

VE475 Intro to Cryptography Homework 3

Taoyue Xia, 518370910087

2021/06/08

1 Ex1–Finite Fields

1. Take X 's value as 0, 1, 2 in $\mathbb{F}_3[X]$:

$$0^2 + 1 = 1 \mod 3 \quad 1^2 + 1 = 2 \mod 3 \quad 2^2 + 1 = 2 \mod 3$$

We can see that for $X \in \mathbb{F}_3[X]$, there doesn't exist an X which makes $X^2 + 1 = 0 \mod 3$. Thus $X^2 + 1$ is irreducible in $\mathbb{F}_3[X]$.

2. In question 1, we proved that $X^2 + 1$ is irreducible in $\mathbb{F}_3[X]$, and the polynomial $1 + 2X$'s degree is less than 2, according to the proof on page 39, c2, Let $P(X) = X^2 + 1$, $A(X) = 1 + 2X$, then there always exists a $B(X)$, such that $A(X)B(X) = 1 \mod P(X)$, which means $B(x)$ is the multiplication inverse of $1 + 2X \mod X^2 + 1$. Proof done.
3. Apply the extended Euclidean algorithm, let a and b be such that $a(1 + 2X) + b(X^2 + 1) = 1 \mod 3$. Then calculate in matrix form (a 's value in the first column, b 's value in the second):

$$\begin{aligned} \begin{pmatrix} 1 & 0 & 1 + 2X \\ 0 & 1 & X^2 + 1 \end{pmatrix} &\Rightarrow \begin{pmatrix} 0 & 1 & X^2 + 1 \\ 1 & 0 & 1 + 2X \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 1 + 2X \\ X & 1 & X + 1 \end{pmatrix} \\ &\Rightarrow \begin{pmatrix} X & 1 & X + 1 \\ X + 1 & 1 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} X + 1 & 1 & 2 \\ X^2 + 2X & X + 1 & 1 \end{pmatrix} \end{aligned}$$

Thus we can find that the multiplication inverse of $1 + 2X \mod X^2 + 1$ is $X^2 + 2X$.

2 Ex2–AES

1. a) The *InvShiftRows* function cyclicly shift each row i 's elements right for $i = 0, 1, 2, 3$.

For example, if the 4×4 matrix is
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix},$$
 then the matrix after the operation *InvShiftRow* would be:
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

- b) The inverse of *AddRoundKey* will make the 4 32-bits words xor with the expansion key, for example, in AES-128, for 11 rounds in the reverse order. This is because anything xor a value twice would keep unchanged.
- c) The 4×4 matrix used for *MixColumns* is (in hexadecimal form):

$$A_1 = \begin{pmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

The 4×4 matrix used for *InvMixColumns* is given by:

$$A_2 = \begin{pmatrix} 00001110 & 00001011 & 00001101 & 00001001 \\ 00001001 & 00001110 & 00001011 & 00001101 \\ 00001101 & 00001001 & 00001110 & 00001011 \\ 00001011 & 00001101 & 00001001 & 00001110 \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix}$$

Then we calculate $A_2 \times A_1$ in $\mathbb{GF}(2^8)$, for example, the four elements of the first row of matrix $B = A_2 \times A_1$ will be:

$$B_{0,1} = (0e \cdot 02) \oplus (0b \cdot 01) \oplus (0d \cdot 01) \oplus (09 \cdot 03) = 01_{16}$$

$$B_{0,2} = (0e \cdot 03) \oplus (0b \cdot 02) \oplus (0d \cdot 01) \oplus (09 \cdot 01) = 00_{16}$$

$$B_{0,3} = (0e \cdot 01) \oplus (0b \cdot 03) \oplus (0d \cdot 02) \oplus (09 \cdot 01) = 00_{16}$$

$$B_{0,4} = (0e \cdot 01) \oplus (0b \cdot 01) \oplus (0d \cdot 03) \oplus (09 \cdot 02) = 00_{16}$$

Using the same method, we can calculate the remaining three rows of B , finally (in hexadecimal form):

$$B = \begin{pmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 01 \\ 00 & 00 & 00 & 01 \end{pmatrix} = \mathbb{I}_4$$

Thus it is the reason why the transformation of *InvMixColumns* is given by the multiplication by matrix A_2 .

2. Firstly, we apply the key expansion schedule to generate round keys according to the original key. Then we perform *AddRoundKey* with $round_keys[40 \sim 43]$, each has one 32-bit word.

In the next nine rounds, we perform *InvShiftRows*, *InvSubBytes*, *AddRoundKey* (with $round_keys[40 - 4i \sim 40 - 4(i - 1)]$ for $i \in [1, 9]$), and *InvMixColumns* in order.

Finally, We perform the last round of *InvShiftRows*, *InvSubBytes*, *AddRoundKey* with $round_keys[0 \sim 3]$, then we can get the original plaintext from the ciphertext.

3. In process of *InvShiftRows*, we just change the position of some elements without changing any value. In process of *InvSubBytes*, we look up the corresponding value of each element in the inverse s-box, and substitute the original value. Therefore, it doesn't matter in which order these two processes are performed. That's why they can be applied on reverse order.
4. a) As *AddRoundKey* will perform an xor between treated text and the key in different rounds, the value of each 32-bit word would probably change. What's more, in process *InvMixColumns*, multiplication and addition in Galois field $\mathbb{GF}(2^8)$ are performed, which can also change the values of words. As addition and multiplication with different values can lead to a completely distinct result, thus the order of application of *AddRoundKey* and *InvMixColumns* cannot be reversed.

b)

$$((m_{i,j} \ (a_{i,j})) \oplus (k_{i,j}))$$

c) As the initial matrix is $(a_{i,j})$, from the above order, we can get:

$$\begin{aligned}(a_{i,j}) &= (m_{i,j})^{-1}((e_{i,j}) \oplus (k_{i,j})) \\ &= (m_{i,j})^{-1}(e_{i,j}) \oplus (m_{i,j})^{-1}(k_{i,j})\end{aligned}$$

So the inverse operation is:

$$(e_{i,j}) \rightarrow (m_{i,j})^{-1}(e_{i,j}) \oplus (m_{i,j})^{-1}(k_{i,j})$$

d) The *InvAddRoundKey* operation will first calculate the multiplication of the *InvMatrix* and the key of the corresponding round, then perform an xor with the text which has been operated by the *InvMixColumns* method.

5. Firstly, we apply the key expansion schedule to generate round keys according to the original key. Then we perform *AddRoundKey* with *round_keys*[40 ~ 43], each has one 32-bit word.

In the next nine rounds, we perform *InvSubBytes*, *InvShiftRows*, *InvMixColumns*, and *InvAddRoundKey* (with *round_keys*[40 - 4*i* ~ 40 - 4(*i* - 1)] for *i* ∈ [1, 9]) in order.

Finally, We perform the last round of *InvSubBytes*, *InvShiftRows*, *AddRoundKey* with *round_keys*[0 ~ 3], then we can get the original plaintext from the ciphertext.

6. The advantage is that the order of non-inverse operations and inverse operations are the same, so that it's easier to understand and implement.

3 Ex3–DES

1. In DES, the size of input text and key are both 64 bits.

a) The input plaintext is enciphered by the following permutation table **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

That is, in the enciphered 64 bits, the first bit is the original 58th bit, the second is the 50th, etc.

- b) Then the key is reduced to 56 bits in the same way as above, looking up value in a table, and replace the original bit.
- c) Then the 64-bit enciphered text is divided into two 32-bit content L_0 and R_0 . Define a operation function f , we can calculate:

$$L_1 = R_0$$

$$R_1 = L_0 \oplus f(R_0, K_0)$$

Repeat the process for 16 rounds, we will get final L_{16} and R_{16} .

- d) For the operation function f , we first extend R_i , ($i \in [0, 16]$) from 32 bits to 48 bits. The method is like that in (b), in a look-up table, generating each bit from the input R_i .

Then the 56-bit key will be splitted into two 28-bit keys, and each shift left for 1 or 2 bits according to each round, the table is shown below:

round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

After that, combine the two parts into a 56-bit key, and reduce it o 48 bits in the same way as (b), with a different look-up table.

Then we xor the extended 48-bit R_i and the key, denote the output as X . X is then divided into 8*6 bits, each 6 bits will pass a S-box S_i , ($i \in [1, 8]$), and each output a 4-bit content.

Finally, join the 8 groups of 4 bits, we will get $f(R_i, K_i)$ which is 32 bits.

- e) After the 16 rounds of transforming, we can get a 64-bit data. Finally, we will perform an inverse of the initial permutation, by the following table 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

Then the whole encryption is done.

For decryption, we just need to reverse the key's order and apply the reverse of each operation.

2. Linear cryptanalysis:

There are two parts for linear cryptanalysis. The first is to construct linear equations relating plaintext, ciphertext and key bits that have a high bias. The second is to use these linear equations in conjunction with known plaintext-ciphertext pairs to derive key bits.

Differential cryptanalysis:

Differential cryptanalysis is usually a chosen plaintext attack. The basic method uses pairs of plaintext related by a constant difference. Difference can be defined in several ways, but the eXclusive OR (XOR) operation is usual.

The attack relies primarily on the fact that a given input/output difference pattern only

occurs for certain values of inputs. Usually the attack is applied in essence to the non-linear components as if they were a solid component (usually they are in fact look-up tables or S-boxes). Observing the desired output difference (between two chosen or known plaintext inputs) suggests possible key values.

3. Triple DES:

We define the encryption of DES as $E_K(P)$, where K stands for key, P stands for original plaintext, and the decryption of DES as $D_K(C)$, where C is the ciphertext.

Triple DES, also known as the TDEA, encodes the plaintext for three rounds of DES to get the ciphertext:

$$C = E_{K_3}(E_{K_2}(E_{K_1}(P)))$$

Then the decode process is the inverse:

$$P = D_{K_1}(D_{K_2}(D_{K_3}(C)))$$

The standard for the three keys K_1 , K_2 , K_3 is defined below:

- (1) Key Option 1: K_1 , K_2 and K_3 are three independent keys.
- (2) Key Option 2: K_1 and K_2 are independent keys, and $K_3 = K_1$.
- (3) Key Option 3: $K_1 = K_2 = K_3$.

Meet-in-the-middle attack is the reason why Double DES is replaced by Triple DES. Its logic is:

$$\begin{aligned} C &= E_{K_2}(E_{K_1}(P)) \\ D_{K_2}(C) &= D_{K_2}(E_{K_2}(E_{K_1}(P))) \\ D_{K_2}(C) &= E_{K_1}(P) \end{aligned}$$

The attacker can compute $E_{K_1}(P)$ for all possible values of K_1 and $D_{K_2}(C)$ for all possible values of K_2 for a total of $2^{K_1} + 2^{K_2}$ operations.

As Double DES uses two keys K_1 and K_2 in same size 56 bits, attackers can bruteforce Double DES in 2^{57} operations and 2^{56} space to get the keys, which is not safe at all.

However, for Triple DES, attackers need up to $2^{K_1+K_2} + 2^{K_3}$, namely 2^{112} operations and 2^{56} space to bruteforce and get the keys, which is safe, but still not secure.

This is the reason why Triple DES is used instead of Double DES.

4. Traditionally, the ***crypt()*** function which encrypts users' passwords by DES, and save the encoded content in `/etc/passwd`. However, DES is proved to be not safe nowadays, so modern Unix systems use **SHA-256** and **MD5**, which is more secure. So if one doesn't directly show his password file to others, it is almost impossible to cause password leak.

4 Ex4–Programming

The codes and makefile are attached in folder **ex4**, with a README file.