CTF 문제에 출제된 암호 공격

IT정보공학과 신명수

- DH key exchange man in the middle attack
- CBC mode Bit-Flipping Attack
- Wiener attack
- Padding oracle attack
- Chosen ciphertext attack
- Etc…

목차

1. DH Man In the Middle Attack

1.1 Diffie-Hellman key Exchange

1.2 Man In the Middle Attack

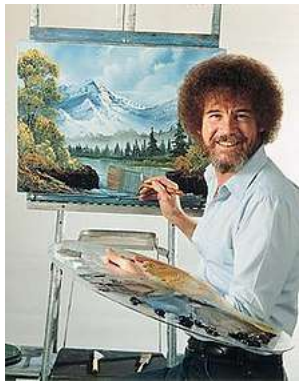
1.1 Diffie-Hellman key exchange

- 두 통신 당사자 사이에 공개되어도 상관없는 정보를 교환함으로써 비밀키를 공유할 수 있는 프로토콜.

- 이산 대수 문제(Discrete Logarithm Problem, DLP)을 기반으로 함. $y = g^x \ mod \ p$ 에서 g, y, p를 알아도 x를 알기 어렵다. (p는 소수) p

1.1 Diffie-Hellman key exchange

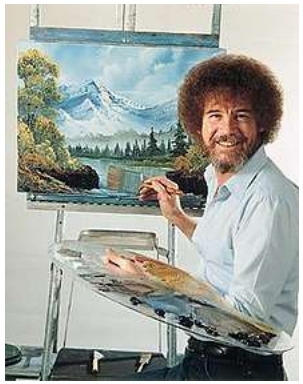
- Alice와 Bob의 통신



1.1 Diffie-Hellman key exchange

1. Generator g와 소수 p를 결정한다.

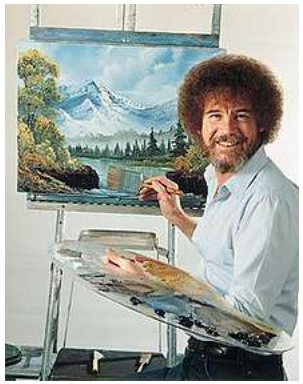
g , p



g , p

1.1 Diffie-Hellman key exchange

2. Alice는 개인키 a를 이용하여 $A = g^a \ mod \ p$ 를 계산한다.



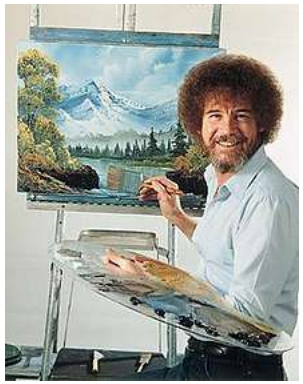
$A = g^a \ mod \ p$

$g \ , p$

1.1 Diffie-Hellman key exchange

3. Bob은 개인키 b를 이용하여 $B = g^b \ mod \ p$ 를 계산한다.

$A = g^a \ mod \ p$



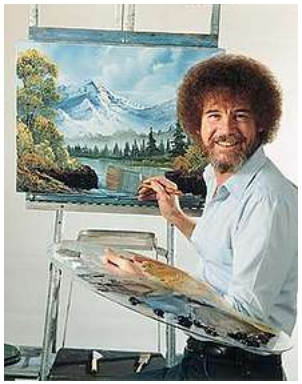
$B = g^b \ mod \ p$

1.1 Diffie-Hellman key exchange

4. Alice는 A를 Bob에게 전송하고, Bob은 B를 Alice 에게 전송한다.

A

B



$A = g^a \ mod \ p$

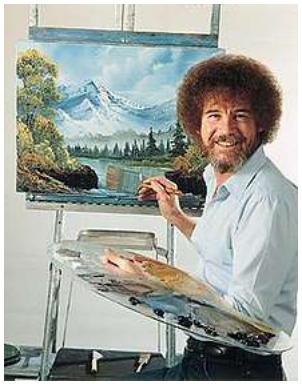
$B = g^b \ mod \ p$

1.1 Diffie-Hellman key exchange

5. 전송받은 값을 밑으로 하고 각자 개인키를 지수로 하는 거듭제곱연산한다.

A

B



$A = g^a \ mod \ p$
$B^a \ mod \ p \ \equiv g^{ab} \ mod \ p$

$B = g^b \ mod \ p$
$A^b \ mod \ p \ \equiv g^{ab} \ mod \ p$

1.1 Diffie-Hellman key exchange

- Computational Diffie-Hellman Problem (CDH)
  비밀값 a, b 없이 g^a, g^b 만으로는 g^{ab}를 계산하는 문제.
  DLP를 풀 수 있다면 CDH 문제도 풀 수 있다.

- DLP는 적어도 CDH문제 만큼 난해하다.
  CDH문제가 적어도 DLP만큼 난해한지는 아직 증명되지 않음.

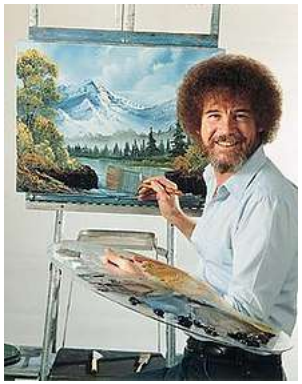
1.2 DH Man In the Middle Attack

A

D

C

B



$A = g^a \ mod \ p$

$D^a \ mod \ p \ \equiv g^{ad} \ mod \ p$

$C = g^c \ mod \ p$

$D = g^d \ mod \ p$

$A^d mod \ p \equiv g^{ad} \ mod \ p$

$B^c mod \ p \equiv g^{bc} \ mod \ p$

$B = g^b \ mod \ p$

$C^b \ mod \ p \ \equiv g^{bc} \ mod \ p$

1.2 DH Man In the Middle Attack

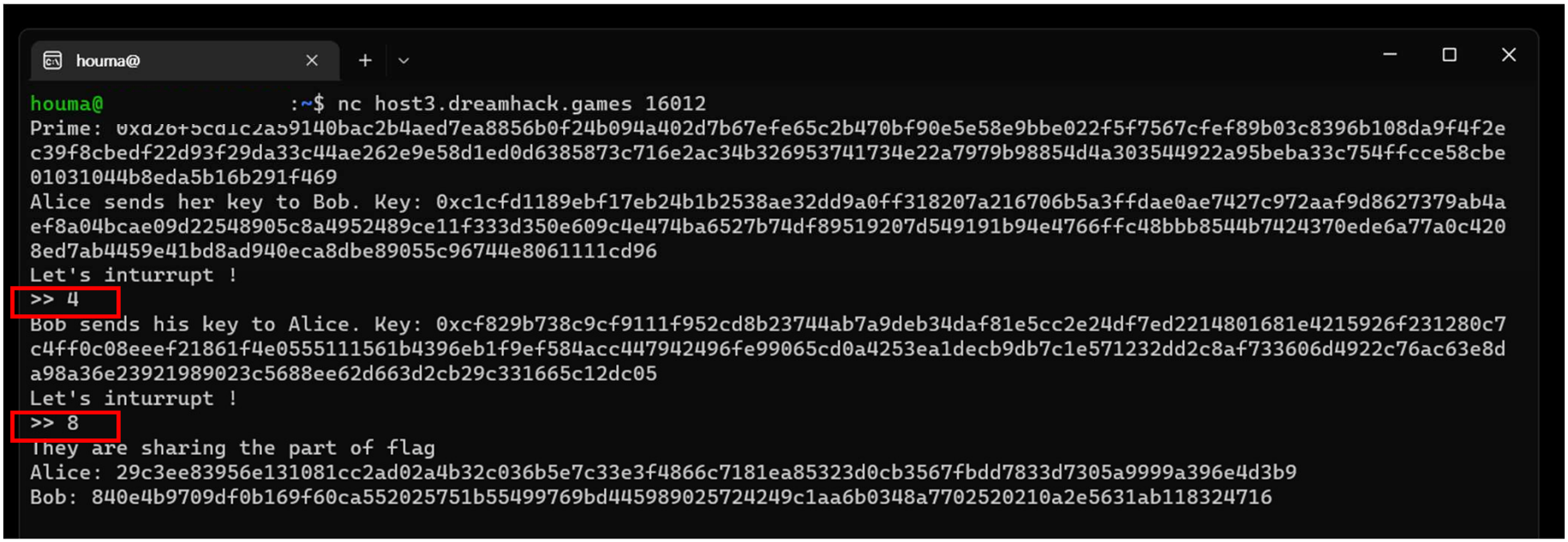
- Dreamhack:textbook-DH
  Alice와 Bob의 통신을 도청중일 때 키 교환 과정을 공격해 flag를 획득해보자.

1.2 DH Man In the Middle Attack

```python
flag = open("flag", "r").read().encode()
prime = getPrime(1024)
print(f"Prime: {hex(prime)}")
alice = Person(prime)
bob = Person(prime)

alice_k = alice.calc_key()
print(f"Alice sends her key to Bob. Key: {hex(alice_k)}")
print("Let's inturrupt !")
alice_k = int(input(">> "))
if alice_k == alice.g:
    exit("Malicious key !!")
bob.set_shared_key(alice_k)

bob_k = bob.calc_key()
print(f"Bob sends his key to Alice. Key: {hex(bob_k)}")
print("Let's inturrupt !")
bob_k = int(input(">> "))
if bob_k == bob.g:
    exit("Malicious key !!")
alice.set_shared_key(bob_k)

print("They are sharing the part of flag")
print(f"Alice: {alice.encrypt(flag[:len(flag) // 2])}")
print(f"Bob: {bob.encrypt(flag[len(flag) // 2:])}")
```

```python
class Person(object):
    def __init__(self, p):
        self.p = p
        self.g = 2
        self.x = random.randint(2, self.p - 1)

    def calc_key(self):
        self.k = pow(self.g, self.x, self.p)
        return self.k

    def set_shared_key(self, k):
        self.sk = pow(k, self.x, self.p)
        aes_key = hashlib.md5(str(self.sk).encode()).digest()
        self.cipher = AES.new(aes_key, AES.MODE_ECB)

    def encrypt(self, pt):
        return self.cipher.encrypt(pad(pt, 16)).hex()

    def decrypt(self, ct):
        return unpad(self.cipher.decrypt(bytes.fromhex(ct)), 16)
```

1.2 DH Man In the Middle Attack



$g = 2$

$C = g^2 \ mod \ Prime = 4$ $\quad\quad$ $D = g^3 \ mod \ Prime = 8$

1.2 DH Man In the Middle Attack

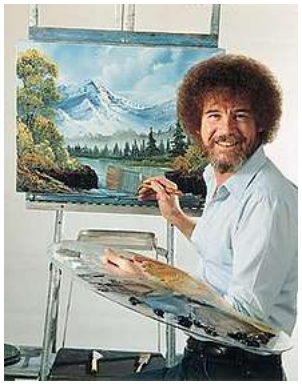
$c = 2, d = 3$

A

8

4

B



$A = 2^a \ mod \ p$

$8^a \ mod \ p \ \equiv 2^{3a} \ mod \ p$

$4 = 2^2 \ mod \ p$

$8 = 2^3 \ mod \ p$

$A^3 mod \ p \equiv 2^{3a} \ mod \ p$

$B^2 mod \ p \equiv 2^{2b} \ mod \ p$

$B = 2^b \ mod \ p$

$4^b \ mod \ p \ \equiv 2^{2b} \ mod \ p$

1.2 DH Man In the Middle Attack

```python
flag = open("flag", "r").read().encode()
prime = getPrime(1024)
print(f"Prime: {hex(prime)}")
alice = Person(prime)
bob = Person(prime)

alice_k = alice.calc_key()
print(f"Alice sends her key to Bob. Key: {hex(alice_k)}")
print("Let's inturrupt !")
alice_k = int(input(">> "))
if alice_k == alice.g:
    exit("Malicious key !!")
bob.set_shared_key(alice_k)

bob_k = bob.calc_key()
print(f"Bob sends his key to Alice. Key: {hex(bob_k)}")
print("Let's inturrupt !")
bob_k = int(input(">> "))
if bob_k == bob.g:
    exit("Malicious key !!")
alice.set_shared_key(bob_k)

print("They are sharing the part of flag")
print(f"Alice: {alice.encrypt(flag[:len(flag) // 2])}")
print(f"Bob: {bob.encrypt(flag[len(flag) // 2:])}")
```

```python
class Person(object):
    def __init__(self, p):
        self.p = p
        self.g = 2
        self.x = random.randint(2, self.p - 1)

    def calc_key(self):
        self.k = pow(self.g, self.x, self.p)
        return self.k

    def set_shared_key(self, k):
        self.sk = pow(k, self.x, self.p)
        aes_key = hashlib.md5(str(self.sk).encode()).digest()
        self.cipher = AES.new(aes_key, AES.MODE_ECB)

    def encrypt(self, pt):
        return self.cipher.encrypt(pad(pt, 16)).hex()

    def decrypt(self, ct):
        return unpad(self.cipher.decrypt(bytes.fromhex(ct)), 16)
```

1.2 DH Man In the Middle Attack

```python
prime = 0xd26f5cd1c2a59140bac2b4aed7ea8856b0f24b094a402d7b67efe65c2b470bf90e5e58e9bbe022f5f7567cfef89b03c8396b108
A = 0xc1cfd1189ebf17eb24b1b2538ae32dd9a0ff318207a216706b5a3ffdae0ae7427c972aaf9d8627379ab4aef8a04bcae09d22548905c
B = 0xcf829b738c9cf9111f952cd8b23744ab7a9deb34daf81e5cc2e24df7ed2214801681e4215926f231280c7c4ff0c08eeef21861f4e05


Alice_flag = '29c3ee83956e131081cc2ad02a4b32c036b5e7c33e3f4866c7181ea85323d0cb3567fbdd7833d7305a9999a396e4d3b9'
Bob_flag = '840e4b9709df0b169f60ca552025751b55499769bd445989025724249c1aa6b0348a7702520210a2e5631ab118324716'

c = 2; d = 3
Asharedkey = pow(A, d, prime)
Aaes_key = hashlib.md5(str(Asharedkey).encode()).digest()
Acipher = AES.new(Aaes_key, AES.MODE_ECB)

Bsharedkey = pow(B, c, prime)
Baes_key = hashlib.md5(str(Bsharedkey).encode()).digest()
Bcipher = AES.new(Baes_key, AES.MODE_ECB)

Aflag = Acipher.decrypt(bytes.fromhex(Alice_flag))
bflag = Bcipher.decrypt(bytes.fromhex(Bob_flag))

print(Aflag
print(bflag
```

```
houma@                MINGW64 /
$ python solution.py
b'DH{6625fe941be9a9ad76f0ca7b3250d83\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
b'd2f8f33ccee8aa5a55a605a3a8e9b7104}\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
```

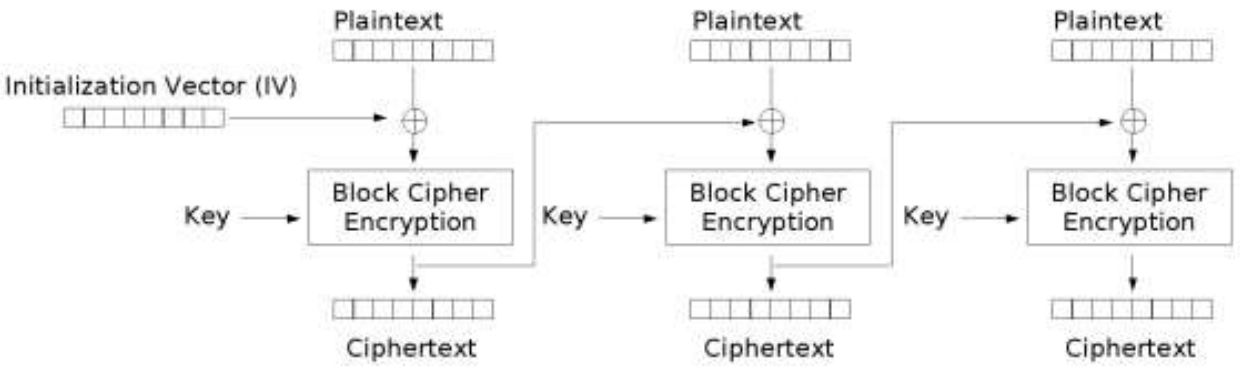
2. CBC Bit-Flipping Attack

2.1 CBC mode

2.2 CBC Bit-Flipping Attack

2.1 CBC mode

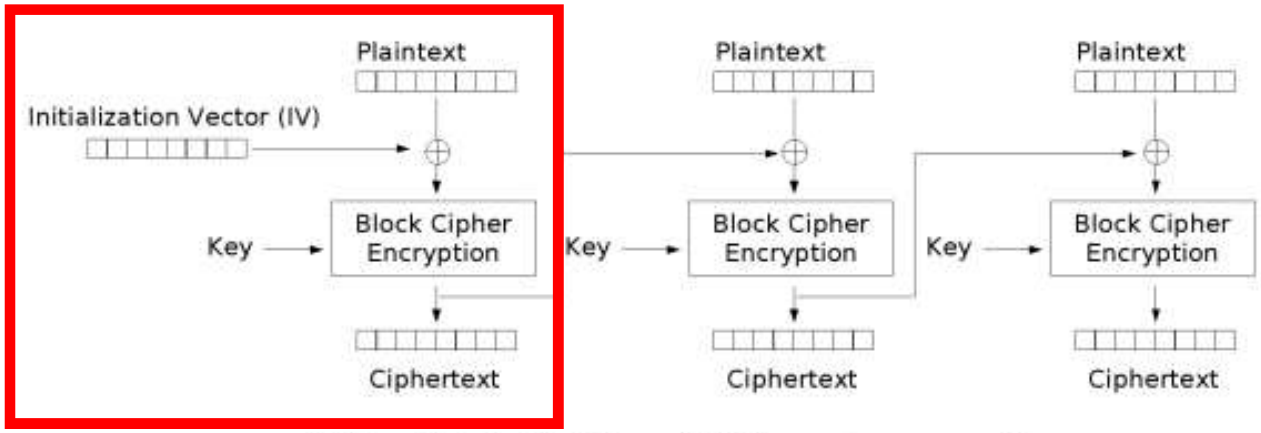
- Cipher Block chaining(CBC) mode



Cipher Block Chaining (CBC) mode encryption

2.1 CBC mode

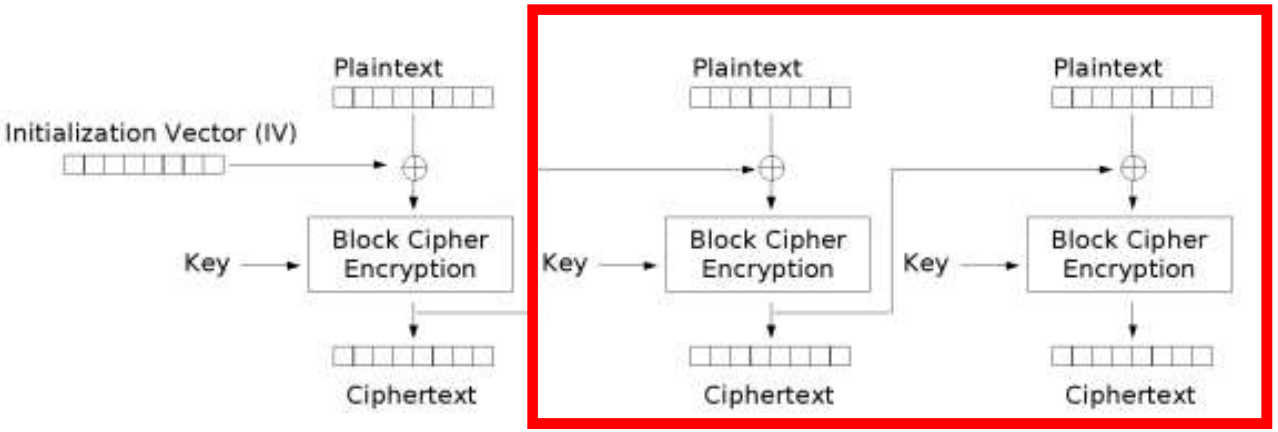
$$CT_0 = Enc(PT_0 \oplus IV, key)$$



Cipher Block Chaining (CBC) mode encryption

2.1 CBC mode

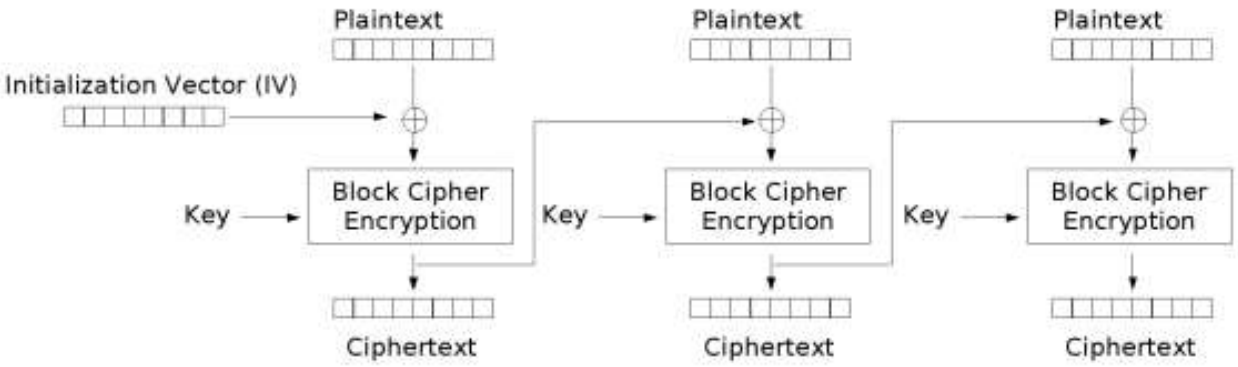
$$CT_i = Enc(PT_i \oplus CT_{i-1}, key)\ (1 \leq i)$$



Cipher Block Chaining (CBC) mode encryption

2.1 CBC mode

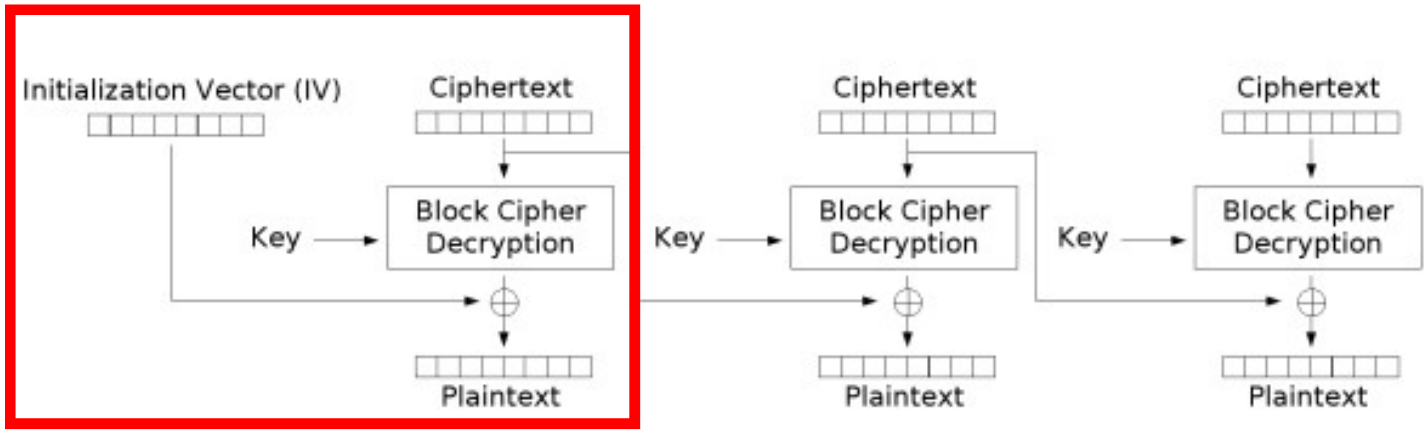
$$CT_0 = Enc(PT_0 \oplus IV, key)$$
$$CT_i = Enc(PT_i \oplus CT_{i-1}, key)\ (1 \leq i)$$



Cipher Block Chaining (CBC) mode encryption

2.1 CBC mode

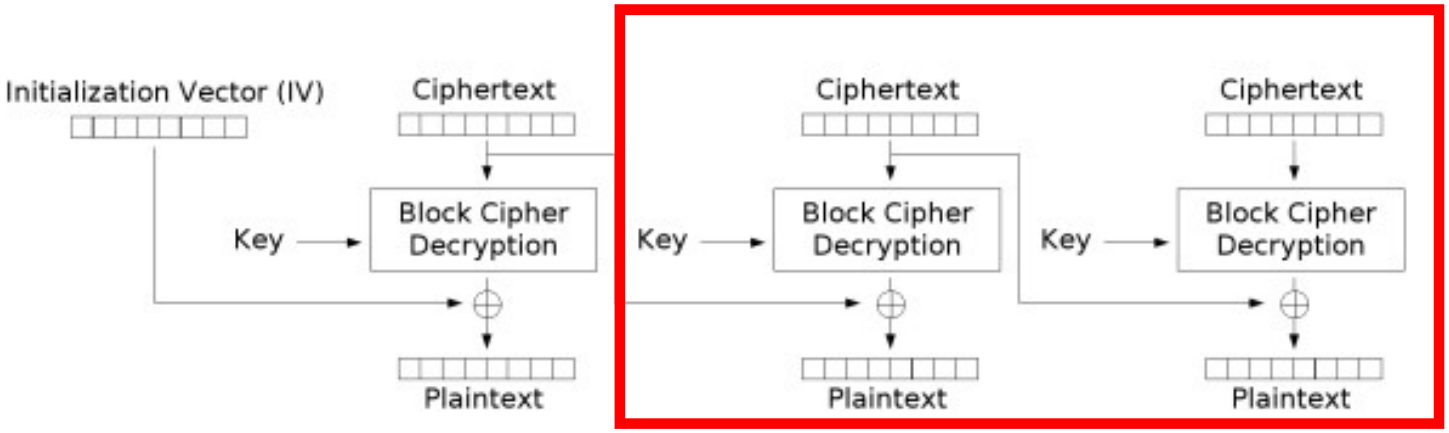
$$PT_0 = Dec(CT_0, key) \oplus IV$$



Cipher Block Chaining (CBC) mode decryption

2.1 CBC mode

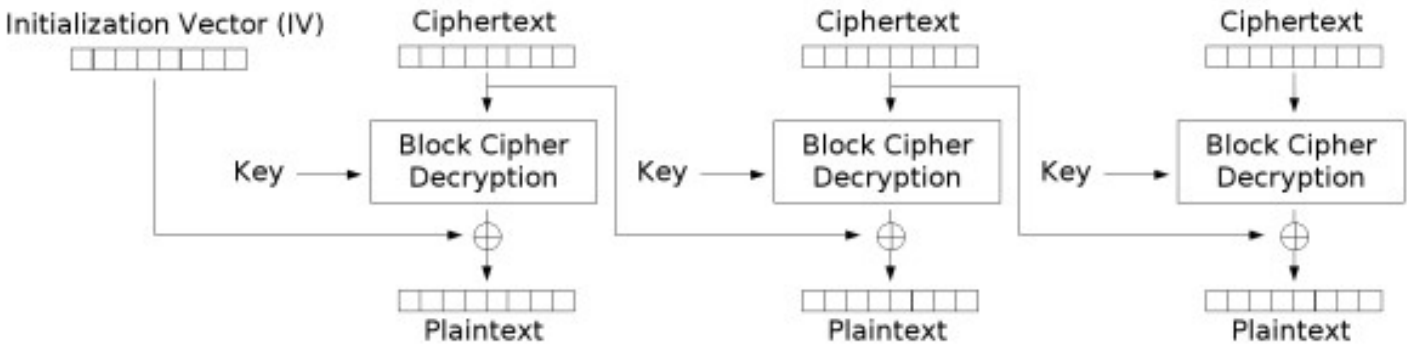
$$PT_i = Dec(CT_i, key) \oplus CT_{i-1} \; (1 \leq i)$$



Cipher Block Chaining (CBC) mode decryption

2.1 CBC mode

$$PT_0 = Dec(CT_0, key) \oplus IV$$
$$PT_i = Dec(CT_i, key) \oplus CT_{i-1}\ (1 \leq i)$$



Cipher Block Chaining (CBC) mode decryption

2.2 CBC Bit-Flipping Attack

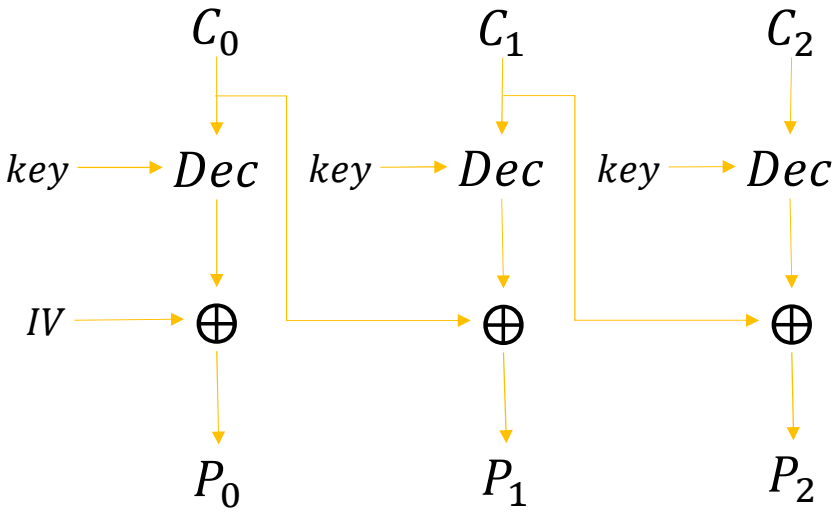
- 암호문을 수정하여 복호화된 평문에 예측가능한 영향을 끼치는 공격.
  공격자는 메시지 복호화를 하지 않더라도 평문을 수정한다.

- 공격자가 평문의 형태를 알고 있다면, 원하는 부분을 수정할 수 있다.

- 암호는 인증이 아님을 입증하는 공격.
  단순히 메시지를 암호화하는 것만으로는 충분하지 않다.

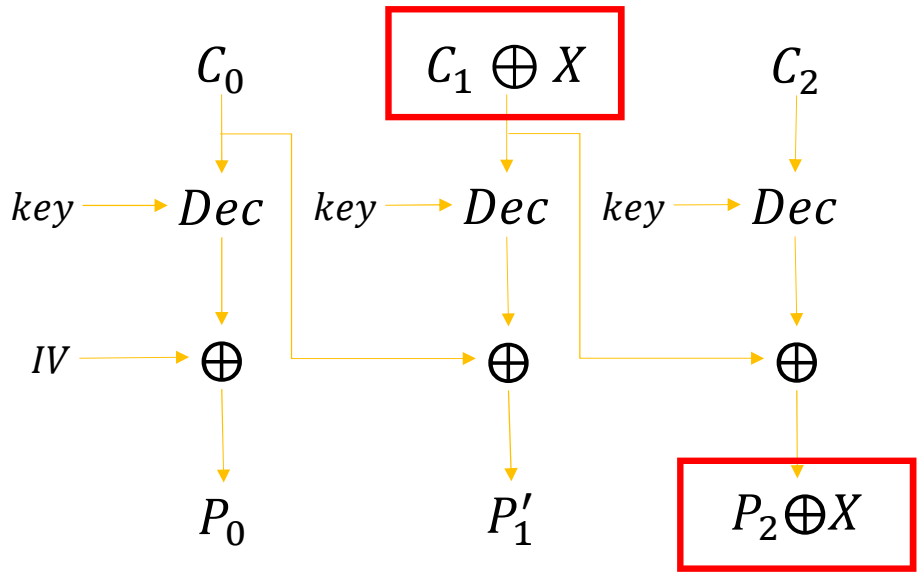
2.2 CBC Bit-Flipping Attack

- 복호화 과정



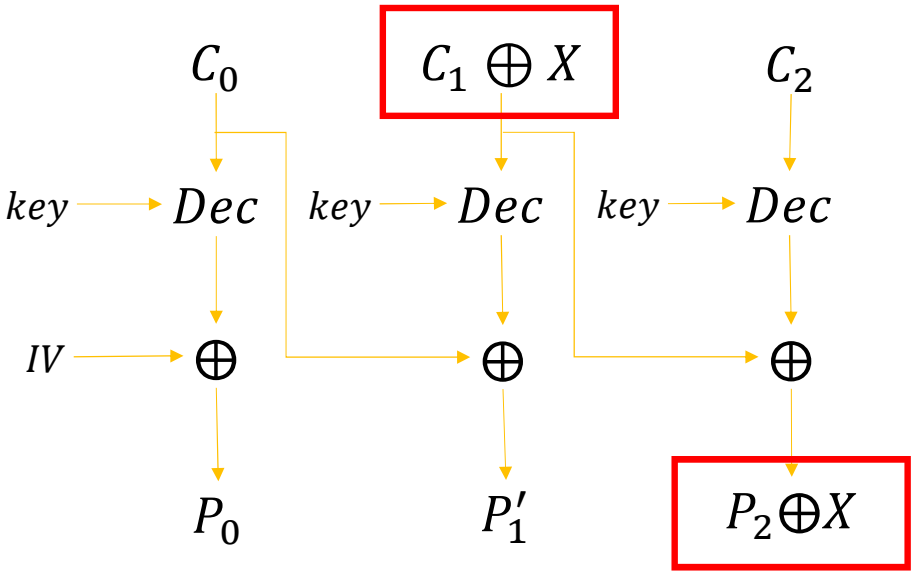
2.2 CBC Bit-Flipping Attack

- 복호화 과정



2.2 CBC Bit-Flipping Attack

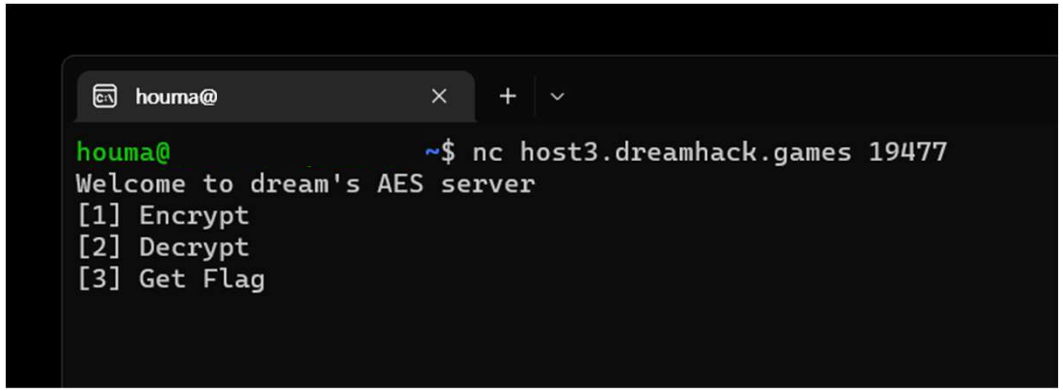
- 복호화 과정



$$P_2 = Dec(C_2, key) \oplus C_1$$
$$Dec(C_2, key) \oplus (C_1 \oplus X) = P_2 \oplus X$$

2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



2.2 CBC Bit-Flipping Attack

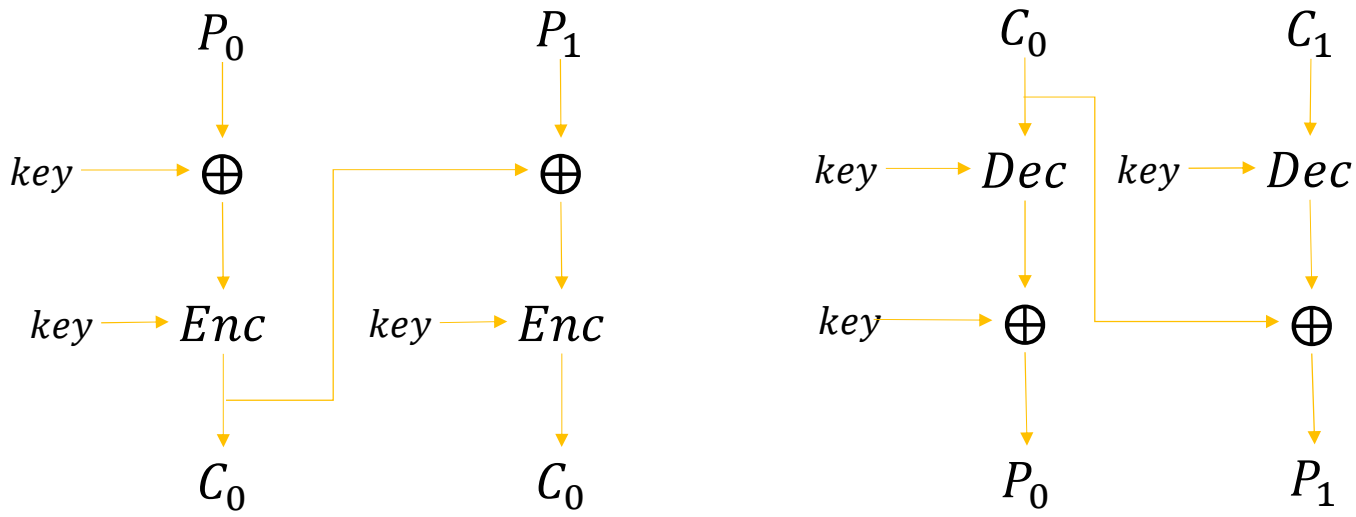
- Dreamhack:textbook-CBC

```python

BLOCK_SIZE = 16
flag = open("flag", "rb").read()
key = bytes(randint(0, 255) for i in range(BLOCK_SIZE))

encrypt = lambda pt: AES.new(key, AES.MODE_CBC, key).encrypt(pad(pt, BLOCK_SIZE))
decrypt = lambda ct: unpad(AES.new(key, AES.MODE_CBC, key).decrypt(ct), BLOCK_SIZE)

print("Welcome to dream's AES server")
```

```python
print("Welcome to dream's AES server")
while True:
    print("[1] Encrypt")
    print("[2] Decrypt")
    print("[3] Get Flag")

    choice = input()

    if choice == "1":
        print("Input plaintext (hex): ", end="")
        pt = bytes.fromhex(input())
        print(encrypt(pt).hex())

    elif choice == "2":
        print("Input ciphertext (hex): ", end="")
        ct = bytes.fromhex(input())
        print(decrypt(ct).hex())

    elif choice == "3":
        print(f"flag = {encrypt(flag).hex()}")
        exit()

    else:
        print("Nope")
```

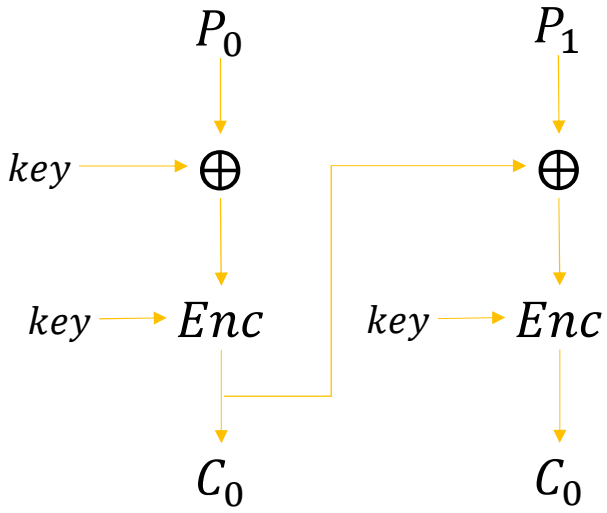
2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



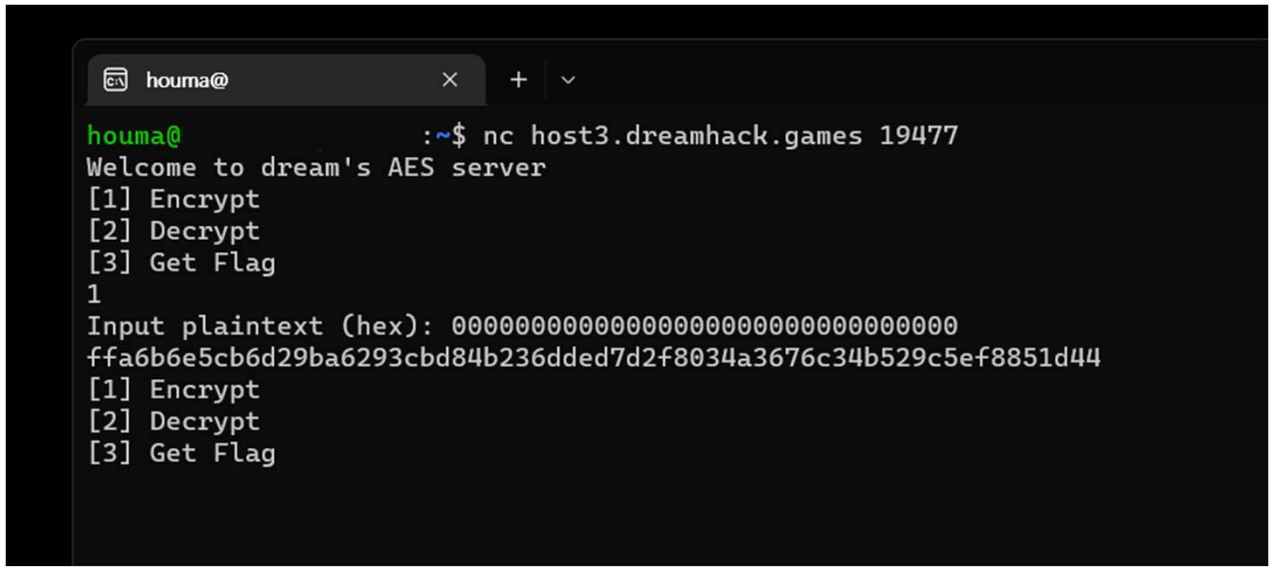
$P_0 = 0x00000000000000000000000000000000$ 일때,

$$P_0 \oplus key = key$$
$$C_0 = Enc(P_0 \oplus key, key)$$
$$C_0 = Enc(key, key)$$

즉, C_0는 key를 암호화한 값이 된다.

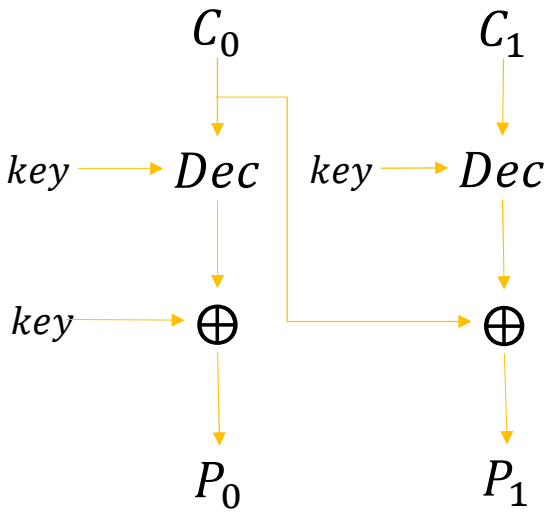
2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



2.2 CBC Bit-Flipping Attack

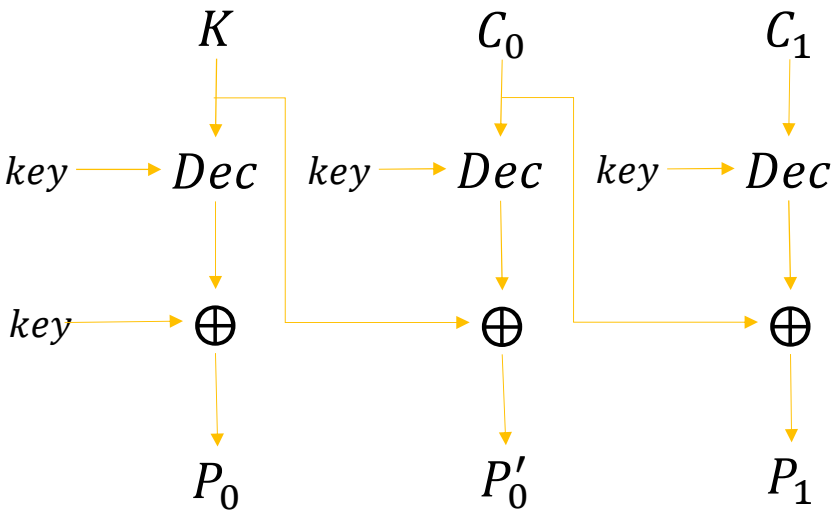
- Dreamhack:textbook-CBC



C_0는 key값을 암호화한 값이고,
C_1은 Padding을 암호화한 값이다.

2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC

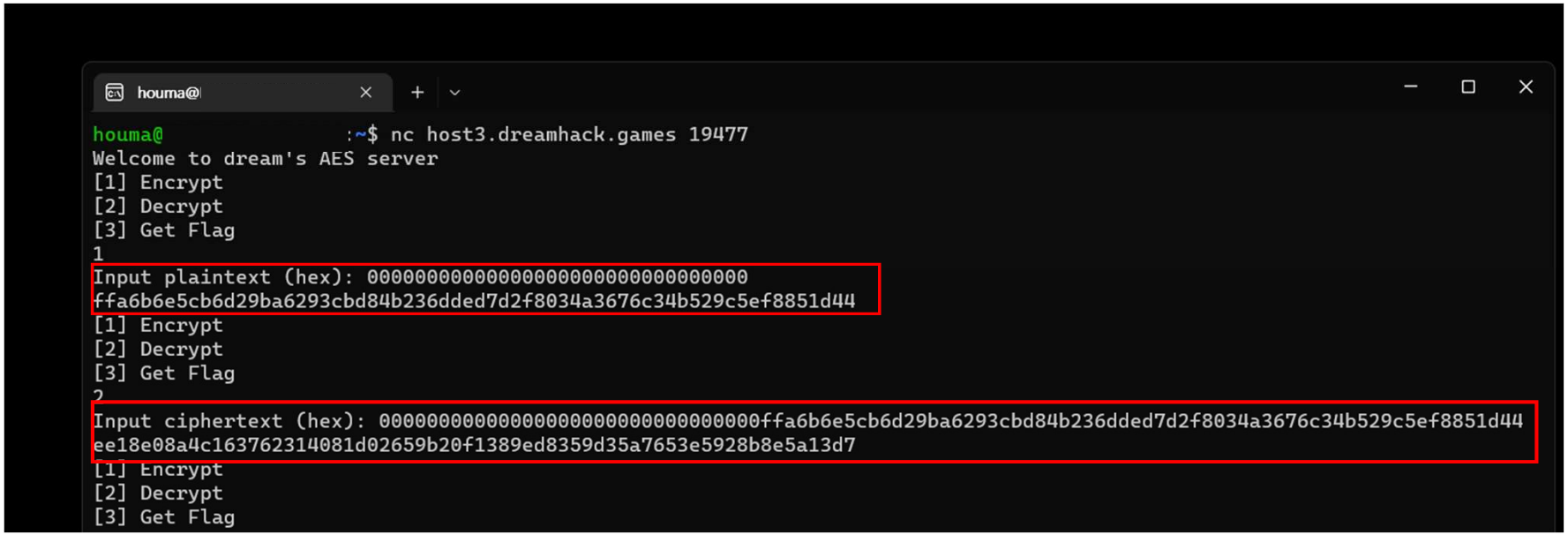


암호문$C_0||C_1$앞에
$K = 0x00000000000000000000000000000000$를 붙여
$K||C_0||C_1$을 들어 복호화를 한다.

C_0는 key를 암호화한 것으로, 복호화 한 후
K와 xor 연산을 하면 P_0'은 key값을 얻을 수 있다.

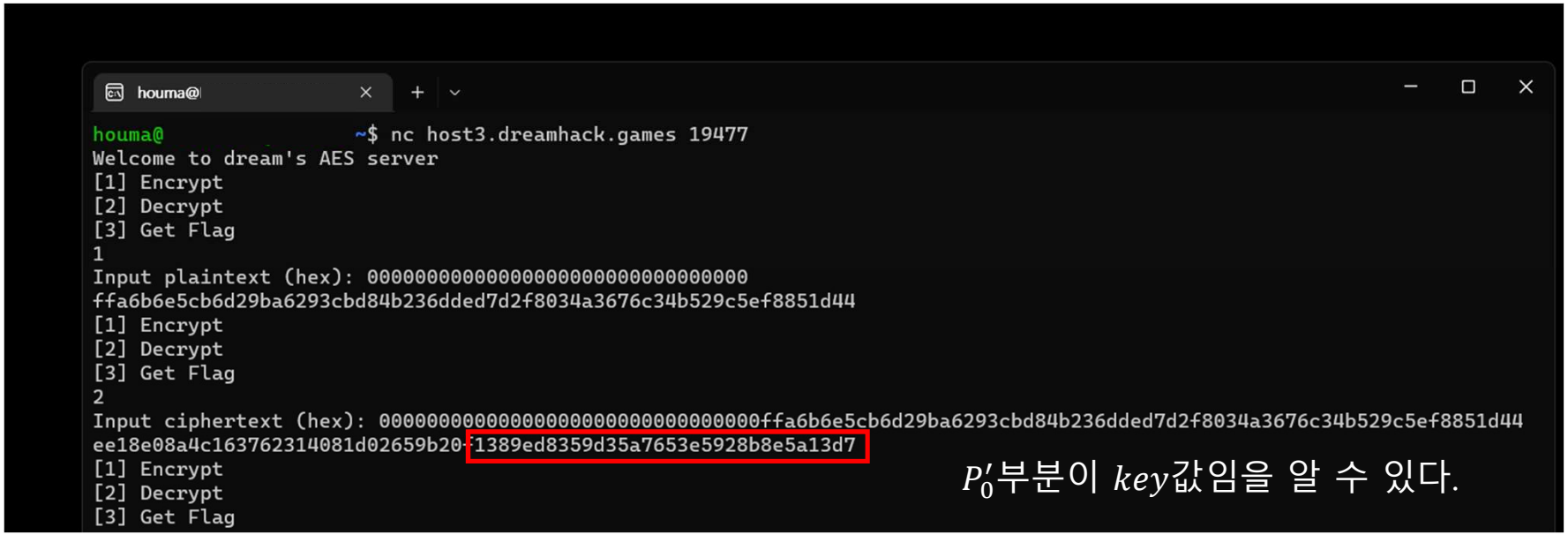
2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



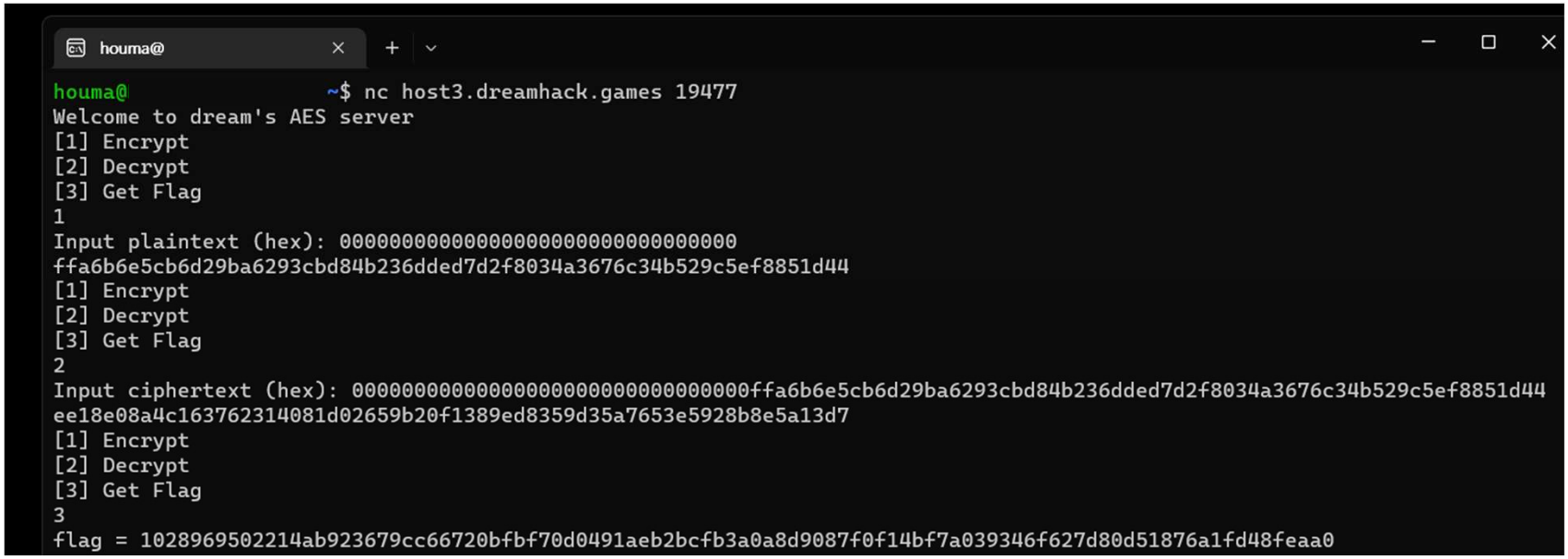
2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC



2.2 CBC Bit-Flipping Attack

- Dreamhack:textbook-CBC

```python
from Crypto.Util.Padding import pad, unpad
from Crypto.Cipher import AES
from pwn import *

encrypt = lambda pt: AES.new(key, AES.MODE_CBC, key).encrypt(pad(pt, 16))
decrypt = lambda ct: unpad(AES.new(key, AES.MODE_CBC, key).decrypt(ct), 16)

key = bytes.fromhex('ee18e08a4c163762314081d02659b20f1389ed8359d35a7653e5928b8e5a13d7')[16:32]
flagct = bytes.fromhex('1028969502214ab923679cc66720bfbf70d0491aeb2bcfb3a0a8d9087f0f14bf7a039346f627d80d51876a1fd48feaa0')

print(decrypt(flagct))

```

```
문제 7  출력  디버그 콘솔  터미널  JUPYTER                    bash

(base)
houma@                MINGW64 /
$ python test.py
b'DH{9666eb07031fbc855428e2223946a4b8}\n'
```

감사합니다.