

Chunyu Xue 518021910698

1 Problem 1

(50 points) Now you' ve learned the RC4 scheme (one of the stream cipher of symmetric key encryption). Denote $KeyGen(\lambda)$ as the key generation algorithm: pick a uniform $k \in \{0, 1\}^{128}$, and output k . Denote m as a message of l bytes and c as the ciphertext. Please use your own words (or pseudocode) to describe the encoding algorithm $Enc(k, m)$ and decoding algorithm $Dec(k, c)$.

Hints: 1) The Enc determines the format of ciphertext output. 2) A stream cipher need synchronized information between encryption and decryption side (why?). Consider a real example: after A and B shared secret key k , A is going to send message m_1 to B in the first day and send message m_2 to B in the second day. In the meanwhile, A and B' s computers running Enc/Dec may shut down or restart due to failure. Take a look at whether your algorithms can support this example safely and conveniently.

(a) $Enc(k, m)$.

Answer: In order to ensure the correctness of **RC4 Algorithm** in different situations, we need to implement the synchronization of this stream ciphers. That is, we can add a number (with n bytes) at the beginning of our ciphertext to inform the decryptor the begin index in the key stream.

Based on the ideas above, we give our encryption algorithm $E(k, m)$ (let $n = 2$):

Algorithm 1: Encryption Algorithm $Enc(k, m)$ of RC4

Input: Uniform secret key $k \in \{0, 1\}^{128}$; Message m of l bytes

Output: Ciphertext c of l bytes.

```
1 byte key[16] ; // secret key
2 byte K[256] ; // keying material
3 byte S[256] ; // internal states

4 for i = 0 to 255 do
5   S[i] = i;
6   K[i] = key[i mod 16];
7 j = 0;
8 for i = 0 to 255 do
9   j = (j + S[i] + K[i]) mod 256;
10  swap(S[i], S[j]);
11 i = j = 0 ; // initialization finished

12 count = 0 ; // Record total rounds
13 for idx = 0 to s do
14   i = (i + 1) mod 256;
15   j = (j + S[i]) mod 256;
16   swap(S[i], S[j]);
17   count ++ ; // s times pass, idle

18 beginIdx = count ; // Record begin index
19 c[0 - 1] = beginIdx ; // Set beginIdx on first two bytes
20 for idx = 2 to l + 2 do
21   i = (i + 1) mod 256;
22   j = (j + S[i]) mod 256;
23   swap(S[i], S[j]);
24   count ++ ;
25   t = (S[i] + S[j]) mod 256;
26   c[idx] = m[idx - 2]  $\oplus$  S[t] ; // Begin encryption
27 return c;
```

(b) $Dec(k, c)$.

Answer: Based on the ideas above, we give our decryption algorithm $D(k, c)$ (let $n = 2$):

Algorithm 2: Decryption Algorithm $Dec(k, c)$ of RC4

Input: Uniform secret key $k \in \{0, 1\}^{128}$; Ciphertext c of l bytes

Output: Message m of l bytes.

```
1 byte key[16] ;                                // secret key
2 byte K[256] ;                                // keying material
3 byte S[256] ;                                // internal states

4 for i = 0 to 255 do
5   S[i] = i;
6   K[i] = key[i mod 16];
7 j = 0;
8 for i = 0 to 255 do
9   j = (j + S[i] + K[i]) mod 256;
10  swap(S[i], S[j]);
11 i = j = 0 ;                                // initialization finished

12 count = 0 ;                                // Record total rounds
13 beginIdx = c[0 - 1] ;                       // obtain begin index
14 for idx = 0 to beginIdx do
15   i = (i + 1) mod 256;
16   j = (j + S[i]) mod 256;
17   swap(S[i], S[j]);
18   count ++ ;                                // beginIdx times pass, idle

19 for idx = 0 to l do
20   i = (i + 1) mod 256;
21   j = (j + S[i]) mod 256;
22   swap(S[i], S[j]);
23   count ++ ;
24   t = (S[i] + S[j]) mod 256;
25   m[idx] = c[idx + 2]  $\oplus$  S[t] ;           // Begin decryption
26 return m;
```

2 Problem 2

(50 points) Suppose the key for a cipher is an l -bit binary string.

(a) What is the key space size of this cipher?

Answer: The key space size of cipher is 2^l .

(b) To find a key by exhaustive key search, how many keys does an attacker need to test on average?

Answer: It's reasonable to assume that the probability distribution of key is uniform distribution. That is, the probability of each value of key is $\frac{1}{2^l}$. Therefore, the average times an attacker need to test by exhaustive key search is:

$$\frac{1}{2^l} \times \sum_{i=1}^{2^l} i = \frac{1}{2^l} \times (1 + 2^l) \times 2^{l-1} = 2^{l-1} + \frac{1}{2}$$

3 Appendix: Another Possible Solution on Problem 1.

We can also modify RC4 Algorithm to design our strategy with reference to **self-synchronized stream cipher***:

1. Let t denotes the number of encryption registers, which stores the ciphertext bytes that transformed before.
2. t bytes in these encryption registers are set to be $1, 2, \dots, t$ and initialized with secret key key .
3. Once we've transfered a byte, we store this byte into one of these encryption registers, and remove it after t rounds. We replace the earlist byte that entered encryption registers with this byte.
4. We use these t bytes in encryption registers as our internal states S , which means that these t bytes will influence the generation of our key stream.
5. We generate our key stream by using maximum and minimum elements in S , which don't pay attention to the order of elements in S (in order to implement self-synchronized). That is, **compared with the change on number order of basic RC4 algorithm, we change the element itself.**

Based on the strategy above, we give the pseudocode of our Encryption algorithm (let $t = 256$):

Algorithm 3: Encryption Algorithm $Enc(k, m)$ of RC4

Input: Uniform secret key $k \in \{0, 1\}^{128}$; Message m of l bytes

Output: Ciphertext c of l bytes.

```
1 byte key[16] ;                                // secret key
2 byte K[256] ;                                // keying material
3 byte S[256] ;                                // internal states (encryption registers)

4 for  $i = 0$  to 255 do
5    $S[i] = i$ ;
6    $K[i] = key[i \bmod 16]$ ;
7  $j = 0$ ;
8 for  $i = 0$  to 255 do
9    $j = (j + S[i] + K[i]) \bmod 256$ ;
10   $S[i] = (S[i] + S[j] + K[j]) \bmod 256$  ;           // initialize S with key
    materials

11 for  $idx = 0$  to  $l$  do
12    $max = \max\{S[i] \mid i = 0, 1, \dots, 255\}$  ;           // maximum element in S
13    $min = \min\{S[i] \mid i = 0, 1, \dots, 255\}$  ;           // minimum element in S
14    $t = (S[max] + S[min]) \bmod 256$  ;                       // Begin encryption
15    $c[idx] = m[idx] \oplus S[t]$ ;
16   Replace earlist byte that entered  $S$  with  $c[idx]$ 

17 return  $c$ ;
```

*Stream Ciphers: https://blog.csdn.net/qq_43721475/article/details/104661762

Proof.

1. **Secrecy.** Obviously, the generated key stream both rely on the secret key and the previous ciphertext bytes, and has enough randomness, which can ensure the secrecy of our algorithm.
2. **Self-synchronized.** If one of the ciphertext bytes comes to failure during transformation, obviously after t rounds of transformation, this failure will be automatically corrected, which means that this failure will only influence t ciphertext bytes during transformation.

The pseudocode of our Decryption algorithm is:

Algorithm 4: Decryption Algorithm $Dec(k, c)$ of RC4

Input: Uniform secret key $k \in \{0, 1\}^{128}$; Ciphertext c of l bytes

Output: Message m of l bytes.

```
1 byte key[16] ;                                // secret key
2 byte K[256] ;                                // keying material
3 byte S[256] ;                                // internal states (encryption registers)

4 for i = 0 to 255 do
5   S[i] = i;
6   K[i] = key[i mod 16];
7 j = 0;
8 for i = 0 to 255 do
9   j = (j + S[i] + K[i]) mod 256;
10  S[i] = (S[i] + S[j] + K[j]) mod 256 ;      // initialize S with key
    materials

11 for idx = 0 to l do
12   max = max{S[i] | i = 0, 1, ..., 255} ;      // maximum element in S
13   min = min{S[i] | i = 0, 1, ..., 255} ;      // minimum element in S
14   t = (S[max] + S[min]) mod 256 ;             // Begin encryption
15   m[idx] = c[idx] ⊕ S[t];
16   Replace earliest byte that entered S with c[idx]

17 return m;
```
