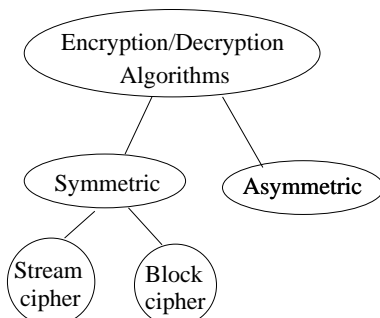


# Symmetric Ciphers: RC4 and RC5

## 1 Ciphers

There are two types of key-based algorithms: symmetric and asymmetric algorithms. In a symmetric algorithm the encryption key and the decryption key are the same. To use a symmetric algorithm, the sender and the receiver agree on a secret key securely before communicate to each other. In an asymmetric algorithm the decryption key is different from the encryption key and is hardly computed from the encryption key. In most asymmetric algorithms the encryption keys are publicly known, however the decryption keys are kept secret.

We consider symmetric algorithm. There are two types of symmetric algorithms: stream ciphers and block ciphers. A stream cipher operates on a stream of plaintext bit by bit, while a block cipher operates on a stream of plaintext block by block. That is a stream cipher encrypts plaintext individually, and an encryption key is used for one bit only in a stream cipher. A block cipher divides a stream of plaintext into blocks, and an encryption key is used for all bits in one block in a block cipher.



To understand the following example, we map 26 characters  $a, b, c, \dots, x, y, z$  into numbers  $1, 2, 3, \dots, 26$ , respectively. Further, a space is mapped into 0. Hence "go to movies tonight" is mapped into

$$7 \ 15 \ 0 \ 20 \ 15 \ 0 \ 13 \ 15 \ 22 \ 9 \ 5 \ 19 \ 0 \ 20 \ 15 \ 14 \ 9 \ 7 \ 8 \ 20 \quad (1)$$

while

$$7 \ 15 \ 15 \ 4 \ 0 \ 13 \ 15 \ 18 \ 14 \ 9 \ 14 \ 7$$

means "good morning". Operation of addition on set  $\{0, 1, 2, \dots, 26\}$  is defined as

$$a + b = \text{the remaining of sum of } a + b \text{ divided by } 27$$

Thus  $25+4$  results in 2, we write it as  $25 + 4 = 2 \pmod{27}$ . Also we have

$$7 + 9 = 16 \pmod{27}, \quad 13 + 14 = 0 \pmod{27}, \quad 26 + 21 = 20 \pmod{27}$$

etc.

**Example 1.1** (Stream cipher) Suppose plaintext is "go to movies tonight". Let encryption keystream be  $k_i = 2i$  for  $i = 0, 1, 2, \dots, 26$ . An encryption algorithm is defined by adding  $k_i$  to the  $i^{\text{th}}$  bit of the plaintext. Then the stream of ciphers is obtained from

$$\begin{aligned} &7 + 2, 15 + 4, 0 + 6, 20 + 8, 15 + 10, 0 + 12, 13 + 14, 15 + 16, 22 + 18, 9 + 20, 5 + 22, 19 + 24, \\ &0 + 26, 20 + 28, 15 + 30, 14 + 32, 9 + 34, 7 + 36, 8 + 38, 20 + 40 \\ &\quad \pmod{27} \end{aligned}$$

This is

$$9 \ 19 \ 6 \ 1 \ 25 \ 12 \ 0 \ 4 \ 13 \ 2 \ 0 \ 16 \ 26 \ 21 \ 18 \ 19 \ 16 \ 16 \ 19 \ 6$$

Back to English characters, it is "isfayl dmb pzu†sppsf".

**Example 1.2** (Block cipher) Suppose stream of plaintext is divided into blocks of length 5. The encryption key for the  $i^{\text{th}}$  block is  $k_i = 3i + 1$ . Then plaintext "go to movies tonight" is divided into 4 blocks which are mapped as follows (from (1))

$$7 \ 15 \ 0 \ 20 \ 15 \quad 0 \ 13 \ 15 \ 22 \ 9 \quad 5 \ 19 \ 0 \ 20 \ 15 \quad 14 \ 9 \ 7 \ 8 \ 20$$

## 2 RC4

The RC4 was developed in 1987 by Ronald Rivest and kept as a trade secret by RSA Data Security. On September 9, 1994, the RC4 algorithm was anonymously posted on the Internet on the Cyperpunks' "anonymous remailers" list.

The RC4 algorithm is a stream cipher. Dealing with stream ciphers, the primary question is how to generate keystream securely. The criteria for having high level security for a keystream include a large period, a high linear complexity, etc. The RC4 generates a pseudo-random sequence and thus a keystream. In the algorithm the keystream is completely independent of the plaintext used. In the following we will describe the RC4 algorithm and show examples.

### 2.1 Pseudo-random generator algorithm

Before going through the RC4 algorithm, first we study the following pseudo-random generator algorithm (PRGA). The PRGA has (i) a secret internal state which is a table  $S$  of all the  $2^n$  possible  $n$ -bit words, and (ii) two counters  $i$  and  $j$ ,  $0 \leq i, j \leq 2^n - 1$ , in it. Counters  $i$  and  $j$  both are initialised to 0, and  $S$  can be initialised randomly to any permutation  $S_0$  of  $(0, 1, 2, \dots, 2^n - 1)$ . At time  $t$ ,  $0 \leq t \leq 2^n - 1$ , it does the following

- give counter  $i$  a value  $i_{t+1} = i_t + 1 \pmod{2^n}$ ;
- give counter  $j$  a value  $j_{t+1} = j_t + S_t(i_{t+1}) \pmod{2^n}$ ;
- swap the entries  $i_{t+1}$  and  $j_{t+1}$  in the previous table  $S_t$ , i.e. let

$$S_{t+1}(i_{t+1}) = S_t(j_{t+1}), S_t(j_{t+1}) = S_t(i_{t+1}) \pmod{2^n}$$

and keep the remaining entries in the previous table  $S_t$  unchanged, i.e. let

$$S_{t+1}(m) = S_t(m), \text{ for all } m \neq i_{t+1}, j_{t+1};$$

- output

$$z_t = S_{t+1}(S_{t+1}(i_{t+1}) + S_{t+1}(j_{t+1})) \pmod{2^n}$$

When time  $t$  running from 0 to  $2^n - 1$ , the algorithm outputs a sequence

$$z_0, z_1, \dots, z_{2^n-1}.$$

#### Pseudo-code

Here is the pseudo-code for PRGA.

```
PRGA( $S$ )
 $j = 0$ 
For ( $i = 0$  to  $2^n - 1$ )
   $i = i + 1 \pmod{2^n}$ 
   $j = j + S[i] \pmod{2^n}$ 
  swap  $S[i]$  and  $S[j]$ 
   $z = S[S[i] + S[j]] \pmod{2^n}$ 
  output  $z$ 
```

**Example 2.1** Suppose  $n = 3$  and  $S$  is initialised to  $S_0 = (0, 1, 2, 3, 4, 5, 6, 7)$ . Let us see how does the algorithm work.

Starting from the initial value  $S_0$  of  $S$  with

$$S_0[0] = 0, S_0[1] = 1, S_0[2] = 2, S_0[3] = 3, S_0[4] = 4, S_0[5] = 5, S_0[6] = 6, S_0[7] = 7$$

let  $i = 0$ . The algorithm does

$$\begin{aligned} i &\leftarrow i + 1 = 1 \\ j &\leftarrow j + S_0[i] = 0 + S_0[1] = 1 \end{aligned}$$

$S$  has its updated value  $S_1$  as

$$S_1[0] = 0, S_1[1] = 1, S_1[2] = 2, S_1[3] = 3, S_1[4] = 4, S_1[5] = 5, S_1[6] = 6, S_1[7] = 7$$

and the following value is obtained

$$z_0 = S_1[S_1[i] + S_1[j]] = S_1[S_1[1] + S_1[1]] = S_1[2] = 2$$

Let  $i = 1$ . The algorithm does

$$\begin{aligned} i &\leftarrow i + 1 = 2 \\ j &\leftarrow j + S_1[i] = 1 + S_1[2] = 3 \\ S_2[2] &= 3, S_2[3] = 2 \end{aligned}$$

$S$  has its updated value  $S_2$  as

$$S_2[0] = 0, S_2[1] = 1, S_2[2] = 3, S_2[3] = 2, S_2[4] = 4, S_2[5] = 5, S_2[6] = 6, S_2[7] = 7$$

and the following value is obtained

$$z_1 = S_2[S_2[i] + S_2[j]] = S_2[S_2[2] + S_2[3]] = S_2[5] = 5$$

Let  $i = 2$ . The algorithm does

$$\begin{aligned} i &\leftarrow i + 1 = 3 \\ j &\leftarrow j + S_2[i] = 3 + S_2[3] = 5 \\ S_3[3] &= 5, S_3[5] = 2 \end{aligned}$$

$S$  has its updated value  $S_3$  as

$$S_3[0] = 0, S_3[1] = 1, S_3[2] = 3, S_3[3] = 5, S_3[4] = 4, S_3[5] = 2, S_3[6] = 6, S_3[7] = 7$$

and the following value is obtained

$$z_2 = S_3[S_3[i] + S_3[j]] = S_3[S_3[3] + S_3[5]] = S_3[7] = 7$$

Let  $i = 3$ . The algorithm does

$$\begin{aligned} i &\leftarrow i + 1 = 4 \\ j &\leftarrow j + S_3[i] = 5 + S_3[4] = 9 = 1 \pmod{8} \\ S_4[1] &= 4, S_4[4] = 1 \end{aligned}$$

*S* has its updated value  $S_4$  as

$$S_4[0] = 0, S_4[1] = 4, S_4[2] = 3, S_4[3] = 5, S_4[4] = 1, S_4[5] = 2, S_4[6] = 6, S_4[7] = 7$$

and the following value is obtained

$$z_3 = S_4[S_4[i] + S_4[j]] = S_4[S_4[4] + S_4[1]] = S_4[5] = 2$$

Let  $i = 4$ . The algorithm does

$$i \leftarrow i + 1 = 5$$

$$j \leftarrow j + S_4[i] = 1 + S_4[5] = 3$$

$$S_5[3] = 2, S_5[5] = 5$$

*S* has its updated value  $S_5$  as

$$S_5[0] = 0, S_5[1] = 4, S_5[2] = 3, S_5[3] = 2, S_5[4] = 1, S_5[5] = 5, S_5[6] = 6, S_5[7] = 7$$

and the following value is obtained

$$z_4 = S_5[S_5[i] + S_5[j]] = S_5[S_5[5] + S_5[3]] = S_5[7] = 7$$

Let  $i = 5$ . The algorithm does

$$i \leftarrow i + 1 = 6$$

$$j \leftarrow j + S_5[i] = 3 + S_5[6] = 9 = 1 \pmod{8}$$

$$S_6[1] = 6, S_6[6] = 4$$

*S* has its updated value  $S_6$  as

$$S_6[0] = 0, S_6[1] = 6, S_6[2] = 3, S_6[3] = 2, S_6[4] = 1, S_6[5] = 5, S_6[6] = 4, S_6[7] = 7$$

and the following value is obtained

$$z_5 = S_6[S_6[i] + S_6[j]] = S_6[S_6[6] + S_6[1]] = S_6[2] = 3$$

Let  $i = 6$ . The algorithm does

$$i \leftarrow i + 1 = 7$$

$$j \leftarrow j + S_6[i] = 1 + S_6[7] = 8 = 0 \pmod{8}$$

$$S_7[0] = 7, S_7[7] = 0$$

*S* has its updated value  $S_7$  as

$$S_7[0] = 7, S_7[1] = 6, S_7[2] = 3, S_7[3] = 2, S_7[4] = 1, S_7[5] = 5, S_7[6] = 4, S_7[7] = 0$$

and the following value is obtained

$$z_6 = S_7[S_7[i] + S_7[j]] = S_7[S_7[7] + S_7[0]] = S_7[7] = 0$$

Let  $i = 7$ . The algorithm does

$$i \leftarrow i + 1 = 0$$

$$j \leftarrow j + S_7[i] = 0 + S_7[0] = 7$$

*S* has its updated value  $S_8$  as

$$S_8[0] = 0, S_8[1] = 6, S_8[2] = 3, S_8[3] = 2, S_8[4] = 1, S_8[5] = 5, S_8[6] = 4, S_8[7] = 7$$

and the following value is obtained

$$z_7 = S_8[S_8[i] + S_8[j]] = S_8[S_8[0] + S_8[7]] = S_8[7] = 7$$

By now, the output sequence is 2, 5, 7, 2, 7, 3, 0, 7.

## 2.2 RC4 algorithm

The RC4 algorithm aims to generate variable length keystream. It starts at a so-called key scheduling algorithm (KSA) which, for a given initialised key string  $K = (K[1], K[2], \dots, K[\ell])$ , generates a table  $S$  of length  $2^n$ , then applies PRGA to generate a keystream, here the output  $S$  from KSA is taken as the initialised value for  $S_0$ . Here is the pseudo-code for KSA. Assume  $S_0 = (0, 1, 2, \dots, 2^n - 1)$ .

```

KSA( $K$ )
 $j = 0$ 
For ( $i = 0$  to  $2^n - 1$ )
     $i = i + 1 \pmod{2^n}$ 
     $j = j + S[i] + K[i \bmod \ell] \pmod{2^n}$ 
    swap  $S[i]$  and  $S[j]$ 

```

**Example 2.2** Let  $n = 2$ ,  $\ell = 3$ , initial key string be  $K = (1, 0, 1)$ . Then running KSA step by step as follows. When  $i = 0$ ,

$$i \leftarrow i + 1 = 1$$

$$j \leftarrow j + S_0[i] + K[i \bmod 3] = 0 + S_0[1] + K[1] = 1$$

this results in an updated  $S_1$  which is identical to  $S_0$  as swapping does not do anything

$$S_1[0] = 0, S_1[1] = 1, S_1[2] = 2, S_1[3] = 3.$$

When  $i = 1$ ,

$$i \leftarrow i + 1 = 2$$

$$j \leftarrow j + S_1[i] + K[i \bmod 3] = 1 + S_1[2] + K[2] = 1 + 2 + 1 = 4 = 0 \pmod{4}$$

$$S_2[0] = 2, S_2[2] = 0$$

this results in an updated  $S_2$  as

$$S_2[0] = 2, S_2[1] = 1, S_2[2] = 0, S_2[3] = 3.$$

When  $i = 2$ ,

$$i \leftarrow i + 1 = 3$$

$$j \leftarrow j + S_2[i] + K[i \bmod 3] = 0 + S_2[3] + K[0] = 0 + 3 + 1 = 0$$

$$S_3[0] = 3, S_3[3] = 2$$

this results in an updated  $S_3$  as

$$S_3[0] = 3, S_3[1] = 1, S_3[2] = 0, S_3[3] = 2.$$

When  $i = 3$ ,

$$i \leftarrow i + 1 = 4 = 0 \pmod{4}$$

$$j \leftarrow j + S_3[i] + K[i \bmod 3] = 0 + S_3[0] + K[0] = 0 + 3 + 1 = 4 = 0 \pmod{4}$$

this results in an updated  $S_4$

$$S_4[0] = 3, S_4[1] = 1, S_4[2] = 0, S_4[3] = 2.$$

Taking this final output  $(3, 1, 0, 2)$  as the initial value for  $S'$ , PRGA is running step by step as follows. When  $i = 0$ ,

$$\begin{aligned} i &\leftarrow i + 1 = 1 \\ j &\leftarrow j + S_0[i] = 0 + S_0[1] = 0 + 1 = 1 \end{aligned}$$

$S'$  has its updated value  $S'_1$  as

$$S'_1[0] = 3, S'_1[1] = 1, S'_1[2] = 0, S'_1[3] = 2$$

and the following value is obtained

$$z_0 = S'_1[S'_1[i] + S'_1[j]] = S'_1[S'_1[1] + S'_1[1]] = S'_1[2] = 0$$

When  $i = 1$ ,

$$\begin{aligned} i &\leftarrow i + 1 = 2 \\ j &\leftarrow j + S'_1[i] = 1 + S'_1[2] = 1 + 0 = 1 \\ S'_2[2] &= 1, S'_2[1] = 0 \end{aligned}$$

$S'$  has its updated value  $S'_2$  as

$$S'_2[0] = 3, S'_2[1] = 0, S'_2[2] = 1, S'_2[3] = 2$$

and the following value is obtained

$$z_1 = S'_2[S'_2[i] + S'_2[j]] = S'_2[S'_2[2] + S'_2[1]] = S'_2[1] = 0$$

When  $i = 2$ ,

$$\begin{aligned} i &\leftarrow i + 1 = 3 \\ j &\leftarrow j + S'_2[i] = 1 + S'_2[3] = 1 + 2 = 3 \end{aligned}$$

$S'$  has its updated value  $S'_3$  as

$$S'_3[0] = 3, S'_3[1] = 0, S'_3[2] = 1, S'_3[3] = 2$$

and the following value is obtained

$$z_2 = S'_3[S'_3[i] + S'_3[j]] = S'_3[S'_3[3] + S'_3[3]] = S'_3[0] = 3$$

When  $i = 3$ ,

$$\begin{aligned} i &\leftarrow i + 1 = 4 = 0 \pmod{4} \\ j &\leftarrow j + S'_3[i] = 3 + S'_3[0] = 3 + 3 = 2 \pmod{4} \\ S'_4[0] &= 1, S'_4[2] = 3 \end{aligned}$$

$S'$  has its updated value  $S'_4$  as

$$S'_4[0] = 1, S'_4[1] = 0, S'_4[2] = 3, S'_4[3] = 2$$

and the following value is obtained

$$z_3 = S'_4[S'_4[i] + S'_4[j]] = S'_4[S'_4[0] + S'_4[2]] = S'_4[0] = 1$$

Now we have got an output sequence  $0, 0, 3, 1$  which is the keystream generated by RC4 algorithm.

### 3 RC5

In this section, we briefly mention the RC5 algorithm. We refer the reader to [1] for more complex account.

The RC5 is a block cipher investigated by Rivest in 1994. We describe the RC5 cipher for 64-bit data blocks, although the RC5 cipher has a variable length block. The encryption algorithm uses  $2r + 2$  32-bit strings,  $S_0, S_1, \dots, S_{2r+1}$ , that are key-dependent and called words. Three operations are involved in the encryption algorithm:  $\oplus$ ,  $+$  ( $\text{mod } 32$ ) and  $\leftarrow$ . The operation  $\oplus$ , called XOR, is bit-wise exclusive-or and each bit is calculated by following the rules:

$$0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0.$$

For example, let

$$\begin{aligned} X &= 01010101000000001111111100001111 \\ Y &= 00000010000100001000010000001111 \end{aligned} \quad (2)$$

Then  $X \oplus Y$  is given as follows

$$X \oplus Y = 01010111000100000111101100000000$$

The operation  $+$  ( $\text{mod } 32$ ) is defined as

$$a + b = \text{the remaining of sum of } a + b \text{ divided by } 32.$$

The operation  $\leftarrow$ , called a rotation, operates on two strings  $X$  and  $Y$  and is defined as follows.

$X \leftarrow Y$  represents that

$X$  is rotated to the left by the number of positions indicated by the value of  $Y$  ( $\text{mod } 32$ )

For example, let  $X$  and  $Y$  be given as in (2). Then the value of  $Y$  is

$$2^{25} + 2^{20} + 2^{15} + 2^{10} + 2^3 + 2^2 + 2 + 2^0 = 15 \pmod{32},$$

and  $X \leftarrow Y$  results in a string obtained by rotating  $X$  to left by 15 positions, that is

$$X \leftarrow Y = 01111111100001111010101010000000$$

#### Encryption algorithm

Suppose 32-bit words  $S_0, S_1, \dots, S_{2r+1}$  are given. The algorithm starts at dividing the 64-bit plaintext into two parts  $L_0, R_0$ , where  $L_0$  is the first left 32-bits and  $R_0$  the next 32-bits. Then the algorithm does the following, for each  $i = 1, 2, \dots, r$ ,

- give a value to  $L_i$  as  $L_i = ((L_{i-1} \oplus R_{i-1}) \leftarrow R_{i-1}) + S_{2i} \pmod{32}$ ;
- give a value to  $R_i$  as  $R_i = ((L_{i-1} \oplus R_{i-1}) \leftarrow L_{i-1}) + S_{2i+1} \pmod{32}$ .

The output is the cipher, a 64-bit string of  $L_r$  followed by  $R_r$ . Here is the pseudo-code.

```

Input plaintext  $L, R$ 
Input words  $S_0, S_1, \dots, S_{2r+1}$ 
For ( $i = 1$  to  $r$ )
     $L = ((L \oplus R) \leftarrow R) + S_{2i} \pmod{32}$ 
     $R = ((L \oplus R) \leftarrow L) + S_{2i+1} \pmod{32}$ 
Output  $L, R$ 

```

The decryption is as easy as encryption is. Here is the pseudo-code for decryption.

Input ciphertext $L, R$
Input words $S_0, S_1, \dots, S_{2r+1}$
For ( $i = r$ down to 1)
$R = ((R - S_{2i+1}) \rightarrow L) \oplus L \pmod{32}$
$L = ((L - S_{2i}) \rightarrow R) \oplus R \pmod{32}$
$R = R - S_1$
$L = L - S_0$
Output $L, R$

where

$X \rightarrow Y$  represents that

$X$  is rotated to the right by the number of positions indicated by the value of  $Y \pmod{32}$

## 4 Summary

Two symmetric ciphers have been introduced. Both RC4 and RC5 are simple and fast algorithms. The advantages of RC4, RC5 are their security and efficiency. There are also attacks to the RC4 and RC5 according to the weakness of the algorithms which can be found in the literature, for example, through websites listed in references below.

## References

- [1] B. Schneier, *Applied Cryptography*, John Wiley & Sons Inc., (1996).
- [2] <http://www.rsasecurity.com/rsalabs/faq/3-6-3.html>
- [3] <http://security.ece.orst.edu/koc/ece575/97Project/Sadagopan+Shah/>