Crypto Engineering Midterm Exam

April, 2023

Student Name : 游建峰

Student ID : 110550164

Department : 資工系

For the following Questions, please write down your answers with explanations

Question 1 :

Let us see what goes wrong when a stream cipher key is used more than once. Below are eleven hex-encoded ciphertexts that are the result of encrypting eleven plaintexts with a stream cipher, all with the same stream cipher key. Your goal is to decrypt the last ciphertext, and submit the secret message within it as solution.

Hint: XOR the ciphertexts together, and consider what happens when a space is XORed with a character in [a-zA-Z].

Ciphertext #1 :

315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b
16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137
dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418
ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f652
86764703de0f3d524400a19b159610b11ef3e

Ciphertext #2 :

234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b
12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29
c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b
2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f

Ciphertext #3 :

32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad
40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c86
6a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5
f41afa8d661cb

Ciphertext #4 :

32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5e
a06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a3
7bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef5204
89e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447
e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa

Ciphertext #5 :

3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe
56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b9
3ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302
cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c
2070

Ciphertext #6 :

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24e
b76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822
a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111
284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e6
93522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc22
9f77ace7aa88a2f19983122b11be87a59c355d25f8e4

Ciphertext #7 :

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba
47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b52
9c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039f
a2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934
777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb02893283
6fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c866
1dd9a4ce

Ciphertext #8 :

315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54c
e56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f
89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9a
a2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565
517f317da9d3

Ciphertext #9 :

271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4a
ad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb
537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f45641
86a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f65
2b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d020
8573a7eef027

Ciphertext #10 :

466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409e
d39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03a
dbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a
2cb2a83

Target Ciphertext (decrypt this one) :

32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85a
ac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b5
7bddef121336cb85ccb8f3315f4b52e301d16e9f52f904

For completeness, here is the python2 script used to generate the ciphertexts. (it doesn't matter if you can't read this)

```
import sys

MSGS = ( ---    11 secret messages    --- )

def strxor(a, b):          # xor two strings (trims the longer input)
        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b)])

def random(size=16):
        return open("/dev/urandom").read(size)

def encrypt(key, msg):
        c = strxor(key, msg)
        print
        print c.encode('hex')
        return c

def main():
        key = random(1024)
        ciphertexts = [encrypt(key, msg) for msg in MSGS]
```

Ans.

```
1  | We can aactor the number 5 with quantum computers. We can also factor the number 1
2  | Euler whuld probably enjoF that now his theorem becomes a corner stone of crypto -
3  | The nicb thing about KeeySoq is now we cryptographers can drive a lot of fancy cars
4  | The cipoertext produced bF a weak encryption algorithm looks as good as ciphertext
5  | You don t want to buy a sZt of car keys from a guy who specializes in stealing cars
6  | There aue two types of crFptography - that which will keep secrets safe from your l
7  | There aue two types of cyOtography: one that allows the Government to use brute for
8  | We can tee the point wherZ the chip is unhappy if a wrong bit is sent and consumes
9  | A (privfte-key)  encryptiPn scheme states 3 algorithms, namely a procedure for gene
10 |  The Coicise OxfordDictioQary (2006) deï¬nes crypto as the art of  writing o r sol
11 | The secuet message is: WhZn using a stream cipher, never use the key more than once

The plaintext is:
The secret message is: When using a stream cipher, never use the key more than once
```

第 11 則 ciphertext 在分析過後可以得出以下 plaintext:

The secret message is: When using a stream cipher, never use the key more than once

第一題的 code:

```python
import string
from itertools import combinations
from collections import defaultdict

CIPHERTEXTS = [
    bytes.fromhex("315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba50"),
    bytes.fromhex("234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb741"),
    bytes.fromhex("32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de812"),
    bytes.fromhex("32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee41"),
    bytes.fromhex("3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de812"),
    bytes.fromhex("32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d"),
    bytes.fromhex("32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af513"),
    bytes.fromhex("315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e941"),
    bytes.fromhex("271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f404"),
    bytes.fromhex("466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d"),
    bytes.fromhex("32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904"),  # <- target
    ]

def possible_xor_results(s: str) -> defaultdict[int, list[tuple[int, int]]]:
    charset = [ord(c) for c in s]
    res = defaultdict(list)
    for a, b in combinations(charset, r=2):
        res[a ^ b] += [(a, b)]
    res[0] = [(a, a) for a in charset]
    return res

if __name__=="__main__":
    possible_xor_pairs = possible_xor_results(string.ascii_letters + " ")

    possible_keys = [defaultdict(int) for _ in range(len(CIPHERTEXTS[0]))]
    for c1, c2 in combinations(CIPHERTEXTS, r=2):
        for keys, b1, b2 in zip(possible_keys, c1, c2):
            possible_cipher_chars = set()
            for x,y in possible_xor_pairs[b1^b2]:
                possible_cipher_chars |= {b1^x, b1^y}
            for c in possible_cipher_chars:
                keys[c] += 1

    # guess the cipher byte that was possible in the most ciphertext pairs
    key_guess = [max(keys, key=keys.get) for keys in possible_keys]
    for i, cs in enumerate(CIPHERTEXTS):
        plaintext_guess = ''.join(chr(x ^ k) for x, k in zip(cs, key_guess))
        print(str(i+1).ljust(2), "|", plaintext_guess)

    print('\n'"The plaintext is: ")
    print("The secret message is: When using a stream cipher, never use the key more than once")
```

Question 2 :

Suppose you are told that the one-time pad encryption of the message “attack at dawn” is “09e1c5f70a65ac519458e7e53f36” (the plaintext letters are encoded as 8-bit ASCII and the given ciphertext is written in hex). What would be the one-time pad encryption of the message “attack at dusk” under the same OTP key?

Ans.

```
The one-time pad encryption of the message "attack at dusk" under the same OTP key is:
0x9e1c5f70a65ac519458e7f13b33
```

第二題的 code:

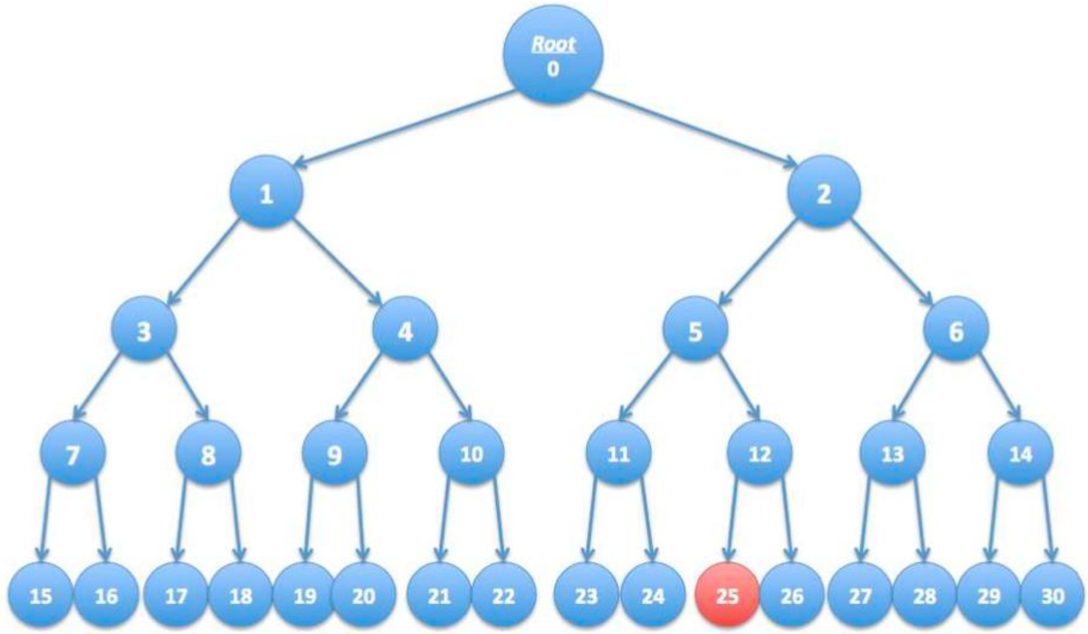
```python
def str_to_int(s):
    return int(s.encode().hex(), 16)

if __name__=="__main__":
    message = 0x09e1c5f70a65ac519458e7e53f36

    key = str_to_int("attack at dawn") ^ message

    print("The one-time pad encryption of the message \"attack at dusk\" under the same OTP key is:")
    print(hex(str_to_int("attack at dusk") ^ key))
```

Question 3 :

The movie industry wants to protect digital content distributed on DVD's. We develop a variant of a method used to protect Blu-ray disks called AACS.
Suppose there are at most a total of n DVD players in the world (e.g. $\mathrm{n} = 2^{32}$). We view these n players as the leaves of a binary tree of height $log_2\mathrm{n}$. Each node in this binary tree contains an AES key k_i. These keys are kept secret from consumers and are fixed for all time. At manufacturing time each DVD player assigned a serial number $\mathrm{i} \in [0, \mathrm{n} - 1]$. Consider the set of nodes S_i along the path from the root to leaf number i in the binary tree. The manufacturer of the DVD player embeds in player number i the keys associated with the nodes in the set S_i. A DVD movie m is encrypted as $E(k_{root}, k) \parallel E(k, m)$ where k is a random AES key called a content-key and k_{root} is the key associated with the root of the tree. Since all DVD players have the key k_{root} all players can decrypt the movie m. We refer to $E(k_{root}, k)$ as the header and $E(k, m)$ as the body. In what follows the DVD header may contain multiple ciphertexts which each ciphertext is the encryption of the content-key k under some key k_i in the binary tree.
Suppose the keys embedded in DVD player number r are exposed by hackers and published on the Internet. In this problem we show that when the movie industry distributes a new DVD movie, they can encrypt the contents of the DVD using a slightly larger header (containing about $log_2 n$ keys) so that all DVD players, except for player number r, can decrypt the movie. In effect, the movie industry disables player number r without affecting other players.

As shown below, consider a tree with $\mathrm{n} = 16$ leaves. Suppose the leaf node labeled 25 corresponds to an exposed DVD player key. Check the set of keys below under which to encrypt the key k so that every player other than player 25 can decrypt the DVD. Only four keys are needed.

因為總共從 root 到 leaf 的 path 上有 $log_2 n$ 個 node，因此需要



(A) 14 (B) 28 (C) 11 (D) 4 (E) 6 (F) 2 (G) 26 (H) 1

Ans. (C)11, (E)6, (G)26, (H)1

因為要讓 25 沒辦法解密，就必須選擇 child 中沒有包括 25 的 key，

而最少的選擇是 1, 6, 11, 26，也就是能夠讓除了 25 以外的人都能夠

decrypt DVD。

Question 4 :

Continuing with the previous question, if there are n DVD players, what is the number of keys under which the content key k must be encrypted if exactly one DVD player’s key needs to be revoked?

(A) 2 (B) n-1 (C) log_2n (D) n/2 (E) $\sqrt{n}$

Ans. (C) log_2n

因為總共從 root 到 leaf 的 path 上有 $log_2 n$ 個 node，因此需要

$log_2 n$ 個 key 才能夠加密。

Question 5 :

In the following let p be a prime. The set $Z_p = \{ x \text{ integer},$ $\text{such that } 0 \leq \text{x} < \text{p}\}$ is a group with respect to addition modulo p (i.e. every element x in Z_p has an inverse $-x \in Z_p$ such that $x + (-x) = 0 \; mod \; p$. The set $Z_p^* = \{ x \text{ integer}, \text{ such that } 0 < \text{x} < \text{p}\}$ is a group with respect to multiplication modulo p (i.e. every element x in Z_p^* has an inverse $x^{-1} \in Z_p^*$ such that $xx^{-1} = 1 \; mod \; p$.

Another cipher with perfect secrecy. Consider the following cipher.

Let Z_p^* be the message space, the key space and the ciphertext space.

Alice and Bob share a key $k \in Z_p^*$ uniformly chosen at random. To send a message $m \in Z_p^*$ to Bob, Alice computes the ciphertext $c = mk \; mod \; p$.

1. Prove that this cipher provides **Perfect Secrecy** using the criterium we proved in class.

Ans.
Recall **Shannon's perfect secrecy** we mentioned in the class.
Let (E, D) be a cipher over (K, M, C) .
(E, D) has perfect secrecy if $\forall \, m_0, \; m_1 \; \in \; M$, ($|m_0| = |m_1|$)
$\{ E(k, m_0) \} = \{ E(k, \; m_1) \}$, where $k \longleftarrow K$

Suppose we have two different messages $m_0, \; m_1 \in Z_p^*$, where $|m_0| = |m_1|$.

And the key $k \in Z_p^*$. We have the following property:

$$c_0 \equiv m_0 k \; mod \; p, \;\; c_1 \equiv m_1 k \; mod \; p - (*)$$

Since every element $x \in Z_p^*$ has an inverse $x^{-1} \in Z_p^*$ such that $x * x^{-1} \equiv 1 \; mod \; p$. We have the following property:

$$c_0 * \; {c_0}^{-1} \equiv 1 \; mod \; p, \; c_1 * \; {c_1}^{-1} \equiv 1 \; mod \; p$$

Furthermore, by $(*)$, we can also know that:

$$c_0 * \; (m_0 \; k)^{-1} \equiv 1 \; mod \; p, \; c_1 * \; (m_1 \; k)^{-1} \equiv 1 \; mod \; p$$

As a result, we have the following property:

$$c_0 \equiv m_0 \; k \; mod \; p, c_1 \equiv m_1 \; k \; mod \; p$$

Which implies that the message space with given key is equal to the ciphertext space, that is:

$$\{ E(k, m_0)\} = \{ E(k, \; m_1)\} = Z_p^*$$

As a result, by **Shannon's perfect secrecy**, we have proved that this cipher provides **Perfect Secrecy.**

2. Why one-time pad are **Perfect Secrecy** and **Semantic Secure**?

Ans.
Perfect Secrecy:
Recall **Shannon's perfect secrecy** we mentioned in the class.
Let (E, D) be a cipher over (K, M, C) .
(E, D) has perfect secrecy if $\forall \, m_0, \; m_1 \; \in \; M$, $(\, |m_0| = \; | \, m_1|)$
$\{ \, E(k, m_0) \, \} \; = \; \{ \, E(k, \; m_1) \, \}$, where $k \longleftarrow K$
Suppose we have two messages m_0, m_1, where $|m_0| = \; | \, m_1|$.
The one-time pads use the following function to encrypt the message:

$$c = E(k, m) = k \oplus m$$

, where c is the ciphertext, k is the key, and m is the message.
As a result, $\{ \, E(k, m_0)\} = \; \{ \, E(k, \; m_1)\} = \{0, 1\}$.
By **Shannon's perfect secrecy**, we have proved that **one-time pads provide Perfect Secrecy.**

Semantic Security:
The definition of semantic security is:
E is semantically secure if for all efficient A , AdvSS[A,E] is negligible.
This implies that for all explicit $m_0, m_1 \in$ M,

$$\{ E(k, m_0)\} \approx_p \{ E(k, m_1)\}$$

For all A: AdvSS[A,OTP] =
$| Pr[A(k \oplus m_0) = 1] - Pr[A(k \oplus m_1) = 1] | = 1/2 - 1/2 = 0$
As a result, **one-time pads are Semantic Secure.**
3. Is the use of one-time pads susceptible to statistical analysis (especially if it is known that the plaintext is in American English)?

Ans. No. 因為 one-time pads 使用的是 xor, 而 xor 的生成 1 或是 0 的機率分別是 1/2, 1/2. 因此使用 one-time pads 會讓原本資料分布變得非常平均，進而讓 statistical analysis 無法發揮功用，因為字母出現頻率的特性被 one-time pads 消除了。此外，one-time pads 的每個 key 都只有使用一次，因此沒辦法透過找到多對 plaintext 和 ciphertext 來嘗試找到 key。

4. Did public-key encryption scheme provide Perfect Secrecy? We assume there is a public-key encryption scheme (KeyGen, Enc, Dec) with perfect correctness (i.e., for all messages M and valid key-pairs (PK, SK), we have $DEC_{sk}(Enc_{pk}$(M)) = M).

Ans.
The threshold for a perfectly secure system is that a computationally unbounded adversary cannot conclude anything about the plaintext from the ciphertext. With a public-key system, the attacker can try to encrypt messages with the real public key; this is not possible with one-time pads. What the attacker can do, quite simply, is to try all one-bit messages, then all two-bit messages, then all three-bit messages, etc., looking for a matching ciphertext.

Question 6 :

Predicting generators. Consider the following *congruential generator*. It uses constants $a, b \in Z_p^*$. The seed is a value $x_0 \in Z_p^*$. The i^{th} value generated is computed as $x_i = ax_{i-1} + b \; mod \; p$.

The sequence output by the generator is $S = x_0, x_1, x_2, \ldots$ Assume that an attacker knows p and witness the sequence.

1. Prove that after a short prefix (i.e. a few of the values x_i's) the attacker is able to predict the rest of the sequence (i.e. the rest of the x_j's).

Ans. Assuming that there exists an integer k, where

$$x_k = ax_{k-1} + b \; mod \; p$$

Such that

$$x_k - x_{k-1} = a(x_{k-1} - x_{k-2}) mod \; p$$

Suppose that $(x_{k-1} - x_{k-2})$ is coprime to p, we have

$$(x_{k-1} - x_{k-2})^{-1}(x_k - x_{k-1}) = a \; mod \; p$$

As a result, we can calculate a.
For b, we can use the relationship:

$$x_k = ax_{k-1} + b \; mod \; p$$

That is,

$$x_k - ax_{k-1} = b \; mod \; p$$

To sum up, after a short prefix (i.e. a few of the values x_i's) the attacker is able to predict the rest of the sequence because the attacker can calculate a and b.

2. What does this say about the security of using the congruential generator as the keystream generator for a stream cipher?

Ans. It is not secure to use congruential generator as the keystream generator for a streamer cipher. If the attacker know the rule of the generator, and know the sequence of the generated sequence, then the attacker can know the keystream.

3. If an attacker knows constants $a, b \in Z_p^*$ and p. How many output bits $S = x_0, x_1, x_2, \ldots$ did the attacker to know to rest sequences.

Ans. Because the attacker know the sequence

$$x_i = ax_{i-1} + b \; mod \; p$$

Furthermore, the attacker also knows constants $a, b \in Z_p^*$ and p.

The attacker only need to know **1 bit** to recover the rest sequences.

4. However, If an attacker knows p but know nothing about constants $a, b \in Z_p^*$. In this case, How many output bits $S = x_0, x_1, x_2, \ldots$ did the attacker need to know to recover the rest sequence?

Ans. By the discussion of 1.,

$$(x_{k-1} - x_{k-2})^{-1}(x_k - x_{k-1}) = a \; mod \; p$$
$$x_k - ax_{k-1} = b \; mod \; p$$

As a result, the attacker only need to know **3 consecutive bits** to recover the rest sequence

Question 7 :

In standard RSA the modulus N is a product of two distinct primes. Suppose we choose the modulus so that it is a product of three distinct primes, namely N = pqr. Given an exponent e relatively prime to $\varphi(N)$ we can derive the secret key as $d = e^{-1} \; mod \; \varphi(N)$. The public key (N, e) and secret key (N, d) work as before. What is $\varphi(N)$ when N is a product of three distinct primes?

(A) $\varphi(N) = pqr - 1$
(B) $\varphi(N) = (p-1)(q-1)(r+1)$
(C) $\varphi(N) = (p+1)(q+1)(r+1)$
(D) $\varphi(N) = (p-1)(q-1)(r-1)$

Ans. (D)

因為如果 P 是質數，$\varphi(P) = P - 1$。因此，

$$\varphi(N) = \varphi(pqr) = \varphi(p)\varphi(q)\varphi(r) = (p-1)(q-1)(r-1)$$

Question 8 :

Suppose we choose the modulus so that it is a product of three distinct primes, namely N = 105. Given an encryption key is 13 which is co-prime to $\varphi(N)$. Please find the secret key as $d = e^{-1} mod\ \varphi(N)$ using Extended Euclidean Algorithm.

Ans.

Because $\varphi(105) = \varphi(3)\varphi(5)\varphi(7) = 2 * 4 * 6 = 48$

$$48 * 3 + 13 * -11 = 1$$

$$The\ inverse\ of\ 13\ is:\ -11 = 37\ mod\ 48$$

As a result, d = 37

The code is like the image below:

```python

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, b):
    g, x, y = egcd(a, b)
    if g != 1:
        raise Exception('modular inverse does not exist')
    return g, x, y

if __name__=="__main__":
    a, b = 48, 13

    gcd, inv_a, inv_b = modinv(a, b)

    print(a, " * ", inv_a, " + ", b, " * ", inv_b, " = ", gcd)
    if inv_b < 0:
        inv_b = inv_b + a
    print("The inverse of 13 is: ", inv_b)
```

Question 9 :

An attacker intercepts the following ciphertext (hex encoded) :

20814804c1767293b99f1d9cab3bc3e7ac1e37bfb15599e5f40eef805488
281d

He knows that the plaintext is the ASCII encoding of the message “Pay Bob 100$” (excluding the quotes). He also knows that the cipher used is CBC encryption with a random IV using AES as the underlying block cipher. Show that the attacker can change the ciphertext so that it will decrypt to “Pay Bob 500$”. What is the resulting ciphertext (hex encoded)? This shows that CBC provides no integrity.

Ans.
This is insecure because the first message block is xored with the random IV

20814804c1767293b99f1d9cab3bc3e7 ac1e37bfb15599e5f40eef805488281d
Pay Bob 100$

9th char
0xb9 decrypts to 1
0xb9 xor ascii (1 xor 5)
0xb9 xor 0x31 xor 0x35
= 0xbd

As a result, the resulting ciphertext is:
20814804c1767293bd9f1d9cab3bc3e7 ac1e37bfb15599e5f40eef805488281d

Question 10 :

Let G be a finite cyclic group (e.g. $G = \mathbb{Z}_p^*$) with generator g. Suppose the Diffie-Hellman function $DH_g(g^x, g^y) = g^{xy}$ is difficult to compute in G. Which of the following functions is also difficult to compute :

As usual, identify the f below for which the contra-positive holds :
if $f(\cdot, \cdot)$ is easy to compute then so is $DH_g(\cdot, \cdot)$.

If you can show that then it will follow that if DH_g is hard to compute in G then so must be f.

(A) $f(g^x, g^y) = g^{2xy}$
(B) $f(g^x, g^y) = (g^2)^{x+y}$
(C) $f(g^x, g^y) = \sqrt{g^{xy}}$
(D) $f(g^x, g^y) = g^{x-y}$
Ans. (A), (C)

因為(B)可以被拆成$(g^2)^{x+y} = g^{2x} * g^{2y}$，而$g^{2x}$和$g^{2y}$都很好計算，

因此(B)很好計算，代表他的 Diffie-Hellman function 也很好計算。

至於(D)也是因為可以被拆成$g^{x-y} = \frac{g^x}{g^y}$，因此(D)也很好計算，代表他的 Diffie-Hellman function 也很好計算。

至於(A),(C)都沒辦法再被拆解，所以很難計算。