Information Security HW2

Problem 1: Pseudo Random Function

$F'(k,(x,x)) = F(k,x) \text{ xor } F(k,x) = 0$ for all x. 所以當我們傳送多個 $F'(k,(x,x))$ 並都是得到 0 這個結果，我們便可以確認他是 $F'(k,(x,y))$。 因此不安全

Problem 2: Triple Encryption

假設 所有的一個 key組合都可以 map 到一個三種 key的組合，所以會撞到這把 key 的機率是 $\frac{2^{56}}{2^{56*3}} = \frac{1}{2^{112}}$ 存在的機率過小可以忽略，而且前面有假設每一個 key 的組合一定可以對到某個三把 key的組合，但實際上可能不是每把 key都有這種組合，因此機率只會更低。

Problem 3: Ciphertext Stealing

由於該方法只會影響到倒數兩塊，因此只討論倒數兩塊的行為

加密:

假設需要 padding T 位

1. P_{n-1} 加密後得到 C_{n-1}
2. 將 C_{n-1} 最後 T bits 貼到 P_n 最後面 作為 padding
3. 對最後一塊 P_n 正常加密得到 C_n，並刪除 C_{n-1}的最後 T bits s以符合原始長度

解密:

1. 將最後一塊解密得到P_n，把最後 T bits 貼回 C_{n-1}
2. 正常對C_{n-1}解密就可以得到 P_{n-1}，

Problem 4: Extended Euclidean Algorithm

- 6
- 1075
- 1844

Problem 5: Euler's Theorem and RSA

仍然適用：
m, N 不互質，分兩種情況討論

- m 與 p 不互質(同等與 q 不互質):

$n = kp$
$n^{ed} = (kp)^{ed} = 0 = kp = n \pmod p$
$n^{ed} = n^{ed-1}n = n^{k(q-1)}n = 1^k n = n \pmod q$

根據中國剩餘定理， $n^{ed} = n \pmod N$

- m 與 p, q 不互質:

$n^{ed} = (kpq)^e d = 0 = kpq \ (\mathrm{mod} N)$ k 必須 = 1，也就是說 $n \leq pq$，以上才合理

Problem 6: Elliptic Curve

我們用 c 取代 λ

$y^2 \bmod p = (x^3 + ax + b) \bmod p$
R, P, Q 共線，假設共線於 $y = cx + d$，得下一個交點等式為:
$(cx + d)^2 = x^3 + ax + b \Rightarrow (cx)^2 + 2cdx + d^2 = x^3 + ax + b$ where $c = \frac{y_P - y_Q}{x_P - x_Q}$

$x^3 - c^2x^2 - 2cdx + ax + b - d^2$
同時也是三個點的的解
$(x - x_P)(x - x_Q)(x - x_R) = x^3 + x^2(-x_P - x_Q - x_R) + x(x_Px_Q + x_Px_R + x_Qx_R) - x_P$

對照係數得:
$-c^2 = -x_P - x_Q - x_R \Rightarrow x_R = (c^2 - x_P - x_Q) \bmod p$
且共線得:
$y_R = (c(x_R - x_P) + y_P)$

If $P = Q$, tengent line:

$y^2 \bmod p = (x^3 + ax + b) \bmod p$ => 微分 => $c = \frac{3x_P^2 + a}{2y_P}$

Problem 7: Common Modulus Protocol Failure

$x_2e_2 = x_1e_1 - 1$
$c_1^{x_1}(c_2^{x_2})^{-1} \bmod n = x^{e_1x_1}(x^{e_2x_2})^{-1} \bmod n = x^{e_1x_1}(x^{e_1x_1-1})^{-1} \bmod n = x$

Problem 7: SEED Lab

Lab 1

```
echo -n "12345" > P
openssl enc -aes-128-cbc -e -in P -out C
openssl enc -aes-128-cbc -d -nopad -in C -out P_new
xxd P_new
echo -n "1234546890" > P
openssl enc -aes-128-cbc -e -in P -out C
openssl enc -aes-128-cbc -d -nopad -in C -out P_new
xxd P_new
echo -n "1234567890123456" > P
openssl enc -aes-128-cbc -e -in P -out C
openssl enc -aes-128-cbc -d -nopad -in C -out P_new
xxd P_new
```

```
[04/16/23]seed@VM:~/Lab/Labsetup$ ./test.sh
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
00000000: 3132 3334 3534 3638 3930 0606 0606 0606  1234546890......
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
```

1. 11 * 0b
2. 6 * 0b
3. None

Lab 2

- Logic

1. 當 K = 1 的時候，得到結果為 0xcf, 代表 padding 為 1 位時的 CC1[15] ^ D2[15] = 1，可以反推原本的 D2[15] = 0xcf ^ 0x01
2. 當 K = 2 的時候，手動修改 CC1[15] = 0xcf ^ 0x01 ^ 0x02，讓結果最後保證為 2 位的答案，得到 0x39 => CC1[14] ^ D2[14] = 0x02 => D2[14] = 0x39 ^ 0x2
3. 當 K = 3 的時後，手動修改 CC1[15] = 0xcf ^ 0x01 ^ 0x03, CC1[14] = 0x39 ^ 0x02 ^ 0x03，得到 0xf2 => CC1[13] ^ D2[13] = 0x03 => D2[13] = 0xf2 ^ 0x03
4. ... 以此類推

- D's

```
D2[10] = 0xea ^ 0x06
D2[11] = 0x40 ^ 0x05
D2[12] = 0x18 ^ 0x04
D2[13] = 0xf2 ^ 0x03
D2[14] = 0x39 ^ 0x02
D2[15] = 0xcf ^ 0x01
#####################################
# In the experiment, we need to iterat
# We will send this CC1 to the oracle,
CC1 = bytearray(16)

CC1[0]  = 0x00
CC1[1]  = 0x00
CC1[2]  = 0x00
CC1[3]  = 0x00
CC1[4]  = 0x00
CC1[5]  = 0x00
CC1[6]  = 0x00
CC1[7]  = 0x00
CC1[8]  = 0x00
CC1[9]  = 0x00
CC1[10] = 0xea
CC1[11] = 0x40 ^ 0x05 ^ 0x06
CC1[12] = 0x18 ^ 0x04 ^ 0x06
CC1[13] = 0xf2 ^ 0x03 ^ 0x06
CC1[14] = 0x39 ^ 0x02 ^ 0x06
CC1[15] = 0xcf ^ 0x01 ^ 0x06
```

- Result

```
C1:   a9b2554b09441180612120098f2f238cd
C2:   779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0xea
CC1:  00000000000000000000ea431af73dc8
P2:   00000000000000000000ccddee030303
```

Lab 3

- Logic

把 Lab 2 轉成程式，當 k = n 的時候 替換 CC1[16 - n + 1 : 16] = D[16 - n + 1 : 16] ^ n 然後用 Lab 2 的邏輯爆搜結果回填 D[16 - n]

- Code

```python
#!/usr/bin/python3
import socket
from binascii import hexlify, unhexlify

# XOR two bytearrays
def xor(first, second):
   return bytearray(x^y for x,y in zip(first, second))

class PaddingOracle:

    def init(self, host, port) -> None:
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.s.connect((host, port))

        ciphertext = self.s.recv(4096).decode().strip()
        self.ctext = unhexlify(ciphertext)

    def decrypt(self, ctext: bytes) -> None:
        self._send(hexlify(ctext))
        return self._recv()

    def _recv(self):
        resp = self.s.recv(4096).decode().strip()
        return resp

    def _send(self, hexstr: bytes):
        self.s.send(hexstr + b'\n')

    def del(self):
        self.s.close()


if name == "main":
    oracle = PaddingOracle('10.9.0.80', 6000)

    # Get the IV + Ciphertext from the oracle
    iv_and_ctext = bytearray(oracle.ctext)
    IV    = iv_and_ctext[00:16]
    C1    = iv_and_ctext[16:32]  # 1st block of ciphertext
    C2    = iv_and_ctext[32:48]  # 2nd block of ciphertext
    C3    = iv_and_ctext[48:64]
    print("C1:  " + C1.hex())
    print("C2:  " + C2.hex())
    print("C3:  " + C3.hex())

    ###############################################################
    # Here, we initialize D2 with C1, so when they are XOR-ed,
    # The result is 0. This is not required for the attack.
    # Its sole purpose is to make the printout look neat.
    # In the experiment, we will iteratively replace these values.
    D2 = bytearray(16)
    D1 = bytearray(16)
    D3 = bytearray(16)

    ###############################################################
    # In the experiment, we need to iteratively modify CC1
    # We will send this CC1 to the oracle, and see its response.
    CC1 = bytearray(16)

    ###############################################################
    # In each iteration, we focus on one byte of CC1.
    # We will try all 256 possible values, and send the constructed
    # ciphertext CC1 + C2 (plus the IV) to the oracle, and see
    # which value makes the padding valid.
    # As long as our construction is correct, there will be
    # one valid value. This value helps us get one byte of D2.
    # Repeating the method for 16 times, we get all the 16 bytes of D2.
    for K in range(1, 17):
```

```python
         for i in range(1, K):
            CC1[16 - i] = D3[16 - i] ^ K
         for i in range(256):
            CC1[16 - K] = i
            status = oracle.decrypt(IV + C1 + CC1 + C3)
            if status == "Valid":
               print("Valid: i = 0x{:02x}".format(i))
               print("CC1: " + CC1.hex())
               D3[16 - K] = CC1[16 - K] ^ K

      for K in range(1, 17):
         for i in range(1, K):
            CC1[16 - i] = D2[16 - i] ^ K
         for i in range(256):
            CC1[16 - K] = i
            status = oracle.decrypt(IV + C1 + CC1 + C2)
            if status == "Valid":
               print("Valid: i = 0x{:02x}".format(i))
               print("CC1: " + CC1.hex())
               D2[16 - K] = CC1[16 - K] ^ K

      for K in range(1, 17):
         for i in range(1, K):
            CC1[16 - i] = D1[16 - i] ^ K
         for i in range(256):
            CC1[16 - K] = i
            status = oracle.decrypt(IV + C1 + CC1 + C1)
            if status == "Valid":
               print("Valid: i = 0x{:02x}".format(i))
               print("CC1: " + CC1.hex())
               D1[16 - K] = CC1[16 - K] ^ K


      ###############################################################

      # Once you get all the 16 bytes of D2, you can easily get P2
      P3 = xor(C2, D3)
      P2 = xor(C1, D2)
      P1 = xor(IV, D1)
      print("P1: " + P1.hex())
      print("P2: " + P2.hex())
      print("P3: " + P3.hex())
```

- Result:

```
P1: 285e5f5e29285e5f5e29205468652053
P2: 454544204c6162732061726520677265
P3: 61742120285e5f5e29285e5f5e290202
```