Cryptography

Part 2 of 3

By: Killua4564

Index

- Symmetric-key algorithm
  - Block cipher
  - Block mode
- Hash function
- Public-key cryptography
  - RSA

Python tools

- pwntools
- pycrypto / pycryptodome
- boltons
- hashpumpy
- gmpy / gmpy2
- owiener

Python support

bytes	`bytes(4)              # b"\x00\x00\x00\x00"`
	`bytes([4])            # b"\x04"`
	`bytes([4]) * 4        # b"\x04\x04\x04\x04"`
	`bytes([1, 2, 3, 4])   # b"\x01\x02\x03\x04"`

xor	`def xor(x: bytes, y: bytes) -> bytes:` `    return bytes(i ^ j for i, j in zip(x, y))`
chunk	`def chunk(data: bytes, k: int = 16) -> list[bytes]:` `    return [data[i:i+k] for i in range(0, len(data), k)]`
	`def chunk(data: str, k: int = 32) -> list[str]:` `    return [data[i:i+k] for i in range(0, len(data), k)]`
	`from boltons import iterutils` `iterutils.chunked(data, block_size)`

Python support

<table>
<tr><td rowspan="2">pad</td><td><pre>from Crypto.Util.Padding import pad
text = pad(text, block_size)</pre></td></tr>
<tr><td><pre>def pad(text: bytes, block_size: int = 16) -> bytes:
    padding = block_size - (len(text) % block_size)
    return text + bytes([padding]) * padding</pre></td></tr>
<tr><td rowspan="2">unpad</td><td><pre>from Crypto.Util.Padding import unpad
text = unpad(text, block_size)</pre></td></tr>
<tr><td><pre>def unpad(text: bytes, block_size: int = 16) -> bytes:
    padding = text[-1]
    assert 1 <= padding <= block_size
    assert text.endswith(bytes([padding]) * padding)
    return text[:-padding]</pre></td></tr>
</table>

Symmetric-key algorithm

Block cipher

Confusion and Diffusion

- Proposed by Claude Shannon (克勞德·夏農) in 1945
- Confusion (混淆)
  - 明文的每個 bit 都應該被 key 的數個部分影響結果
  - 讓兩者之間有複雜的關聯性
  - e.g. substitution (s-box)、mult
- Diffusion (擴散)
  - 改變明文的一個 bit，一半以上 bits 的密文應該要被改變
  - 改變密文的一個 bit，應該要有接近一半 bits 的明文被改變
  - 被改變 bits 的位置要盡量看起來是隨機的
  - e.g. permutation (p-box)、rotation

Padding

- Bit Padding (擴充成 byte 版本為 ISO/IEC 7816-4)
  - bin(message) || bit(1) || bit(0) * (k - 1)

```
... | 1011 1001 1101 0100 0010 0111 0000 0000 |
```

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 80 00 00 00 |
```

- Byte Padding
  - Zero padding
    - message || byte(0) * k

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 00 |
```

  - ANSI X9.23
    - message || byte(0) * (k - 1) || byte(k)

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |
```

  - ISO 10126
    - message || byte(nonce) * (k - 1) || byte(k)

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 81 A6 23 04 |
```

  - PKCS#5、PKCS#7
    - message || byte(k) * k

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

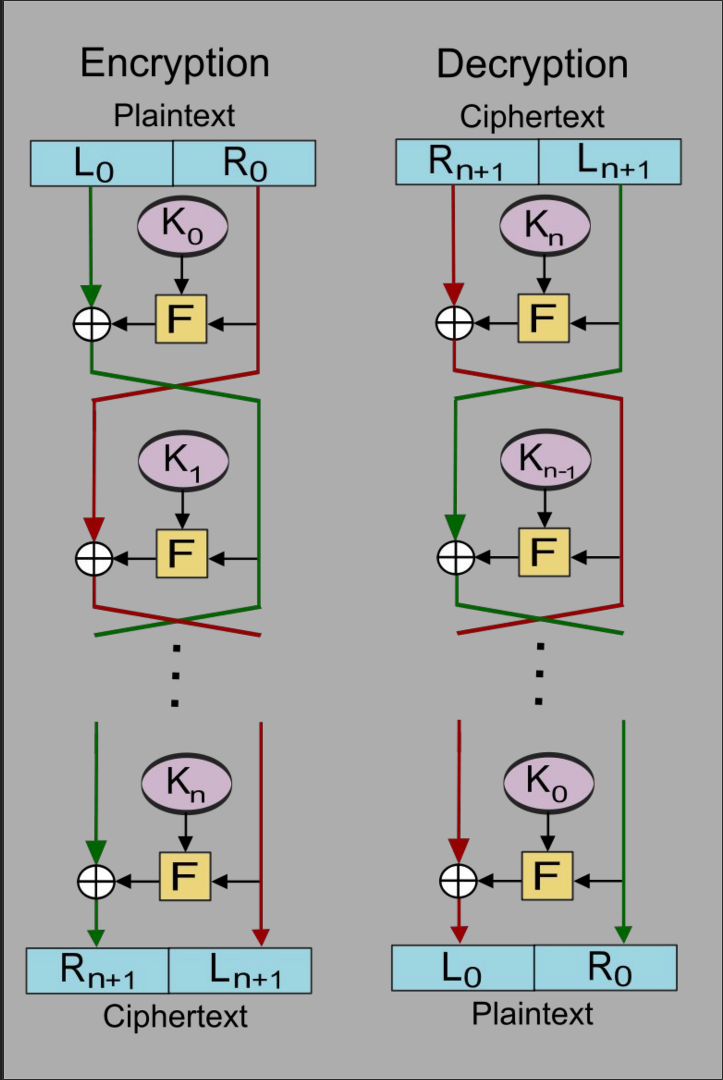
- [Padding the world wonders]

Feistel network

- 又稱 Feistel cipher
- 加密和解密很相似
- F 函式不用是可逆的
- 程式簡單容易寫
- for all i = 1, 2, 3, …, n

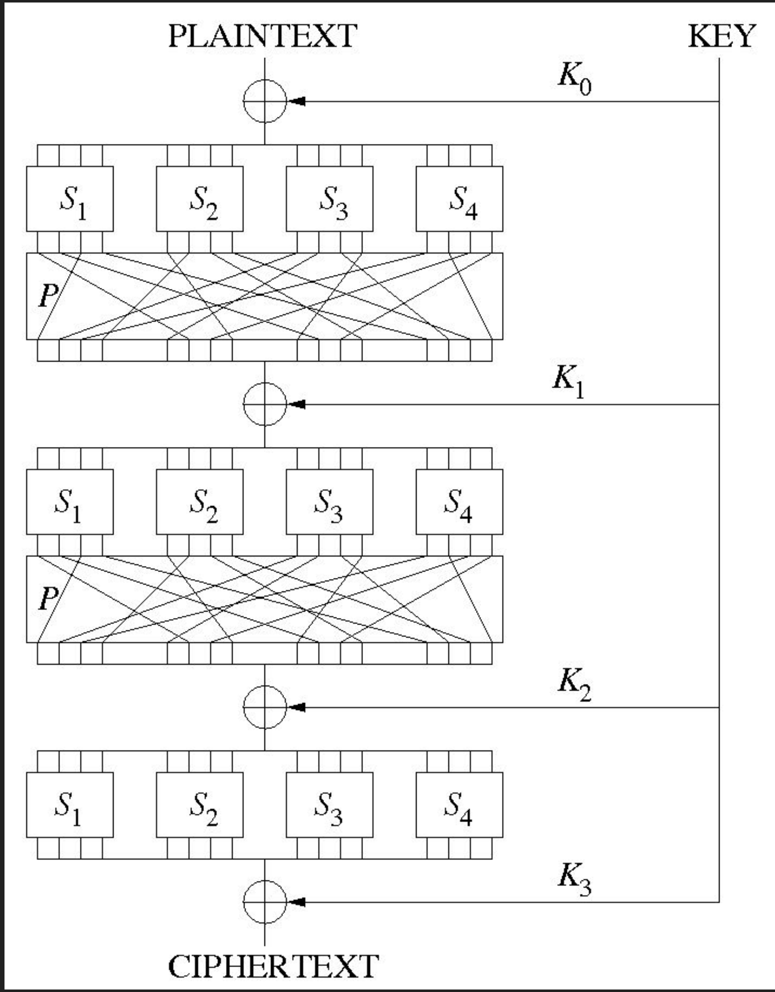
$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus \mathrm{F}(R_i, K_i).$$

- e.g. DES、TEA



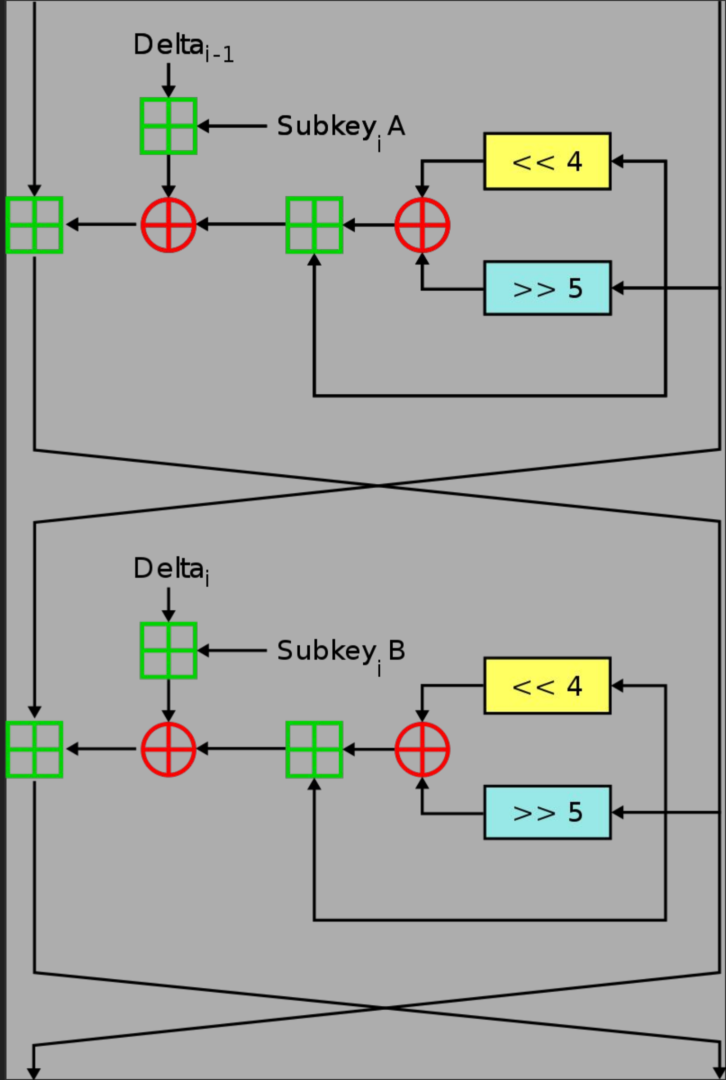
Substitution-Permutation network

- 簡稱 SPN / SP network
- 使用明文塊和金鑰塊產生密文塊
- 結構特性
  - 改變一個 bit
  - 經過 S-box 改變多個 bits
  - 通過 P-box 置換到多個 S-box
  - 再改變更多的 bits …
- e.g. AES、3-Way、SHARK



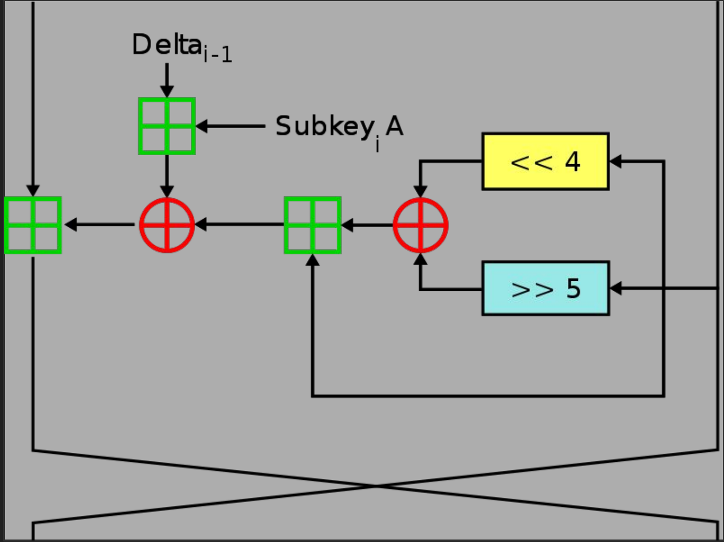
XTEA

- eXtended Tiny Encryption Algorithm
- Feistel network
- key size: 128 bits
- block size: 64 bits



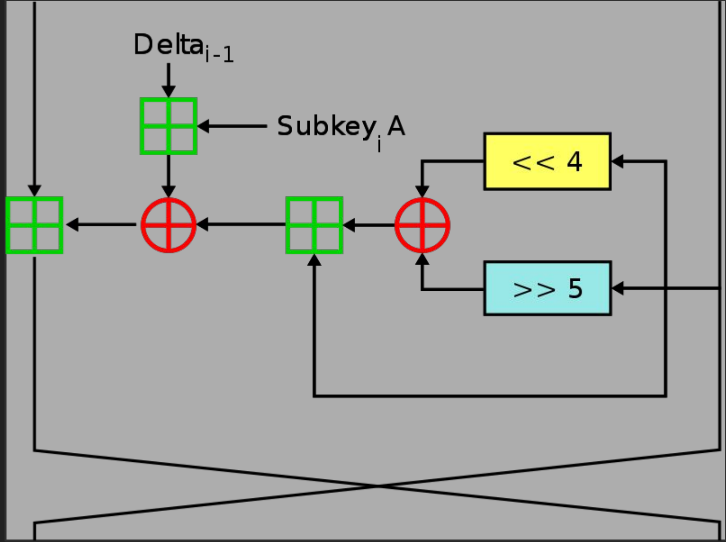
XTEA

- eXtended Tiny Encryption Algorithm
- Feistel network
- key size: 128 bits
- block size: 64 bits




```python
def encrypt(self, plaintext: bytes) -> bytes:
    ciphertext = b""
    plaintext = self.pad(plaintext)
    for i in range(0, len(plaintext), 8):
        v0, v1 = struct.unpack("<2L", bytes(plaintext[i:i+8]))
        _sum, delta, mask = 0, 0x9E3779B9, 0xFFFFFFFF
        for _ in range(self.rounds):
            v0 = (v0 + (((v1 << 4 ^ v1 >> 5) + v1) ^ (_sum + self.key[_sum & 3]))) & mask
            _sum = (_sum + delta) & mask
            v1 = (v1 + (((v0 << 4 ^ v0 >> 5) + v0) ^ (_sum + self.key[_sum >> 11 & 3]))) & mask
        ciphertext += struct.pack("<2L", v0, v1)
    return ciphertext
```

XTEA



- eXtended Tiny Encryption Algorithm
- Feistel network
- key size: 128 bits
- block size: 64 bits

```python
def decrypt(self, ciphertext: bytes) -> bytes:
    plaintext = b""
    for i in range(0, len(ciphertext), 8):
        v0, v1 = struct.unpack("<2L", bytes(ciphertext[i:i+8]))
        _sum, delta, mask = (0x9E3779B9 * self.rounds) & 0xFFFFFFFF, 0x9E3779B9, 0xFFFFFFFF
        for _ in range(self.rounds):
            v1 = (v1 - (((v0 << 4 ^ v0 >> 5) + v0) ^ (_sum + self.key[_sum >> 11 & 3]))) & mask
            _sum = (_sum - delta) & mask
            v0 = (v0 - (((v1 << 4 ^ v1 >> 5) + v1) ^ (_sum + self.key[_sum & 3]))) & mask
        plaintext += struct.pack("<2L", v0, v1)
    plaintext = self.unpad(plaintext)
    return plaintext
```

DES (Data Encryption Standard)

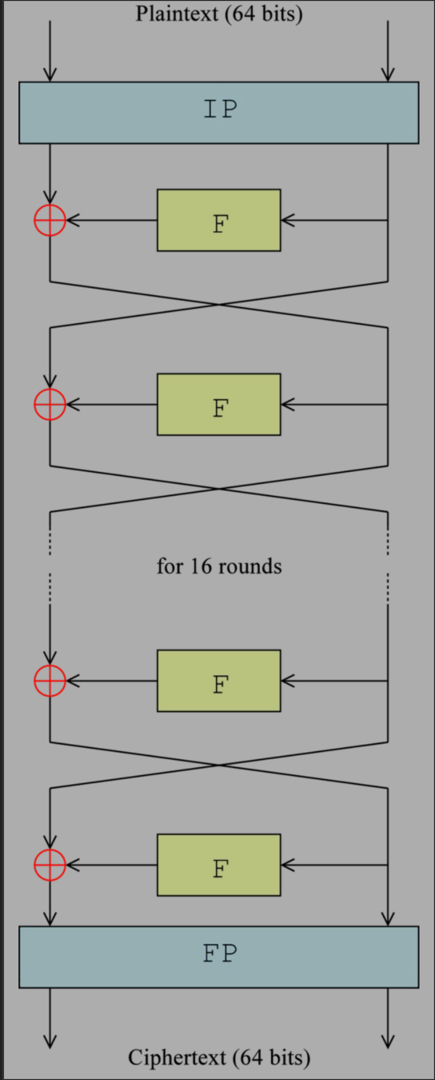
- Feistel network
- key size: 64 bits → 56 bits
- block size: 64 bits

IP

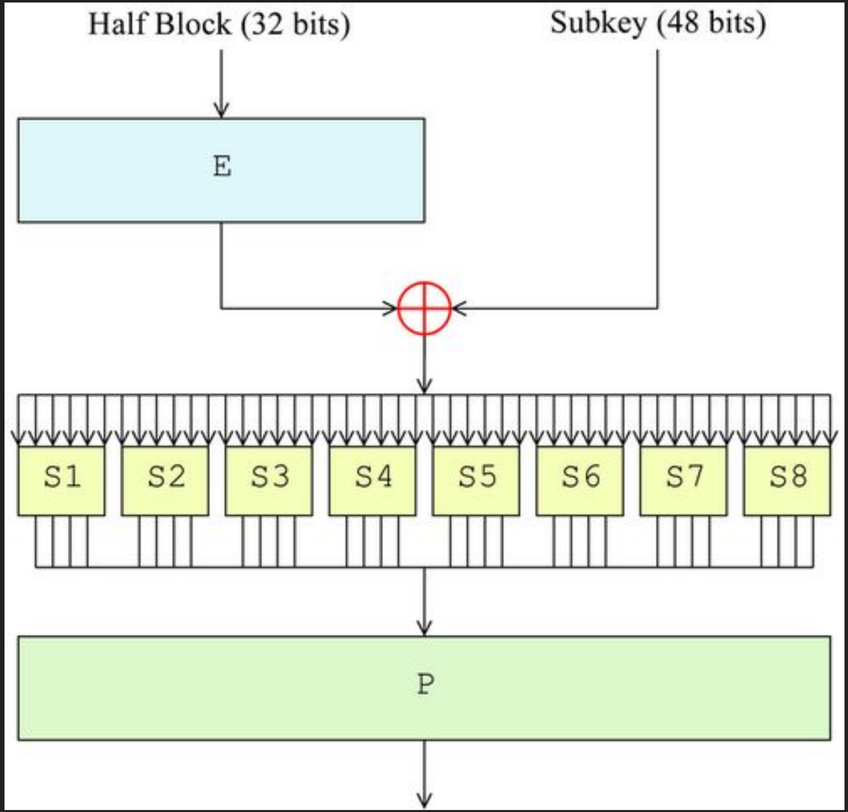
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25



DES (Data Encryption Standard)

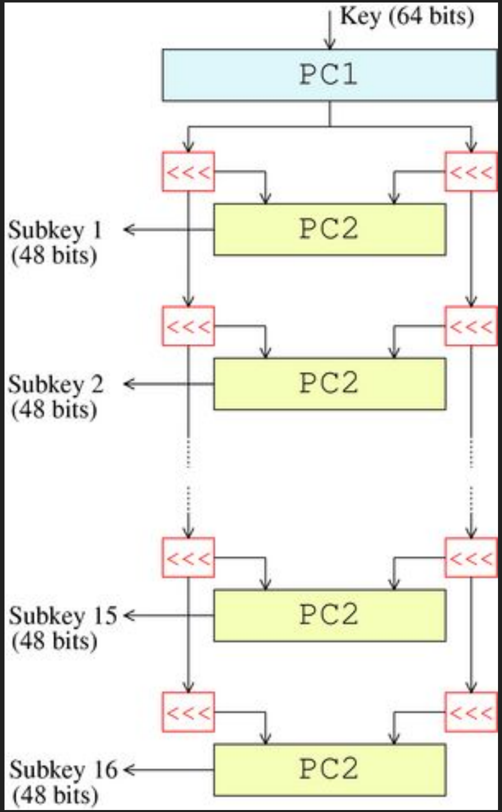


E

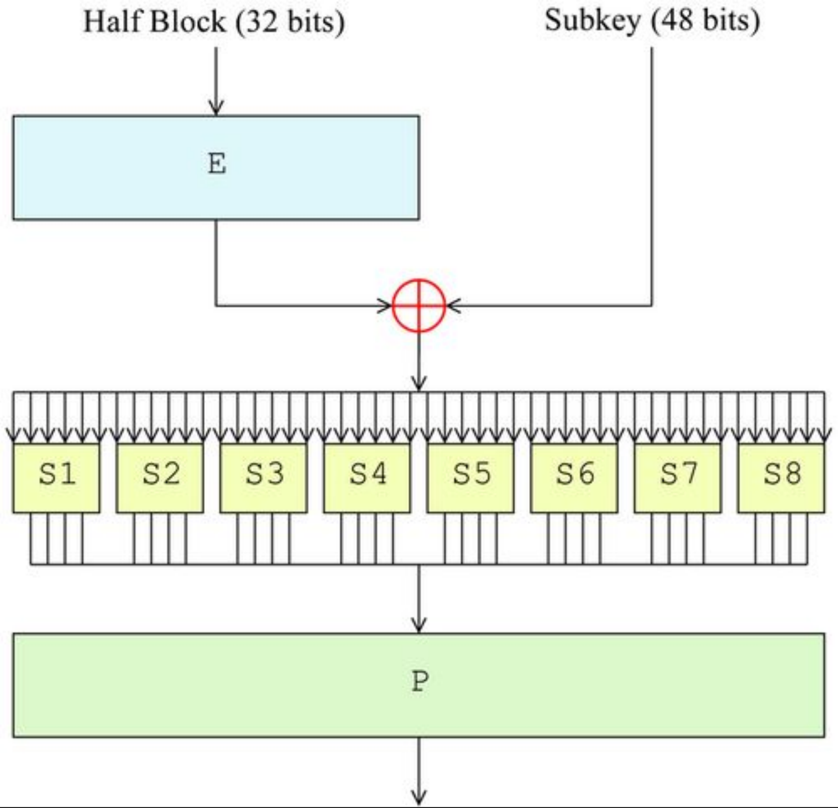
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25



DES (Data Encryption Standard)



S_1

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
0yyyy1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
1yyyy0	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
1yyyy1	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
0yyyy1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
1yyyy0	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1yyyy1	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
0yyyy1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
1yyyy0	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
1yyyy1	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

DES (Data Encryption Standard)

PC–1

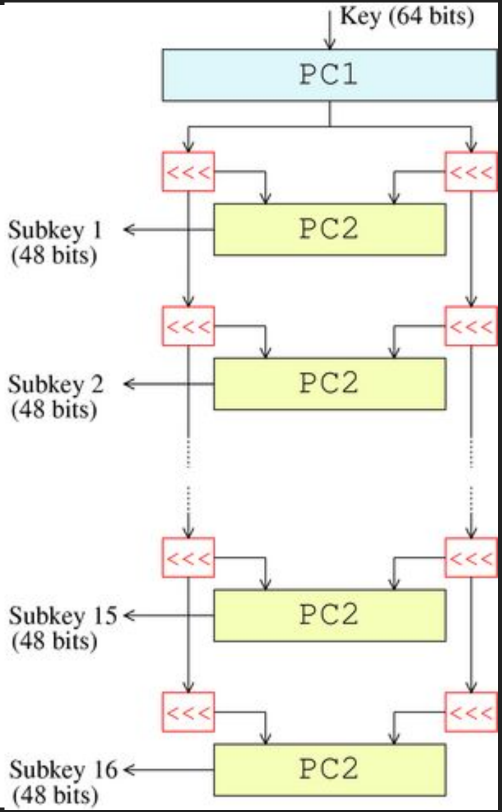
左						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
右						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC–2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

移位

回次	左移位數
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1



DES (Data Encryption Standard)

- Brute force attack
  parity check bits
- 二補數特性
  $E_K(P) = C \Leftrightarrow E_{\overline{K}}(\overline{P}) = \overline{C}$
- 弱金鑰
  $E_K(E_K(P)) = P$
- 半弱金鑰
  $E_{K_1}(E_{K_2}(P)) = P$
- 差分和線性密碼攻擊

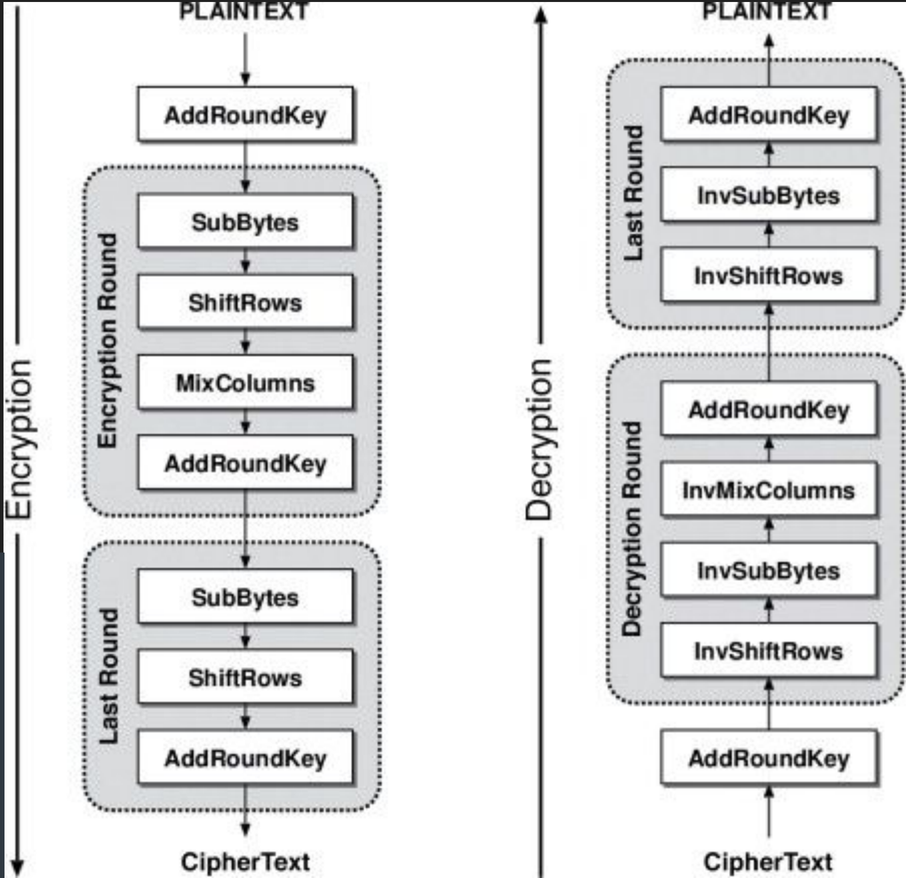
3DES (Triple Data Encryption Algorithm)

- 定義
  - Enc(Dec(Enc(plaintext, key1), key2), key3)
- 3TDEA
  - key1 != key2 != key3
- 2TDEA
  - key1 = key3 != key2

AES (Advanced Encryption Standard)

- Block cipher
- Substitution-Permutation network
- Square State
- key size: 128, 192, 256 bits
- block size: 128 bits
- rounds: 10, 12, 14

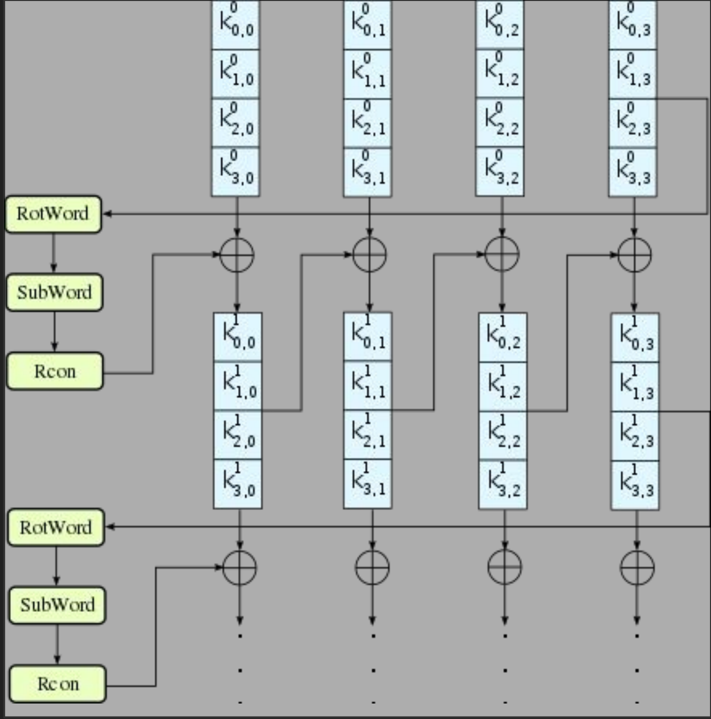
```python
def encrypt(self, plaintext: list[int]) -> list[int]:
    state = list(plaintext)
    state = self._add_round_key(state, self.round_keys[0])
    for round_idx in range(1, 11):
        state = self._sub_bytes(state)
        state = self._shift_rows(state)
        if round_idx < 10:
            state = self._mix_columns(state)
        state = self._add_round_key(state, self.round_keys[round_idx])
    return state
```



AES (Advanced Encryption Standard)

- key schedule

```python
@classmethod
def _rot_bytes(cls, state: list[int]) -> list[int]:
    return state[1:] + [state[0]]

@classmethod
def generate_round_keys(cls, state: list[int]) -> list[int]:
    round_keys: list[list[int]] = [list(state)]
    for round_idx in range(10):
        round_state: list[int] = cls._sub_bytes(cls._rot_bytes(state[-4:]))
        round_state[0] = round_state[0] ^ cls.rcon[round_idx]
        for idx in range(16):
            state[idx] = state[idx] ^ round_state[idx % 4]
            if idx % 4 == 3:
                round_state = state[idx-3:idx+1]
        round_keys.append(list(state))
    return round_keys
```



$$rc_i = x^{i-1} \quad \mathrm{GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)}$$

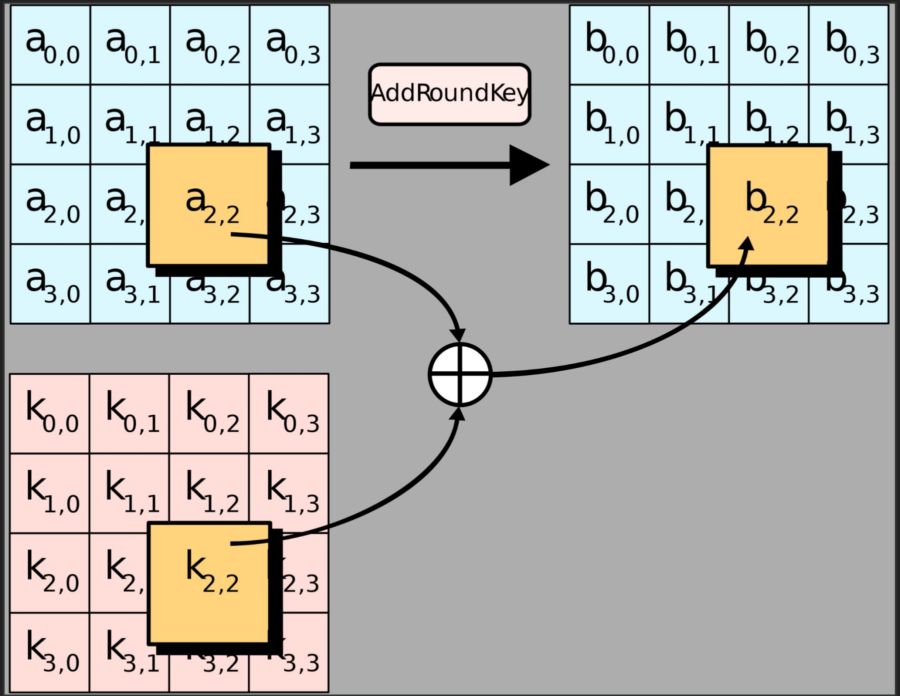
$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11\mathrm{B}_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases}$$

$$rcon_i = [\, rc_i \quad 00_{16} \quad 00_{16} \quad 00_{16} \,]$$

i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1B	36

AES (Advanced Encryption Standard)

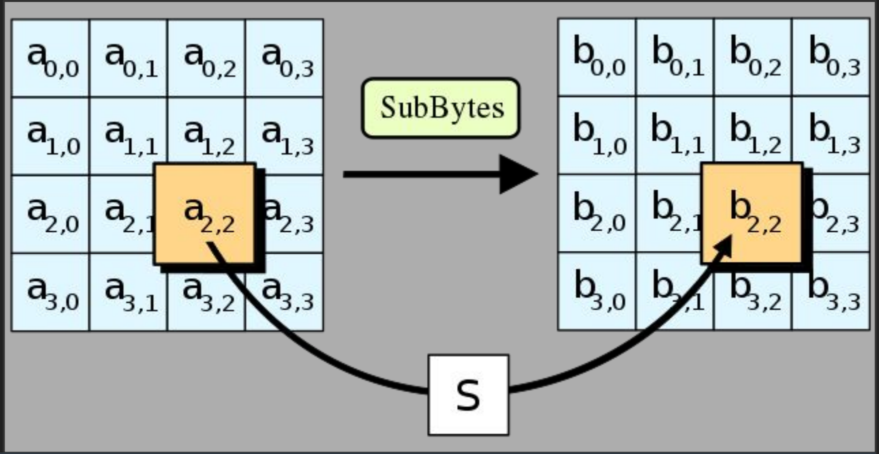
- **AddRoundKey**
- SubBytes
- ShiftRows
- MixColumns




```python
@classmethod
def _add_round_key(cls, state: list[int], round_key: list[int]) -> list[int]:
    for i in range(16):
        state[i] = state[i] ^ round_key[i]
    return state
```

AES (Advanced Encryption Standard)

- AddRoundKey
- SubBytes
- ShiftRows
- MixColumns

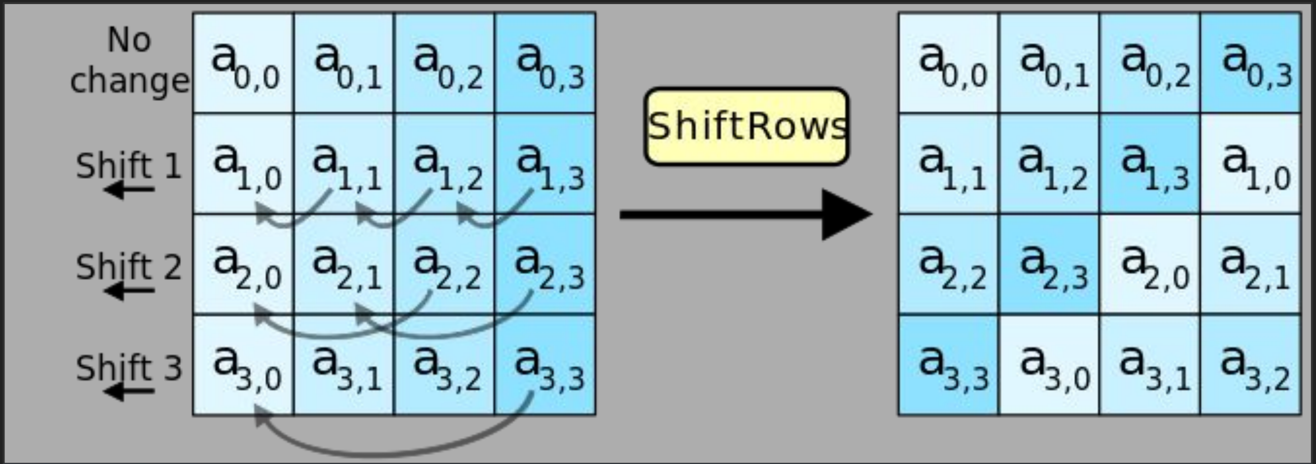



```python
s_box: tuple[int] = (
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16,
)
```

```python
@classmethod
def _sub_bytes(cls, state: list[int]) -> list[int]:
    return [cls.s_box[i] for i in state]
```

AES (Advanced Encryption Standard)

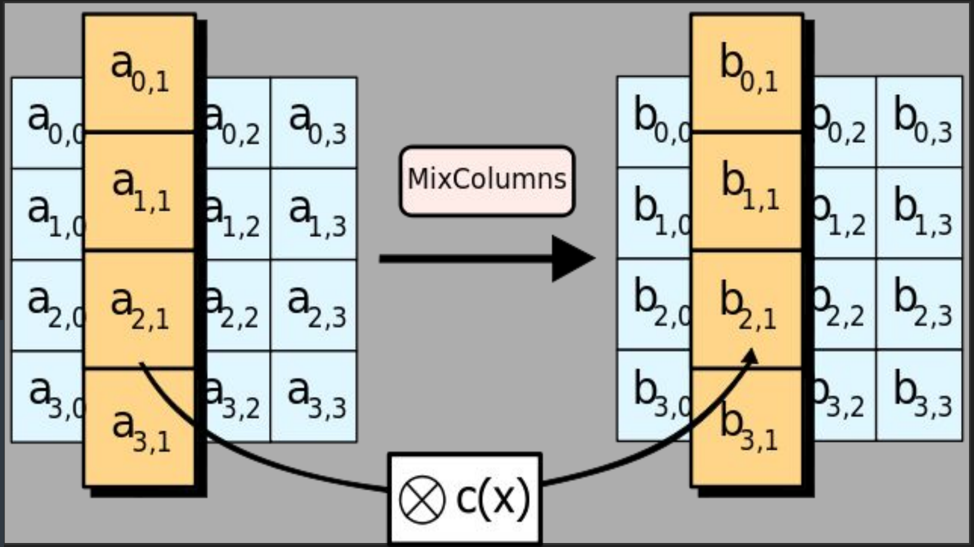
- AddRoundKey
- SubBytes
- ShiftRows
- MixColumns




```python
@classmethod
def _shift_rows(cls, state: list[int]) -> list[int]:
    state[1], state[5], state[9], state[13] = state[5], state[9], state[13], state[1]
    state[2], state[6], state[10], state[14] = state[10], state[14], state[2], state[6]
    state[3], state[7], state[11], state[15] = state[15], state[3], state[7], state[11]
    return state
```

AES (Advanced Encryption Standard)

- AddRoundKey
- SubBytes
- ShiftRows
- MixColumns




```python
@classmethod
def _mix_columns(cls, state: list[int]) -> list[int]:
    return [
        cls.gal2[state[0]] ^ cls.gal3[state[1]] ^ state[2] ^ state[3],
        state[0] ^ cls.gal2[state[1]] ^ cls.gal3[state[2]] ^ state[3],
        state[0] ^ state[1] ^ cls.gal2[state[2]] ^ cls.gal3[state[3]],
        cls.gal3[state[0]] ^ state[1] ^ state[2] ^ cls.gal2[state[3]],

        cls.gal2[state[4]] ^ cls.gal3[state[5]] ^ state[6] ^ state[7],
        state[4] ^ cls.gal2[state[5]] ^ cls.gal3[state[6]] ^ state[7],
        state[4] ^ state[5] ^ cls.gal2[state[6]] ^ cls.gal3[state[7]],
        cls.gal3[state[4]] ^ state[5] ^ state[6] ^ cls.gal2[state[7]],

        cls.gal2[state[8]] ^ cls.gal3[state[9]] ^ state[10] ^ state[11],
        state[8] ^ cls.gal2[state[9]] ^ cls.gal3[state[10]] ^ state[11],
        state[8] ^ state[9] ^ cls.gal2[state[10]] ^ cls.gal3[state[11]],
        cls.gal3[state[8]] ^ state[9] ^ state[10] ^ cls.gal2[state[11]],

        cls.gal2[state[12]] ^ cls.gal3[state[13]] ^ state[14] ^ state[15],
        state[12] ^ cls.gal2[state[13]] ^ cls.gal3[state[14]] ^ state[15],
        state[12] ^ state[13] ^ cls.gal2[state[14]] ^ cls.gal3[state[15]],
        cls.gal3[state[12]] ^ state[13] ^ state[14] ^ cls.gal2[state[15]],
    ]
```

AES (Advanced Encryption Standard)

- AddRoundKey
- SubBytes
- ShiftRows
- MixColumns
  - $c(x) = 3x^3 + x^2 + x + 2$
  - 乘法 $\otimes$ 要 mod $x^4 + 1$
  - 加法 $\oplus$ 是在 GF(2^8) 底下
  - 推導後可簡化成

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

```python
gal2: tuple[int] = (
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1a, 0x1c, 0x1e,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e,
    0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,
    0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7a, 0x7c, 0x7e,
    0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e,
    0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,
    0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde,
    0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe,
    0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d, 0x03, 0x01, 0x07, 0x05,
    0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d, 0x23, 0x21, 0x27, 0x25,
    0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45,
    0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,
    0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d, 0x83, 0x81, 0x87, 0x85,
    0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5,
    0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
    0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed, 0xe3, 0xe1, 0xe7, 0xe5,
)
```

```python
gal3: tuple[int] = (
    0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d, 0x14, 0x17, 0x12, 0x11,
    0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21,
    0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71,
    0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d, 0x44, 0x47, 0x42, 0x41,
    0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1,
    0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1,
    0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd, 0xb4, 0xb7, 0xb2, 0xb1,
    0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81,
    0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a,
    0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6, 0xbf, 0xbc, 0xb9, 0xba,
    0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xe9, 0xea,
    0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda,
    0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46, 0x4f, 0x4c, 0x49, 0x4a,
    0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a,
    0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a,
    0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16, 0x1f, 0x1c, 0x19, 0x1a,
)
```

```python
@classmethod
def _mix_columns(cls, state: list[int]) -> list[int]:
    return [
        cls.gal2[state[0]] ^ cls.gal3[state[1]] ^ state[2] ^ state[3],
        state[0] ^ cls.gal2[state[1]] ^ cls.gal3[state[2]] ^ state[3],
        state[0] ^ state[1] ^ cls.gal2[state[2]] ^ cls.gal3[state[3]],
        cls.gal3[state[0]] ^ state[1] ^ state[2] ^ cls.gal2[state[3]],

        cls.gal2[state[4]] ^ cls.gal3[state[5]] ^ state[6] ^ state[7],
        state[4] ^ cls.gal2[state[5]] ^ cls.gal3[state[6]] ^ state[7],
        state[4] ^ state[5] ^ cls.gal2[state[6]] ^ cls.gal3[state[7]],
        cls.gal3[state[4]] ^ state[5] ^ state[6] ^ cls.gal2[state[7]],

        cls.gal2[state[8]] ^ cls.gal3[state[9]] ^ state[10] ^ state[11],
        state[8] ^ cls.gal2[state[9]] ^ cls.gal3[state[10]] ^ state[11],
        state[8] ^ state[9] ^ cls.gal2[state[10]] ^ cls.gal3[state[11]],
        cls.gal3[state[8]] ^ state[9] ^ state[10] ^ cls.gal2[state[11]],

        cls.gal2[state[12]] ^ cls.gal3[state[13]] ^ state[14] ^ state[15],
        state[12] ^ cls.gal2[state[13]] ^ cls.gal3[state[14]] ^ state[15],
        state[12] ^ state[13] ^ cls.gal2[state[14]] ^ cls.gal3[state[15]],
        cls.gal3[state[12]] ^ state[13] ^ state[14] ^ cls.gal2[state[15]],
    ]
```

AES (Advanced Encryption Standard)

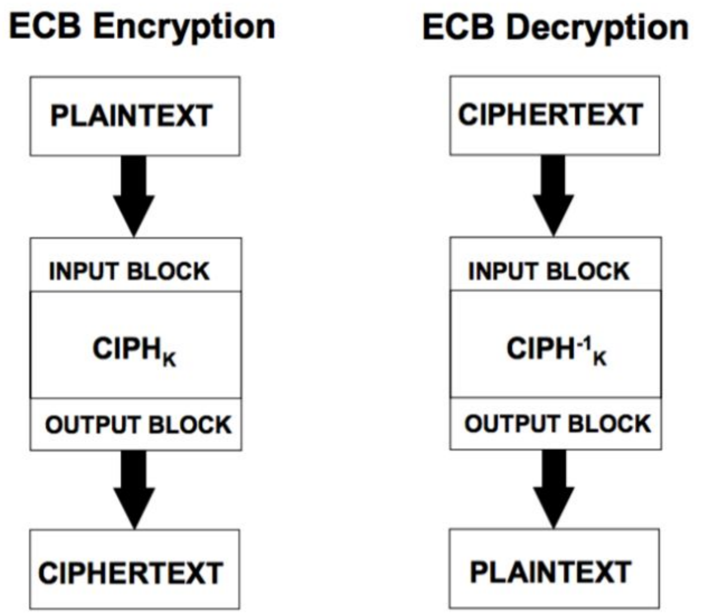
- 旁道攻擊 (side-channel attack)
  - 不是攻擊演算法本身，而是不安全/會洩漏資料的系統
  - 2005 年發表對於 OpenSSL 的伺服器用「時序攻擊法 」進行破解
    - 可是途中需要高達兩億次篩選過的明文去讓它加密
- 關聯密碼攻擊 (related-key attack)
  - 攻擊者可以觀察很多組有關聯的金鑰各自加密的過程和結果
  - 目前最佳可用 2 組金鑰在 $2^{39/45/70}$ 次嘗試內破解 256 bits + 9/10/11 rounds
- 選擇明文攻擊 (chosen-plaintext attack)
  - 攻擊者可以選擇特定關係的明文去讓演算法用相同的金鑰加密
  - 目前最佳可破解 128 bits + 7 rounds 和 192/256 bits + 8 rounds
- 中途相遇攻擊 (meet-in-the-middle attack)
  - Diffie 和 Hellman 提出，利用空間換取時間，讓組合數從相乘變成相加
  - 其用完全二部圖 (Biclique) 的架構，可在 $2^{126.1}$, $2^{189.7}$, $2^{254.4}$ 複雜度內破解

Symmetric-key algorithm

Block mode

ECB mode (Electronic CodeBook)

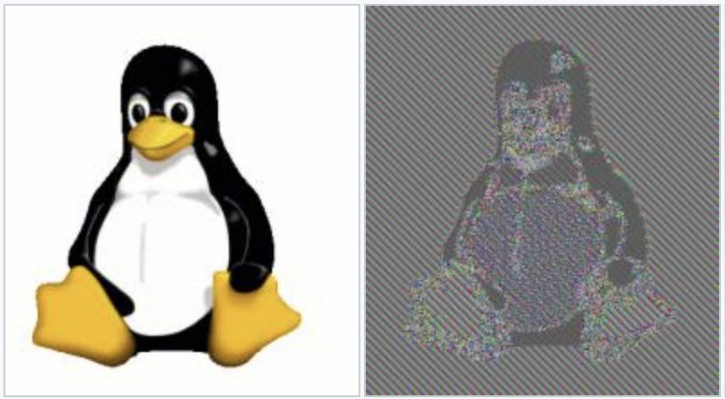
- 電子密碼本模式
- 相同明文 → 相同密文
- 混淆性不夠
  - 可以用複製貼上做攻擊



|usr=a&pw|=a&root=|N........| → |A|B|C|

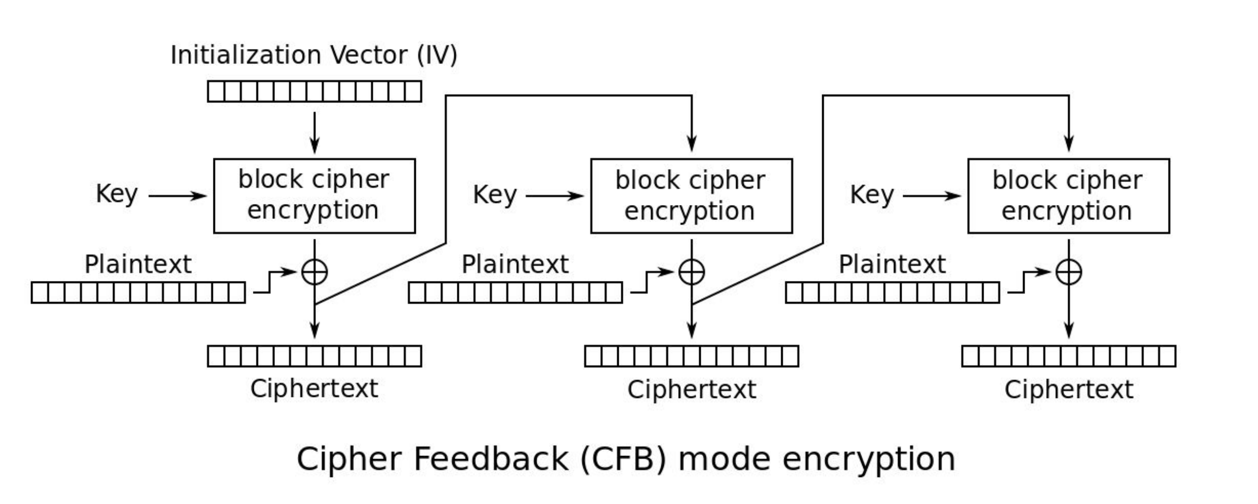
|usr=abcd|Y&pw=aaa|&root=N.| → |D|E|F|

|usr=a&pw|=a&root=|Y&pw=aaa|N........| → |A|B|E|C|



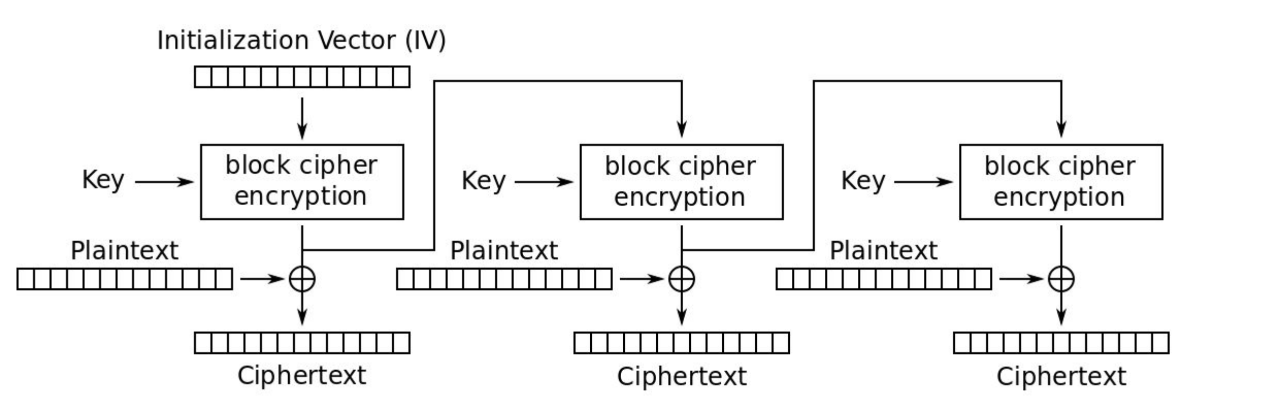
CFB mode (Cipher FeedBack)

- 密文反饋模式
- Self-synchronizing stream cipher
- 可自定義 segment size
- 解密快，可隨機讀取



OFB mode (Output FeedBack)

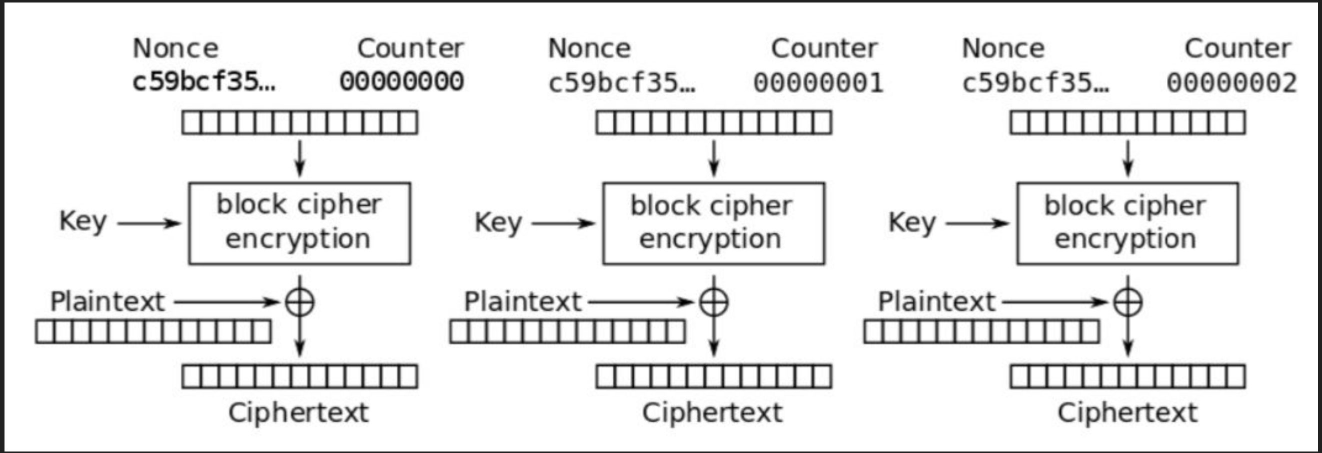
- 輸出反饋模式
- Synchronous stream cipher
- 加解密不可平行，也不可隨機讀取
- 若已知明文限制在一定範圍並有大量的密文，可以實施暴力破解



Output Feedback (OFB) mode encryption

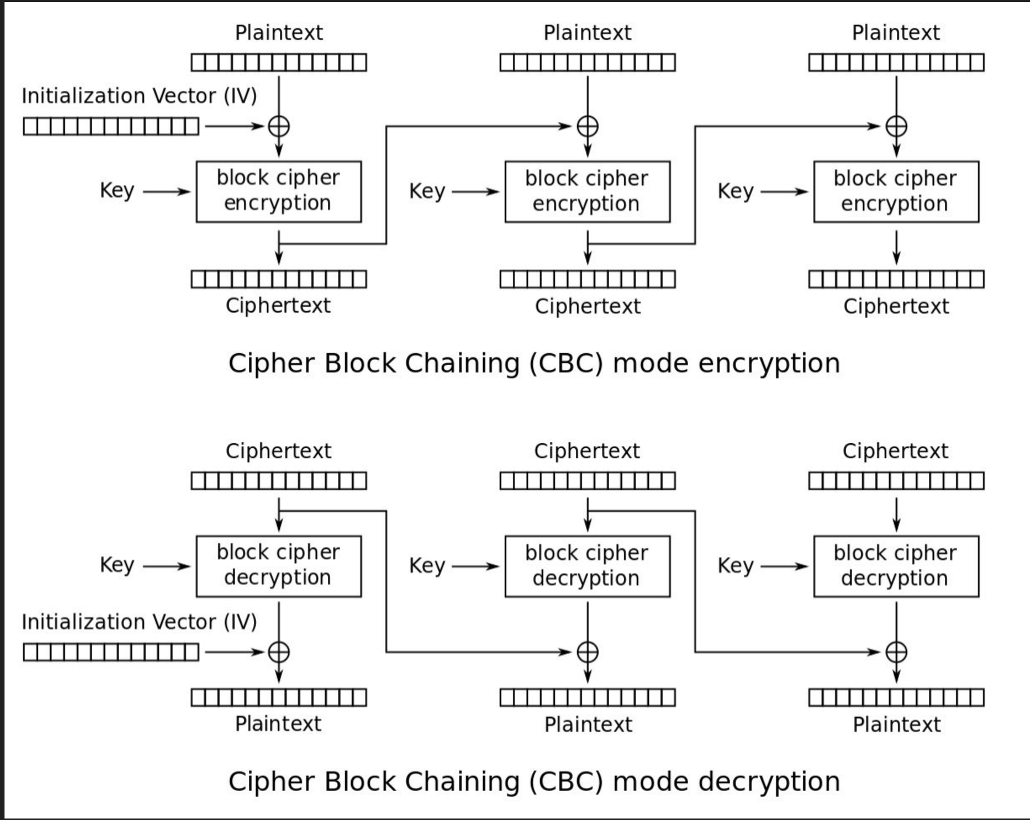
CTR mode (Counter)

- 計數器模式
- 由 Nonce || Counter 來產生隨機的 IV
- 確保每次的 IV 皆不同，以達到最大效能的保密性
- 加解密都可平行，也可以隨機讀取
- 攻擊：位元翻轉攻擊 (Bit-flipping attack)

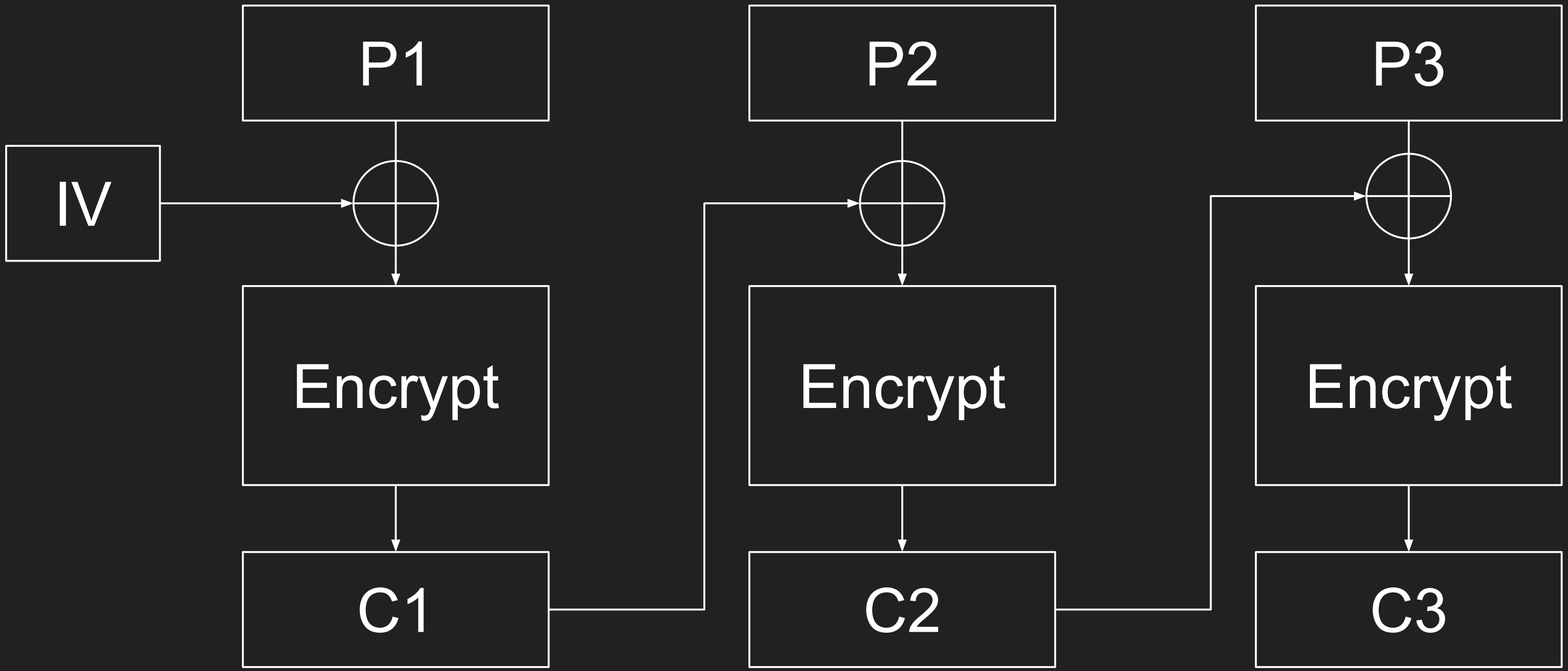


CBC mode (Cipher Block Chaining)

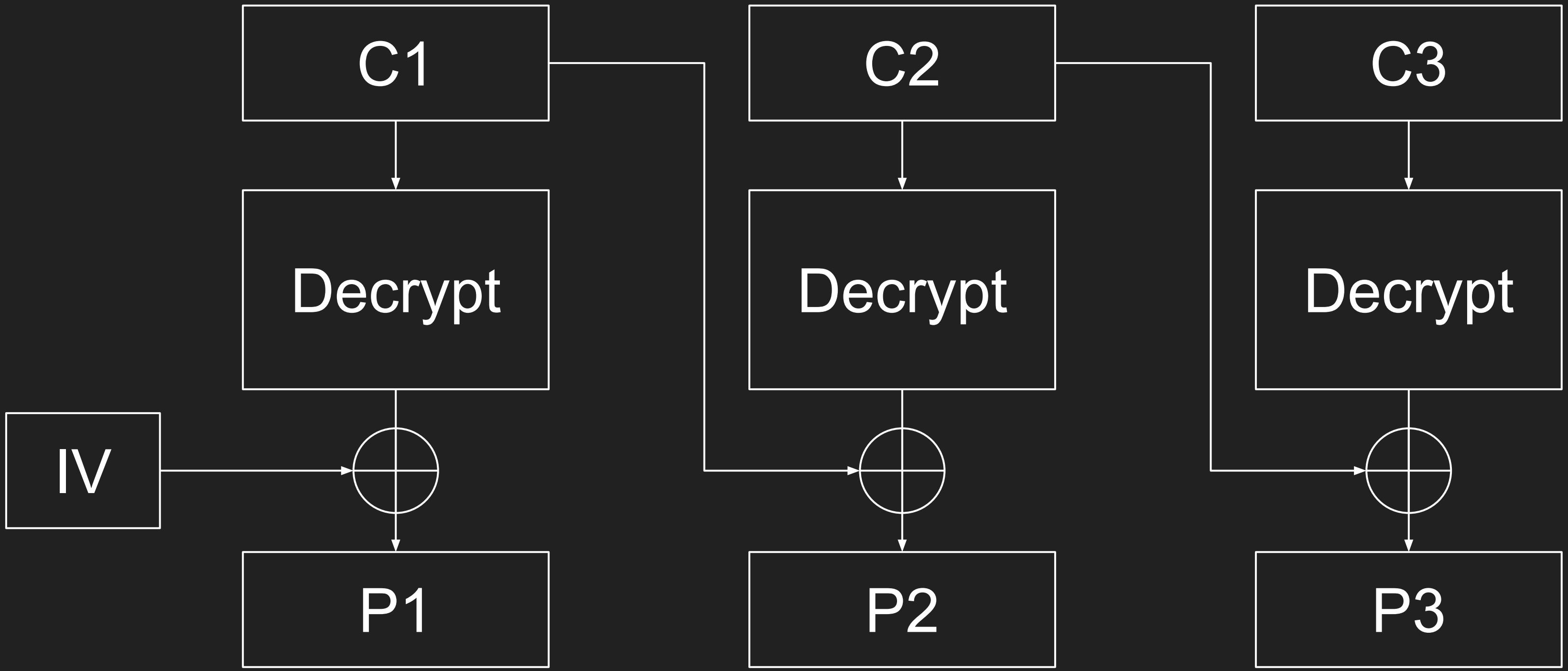
- 密碼區塊鏈模式
- IV: 相同明文 → 不同密文
- 加密慢解密快
- 可隨機讀取
- TLS 最常見的模式
- 攻擊
  - Padding Oracle (POODLE)
  - BEAST Attack
  - EFAIL



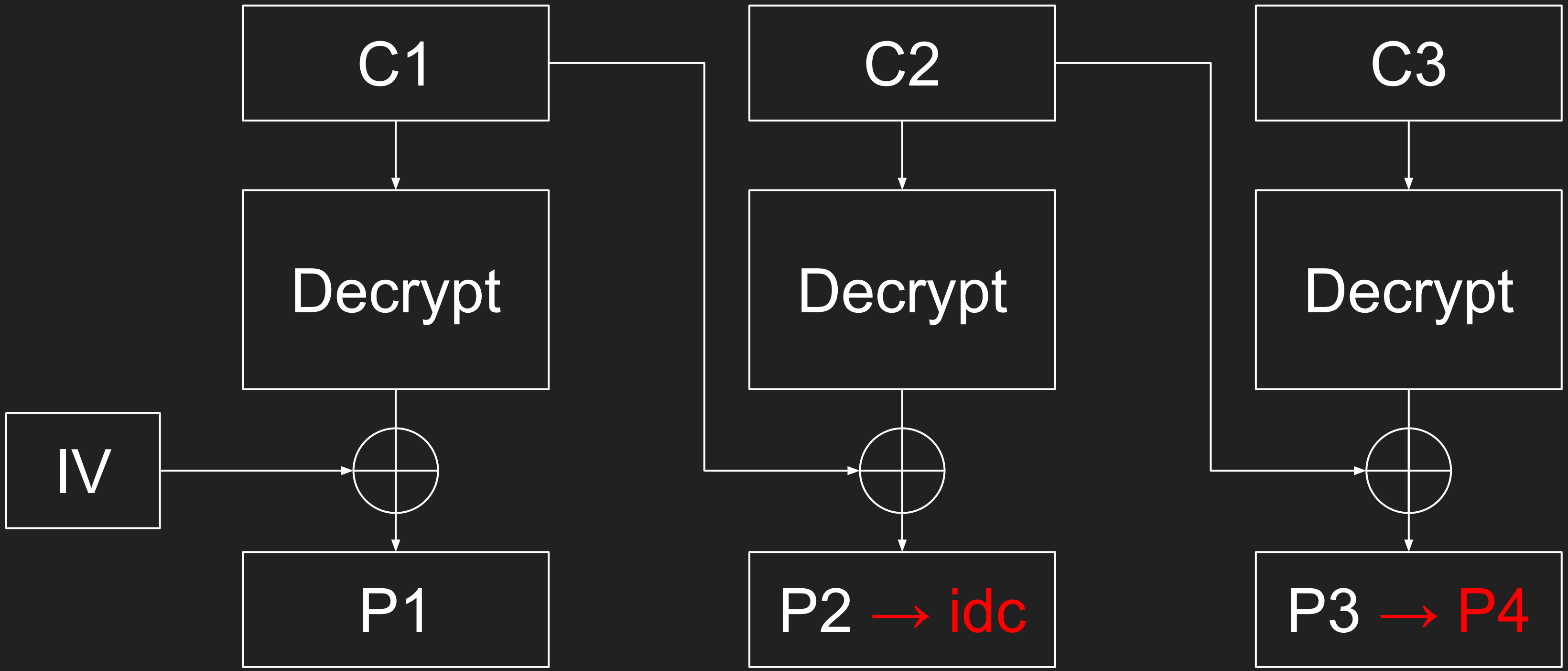
Bit-flipping attack



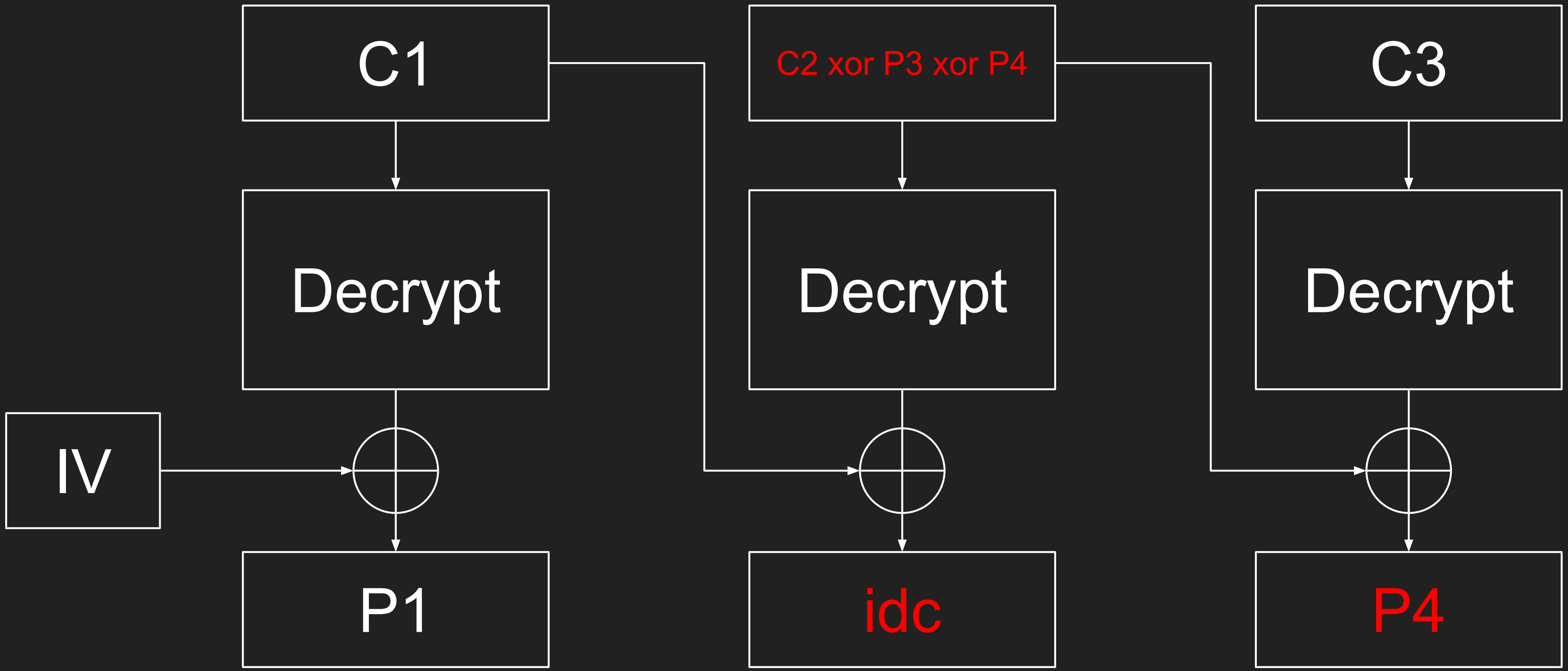
Bit-flipping attack



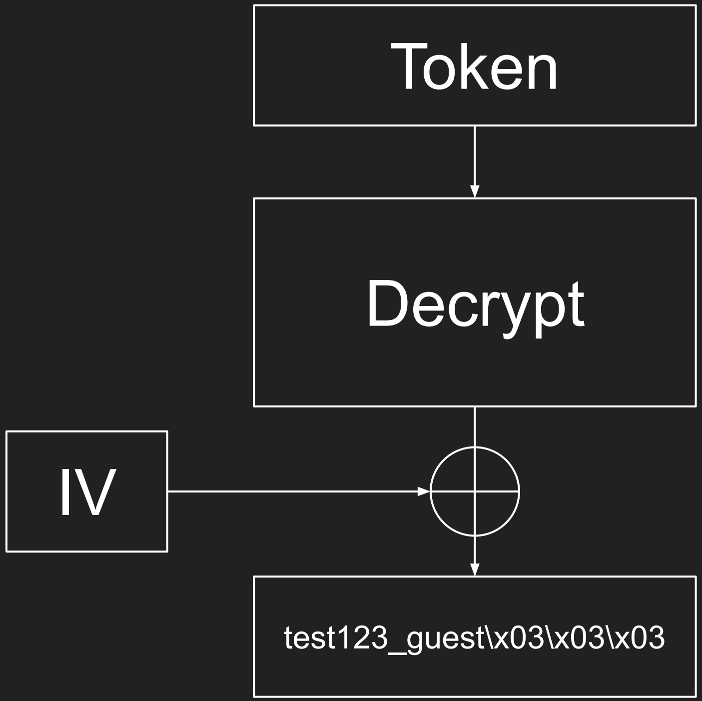
Bit-flipping attack



Bit-flipping attack

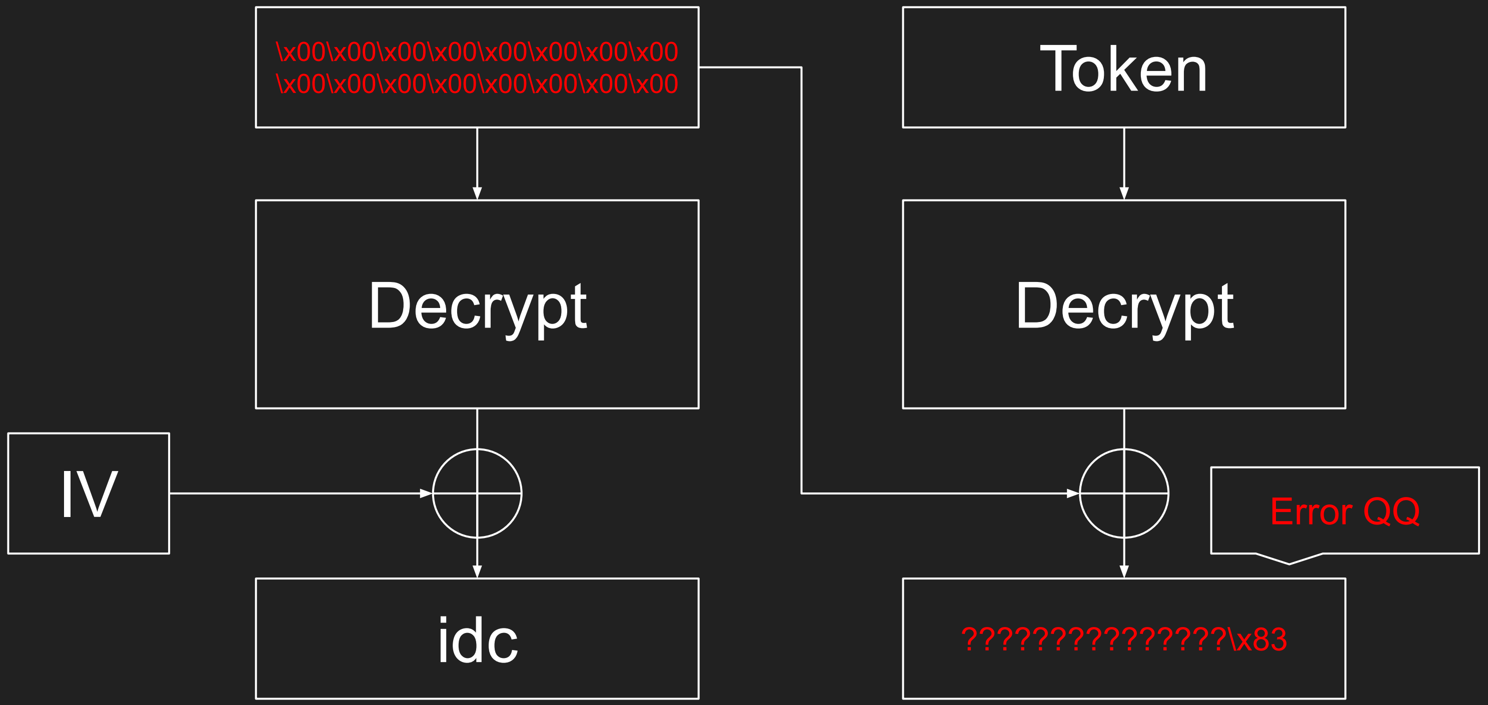


Padding Oracle attack

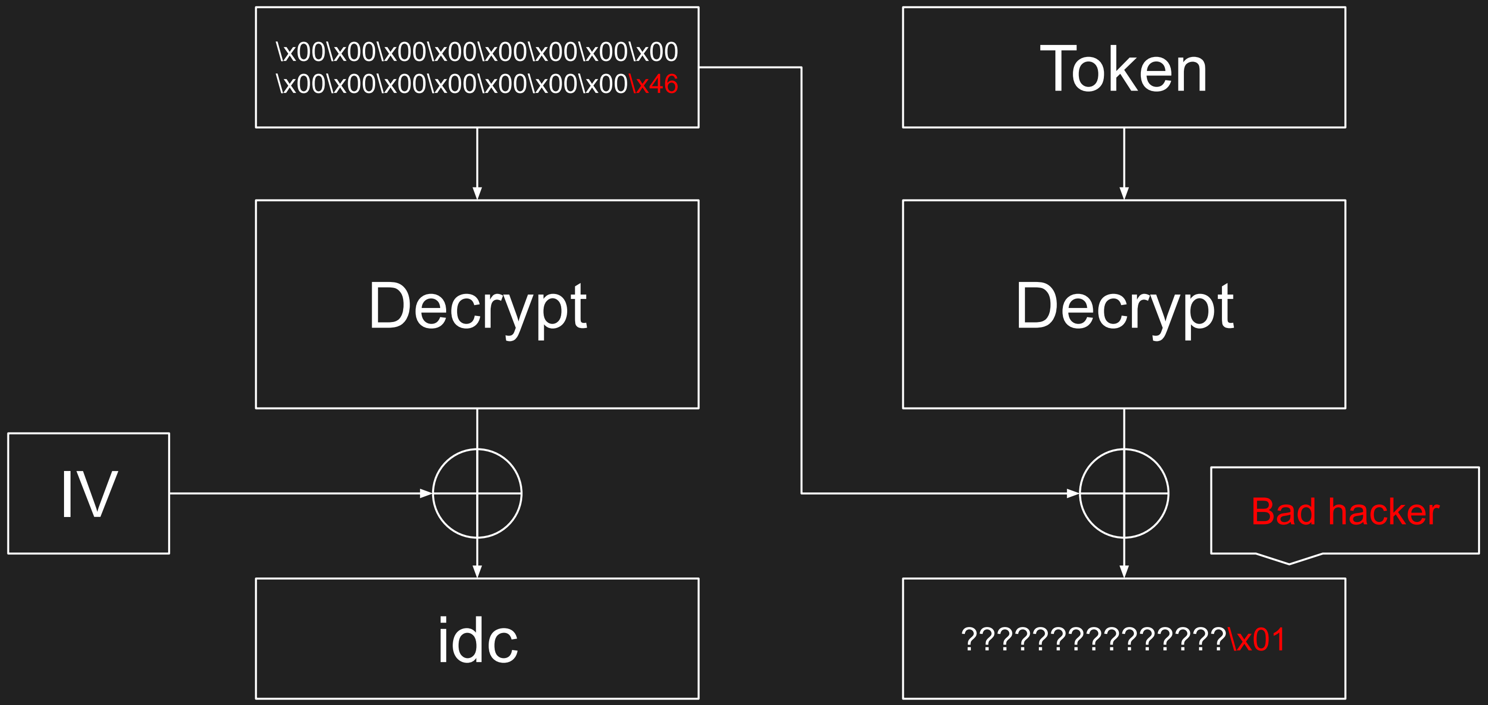



```python
def decrypt(ciphertext: bytes) -> bytes:
    try:
        cipher = AES.new(key, AES.MODE_CBC, iv)
        plaintext = cipher.decrypt(ciphertext)
        return unpad(plaintext)
    except AssertionError:
        return b"Error"


def verify():
    data = decrypt(bytes.fromhex(input("> Token: ")))
    if b"test123_guest" in data:
        print("Hi guest!")
        return

    if b"user456_admin" in data:
        print(f"Hi admin! Here is your flag: {FLAG}")
        sys.exit()

    if b"Error" in data:
        print("Error QQ")
        return

    print("Bad hacker")
```

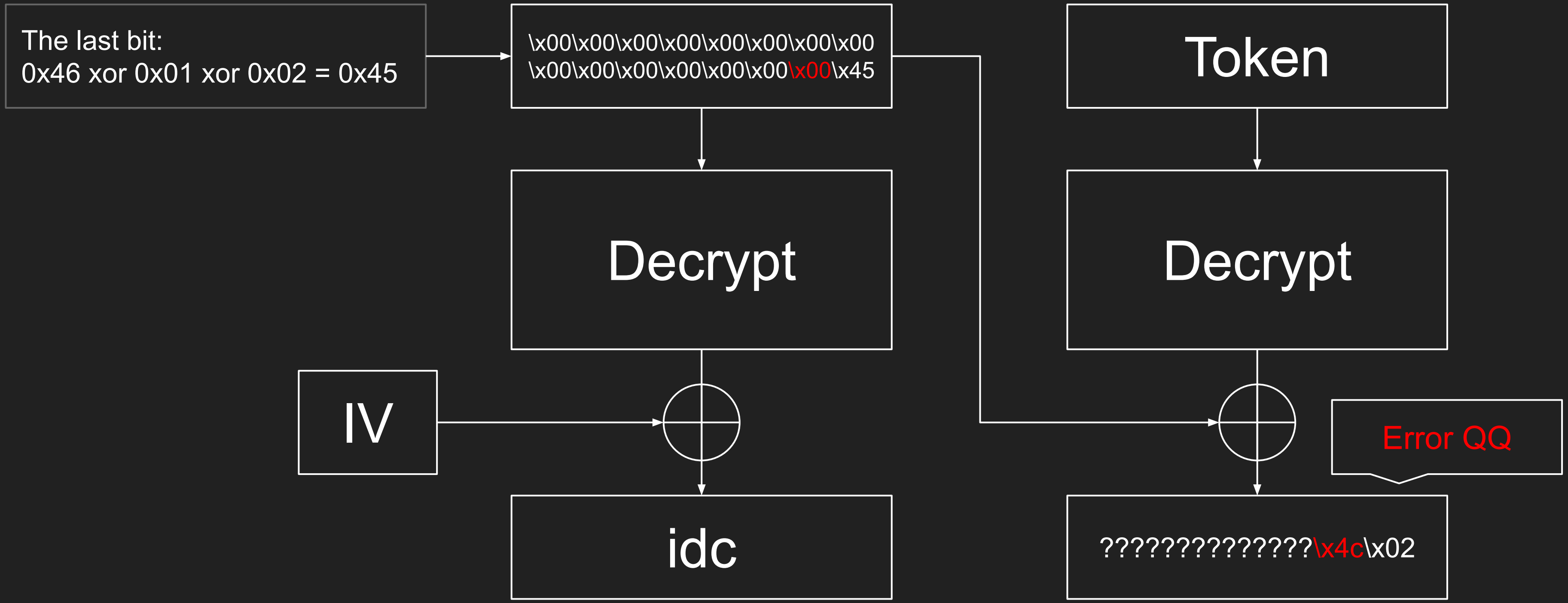
Padding Oracle attack



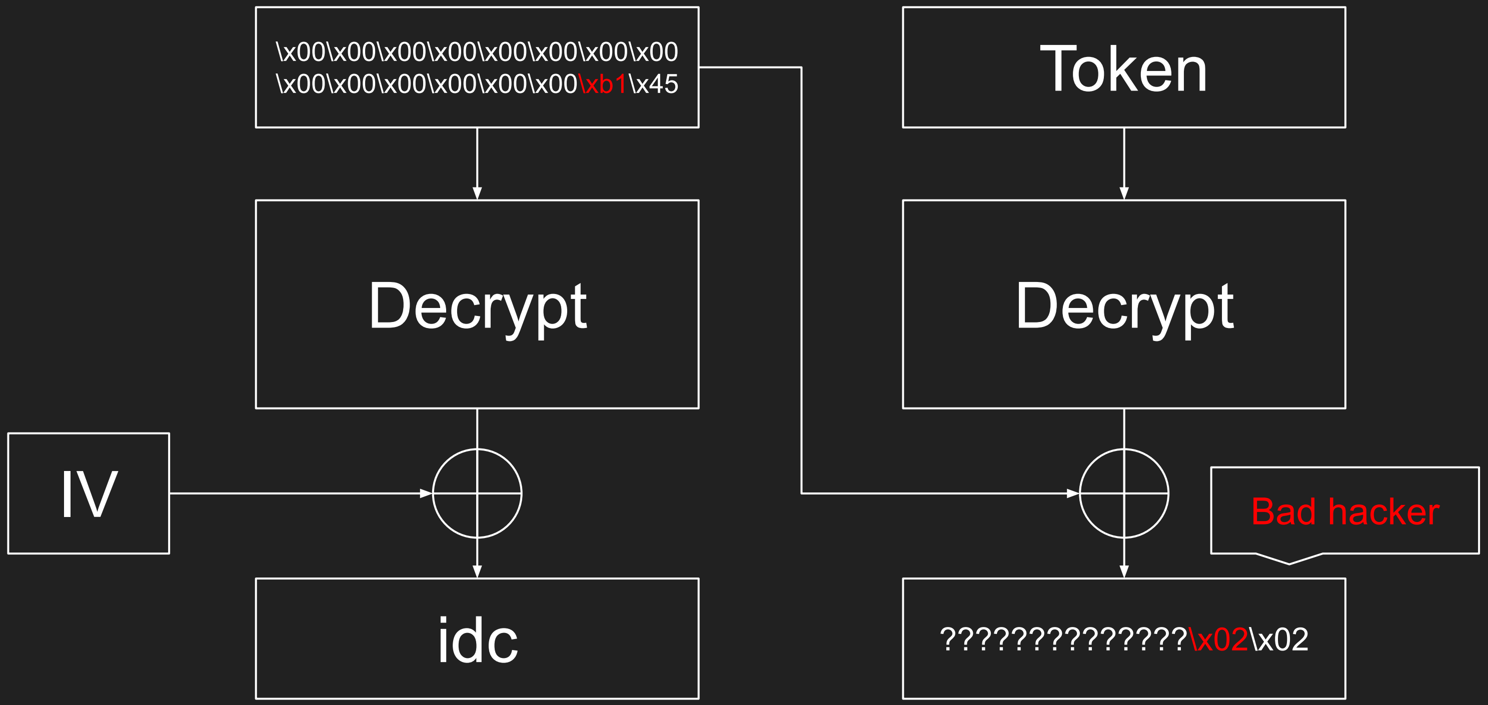
Padding Oracle attack



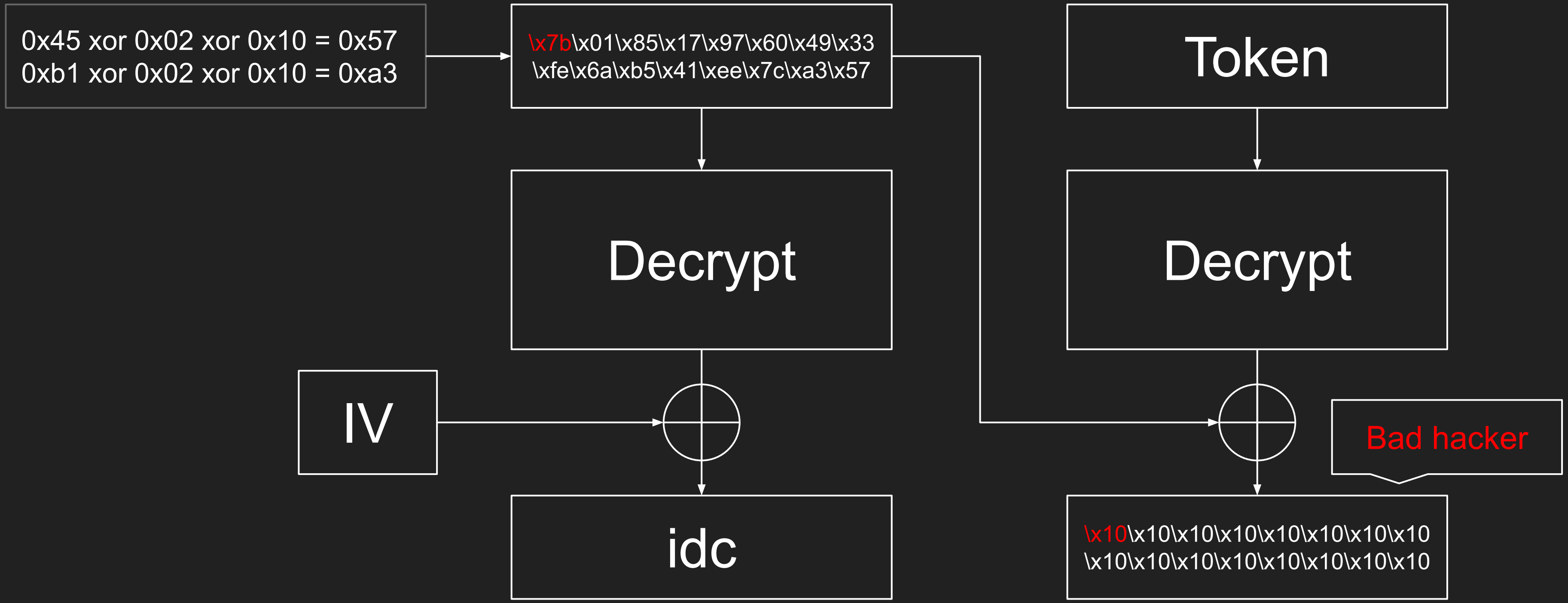
Padding Oracle attack



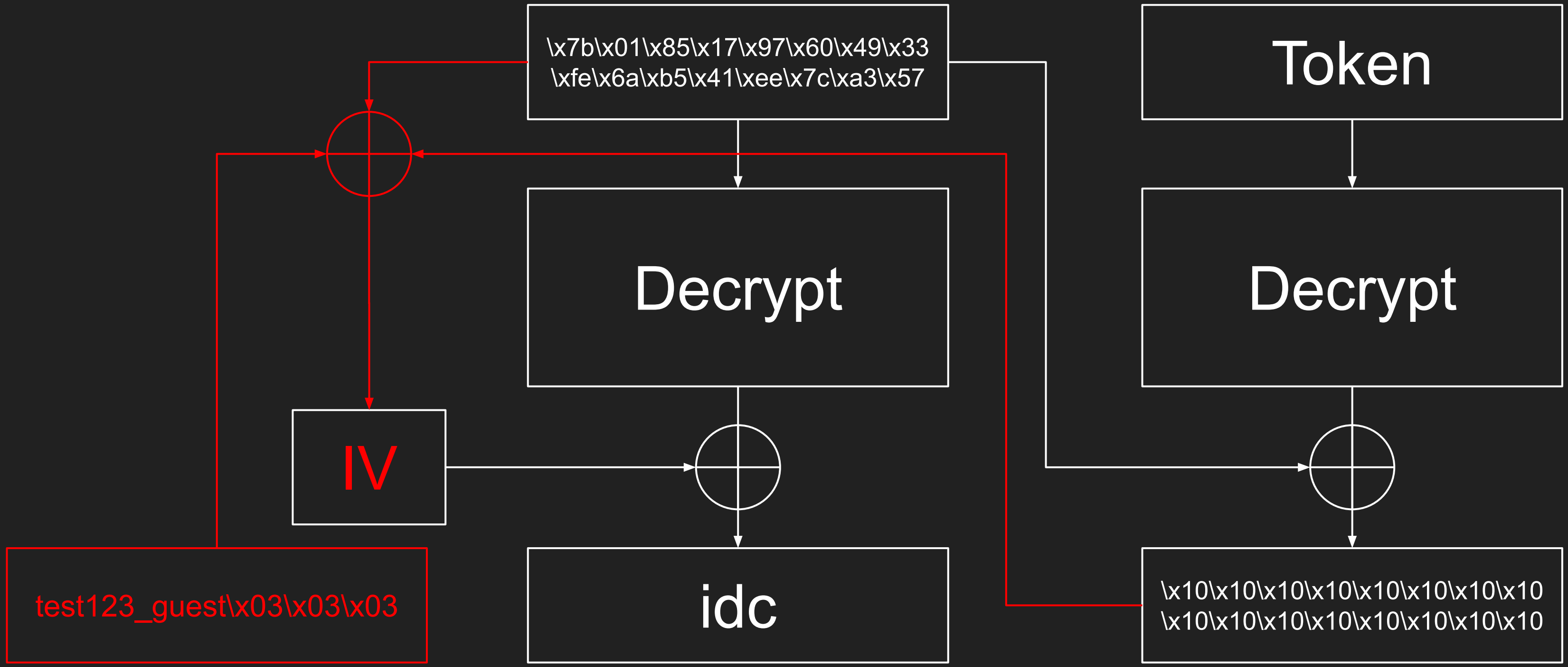
Padding Oracle attack



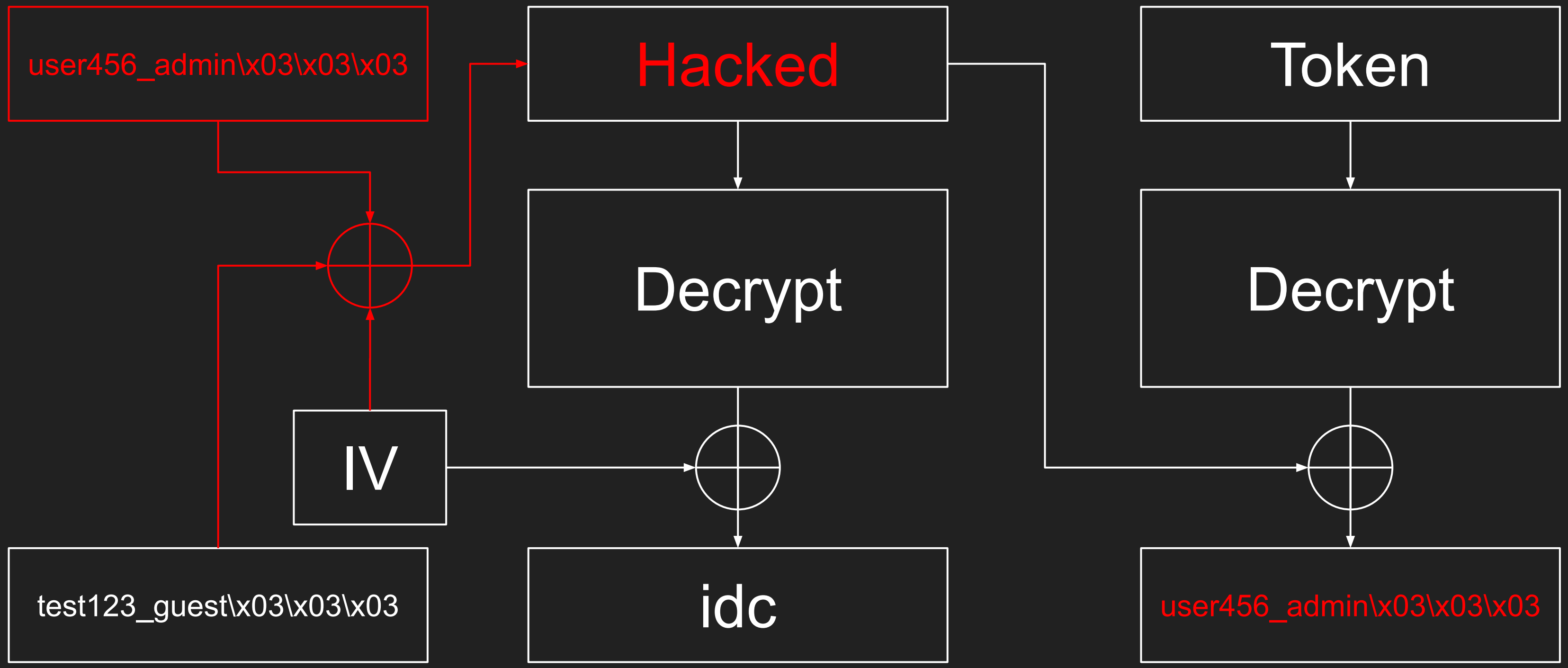
Padding Oracle attack



Padding Oracle attack

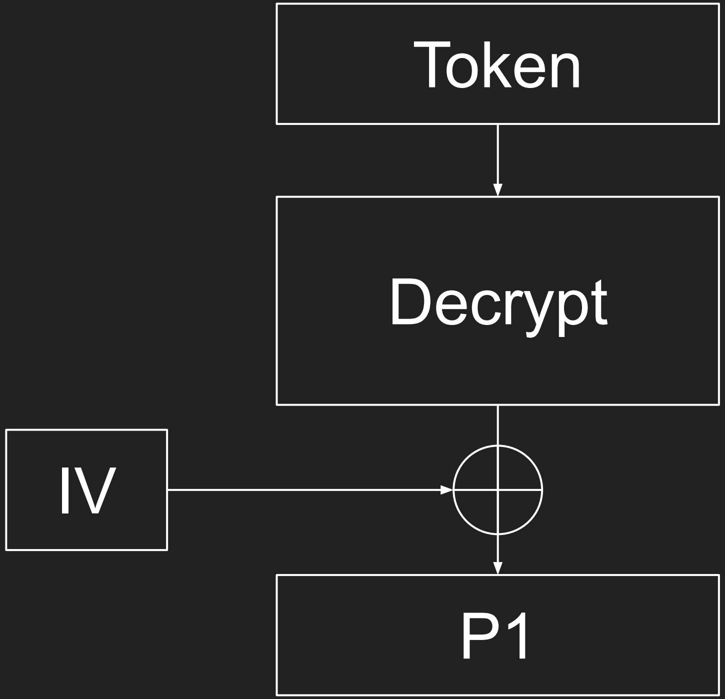


Padding Oracle attack



Padding Oracle attack

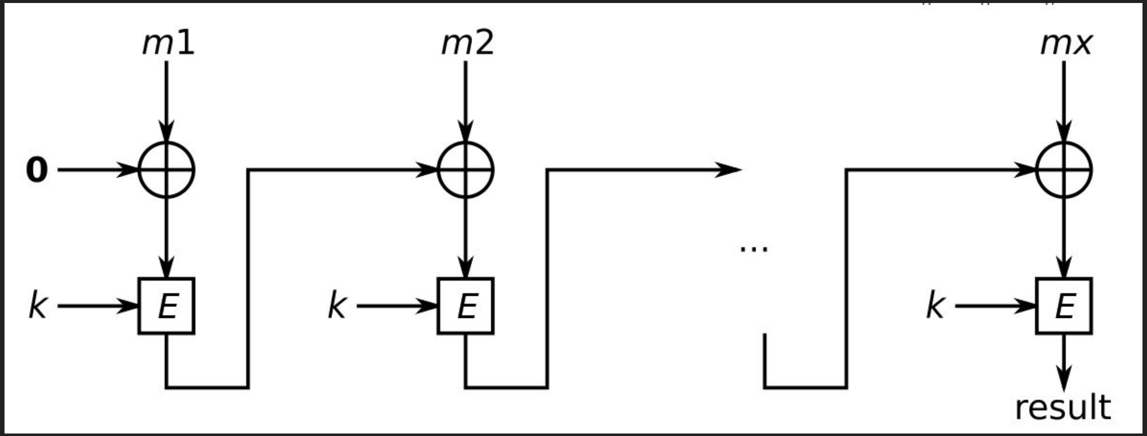
- P0 = \x7b\x01\x85\x17\x97\x60\x49\x33 \xfe\x6a\xb5\x41\xee\x7c\xa3\x57
- Padding = \x10\x10\x10\x10\x10\x10\x10\x10 \x10\x10\x10\x10\x10\x10\x10\x10
- IV = P0 xor P1 xor Padding
- Hacked = P0 xor P2 xor Padding (P1 → P2)



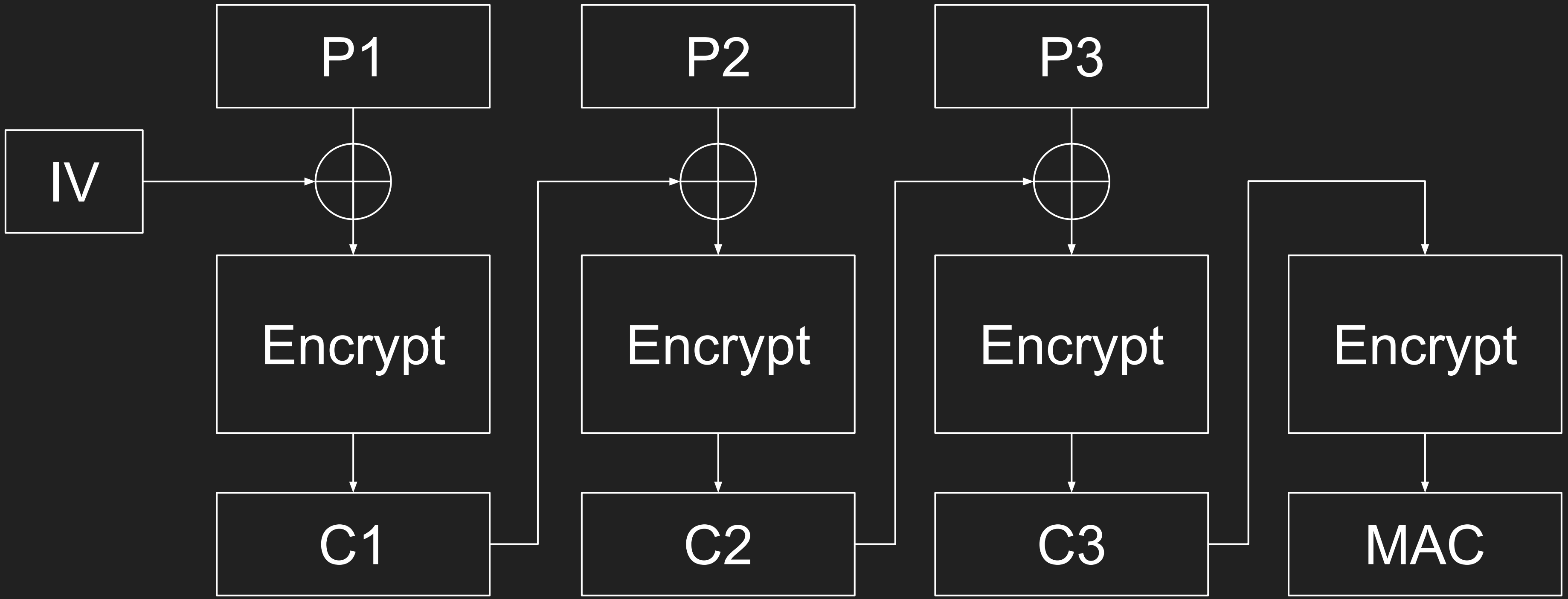
CBC-MAC

- MAC (Message Authentication Code)
- 輸入 message 先用 CBC mode 加密
- 將最後一個 block 再用 ECB mode 加密一次
- 其中 CBC 和 ECB mode 的 key 是相同的

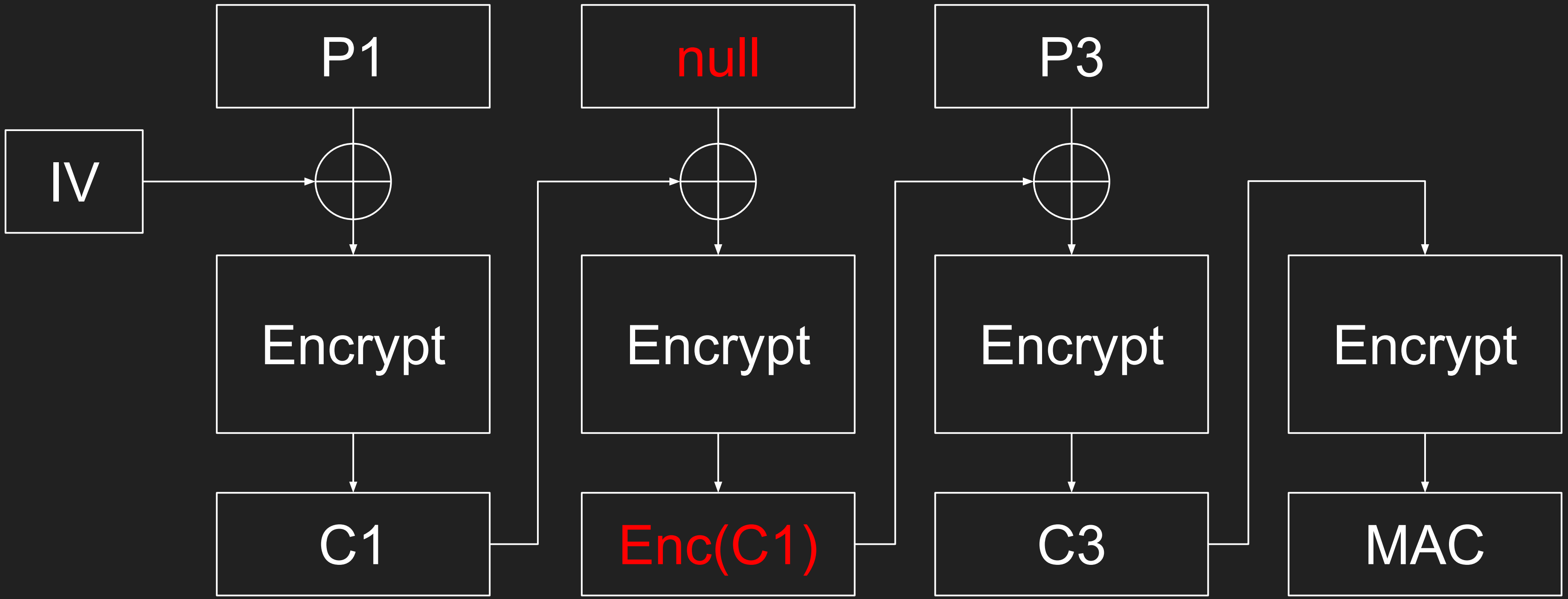
```
def generate_cbc_mac(text: bytes) -> bytes:
    cipher = AES.new(key, AES.MODE_CBC, iv)
    rawmac = cipher.encrypt(pad(text))
    cipher = AES.new(key, AES.MODE_ECB)
    cbcmac = cipher.encrypt(rawmac[-16:])
    return cbcmac
```



CBC-MAC

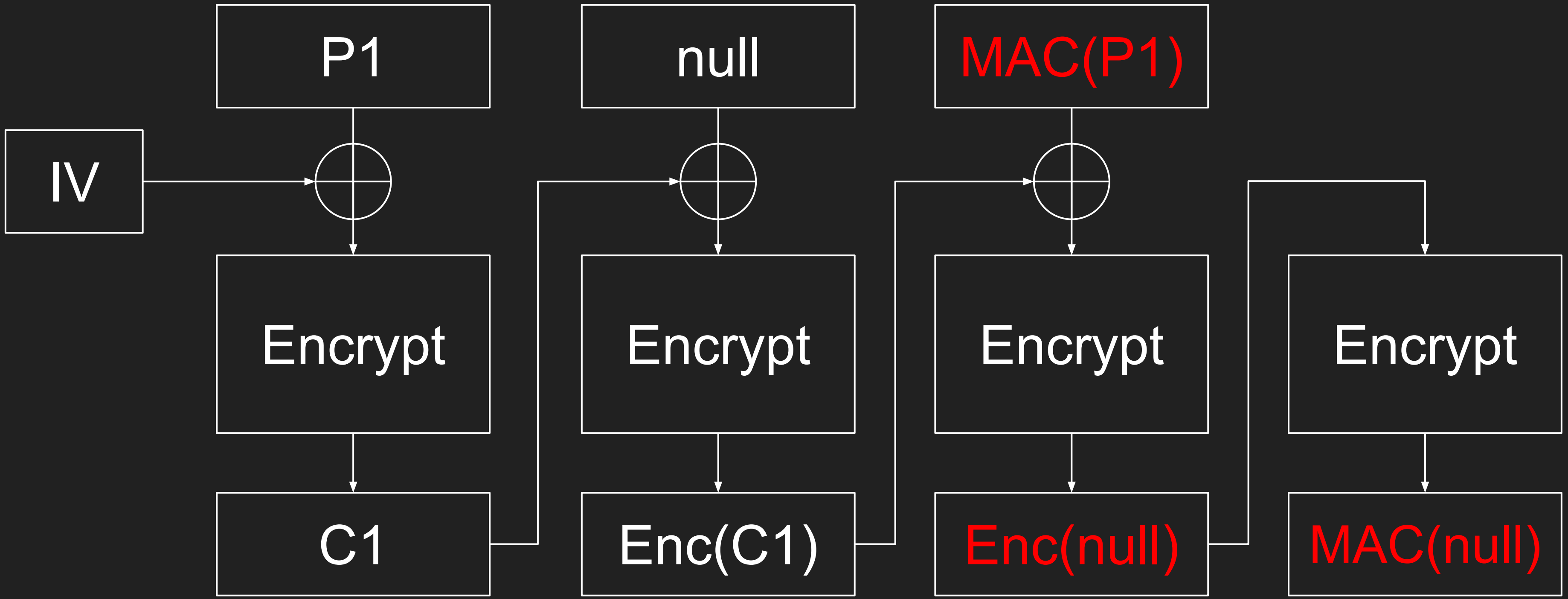


CBC-MAC



P1
null
P3
IV
Encrypt
Encrypt
Encrypt
Encrypt
C1
Enc(C1)
C3
MAC

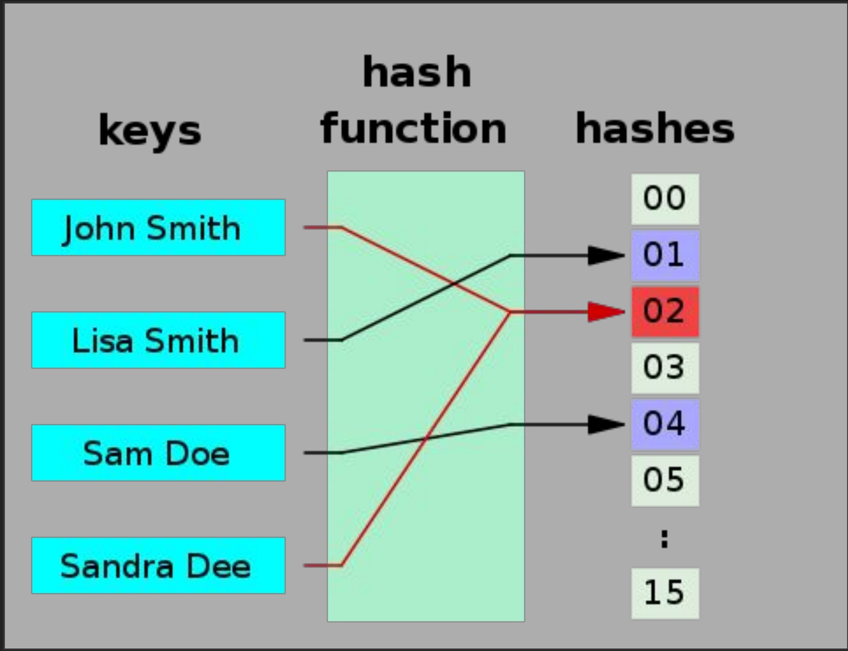
CBC-MAC


P1
null
MAC(P1)
IV
Encrypt
Encrypt
Encrypt
Encrypt
C1
Enc(C1)
Enc(null)
MAC(null)

Hash function

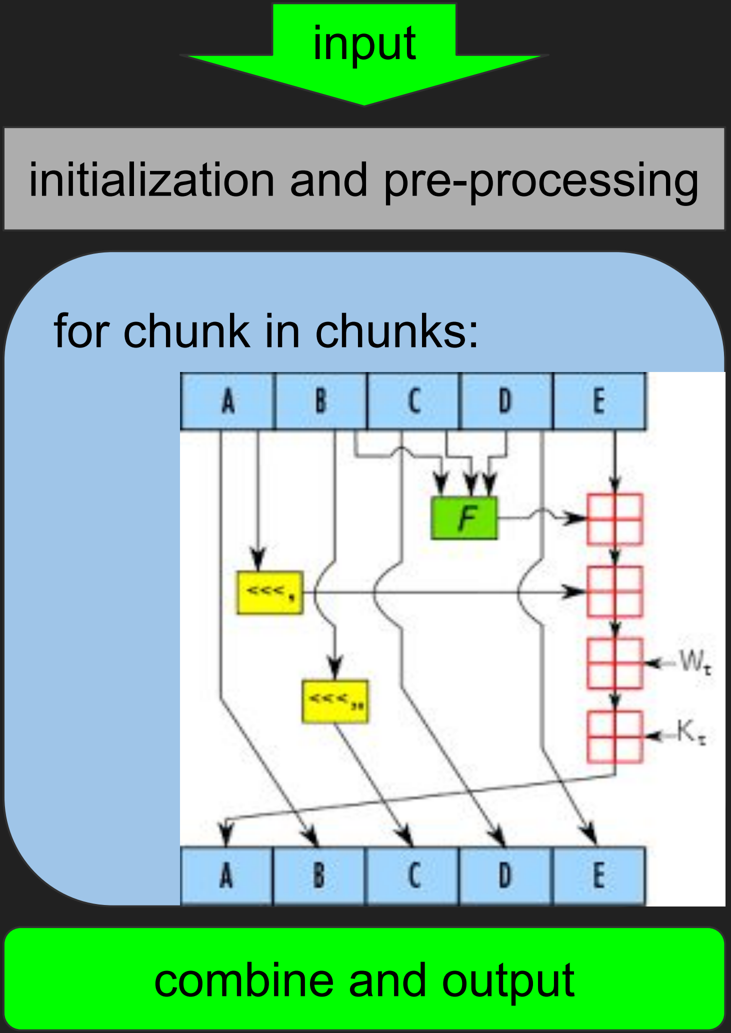
Hash function

- 雜湊函數，又稱雜湊演算法、哈希函數
- 有不可逆的特性
  - 儲存密碼
  - 製作簽章/校驗碼
- 為多對一的函數
  - 有可能會被碰撞
  - 是否可惡意碰撞或偽造



SHA (Secure Hash Algorithm)

- MD5 (Merkle–Damgård construction)
- SHA1
- SHA2
    - SHA-224
    - SHA-256
    - SHA-384
    - SHA-512
- SHA3 (Keccak)
    - SHA3-224
    - SHA3-256
    - SHA3-384
    - SHA3-512



SHA (Secure Hash Algorithm)

```
# initialization variables
h0, h1, h2, h3, h4 = 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0

# pre-processing
message += b"\x80"
message += b"\x00" * ((56 - len(message) % 64) % 64)
message += message_length.to_bytes(8, byteorder="big")

# break the message in 512bits chunks
chunks = [message[i:i+64] for i in range(0, len(message), 64)]
for chunk in chunks:
    # break chuck into sixteen 32bits big-endian words and extend to 80 words
    # initialize hash value for this chunk
    # add this chunk's hash to result so far

# produce the final hash value
digest = "".join(map(lambda x: x.to_bytes(4, byteorder="big").hex(), (h0, h1, h2, h3, h4)))
print(digest)
```

SHA (Secure Hash Algorithm)

```
for chunk in chunks:
    # break chuck into sixteen 32bits big-endian words and extend to 80 words
    w = [int.from_bytes(chunk[i:i+4], byteorder="big") for i in range(0, len(chunk), 4)]
    for i in range(16, 80):
        w.append(left_rotate(w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16], 1))
    # initialize hash value for this chunk
    a, b, c, d, e = h0, h1, h2, h3, h4
    for i in range(80):
        if 0 <= i <= 19:    f, k = (b & c) | (~b & d),          0x5A827999
        elif 20 <= i <= 39: f, k = b ^ c ^ d,                   0x6ED9EBA1
        elif 40 <= i <= 59: f, k = (b & c) | (b & d) | (c & d), 0x8F1BBCDC
        elif 60 <= i <= 79: f, k = b ^ c ^ d,                   0xCA62C1D6
        a, b, c, d, e = (left_rotate(a, 5) + f + e + k + w[i]) & 0xffffffff, a, left_rotate(b, 30), c, d
    # add this chunk's hash to result so far
    h0 = (h0 + a) & 0xffffffff
    h1 = (h1 + b) & 0xffffffff
    h2 = (h2 + c) & 0xffffffff
    h3 = (h3 + d) & 0xffffffff
    h4 = (h4 + e) & 0xffffffff
```

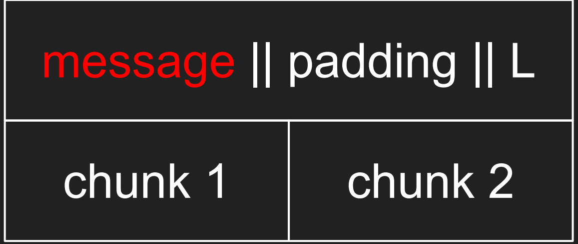
SHA (Secure Hash Algorithm)

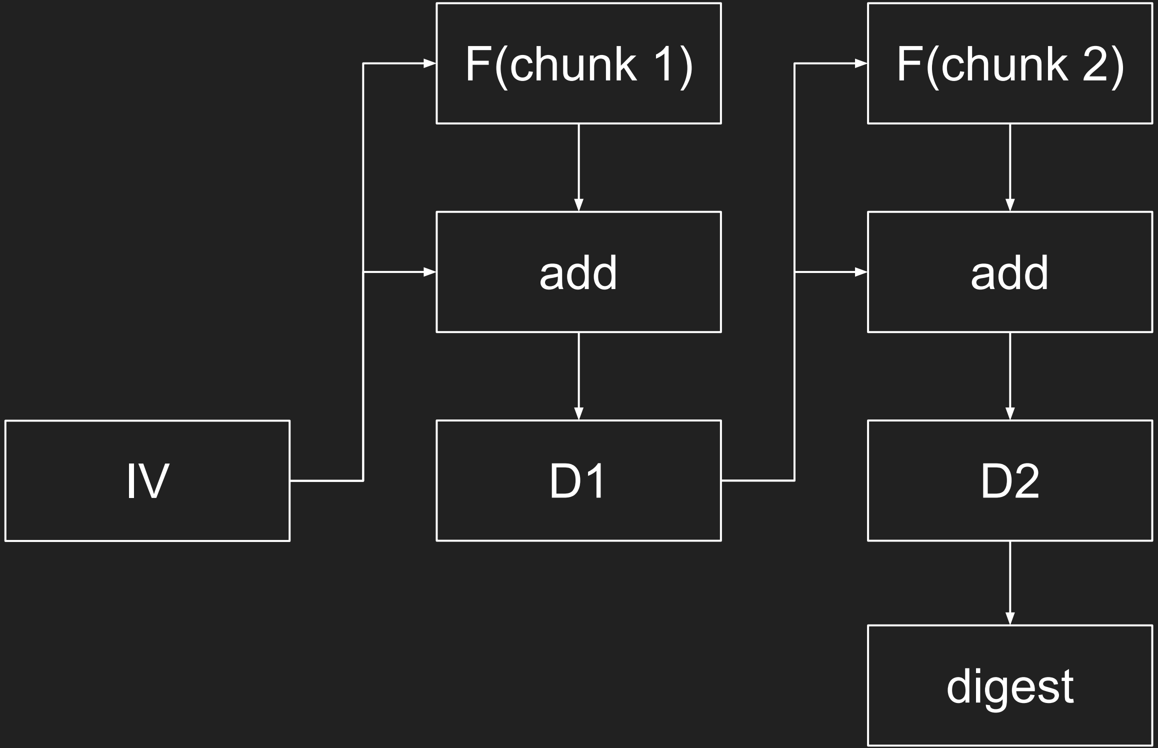
Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security against collision attacks (bits)	Security against length extension attacks (bits)	Performance on Skylake (median cpb)[1]		First published
									Long messages	8 bytes	
MD5 (as reference)		128	128 (4 × 32)	512	64	And, Xor, Or, Rot, Add (mod 2^{32})	≤ 18 (collisions found)[2]	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Or, Rot, Add (mod 2^{32})	< 34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1							< 63 (collisions found)[3]		3.47	52.00	1995
SHA-2	*SHA-224* *SHA-256*	224 256	256 (8 × 32)	512	64	And, Xor, Or, Rot, Shr, Add (mod 2^{32})	112 128	32 0	7.62 7.63	84.50 85.25	2004 2001
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Or, Rot, Shr, Add (mod 2^{64})	192	128 (≤ 384)	5.12	135.75	2001
	SHA-512	512					256	0[4]	5.06	135.50	2001
	SHA-512/224 *SHA-512/256*	224 256					112 128	288 256	≈ SHA-384	≈ SHA-384	2012
SHA-3	*SHA3-224* *SHA3-256* *SHA3-384* *SHA3-512*	224 256 384 512	1600 (5 × 5 × 64)	1152 1088 832 576	24[5]	And, Xor, Rot, Not	112 128 192 256	448 512 768 1024	8.12 8.59 11.06 15.88	154.25 155.50 164.00 164.00	2015
	SHAKE128 *SHAKE256*	*d* (arbitrary) *d* (arbitrary)		1344 1088			min(d/2, 128) min(d/2, 256)	256 512	7.08 8.59	155.25 155.50	

LEA (Length Extension Attacks)

- 長度擴展攻擊
- 在知道訊息長度的情況下，對雜湊值做訊息的擴充
- 利用雜湊函會將 state 串接直接變成雜湊值的弱點(洩漏)
- 已知條件
  - k = len(message)
  - sig = hash(message)
- 可擴充成
  - new sig = hash(message || data)
- 舉例
  - k = len(secret)
  - sig = hash(secret || user=guest&admin=0)
  - new sig = hash(secret || user=guest&admin=0 || idc || &admin=1)

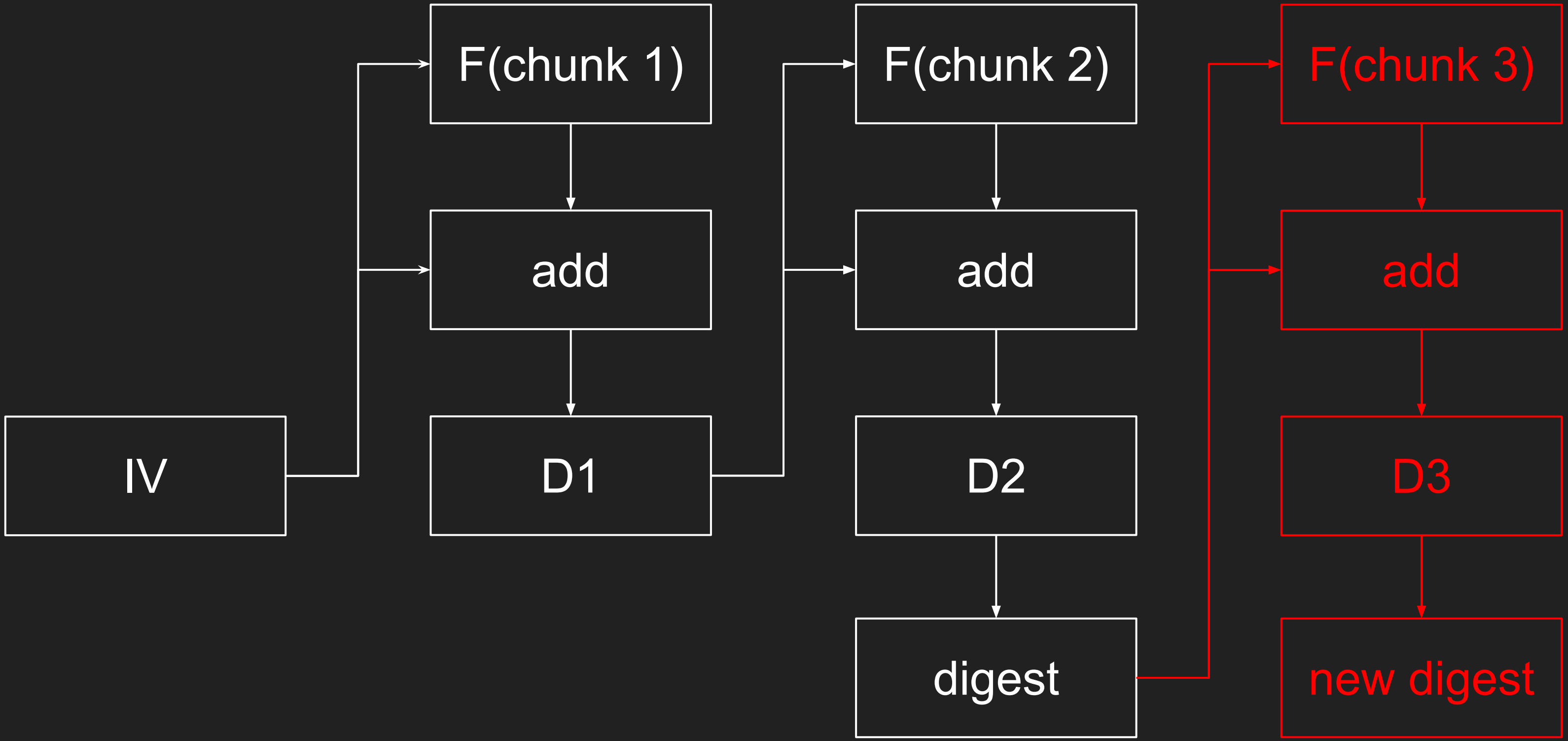
LEA (Length Extension Attacks)





LEA (Length Extension Attacks)

message || padding || L		data
chunk 1	chunk 2	chunk 3



LEA (Length Extension Attacks)

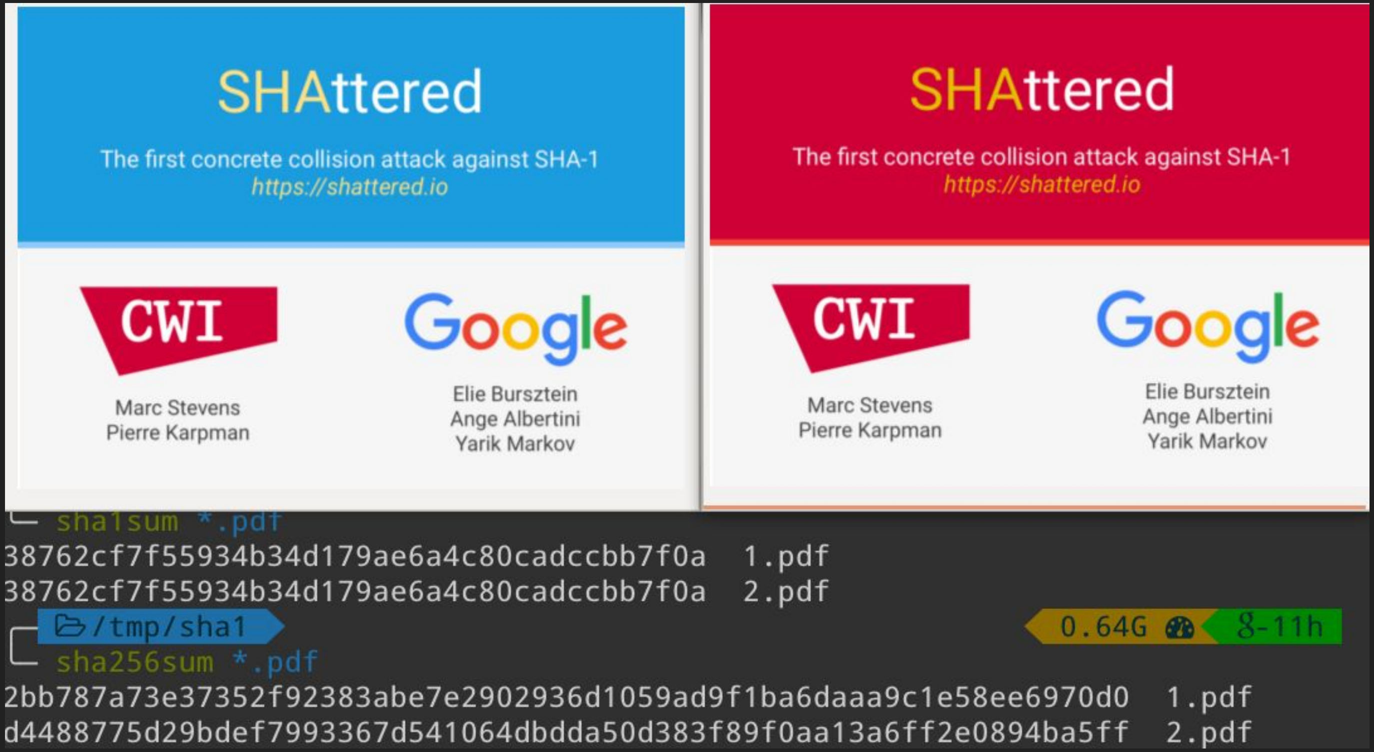
- 舉個例子
  - sig = hash(secret || user=guest&admin=0)
  - len(secret) = 50
- 想把 message 擴充(竄改)成
  - message = secret || user=guest&admin=0 || idc || &admin=1
- 原本 sig 真正組成的 message
  - secret || user=guest&admin=0 || padding || L=68
- 所以 new sig 真正要算的 message
  - secret || user=guest&admin=0 || padding || L=68 || &admin=1 || padding_2 || L=136
- 真正要擴充上去的 data
  - padding || L=68 || &admin=1

MD5 Collision

- 不同情況下，產生兩組相同的 MD5 資料
  - HashClash (docker)
    - 可指定特定的前綴
  - MD5 tunneling
    - 可指定特定的初始 IV

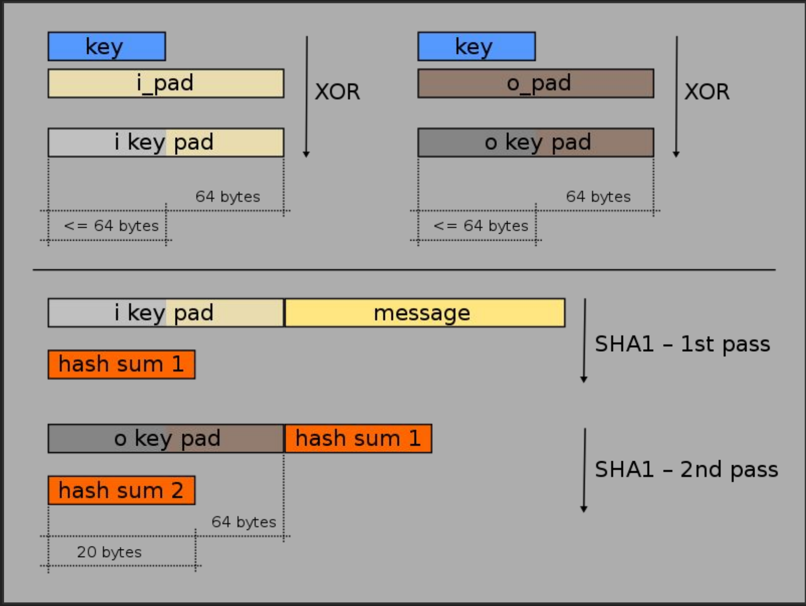
```
Killua4564:NTUSTISC Killua4564$ md5sum collision1.bin collision2.bin
9c96362cb7fccb439f912f5631681883  collision1.bin
9c96362cb7fccb439f912f5631681883  collision2.bin
Killua4564:NTUSTISC Killua4564$ xxd collision1.bin
00000000: 4e54 5553 5449 5343 a525 3162 2392 70fd  NTUSTISC.%1b#.p.
00000010: b4a4 e0c4 bbfc af03 be49 e95a 06da a291  .........I.Z....
00000020: 17cb 5ba6 a96b a537 27d2 4551 de30 0745  ..[..k.7'.EQ.0.E
00000030: 6929 d4f8 4f8a 3d14 91da b14e 9ba8 e14c  i)..O.=....N...L
00000040: 03f4 500b 2410 b56a 3baf fc2c cf39 c2d4  ..P.$..j;..,.9..
00000050: c9b6 a43e 9998 db9a 6b54 abaf 7c1c 3463  ...>....kT..|.4c
00000060: 6826 3b02 f00c ccd3 7a29 c14f 1835 44d4  h&;.....z).O.5D.
00000070: d68b dac0 80db 09e5 aff8 b339 54da 0f2e  ...........9T...
Killua4564:NTUSTISC Killua4564$ xxd collision2.bin
00000000: 4e54 5553 5449 5343 a526 3162 2392 70fd  NTUSTISC.&1b#.p.
00000010: b4a4 e0c4 bbfc af03 be49 e95a 06da a291  .........I.Z....
00000020: 17cb 5ba6 a96b a537 27d2 4551 de30 0745  ..[..k.7'.EQ.0.E
00000030: 6929 d4f8 4f8a 3d14 91da b14e 9ba8 e14c  i)..O.=....N...L
00000040: 03f4 500b 2410 b56a 3bae fc2c cf39 c2d4  ..P.$..j;..,.9..
00000050: c9b6 a43e 9998 db9a 6b54 abaf 7c1c 3463  ...>....kT..|.4c
00000060: 6826 3b02 f00c ccd3 7a29 c14f 1835 44d4  h&;.....z).O.5D.
00000070: d68b dac0 80db 09e5 aff8 b339 54da 0f2e  ...........9T...
```

SHA1 Collision



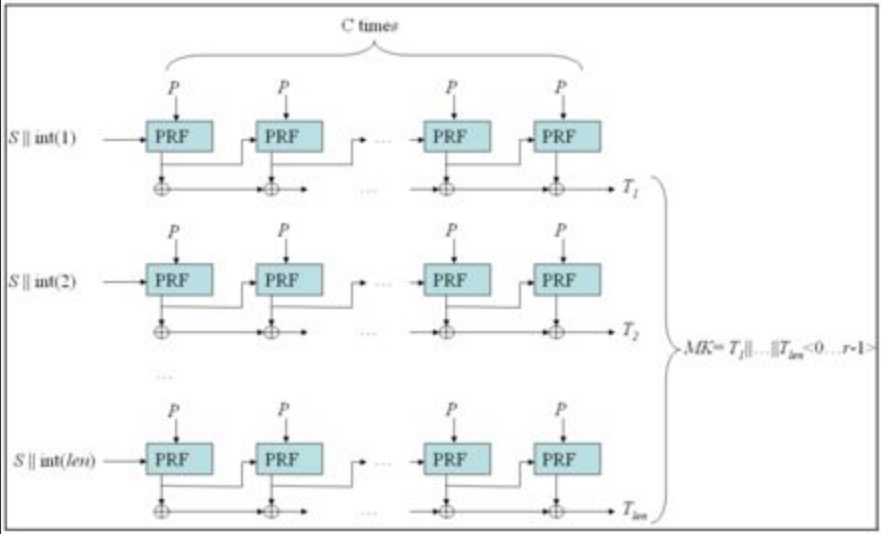
HMAC

- 雜湊訊息鑑別碼
- $$\mathrm{HMAC}(K, m) = \mathrm{H}\Big(\big(K' \oplus opad\big) \,\|\, \mathrm{H}\big(\big(K' \oplus ipad\big) \,\|\, m\big)\Big)$$
  $$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$
- 很好的抵禦 LEA
- 有機會用不同 key 算出相同 HMAC
  - 找任意足夠大的 K
  - 則 (K, H(K)) 會產生相同 K’
- 使用 SHA256 的 HMAC
  - 通常叫 HMAC-SHA256
  - 在 JWT 裡叫 HS256



PBKDF2

- 金鑰衍生函式 (KDF)
  - 將一組金鑰擴充成很多組的方式
- $DK = \mathrm{PBKDF2}(\mathrm{PRF}, \mathit{Password}, \mathit{Salt}, c, \mathit{dkLen})$
  - PRF 是能產 MAC 的函式
- 適合保護密碼的雜湊
- 實作流程 (ry
  - 偽造的思路是在 PRF 身上
- PKCS #5 v2.0
- OWASP 推薦 iterations
  - PBKDF2-HMAC-SHA256 (310000)
  - PBKDF2-HMAC-SHA512 (120000)



Public-key cryptography

RSA

Low public exponent attack

- 低指數公鑰攻擊、低加密指數攻擊
- e 很小的時候 (例如 e=3) 直接爆破出來 m
- $m^e \bmod n = c$
  - $\rightarrow m^3 \bmod n = c$
  - $\rightarrow m^3 = c + k * n$
- 窮舉一下 k 開 e 方根
- 目的
  - 為了節省加密和驗證的時間 $(F_x = 2^{2^x} + 1)$
  - 通常 e 會從 Fermat number 挑
    - 也通常挑 $F_0=3$, $F_2=17$, $F_4=65537$

Wiener's attack

- 維納攻擊，是種低解密指數攻擊
- 當 e 非常大導致 d 很小時用 (e, n) 直接推算 d
- 當 $d < 1/3 * n^{1/4}$ 和 $|p - q| < \min(p, q)$ 條件符合時可以利用 (e, n) 來估計 (d, φ(n))
- $ed \equiv 1 \pmod{\varphi(n)}$
  - $\rightarrow k \in N, ed = k * \varphi(n) + 1$
  - $\rightarrow e / \varphi(n) = k / d + 1 / (d * \varphi(n))$
  - $\rightarrow e / \varphi(n) \approx k / d$
  - $\rightarrow e / n \approx k / d$

Wiener's attack - Lemma 1

- 滿足 $|p - q| < \min(p, q)$
  - 也就是 $q < p < 2q$
- $n - \varphi(n)$
  - $\rightarrow n - (p - 1) * (q - 1)$
  - $\rightarrow n - pq + p + q - 1$
  - $\rightarrow p + q - 1 < 3\text{sqrt}(n)$
- 得到 $n - \varphi(n) < 3\text{sqrt}(n)$

Wiener's attack - Lemma 2

- 滿足 $d < 1/3 * n^{1/4}$
- $ed \equiv 1 \pmod{\varphi(n)}$
  - $\rightarrow ed = k * \varphi(n) + 1$
  - $\rightarrow k * \varphi(n) = ed - 1$
  - $\rightarrow k * \varphi(n) < ed$
  - $\rightarrow k * \varphi(n) < \varphi(n) * d$
  - $\rightarrow k < d < 1/3 * n^{1/4}$
  - $\rightarrow k < 1/3 * n^{1/4}$
- 得到 $k < 1/3 * n^{1/4}$

Wiener's attack - Lemma 3

- 滿足 $d < 1/3 * n^{1/4}$
  - $\rightarrow d < 1/3 * n^{1/4}$
  - $\rightarrow 3d < n^{1/4}$
  - $\rightarrow 2d < n^{1/4}$
  - $\rightarrow 1 / 2d > 1 / n^{1/4}$
- 得到 $1 / n^{1/4} < 1 / 2d$

Wiener's attack - proof

- Lemma 1: $n - \varphi(n) < 3\text{sqrt}(n)$
- Lemma 2: $k < 1/3 * n^{1/4}$
- Lemma 3: $1 / n^{1/4} < 1 / 2d$
- 計算 $|e / n - k / d|$
  - → $|(ed - nk) / dn| = |(1 + k\varphi(n) - nk) / dn|$
  - → $(k(n - \varphi(n)) - 1) / dn < (3k * \text{sqrt}(n) - 1) / dn < 3k * \text{sqrt}(n) / dn$
  - → $3k * \text{sqrt}(n) / dn < n^{3/4} / dn = 1 / dn^{1/4}$
  - → $1 / dn^{1/4} < 1 / 2d^2$
- 發現 e / n 和 k / d 相差小於 $1 / 2d^2$
  - 可利用 e / n 將 k / d 逼出來

Wiener's attack

- e / n ≈ k / d
- 連分數

$$\frac{e}{N}=\frac{17993}{90581}=\cfrac{1}{5+\cfrac{1}{29+\cdots+\cfrac{1}{3}}}=[0,5,29,4,1,3,2,4,3]$$

- 推算 (k, d)

$$\frac{k}{d}=0,\frac{1}{5},\frac{29}{146},\frac{117}{589},\frac{146}{735},\frac{555}{2794},\frac{1256}{6323},\frac{5579}{28086},\frac{17993}{90581}$$

Wiener's attack

- 最後 $\varphi(n)$ 呢？
  - $ed = k * \varphi(n) + 1$
  - $\rightarrow \varphi(n) = (ed - 1) / k$
- 用途 1 : 驗證 d
  - assert d == inverse(e, φ(n))
- 用途 2 : 分解 (p, q)
  - $\varphi(n) = (p - 1) * (q - 1) = pq - (p + q) + 1 = n - (p + q) + 1$
  - $\rightarrow p + q = n - \varphi(n) + 1$
  - 考慮方程 $x^2 - (p + q)x + pq = 0$
    - 則 x 的兩個解為 (p, q)

LSB Oracle attack

- 情境:給密文 c 回傳解密後明文 mod r 的結果
  - 假設 r = 2, 則回傳明文的最後一個 bit
- 利用同態加密 (Homomorphic encryption) 的特性製成 $r^{ke}c \bmod n$ 送出
  - 如果 $0 \le m \le n/2$, 則 $2m \bmod 2 = 0$
    - 如果 $0 \le m \le n/4$, 則 $4m \bmod 2 = 0$
    - 如果 $n/4 \le m \le n/2$, 則 $(4m - n) \bmod 2 = 1$
  - 如果 $n/2 \le m \le n$, 則 $(2m - n) \bmod 2 = 1$
    - 如果 $n/2 \le m \le 3n/4$, 則 $(4m - 2n) \bmod 2 = 0$
    - 如果 $3n/4 \le m \le n$, 則 $(4m - 3n) \bmod 2 = 1$
  - 依此類推可以 oracle 出來原始的明文 m
    - 其中 $(r^k m - in) \bmod r$ 的值可以由 $(-in \bmod r)$ 得出

Bleichenbacher attack

- 又稱 million message attack
- 發生在 TLS 使用 PKCS#1 v1.5 版本的 RSA 公開金鑰交換
  - 其中 padding 會在 message 前面加上固定的 0x00 0x02 去做填充

00	02	padding string	00	data block

  - 而在 TLS 中如果 unpad 失敗會回覆 Bad data
  - 可以利用同態加密的特性做到 MSB Oracle
- 收到密文 c 之後遍歷每個 s，計算每個 $s^e c \bmod n$ 去嘗試 unpad
  - 如果 unpad 成功表示 $2B \le sm \bmod n < 3B$ (B 表示所有低位)
  - 就可以推出 $(2B + kn) / s \le m < (3B + kn) / s$
  - 雖然不知道 k 但可以遍歷所有 k 直到不滿足 $0 < m < n$
  - 把所有 s 中 k 的可能性做交集則可以擠出 m
- 後續影響：[DROWN attack]

To be continued