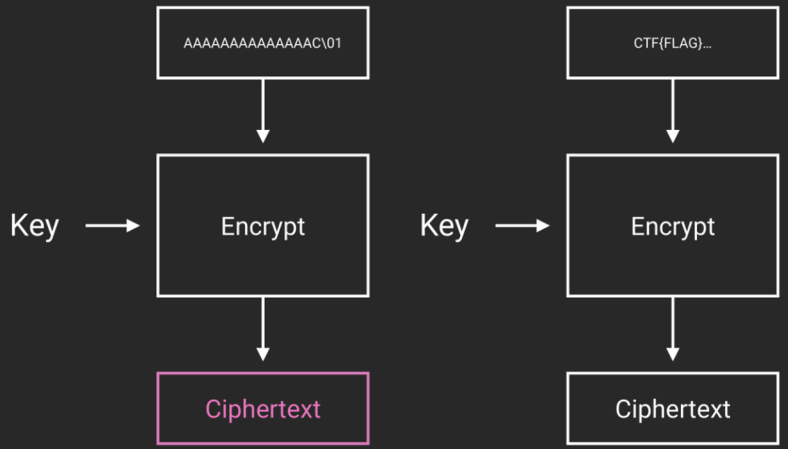
找 flag[1]

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



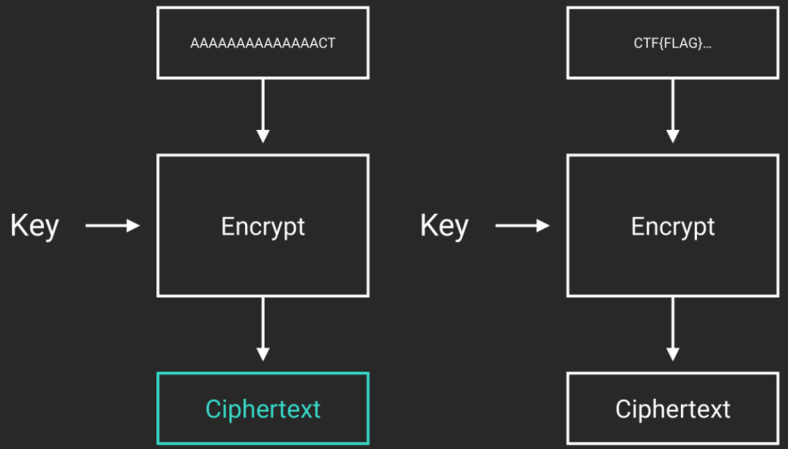
找 flag[1]

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

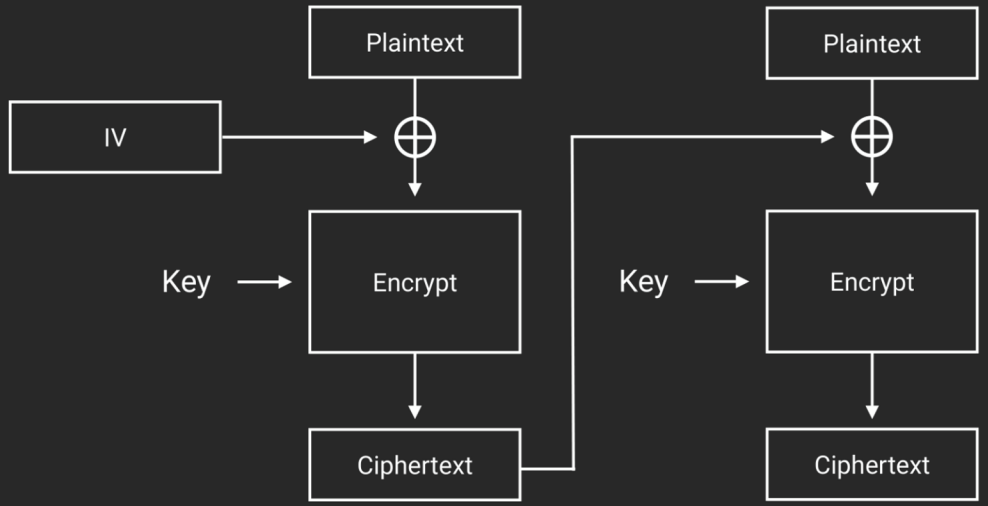
- 這樣一直做下去就能解密出 flag 的所有字元了
- 最多需要做 len(flag) * 257 次

CTF Challenges

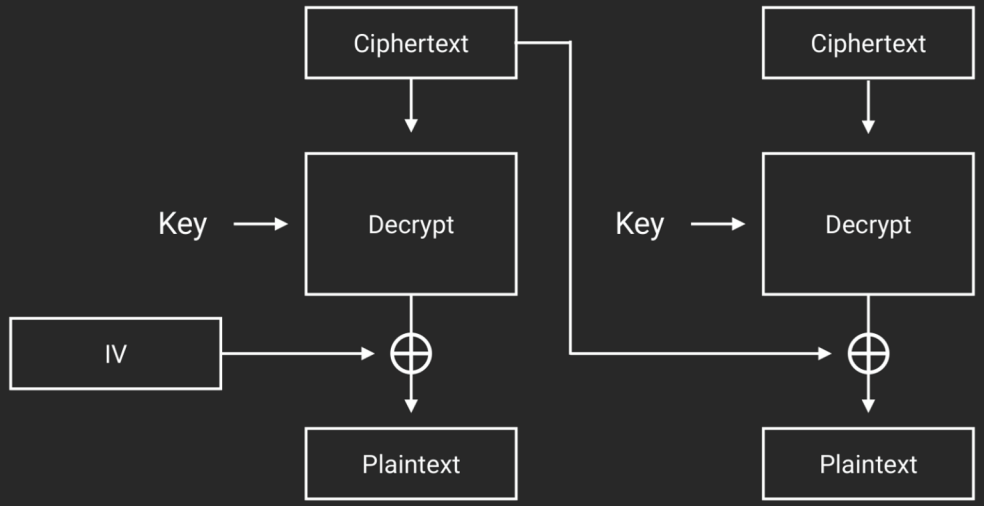
- **UTCTF 2020 - Random ECB**
- **TUCTF 2018 - AESential Lesson**
- **cryptohack.org - ECB Oracle**

CBC Mode

CBC Mode Encryption



CBC Mode Decryption

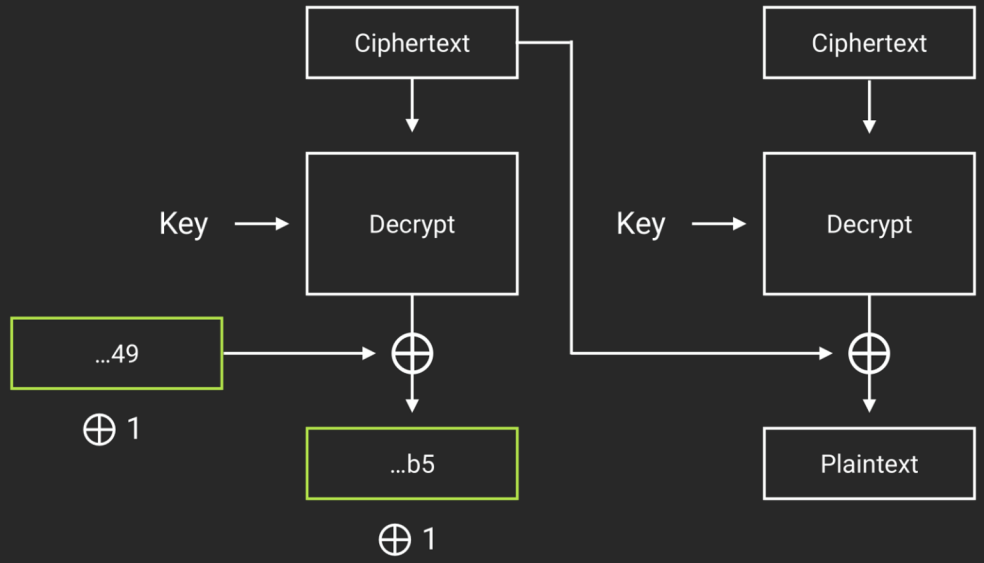


CBC Mode

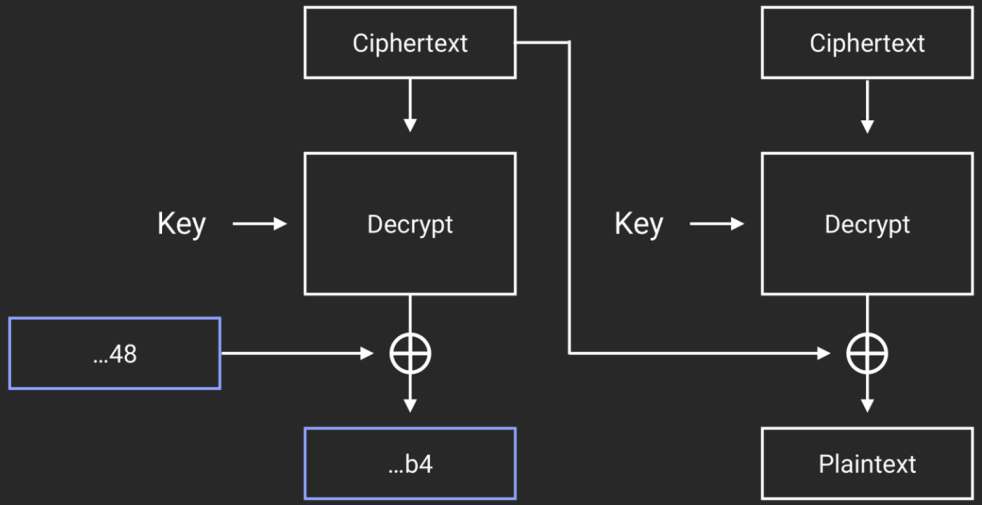
- 每塊密文都依賴前面所有的明文
- 相同的區塊明文會加密出不同的區塊密文
- 有一個初始化向量 (Initial Vector)，簡稱 IV

CBC Mode / Bit-Flipping Attack

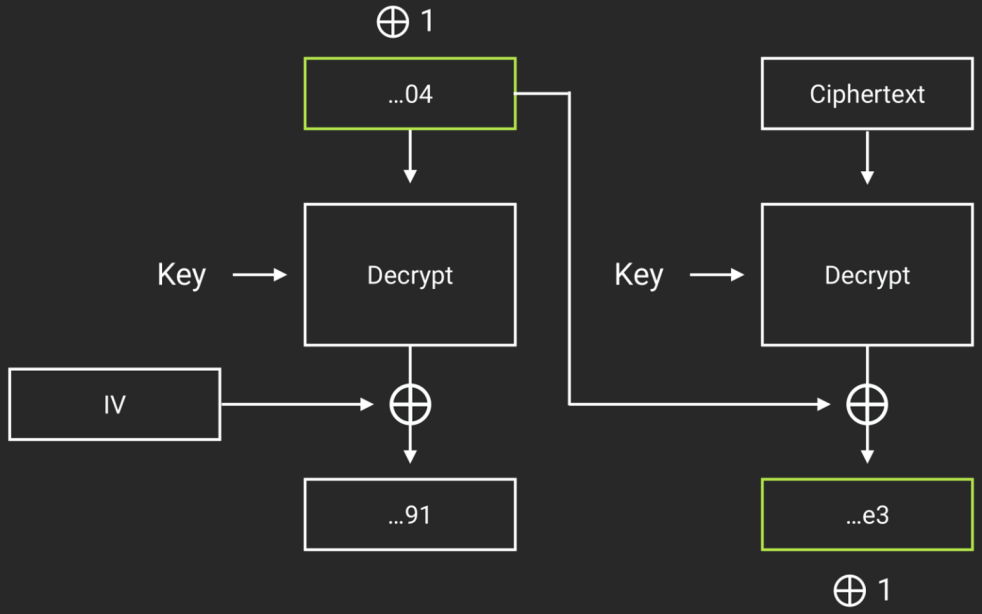
CBC Mode / Bit-Flipping Attack



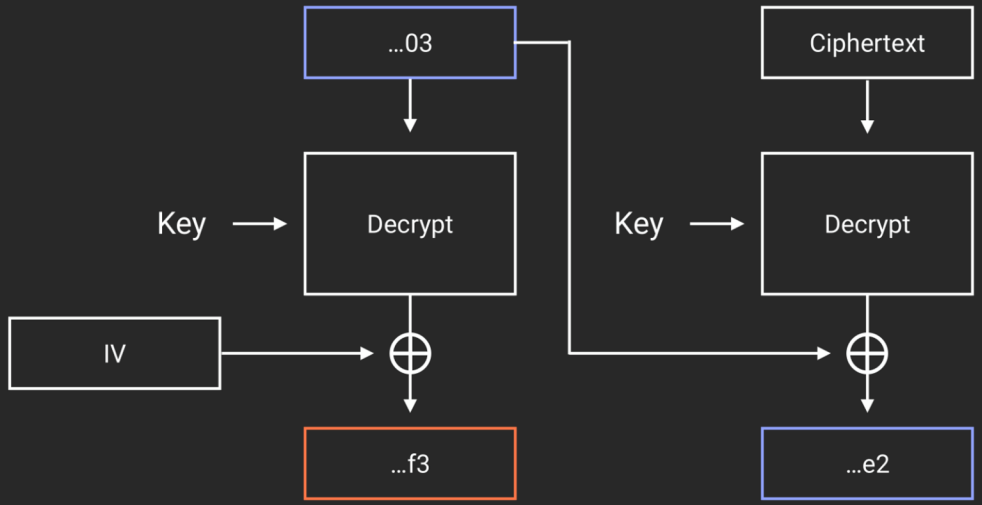
CBC Mode / Bit-Flipping Attack



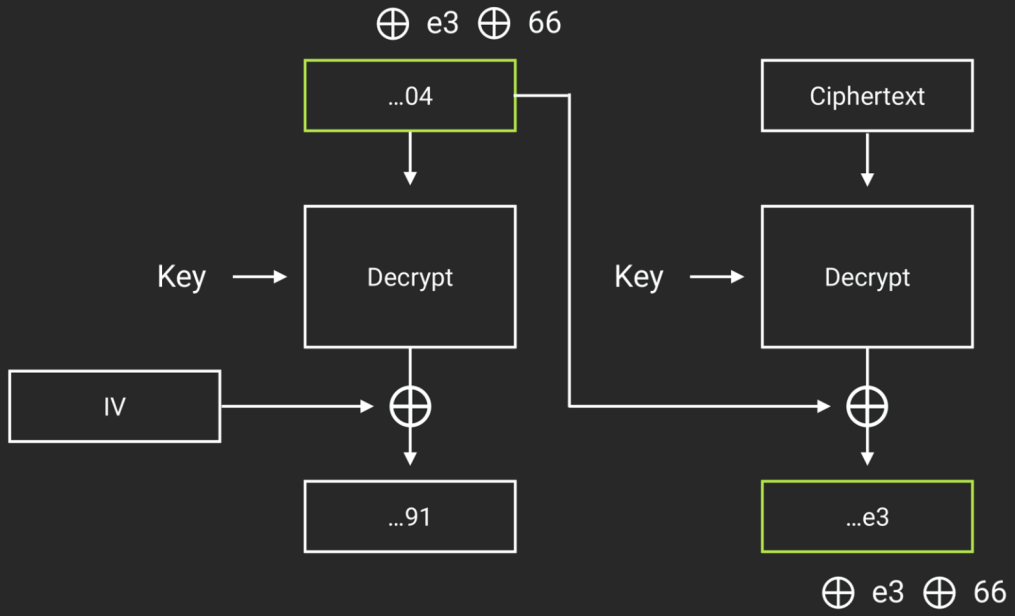
CBC Mode / Bit-Flipping Attack



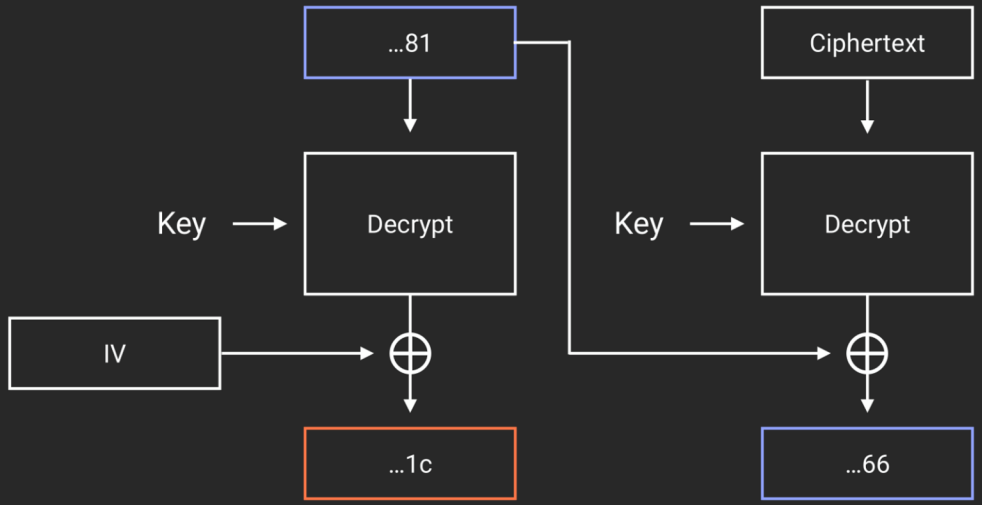
CBC Mode / Bit-Flipping Attack



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

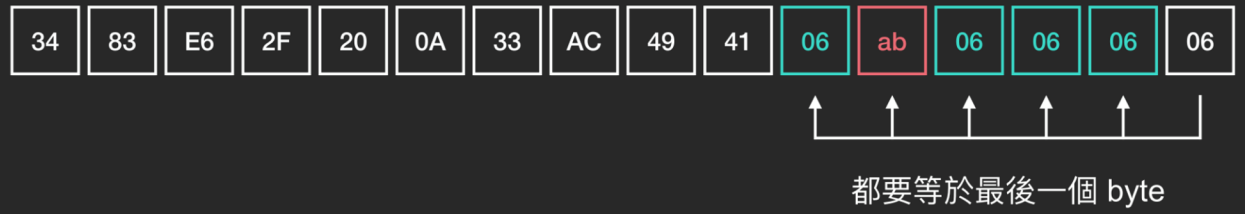
PKCS#7 : Cryptographic Message Syntax

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



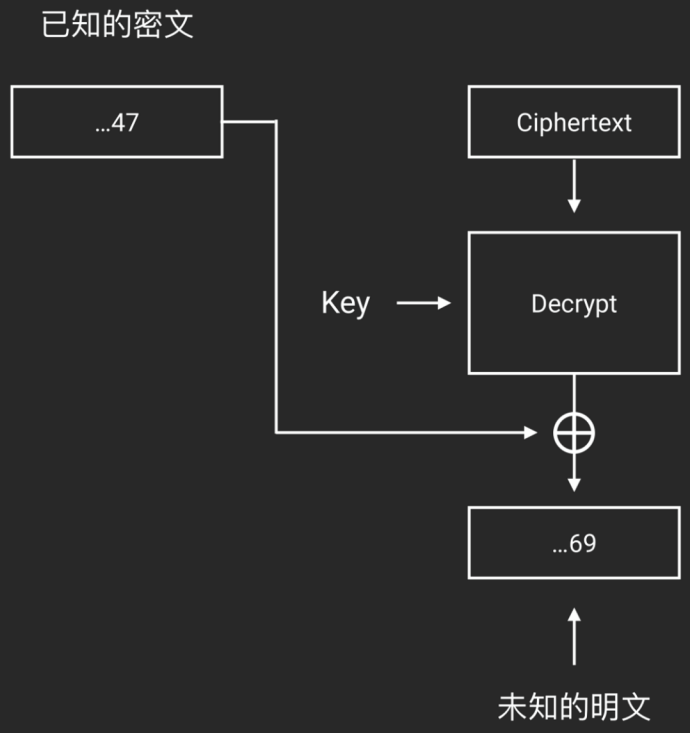
Python Implementation

```python
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]]
```

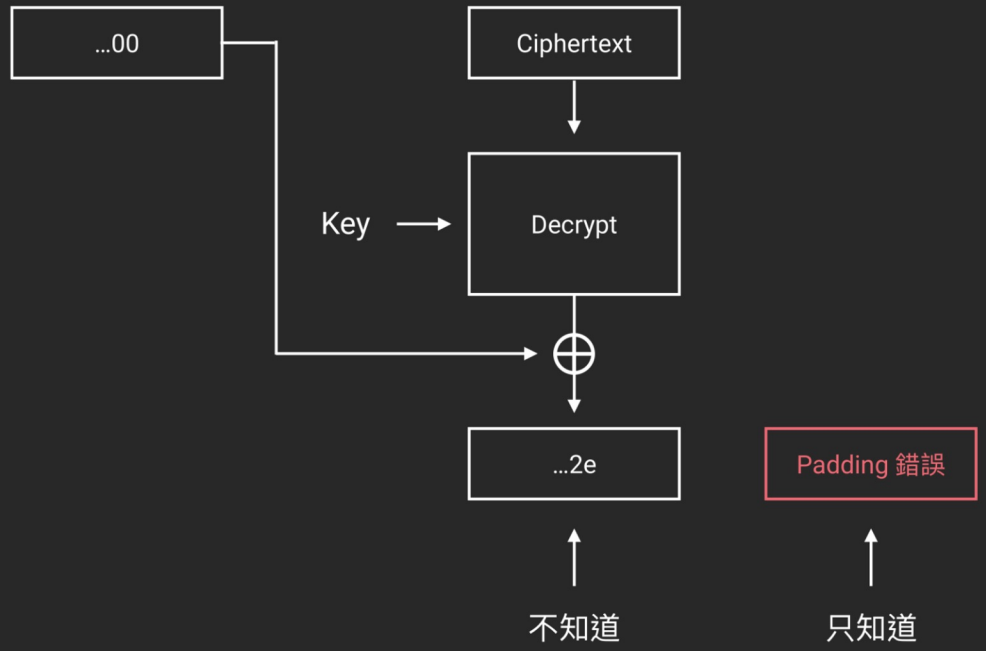
解 plaintext[-1]

Padding Oracle Attack



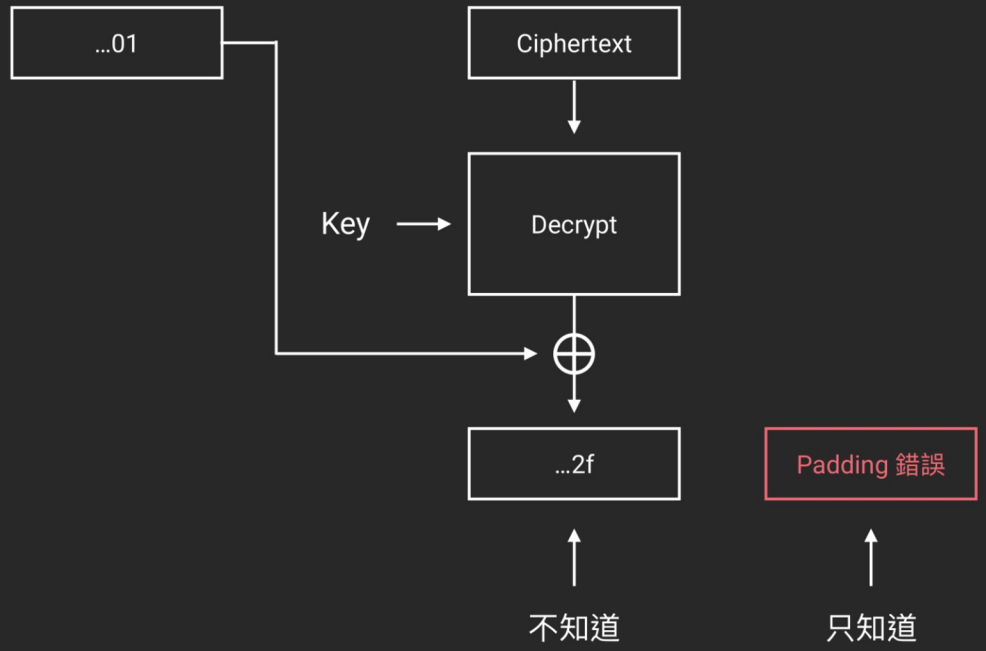
Padding Oracle Attack

暴力嘗試最後一個 byte



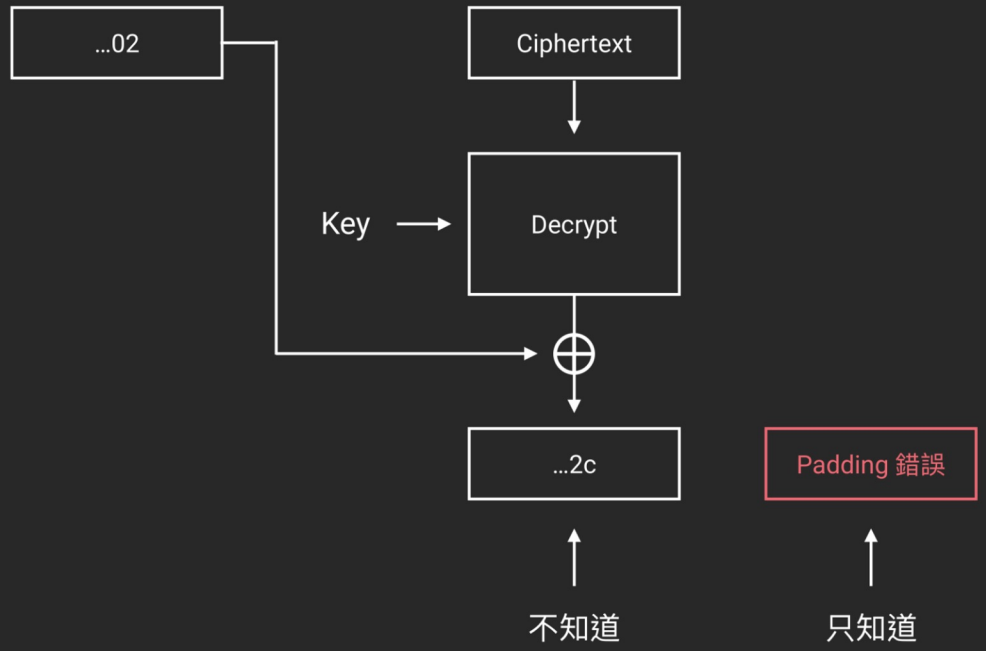
Padding Oracle Attack

暴力嘗試最後一個 byte



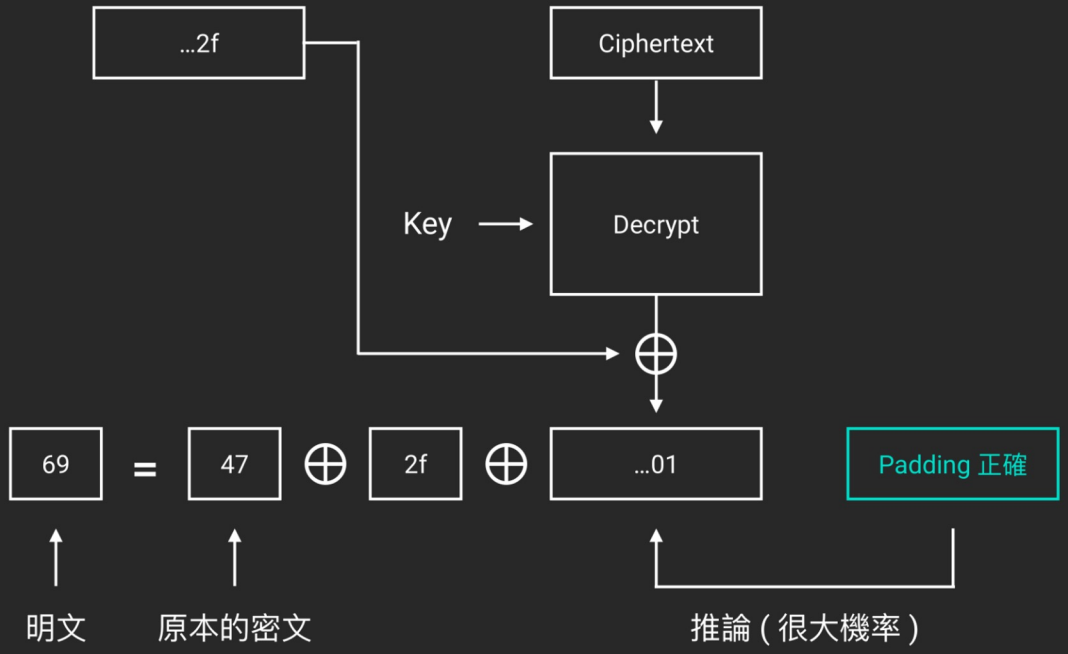
Padding Oracle Attack

暴力嘗試最後一個 byte



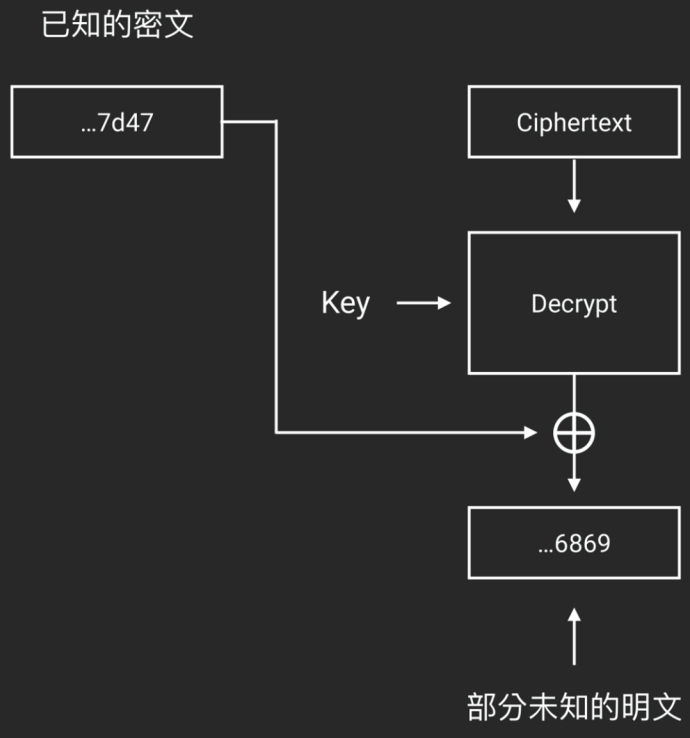
Padding Oracle Attack

暴力嘗試最後一個 byte

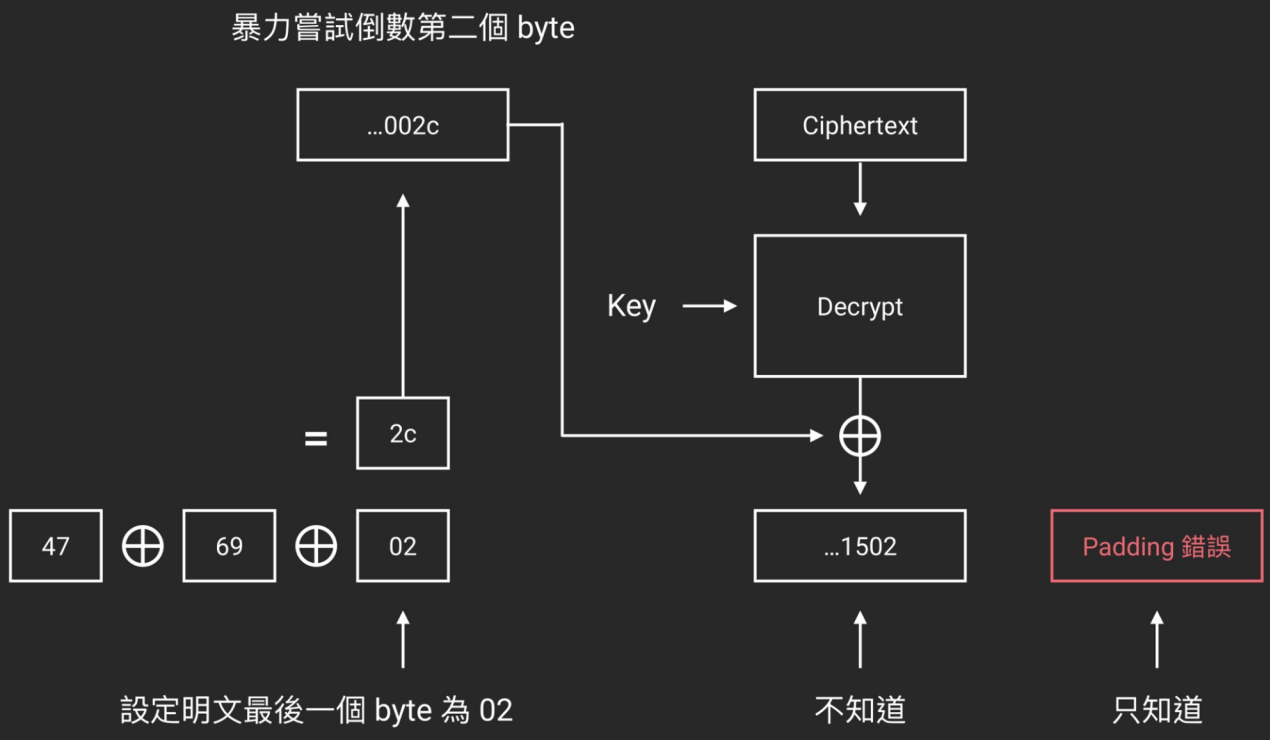


解 plaintext[-2]

Padding Oracle Attack

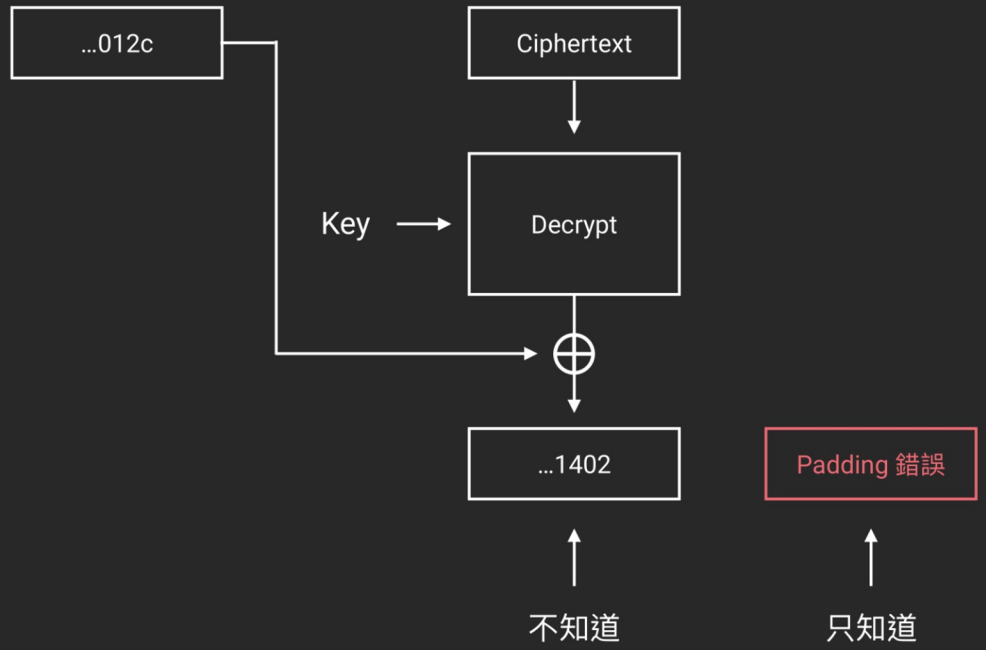


Padding Oracle Attack



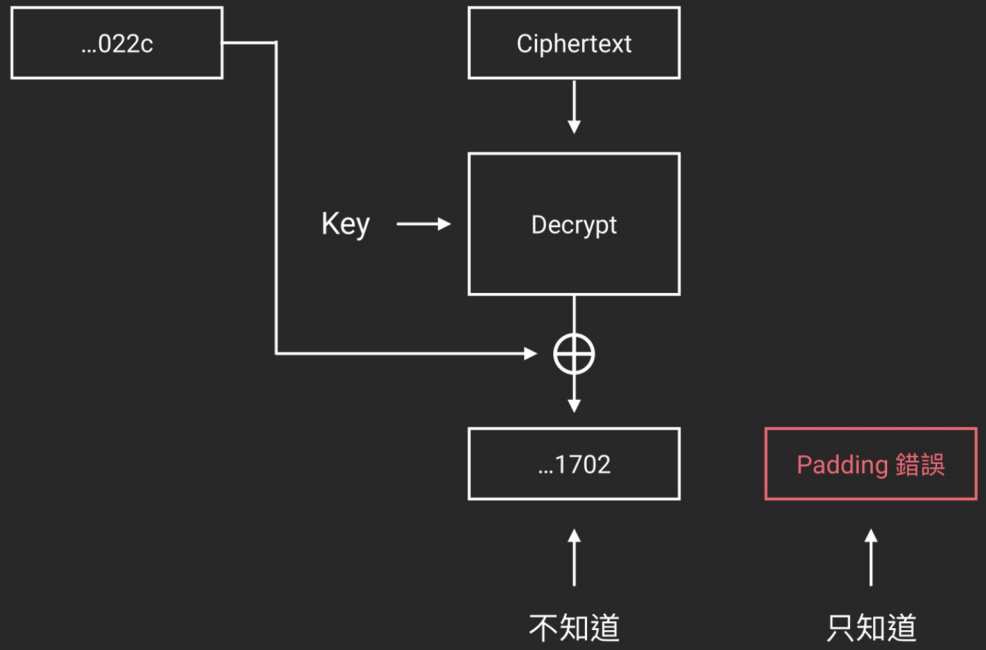
Padding Oracle Attack

暴力嘗試倒數第二個 byte



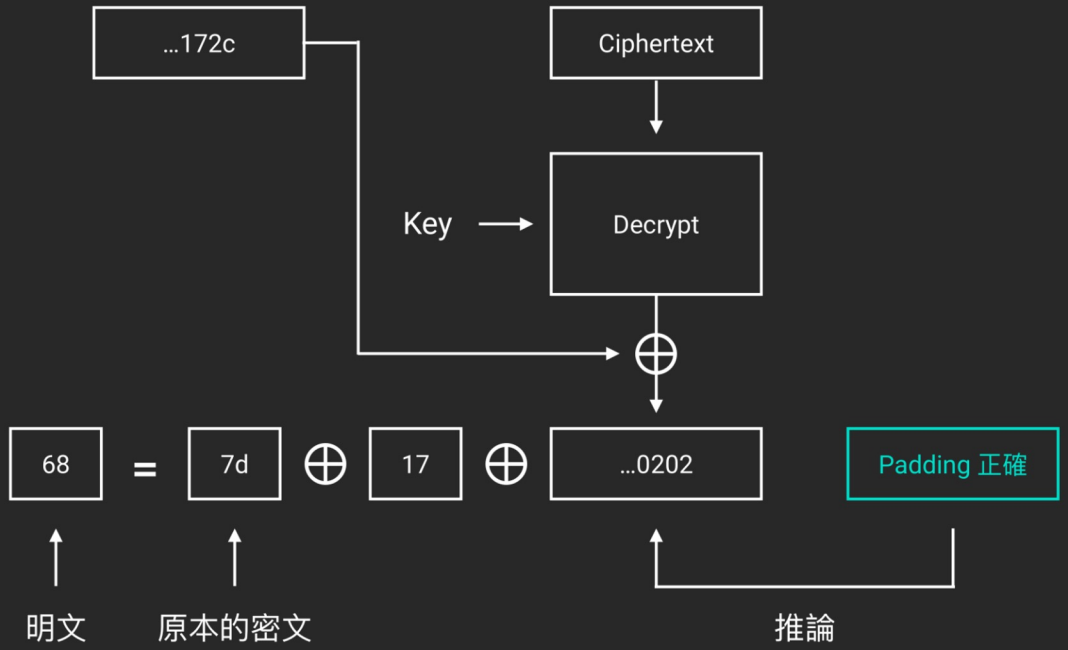
Padding Oracle Attack

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 :

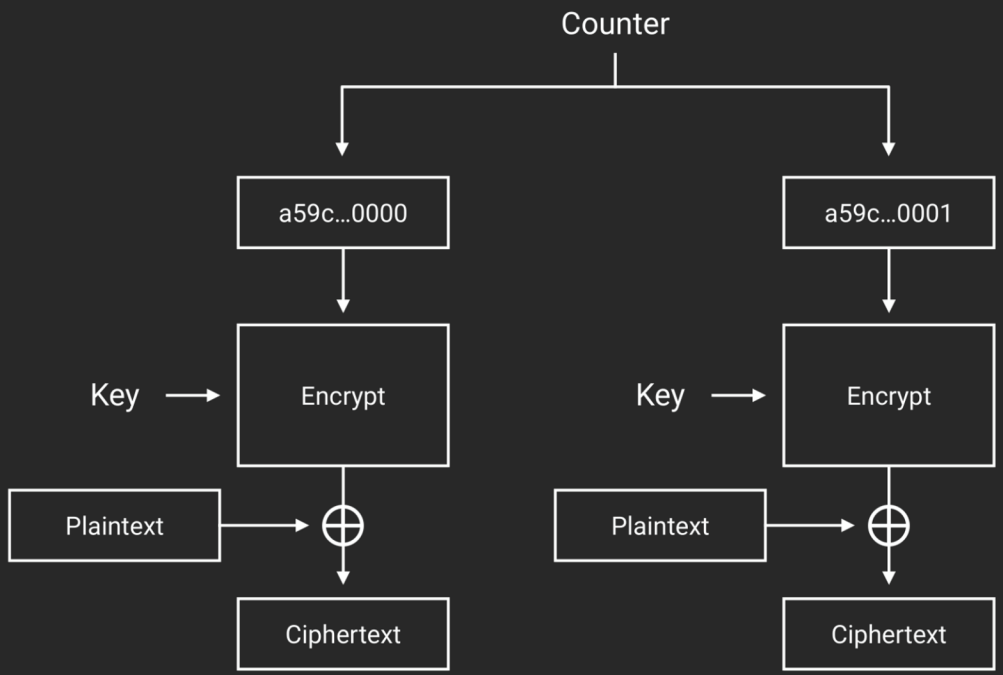
- CSAW CTF 2016 Quals - Neo
- HITCON CTF 2016 Quals - Hackpad
- BAMBOOFOX CTF 2018 - mini-padding

padding 相關攻擊技巧 :

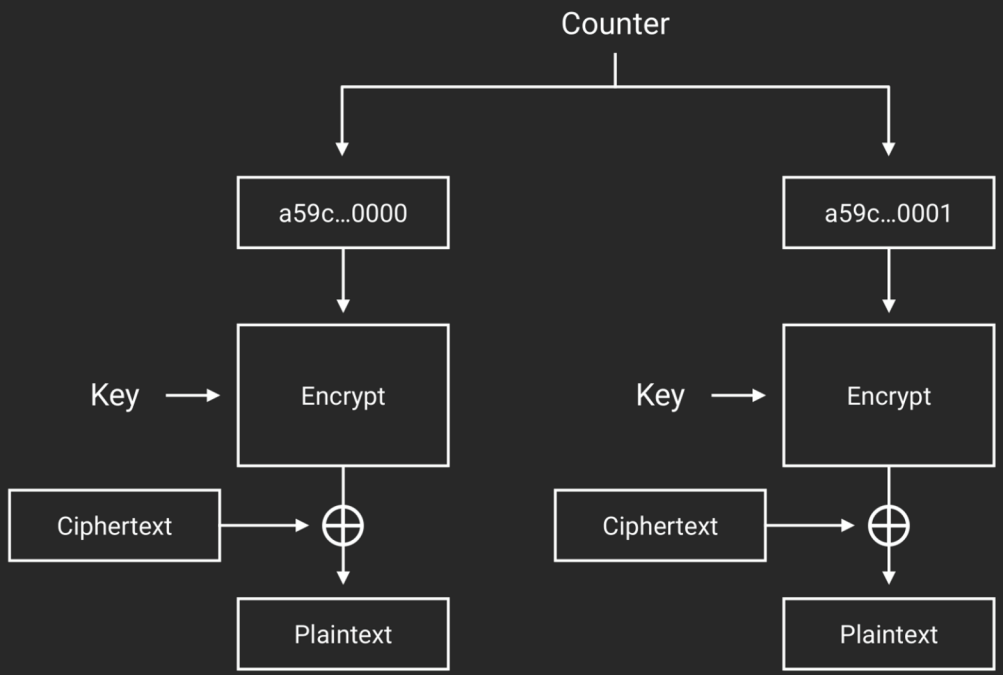
- HITCON CTF 2017 Quals - Secret Server
- HITCON CTF 2017 Quals - Secret Server Revenge
- BAMBOOFOX CTF 2018 - baby-lea-revenge
- BAMBOOFOX CTF 2018 - baby-lea-impossible

CTR Mode

CTR Mode Encryption



CTR Mode Decryption



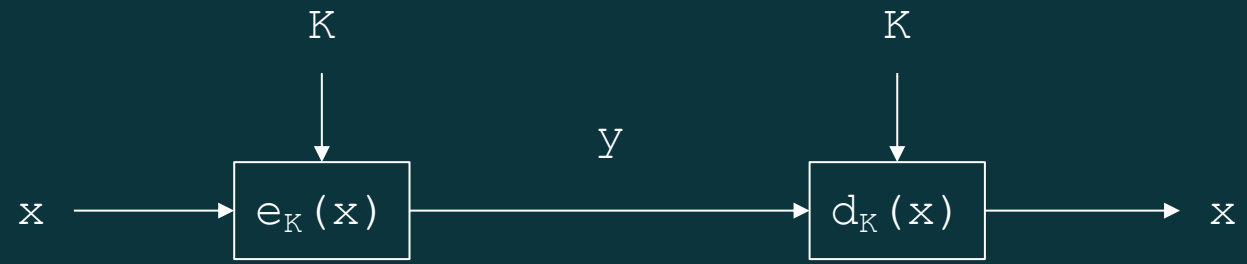
CTR Mode

- 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

Lab 2: Padding Oracle Attack

Asymmetric cryptography

Symmetric Cryptography Revisited

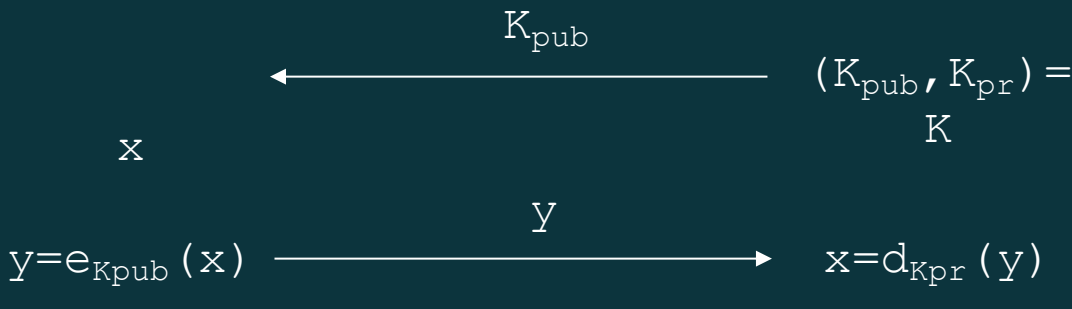


- The same secret key K is used for encryption and decryption
- Encryption and Decryption are very similar (or even identical) functions

Symmetric Cryptography: Shortcomings

- Key distribution problem: The secret key must be **transported securely**
- Number of keys: n users in the network require n(n-1)/2 keys, each user stores (n-1) keys
- Alice or Bob can **cheat each other**, because they have identical keys

Asymmetric (Public-Key) Cryptography



- During the key generation, a key pair K_{pub} and K_{pr} is computed
- Alice encrypts a message with the **not secret** public key K_{pub}
- Only Bob has the **secret** private key K_{pr} to decrypt the message

Basic Key Transport Protocol

Hybrid systems: incorporating asymmetric and symmetric algorithms

- **Key exchange** (for symmetric schemes) are performed with (slow) **asymmetric** algorithms
- **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

How to build Public-Key Algorithms

- Asymmetric schemes are based on a "**one-way function**" f:
  - Computing $y = f(x)$ is computationally easy
  - Computing $x = f^{-1}(y)$ is computationally infeasible
- One-way functions are based on mathematically hard problems. Three main families:
  - **Factoring Integers** (RSA): Given a composite integer n, find its prime factors (Multiply two primes: easy)
  - **Discrete Logarithm** (Diffie-Hellman, Elgamal, DSA): Given a, y and m, find x such that $a^x = y \bmod m$ (Exponentiation a^x : easy)
  - **Elliptic Curves** (ECDH, ECDSA): Generalization of discrete logarithm

Key Lengths and Security Levels

Symmetric	*ECC*	*RSA, DL*	*Remark*
64 Bit	128 Bit	≈ 700 Bit	Only short term security (a few hours or days)
80 Bit	160 Bit	≈ 1024 Bit	Medium security (except attacks from big governmental institutions etc.)
128 Bit	256 Bit	≈ 3072 Bit	Long term security (without quantum computers)

RSA

- Introduction
- How to choose N?
- How to choose e?
- How to choose d?
- Enough? 😱

Key Generation

- Choose 2 large primes p, q, compute
  - $n = pq$
  - $\varphi(n) = (p-1)(q-1)$
  - $a^{\varphi(n)} = 1 \pmod{n}$
- Choose e such that $GCD(e, \varphi(n)) = 1$, compute
  - $d = e^{-1} \bmod \varphi(n)$
- Return $K_{pub} = (e, n)$, $K_{pr} = d$

Encryption & Decryption

- Encryption
  - $c = m^e \pmod n$
- Decryption
  - $m = c^d \pmod n$
- Correctness
  - $m^{\varphi} = (m^{p-1})^{q-1} = 1^{q-1} = 1 \pmod p$ (**Fermat's little theorem**)
  - $m^{\varphi} = (m^{q-1})^{p-1} = 1^{p-1} = 1 \pmod q$
  - $\Rightarrow m^{\varphi} = 1 \pmod n$ (**Chinese remainder theorem**)
  - $c^d = m^{ed} = m^{k\varphi+1} = m \pmod n$

Factorization Algorithm

- General Purpose
  - running time does not depend on the properties of n
  - fastest algorithm has running time of subexponential of logn

- Special Purpose
  - running time depends on the properties of n
  - |p-q| is small ⇒ Fermat's factorization
  - p-1 has small factors ⇒ Pollard's p-1 algorithm
  - p+1 has small factors ⇒ Williams' p+1 algorithm

Fermat's factorization

$$n = pq = (\frac{p+q}{2})^2 - (\frac{p-q}{2})^2$$

- Number of steps:

$$(p+q)/2 - \sqrt{n} = (\sqrt{p} - \sqrt{q})^2/2 = (\sqrt{n} - p)^2/2p$$

```
def fermatFactor(n):
    a = isqrt(n)
    b2 = a * a - n
    while not isqrt(b2)**2 == b2:
        a = a + 1
            b2 = a * a - n
    return a - isqrt(b2), a + isqrt(b2)
```

Pollard's p-1 Algorithm

- p-1 is B-smooth, i.e. p-1's biggest prime factor ≤ B
  - $p - 1 \mid 1 \times 2 \times \ldots \times B$
  - $2^{1 \times 2 \times \ldots \times B} = 2^{k(p-1)} = 1 \pmod{p}$
  - $\text{GCD}(2^{1 \times 2 \times \ldots \times B} - 1, n) > 1$

```
def pollard(n):
    a = 2
    b = 2
    while True:
        a = pow(a, b, n)
        d = gcd(a - 1, n)
        if 1 < d < n: return d
        b += 1
```

Factoring Tools

- http://factordb.com/index.php
- https://github.com/DarkenCode/yafu

How to Choose Public Exponent e

- e too small ⇒ direct e-th root, broadcast attack
- e too big ⇒ slow encryption
- Usually choose prime of form $2^x + 1$, e.g. $2^{16} + 1 = 65537$
  - 16 + 1 calculations in Square and Multiply

```
def Square_and_Multiply(x, y):
    if y == 0: return 1
    k = Square_and_Multiply(x, y // 2) ** 2
    return k * x if y % 2 else k
```

Direct e-th Root

- m, e are small such that $m^e < n$
- Find e-th root of m^e in integral domain
- Requrie random padding on m

Franklin-Reiter related-message attack

- e is small, $m_1 = f(m_2)$ for some linear polynomial $f = ax+b$
  - $c_1 = m_1^e \pmod n$
  - $c_2 = m_2^e = (am_1 + b)^e \pmod n$
- Given (n, e, c_1, c_2, f), attacker can recover m_1, m_2 efficiently
  - m_1 is a root of $g_1(x) = x^e - c_1$
  - m_1 is a root of $g_2(x) = f(x)^e - c_2$
  - $(x - m_1)$ divides both g_1, g_2
  - $\text{GCD}(g_1, g_2) = x - m_1$

Broadcast Attack

- Same message m was encrypted 3 times using the encryption exponent e = 3 but different moduli n_1, n_2, and n_3
  - $m^3 = c_1 \bmod n_1$
  - $m^3 = c_2 \bmod n_2$
  - $m^3 = c_3 \bmod n_3$
  - Using CRT, $m^3 = c \bmod n_1 n_2 n_3$
  - Since $m^3 < n_1 n_2 n_3$, $m^3 = c \Rightarrow$ cube root
- Generally require e different ciphertext to recover m

How to Choose Private Exponent d

- d too small ⇒ Wiener's attack, Boneh-Durfee's attack

Bound for d	Assume Interval for γ	Year
$d < \frac{1}{3}N^{\frac{1}{4}}$	No γ	1990
$d < \frac{1}{8}N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2002
$d < N^{\frac{1-\gamma}{2}}$	$0.25 \leq \gamma < 0.5$	2008
$d < N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2009
$d < \frac{\sqrt{6\sqrt{2}}}{6}N^{\frac{1}{4}}$	No γ	2013
$d < \frac{1}{2}N^{\frac{1}{4}}$	No γ	2015
$d < \frac{\sqrt{3}}{\sqrt{2}}N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2019

Reference: Ariffin, K., Rezal, M., Abubakar, S. I., Yunos, F., and Asbullah, M. A. (2019). New cryptanalytic attack on rsa modulus n = pq using small prime difference method.

Continued Fraction

- $\frac{69}{420} = 0 + \cfrac{1}{6 + \cfrac{1}{11 + \frac{1}{2}}} \rightarrow [0; 6, 11, 2]$
- $\sqrt{19} = 4 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{3 + \ldots}}} = [4; 2, 1, 3, 1, 2, 8, 2, 1, 3, 1, 2, 8, \ldots]$

$e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, \ldots]$

Wiener's Attack

Theorem 1 (Continued-Fractions). *Let a, b, c and d be integers satisfying*

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}, \tag{1}$$

where a/b and c/d are in lowest terms (i.e., $\gcd(a,b) = \gcd(c,d) = 1$). Then c/d is one of the convergents in the continued fraction expansion of a/b. Further, the continued fraction expansion of a/b is finite with the total number of convergents being polynomial in $\log(b)$.

- $ed = 1 + k\phi(N) = 1 + k(N - p - q + 1)$

$$\Rightarrow \frac{e}{N} - \frac{k}{d} = \frac{1}{dN} - \frac{k(p+q-1)}{dN}$$

- $k < d < \frac{1}{3}N^{\frac{1}{4}}, p + q - 1 < 3N^{\frac{1}{2}}$

$$\Rightarrow \left|\frac{e}{n} - \frac{k}{d}\right| < \left|\frac{k(p+q-1)}{dN}\right| < \frac{1}{2d^2}$$

Wiener's Attack (cont.)

- k/d will be one of the convergents in the continued fraction expansion of e/n
- $\varphi = (ed - 1)/k = (p - 1)(q - 1) = n - p - q + 1$
- Solve $x^2 - (n-\varphi+1)x + n = 0$
  - $x = p$ or q

Even if you choose every parameters wisely……

- Common Factor Attack
- Common Modulus Attack
- Chosen Ciphertext Attack
- LSB Oracle

Common Factor Attack

- (e, n_1), (e, n_2) such that $GCD(n_1, n_2) \neq 1$
- Fast pairwise GCD computation
  - https://factorable.net/

Common Modulus Attack

- Same message, same modulus, different public exponent
  - $GCD(e_1, e_2) = 1$
  - $c_1 = m^{e1} \bmod n$
  - $c_2 = m^{e2} \bmod n$
- Bézout's identity
  - Exist a_1, a_2 such that $a_1e_1 + a_2e_2 = GCD(e_1, e_2) = 1$
  - a_1, a_2 can be found by extended Euclidean algorithm
- $c_1^{a_1}c_2^{a_2} = m^{a_1e_1+a_2e_2} = m \pmod n$

Chosen Ciphertext Attack

- Homomorphism
  - $f(x \circ y) = f(x) * f(y)$
- RSA encryption is homomorphic
  - $e(m_1 m_2) = (m_1 m_2)^e = e(m_1) e(m_2)$
- Server can decrypt anything except $c = m^e$
  - $d(2^e c) = 2m$
  - $2^{-1} \cdot 2m = m \pmod n$

LSB Oracle

- Server can decrypt any c, but only return the least significant bit of m

LSB Oracle

- To get first bit (LSB), oracle
  - $c \rightarrow m$
- Inference
  - 

  - $m = 2y_1 + x_0$
  - $r = 2y_1 + x_0 = x_0 \pmod 2$
  - $\Rightarrow x_0 = r$

LSB Oracle (cont.)

- Oracle
  - $(2^{-1})^e c \rightarrow 2^{-1}m$
- Inference
  - 

  - $2^{-1}m = 2y_2 + x_1 + 2^{-1}x_0$
  - $r = [2y_2 + x_1 + 2^{-1}x_0]_{\text{modn}} \pmod 2$
    $= [2^{-1}x_0]_{\text{modn}} + x_1 \pmod 2$
  - $\Rightarrow x_1 = r - [2^{-1}x_0]_{\text{modn}} \pmod 2$

LSB Oracle (cont.)

- Oracle
  - $(2^{-2})^e c \rightarrow 2^{-2}m$
- Inference
  - 

  - $2^{-2}m = 2y_3 + x_2 + 2^{-1}x_1 + 2^{-2}x_0$
  - $r = [2y_3 + x_2 + 2^{-1}x_1 + 2^{-2}x_0]_{modn} \pmod 2$
    $= [2^{-2}x_0 + 2^{-1}x_1]_{modn} + x_2 \pmod 2$
  - $\Rightarrow x_2 = r - [2^{-2}x_0 + 2^{-1}x_1]_{modn} \pmod 2$

LSB Oracle (cont.)

- Can recover one bit every oracle
- Need log(n) oracles totally

Lab 3:
LSB Oracle