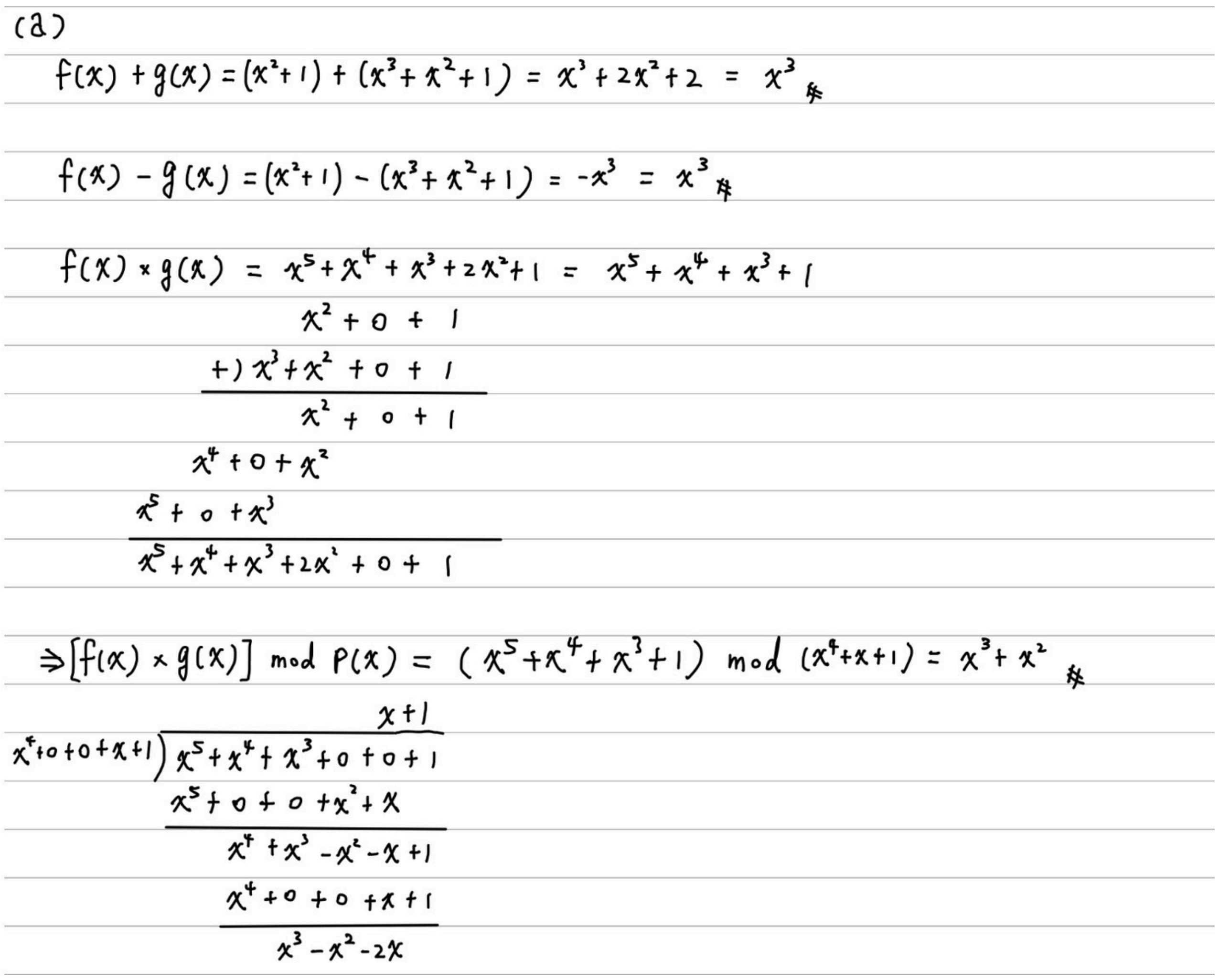
密碼工程 midterm

Problem 1

(a)



(a)

$f(x)+g(x)=(x^2+1)+(x^3+x^2+1)=x^3+2x^2+2=x^3$ #

$f(x)-g(x)=(x^2+1)-(x^3+x^2+1)=-x^3=x^3$ #

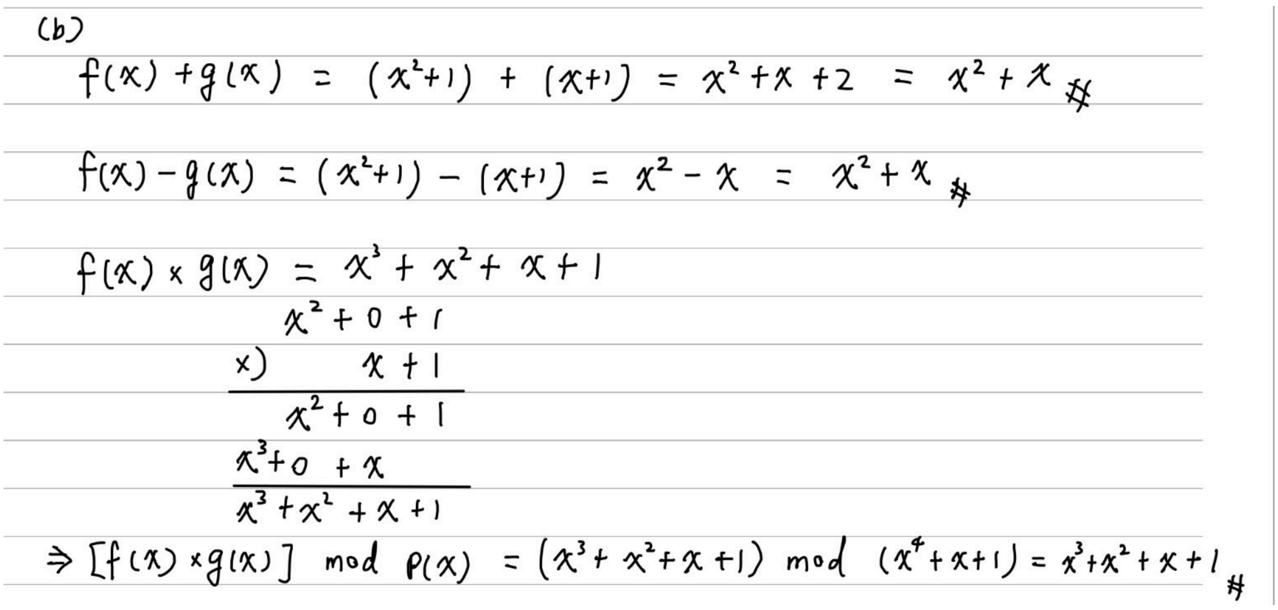
$f(x)\times g(x)=x^5+x^4+x^3+2x^2+1=x^5+x^4+x^3+1$

$$\begin{array}{r} x^2+0+1 \\ +)\ x^3+x^2+0+1 \\ \hline x^2+0+1 \\ x^4+0+x^2\quad\ \ \\ x^5+0+x^3\qquad\quad \\ \hline x^5+x^4+x^3+2x^2+0+1 \end{array}$$

$\Rightarrow [f(x)\times g(x)] \bmod P(x) = (x^5+x^4+x^3+1) \bmod (x^4+x+1) = x^3+x^2$ #

$$\begin{array}{r|l} & \quad\quad\quad x+1 \\ \hline x^4+0+0+x+1 & x^5+x^4+x^3+0+0+1 \\ & x^5+0+0+x^2+x \\ \hline & \quad x^4+x^3-x^2-x+1 \\ & \quad x^4+0+0+x+1 \\ \hline & \quad x^3-x^2-2x \end{array}$$

(b)



(b)

$f(x)+g(x) = (x^2+1) + (x+1) = x^2+x+2 = x^2+x$ #

$f(x)-g(x) = (x^2+1) - (x+1) = x^2-x = x^2+x$ #

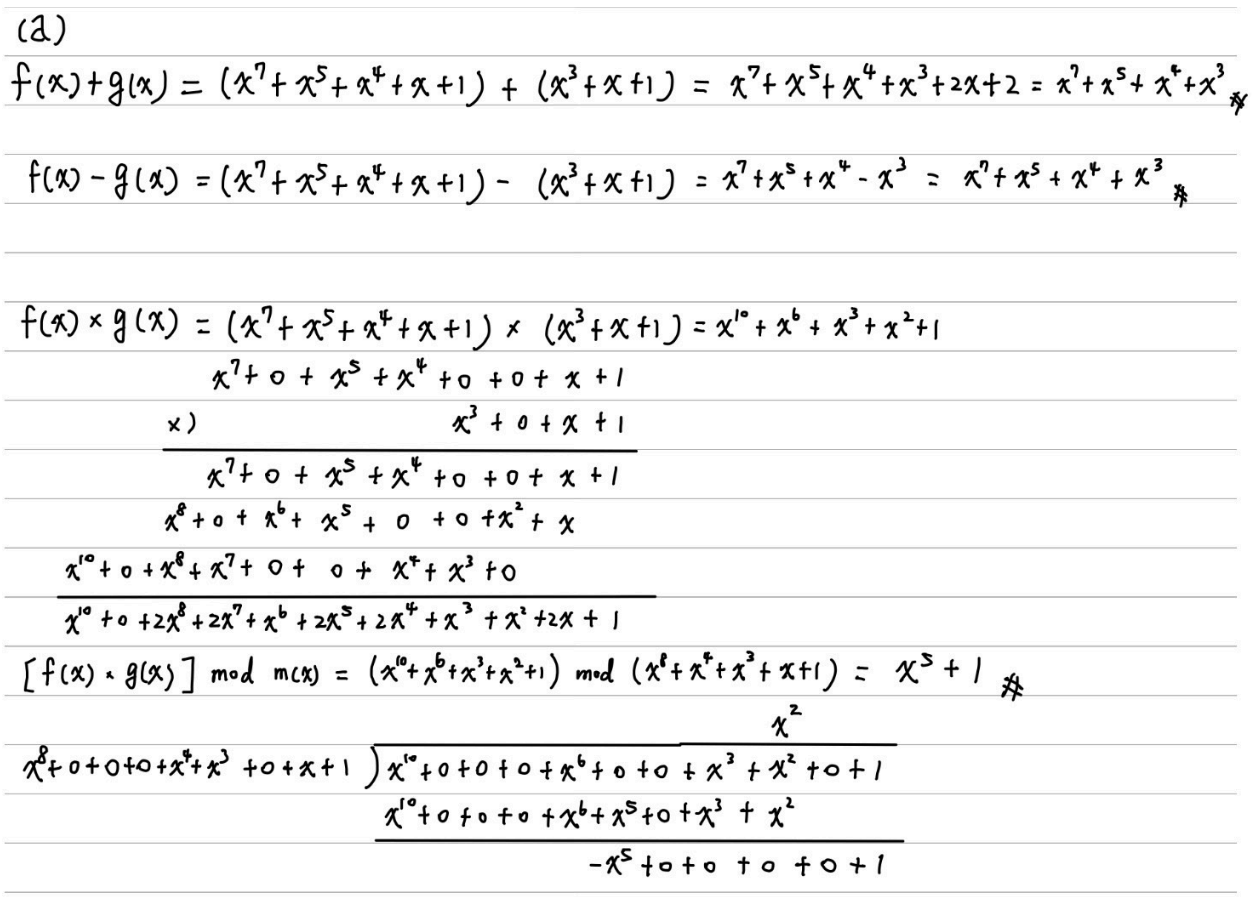
$f(x)\times g(x) = x^3+x^2+x+1$

$$\begin{array}{rl} & x^2+0+1 \\ \times) & x+1 \\ \hline & x^2+0+1 \\ & x^3+0+x \\ \hline & x^3+x^2+x+1 \end{array}$$

$\Rightarrow [f(x)\times g(x)] \bmod p(x) = (x^3+x^2+x+1) \bmod (x^4+x+1) = x^3+x^2+x+1$ #

Problem 2

(a)



(a)

$f(x)+g(x) = (x^7+x^5+x^4+x+1) + (x^3+x+1) = x^7+x^5+x^4+x^3+2x+2 = x^7+x^5+x^4+x^3$ #

$f(x)-g(x) = (x^7+x^5+x^4+x+1) - (x^3+x+1) = x^7+x^5+x^4-x^3 = x^7+x^5+x^4+x^3$ #

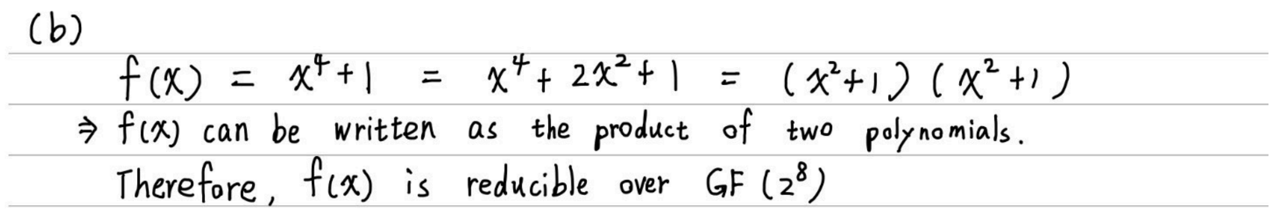
$f(x)\times g(x) = (x^7+x^5+x^4+x+1)\times(x^3+x+1) = x^{10}+x^6+x^3+x^2+1$

$$\begin{array}{rl} & x^7+0+x^5+x^4+0+0+x+1 \\ \times) & x^3+0+x+1 \\ \hline & x^7+0+x^5+x^4+0+0+x+1 \\ & x^8+0+x^6+x^5+0+0+x^2+x \\ & x^{10}+0+x^8+x^7+0+0+x^4+x^3+0 \\ \hline & x^{10}+0+2x^8+2x^7+x^6+2x^5+2x^4+x^3+x^2+2x+1 \end{array}$$

$[f(x)\times g(x)] \bmod m(x) = (x^{10}+x^6+x^3+x^2+1) \bmod (x^8+x^4+x^3+x+1) = x^5+1$ #

$$\begin{array}{rl} & x^2 \\ x^8+0+0+0+x^4+x^3+0+x+1 \,) & x^{10}+0+0+0+x^6+0+0+x^3+x^2+0+1 \\ & x^{10}+0+0+0+x^6+x^5+0+x^3+x^2 \\ \hline & -x^5+0+0+0+0+1 \end{array}$$

(b)



(b)

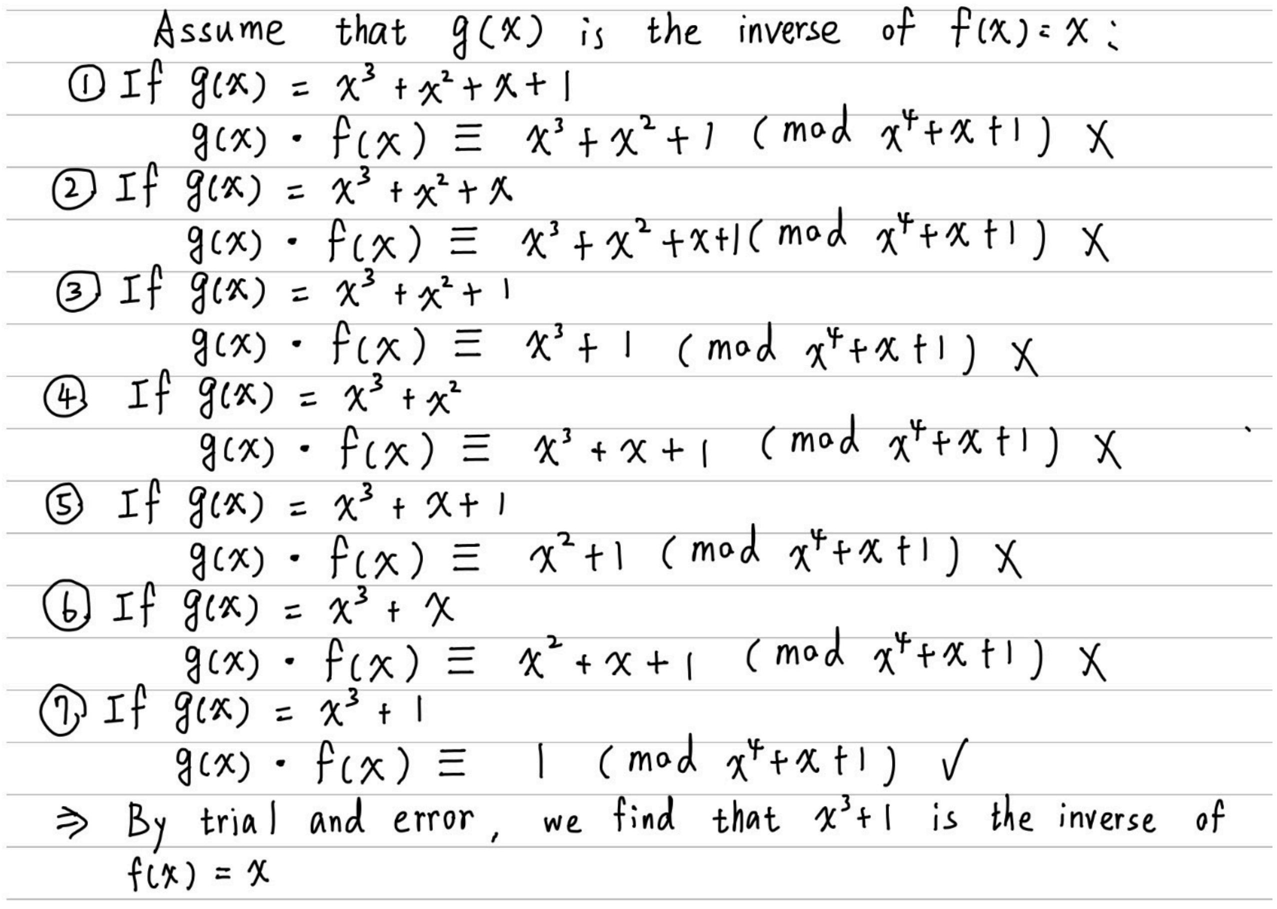
$f(x) = x^4+1 = x^4+2x^2+1 = (x^2+1)(x^2+1)$

$\Rightarrow$ $f(x)$ can be written as the product of two polynomials.

Therefore, $f(x)$ is reducible over $GF(2^8)$

Problem 3

(a)



Assume that $g(x)$ is the inverse of $f(x) = x$:

① If $g(x) = x^3+x^2+x+1$

$g(x) \cdot f(x) \equiv x^3+x^2+1 \pmod{x^4+x+1}$ X

② If $g(x) = x^3+x^2+x$

$g(x) \cdot f(x) \equiv x^3+x^2+x+1 \pmod{x^4+x+1}$ X

③ If $g(x) = x^3+x^2+1$

$g(x) \cdot f(x) \equiv x^3+1 \pmod{x^4+x+1}$ X

④ If $g(x) = x^3+x^2$

$g(x) \cdot f(x) \equiv x^3+x+1 \pmod{x^4+x+1}$ X

⑤ If $g(x) = x^3+x+1$

$g(x) \cdot f(x) \equiv x^2+1 \pmod{x^4+x+1}$ X

⑥ If $g(x) = x^3+x$

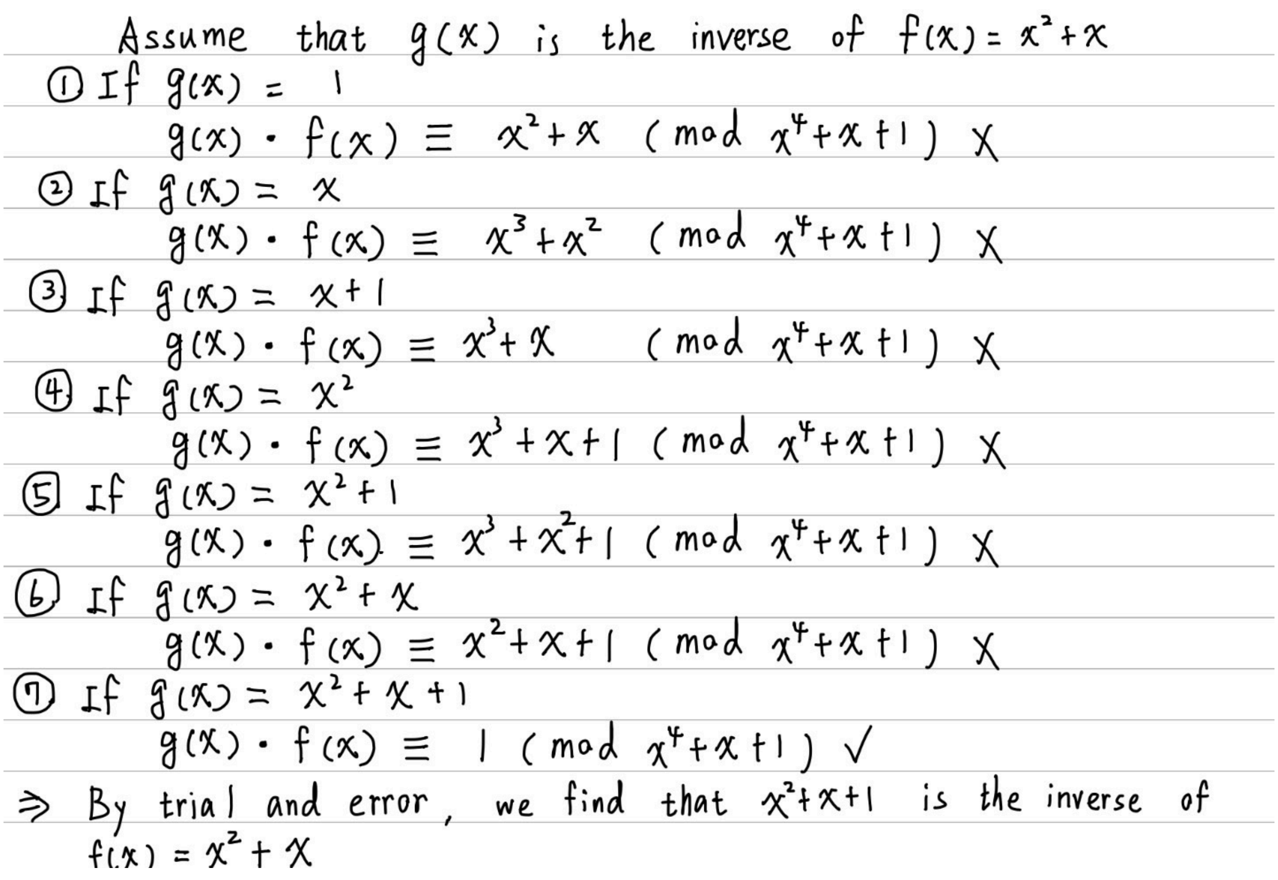
$g(x) \cdot f(x) \equiv x^2+x+1 \pmod{x^4+x+1}$ X

⑦ If $g(x) = x^3+1$

$g(x) \cdot f(x) \equiv 1 \pmod{x^4+x+1}$ ✓

$\Rightarrow$ By trial and error, we find that x^3+1 is the inverse of $f(x) = x$

(b)



Assume that $g(x)$ is the inverse of $f(x) = x^2+x$

① If $g(x) = 1$

$g(x) \cdot f(x) \equiv x^2+x \pmod{x^4+x+1}$ ✗

② If $g(x) = x$

$g(x) \cdot f(x) \equiv x^3+x^2 \pmod{x^4+x+1}$ ✗

③ If $g(x) = x+1$

$g(x) \cdot f(x) \equiv x^3+x \pmod{x^4+x+1}$ ✗

④ If $g(x) = x^2$

$g(x) \cdot f(x) \equiv x^3+x+1 \pmod{x^4+x+1}$ ✗

⑤ If $g(x) = x^2+1$

$g(x) \cdot f(x) \equiv x^3+x^2+1 \pmod{x^4+x+1}$ ✗

⑥ If $g(x) = x^2+x$

$g(x) \cdot f(x) \equiv x^2+x+1 \pmod{x^4+x+1}$ ✗

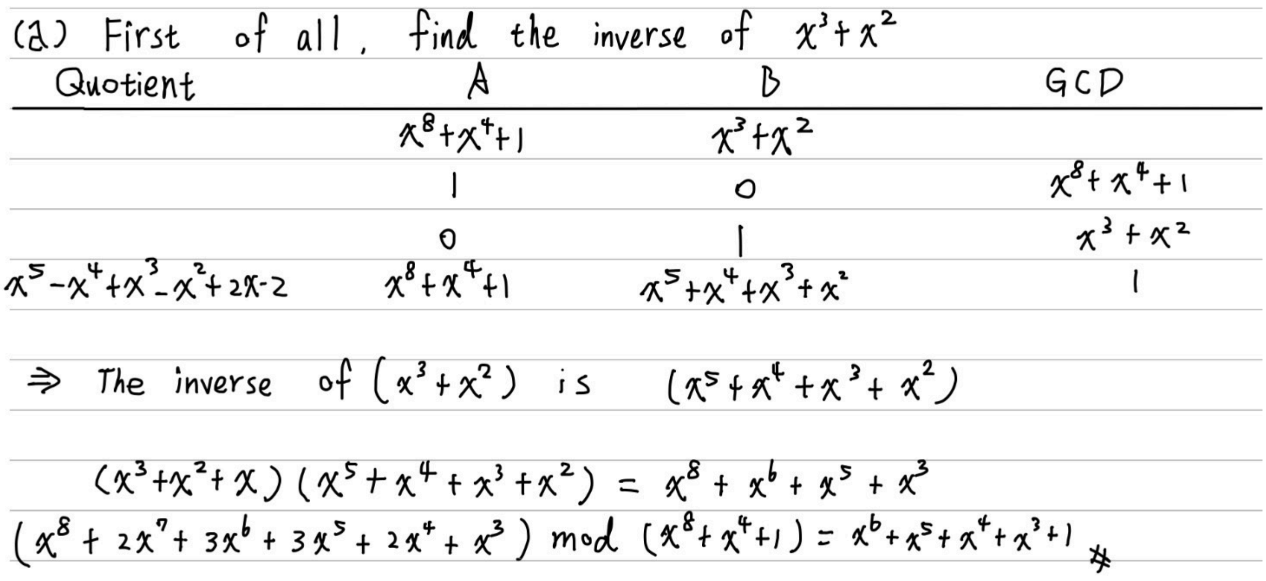
⑦ If $g(x) = x^2+x+1$

$g(x) \cdot f(x) \equiv 1 \pmod{x^4+x+1}$ ✓

⇒ By trial and error, we find that x^2+x+1 is the inverse of $f(x) = x^2+x$

Problem 4

(a)



(a) First of all, find the inverse of x^3+x^2

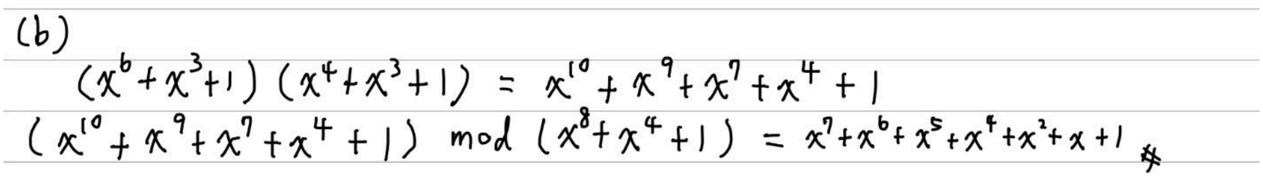
Quotient	A	B	GCD
	x^8+x^4+1	x^3+x^2	
	1	0	x^8+x^4+1
	0	1	x^3+x^2
$x^5-x^4+x^3-x^2+2x-2$	x^8+x^4+1	$x^5+x^4+x^3+x^2$	1

⇒ The inverse of (x^3+x^2) is $(x^5+x^4+x^3+x^2)$

$(x^3+x^2+x)(x^5+x^4+x^3+x^2) = x^8+x^6+x^5+x^3$

$(x^8+2x^7+3x^6+3x^5+2x^4+x^3) \bmod (x^8+x^4+1) = x^6+x^5+x^4+x^3+1$ #

(b)



(b)

$(x^6+x^3+1)(x^4+x^3+1) = x^{10}+x^9+x^7+x^4+1$

$(x^{10}+x^9+x^7+x^4+1) \bmod (x^8+x^4+1) = x^7+x^6+x^5+x^4+x^2+x+1$ #

Problem 5

(a)

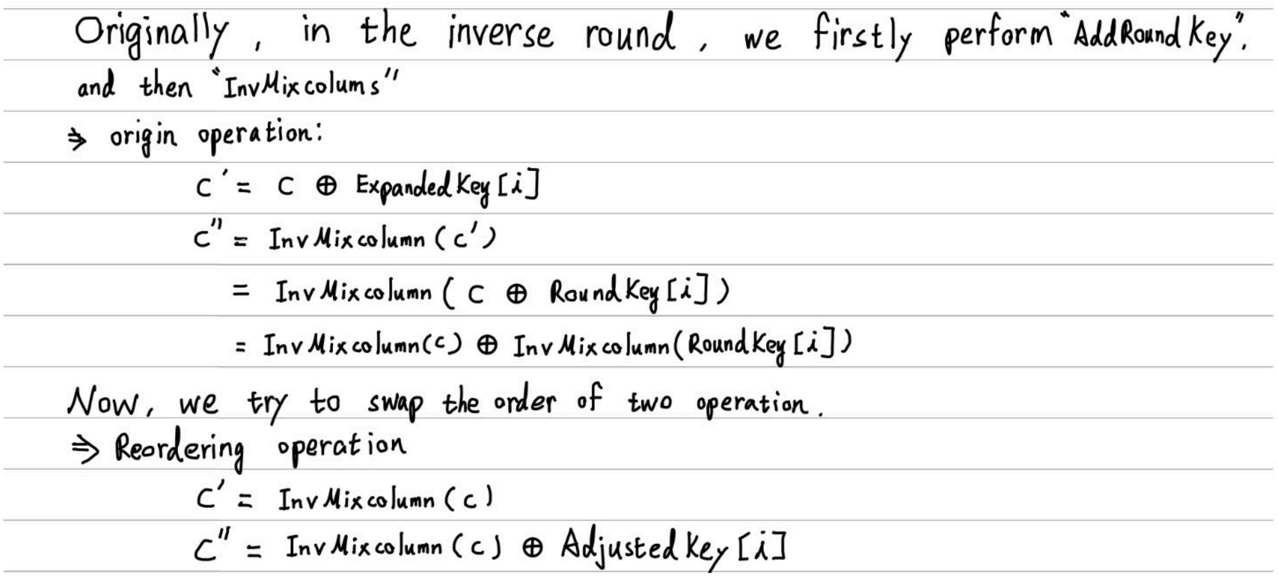
To perform mix column operation, we sould do the multiplication and XOR several times. If we keep calculating again and again, it will take lots of time. With a view to solving this problem, we create a lookup table recording the result of the multiplication of each possible byte value with the fixed matrix. In this way, we can find the result we want quickly. After that, we only need to perform XOR operation to get the final ansewer.

(b)

It is the same concept of the problem a. If we perform byte substitution, shift row and mix column seperately and repeatedly, it will take lots of time. Therefore, we create three lookup tables for byte substitution, shift row and mix column respectively. Next, combine the byte substitution table and shift row table by interconnecting the corresponding bytes. Subsequently, do the same process to combine the mix column multiplication lookup table and the table we created last step. Now, we have a big table which records the results undergone the three operations. In this way, we can find the result we want quickly. After that, we only need to perform XOR operation to get the final ansewer.

problem 6

First of all, InvSubBytes is an inverse operation that performs substitution step where each byte is replaced with another. It will operates on individual byte. As for InvShiftRows, it is an inverse transposition step where each row of the state is shifted cyclically by a certain number of bytes. It affects the order of whole bytes. Due to the different effect, the order of InvSubBytes and InvShiftRows is indifferent.
Secondly,



Originally, in the inverse round, we firstly perform "AddRoundKey", and then "InvMixcolums"

$\Rightarrow$ origin operation:

$$C' = C \oplus \text{ExpandedKey}[i]$$
$$C'' = \text{InvMixcolumn}(C')$$
$$= \text{InvMixcolumn}(C \oplus \text{RoundKey}[i])$$
$$= \text{InvMixcolumn}(C) \oplus \text{InvMixcolumn}(\text{RoundKey}[i])$$

Now, we try to swap the order of two operation.

$\Rightarrow$ Reordering operation

$$C' = \text{InvMixcolumn}(C)$$
$$C'' = \text{InvMixcolumn}(C) \oplus \text{AdjustedKey}[i]$$

→ It can be observed that C'' will be identical if AdjustedKey[i] is equal to InvMixColumns(RoundKey[i]). Therefore, if we precompute a AdjustedKey as InvMixColumns(RoundKey[i]), then it can invert the order of AddRoundKey operation and InvMixColumns operation without changing the results.
Encryption per round:

```
Round(State, ExpandedKey[i]) {
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State, ExpandedKey[i]);
}
```

Modified decryption per round by using the property we showed above:

```
InvRound (State, EqExpandedKey[i]) {
    InvSubBytes(State);
    InvShiftRows(State);
    InvMixColumns(State);
    AddRoundKey(State, EqExpandedKey[i]);
}
```

However, the InvRound structure, combined with the initial AddRoundKey, is not the inverse of the encryption.The reason is that the last MixColumns of encryption has no corresponding InvMixColumns, and the last InvMixColumns of decryption has no corresponding MixColumns, either. Therefore, if we omit MixColumns operation from the last encryption round and InvMixColums operation from the last decryption round, then we can completly solve it. In this way, we not only make InvRound be the inverse of Round but also make make the structure of them quite similar, leading to the improvement of sharing some of the code (for soft ware implementation) or chip area (for hardware implementation) for AES encryption and decryption.

problem 7

Overall, AES is better than 3DES. AES has a larger key size (the largest is 256 bits) compared to 3DES's 168-bit effective key size. Moreover, AES has undergone extensive cryptanalysis, which can give more security. As for efficiency, AES's algorithm is simpler and more streamlined, allowing for faster encryption and decryption speeds. Furthermore, most CPUs ship with AES accelerators nowadays, which means that AES is even faster.
Therefore, if there are no any limits, using AES is the best choice. However, in reality, changing management is hard, certain smart card or hardware module do not support AES, but support 3DES. It forces people to use 3DES without any choice if they do not decide to change the hardware.

3DES is susceptible to Meet-in-the-Middle attacks like 2DES, but its attack complexity is higher. We need to use 2^{112} operations to encrypt the plaintext with all possible keys in the first encryption phase and use 2^{56} operations to decrypt the ciphertext with all possible keys in the second decryption phase. Subsequently, store the 2^{56} intermediate results to figure out the potential match pairs of keys.

problem 8

To revoke the old key, we should check all instances where the key is used fistly. Next, destroy all copies of the old key to ensure that the fired CTO does not have access to it. Finally, update access control lists and revoke the fired CTO's access to any systems or resources where the old key might still be present.
After that, we can use a NIST-approved key generation algorithm to generate a new key. Subsequently, deploy the new key we generated to authorized entities and systems that require it. During the distribution, we should use NIST-approved key wrapping or key encryption techniques to avoid leaking the information of new key.

problem 9

(a)

The first problem is that e Alice chose might not coprime with (p-1)*(q-1), where p and q are prime numbers. For example, if Alice chose 7 to be e, and p is equal to 29. It will not satisfy the requirement of RSA, which may lead to failure to generate a decryption key.
The second problem is that it may be easily attacked by attacker. Assume that an attacker collects two ciphertexts C_1 and C_2, whose public key are (n, 3) and (n, 5) respectively.
$C_1 = M^3 \bmod n$
$C_2 = M^5 \bmod n$
Next, utilize the extended Euclidean algorithm to find integers a and b such that:
$a * 3 + b * 5 = 1$ (Note that e here is always a prime number, so the gcd of two different e is always 1)
Subsequently, we use the imformation we've known above and try to calculate C, which is equal to $(C_1)^a$ multiply $(C_2)^b$ mod n.
$C = (C_1)^a * (C_2)^b \bmod n = (M^3)^a * (M^5)^b \bmod n = M^{3a+5b} = M$
In this way, the attacker successfully recover the original message M.

(b)

Note that in the following description, p_i is smaller than q_i, which i = 1, 2, 3...
Alice generates her initial public key (n_1, e_1) where n_1 is equal to the product of two large prime numbers p_1 and q_1.

In the subsequent months, Alice updates her public keys to (n_2, e_2), (n_3, e_3), ..., where $n_2 = p_2 * q_2$, $n_3 = p_3 * q_3$, and so on, by incrementing p and q to the next prime numbers.
Now, an attacker starts to observe the sequence of n and try to find the gcd of n_1 and n_i, which i = 2, 3, 4...
If the gcd of n_1 and n_i is 1, it means that p_1, q_1, p_i, and q_i are four different prime numbers. Then, we will try next n and find the gcd of them again until the gcd of n_1 and n_i is x which is not equal to 1. It means that $q_1 = p_i = x$.
Once q_1 is known, the attacker can easily compute p_1 as $p_1 = n_1 / q_1$. Next, the attacker can start utilize p_1 and q_1 to calculate the decrption key d_1.
$d_1 = e_1$-1 mod ((p_1-1)(q_1-1))
Once the attacker has the decryption key, it can utilize it to decrypt the ciphertext and get the message he/she wants.

problem 10

(a)

If we want to perform Inverse MixColumns step, it is neccessary to multiply the state matrix and the matrix for the Inverse MixColumns operation. The roughly rule of the matrix multiplication here is identical to the standard matrix multiplication. For example, if we want to get the value in the second row and third column, we need to compute the dot product of the second row vector of the state matrix and the third column vector of the matrix for the Inverse MixColumns operation. However, there are still some differences. The matrix multiplication here sould use arithmetic in the finite field GF(2^8). Following are the rules of addition and multiplication.
Addition: Addition is performed using bitwise XOR.
Multiplication: First of all, perform the normal multiplication. Next, find the value of each coefficient mod 2. Finally, Take the polynomial obtained in the second step modulo $x^8+x^4+x^3+x+1$ and get the answer we want.

(b)

Multiplication by 9 :

$9 \cdot x = 8 \cdot x + x = 2(2(2 \cdot x)) + x$ #

Multiplication by 11 ;

$11 \cdot x = 10 \cdot x + x = 2 \cdot 5 \cdot x + x = 2(4x + x) + x = 2(2(2 \cdot x) + x) + x$ #

Multiplication by 13 ;

$13 \cdot x = 12 \cdot x + x = 2(2 \cdot 3 \cdot x) + x = 2(2(2 \cdot x + x)) + x$ #

Multiplication by 14 ;

$14 \cdot x = 2(7 \cdot x) = 2(2 \cdot 3 \cdot x + x) = 2(2(2 \cdot x + x) + x)$ #

(c)

In the implementation of Inverse MixColumns in AES decryption, it should perform multiplication many times, which can take enormous amount of time. Therefore, we can precompute the results of multiplications by the value in the matrix for the Inverse MixColumns operation for all possible byte values. In this way, when we start to implement Inverse MixColumns, all we need to do is to look up the table to get the values we want and do the addition. The advantage of it is that we can save lots of time by accelerating our calculation speed. However, every coin has two sides. The drawback of using LUTs is that it requires additional memory to store the precomputed values. If the memory is not big enough, it may lead to a problem. Moreover, it is vulnerable to cache timing attack, which is a special type of side channel attack.

Problem 11

$C_0 = \text{Enc}(k, m_0) \oplus IV$

$C_1 = \text{Enc}(k, m_1) \oplus m_0 = \text{Enc}(k, x) \oplus m_0$

$C_2 = \text{Enc}(k, m_2) \oplus m_1 = \text{Enc}(k, x) \oplus x$

$\Rightarrow C_1 \oplus C_2 = (\text{Enc}(k, x) \oplus m_0) \oplus (\text{Enc}(k, x) \oplus x)$

$= m_0 \oplus x$

$\Rightarrow$ Since C_1 and C_2 are public, we can easily calculate $C_1 \oplus C_2$.

Besides, x is known. Therefore, we can easily calculate m_0 by utilizing the value of $C_1 \oplus C_2$ and x.

problem 12

Divide M_1 to three parts : $M_1[0]$ · $M_1[1]$ · $M_1[2]$

Divide M_2 to three parts : $M_2[0]$ · $M_2[1]$ · $M_2[2]$

$T_1 = Enc(Enc(Enc(M_1[0] \oplus IV_1, k) \oplus M_1[1], k) \oplus M_1[2], k)$

$T_2 = Enc(Enc(Enc(M_2[0] \oplus IV_2, k) \oplus M_2[1], k) \oplus M_2[2], k)$

Assume that $m = M_2[0] \oplus IV_2 \oplus T_1$

$M_3 = M_1[0] \| M_1[1] \| M_1[2] \| m \| M_2[1] \| M_2[2]$

$\Rightarrow M_1[0] \| M_1[1] \| M_1[2]$ will generate T_1

$Enc(Enc(Enc(T_1 \oplus (M_2[0] \oplus IV_2 \oplus T_1), k) \oplus M_2[1], k) \oplus M_2[2], k)$

$= Enc(Enc(Enc(M_2[0] \oplus IV_2, k) \oplus M_2[1], k) \oplus M_2[2], k)$

$= T_2$

$\Rightarrow$ We prove that we can forge a MAC (IV_1, T_2) to another message M_3 by utilizing the known information.

Extra Credit 2

(a)

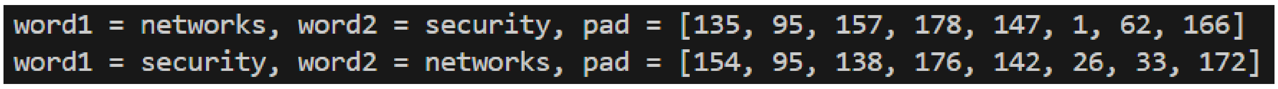
$$\left.\begin{aligned} C_1 &= M_1 \oplus p \\ C_2 &= M_2 \oplus p \end{aligned}\right\} \Rightarrow C_1 \oplus C_2 = (M_1 \oplus p) \oplus (M_2 \oplus p)$$

$$= M_1 \oplus M_2$$

→ Due to the deriation above, we can know that the result of exclusive or between two messages will be identical to the result of exclusive or between two ciphertexts. Therefore, we can use this property to figure out the words. However, there are still lots of possible results. In order to find the valid words, I use python to check. Following is my python code:

```
C1 = [0xe9, 0x3a, 0xe9, 0xc5, 0xfc, 0x73, 0x55, 0xd5]
C2 = [0xf4, 0x3a, 0xfe, 0xc7, 0xe1, 0x68, 0x4a, 0xdf]

# perform XOR to M1 and M2
M1_XOR_M2 = []
for num1, num2 in zip(C1, C2):
    M1_XOR_M2.append(num1 ^ num2)

with open('C:/Users/Microsoft/Desktop/midterm/dictionary.txt', 'r') as words_fi
    words = words_file.read().split()
    words = set([word for word in words if len(word) == len(M1_XOR_M2)])

    # Try to use M1 in dictionary to find M2 which is valid.
    for word1 in words:
        M1 = [ord(x) for x in word1]
        M2 = [x ^ y for x, y in zip(M1, M1_XOR_M2)]
        word2 = ''.join([chr(x) for x in M2])

        if word2 in words:
            pad = [x ^ y for x, y in zip(M1, C1)]
            print('word1 = %s, word2 = %s, pad = %s' % (word1, word2, pad))
```

First of all, we perform XOR to M1 and M2. Next, download the English word lists from the Internet so as to check whether M1 and M2 we get is valid or not. Subsequently, find all the words whose lengh is identical to 8. Utilize those words to perform XOR to M1 and M1_XOR_M2. In this way, we can get M2. Finally, check if M2 is valid. If it is valid, we just print M1, M2, and the pad.
result:



```
word1 = networks, word2 = security, pad = [135, 95, 157, 178, 147, 1, 62, 166]
word1 = security, word2 = networks, pad = [154, 95, 138, 176, 142, 26, 33, 172]
```