جامـــــعة المـــــستقبل
AL MUSTAQBAL UNIVERSITY

قســـم الامـــــن الـــــــسيبرانـــــــي

Department of Cyber Security

Subject:

RC5

Class:

Second

Lecturer:

Asst. lect. Mustafa Ameer Awadh

Lecture: (5)

Lecture 5:

RC5

RC5

The RC5 encryption algorithm was designed by Ronald Rivest of Massachusetts Institute of Technology (MIT) and it first appeared in December 1994. RSA Data Security, Inc. estimates that RC5 and its successor, RC6, are strong candidates for potential successors to DES. RC5 analysis (RSA
Laboratories) is still in progress and is periodically updated to reflect any additional findings.

Description of RC5

RC5 is a symmetric block cipher designed to be suitable for both **software** and **hardware** implementation . It is a parameterized algorithm, with a ***variable block size***, a ***variable number of rounds*** and a ***variable-length key***. This provides the opportunity for great **flexibility** in both performance characteristics and the level of security.

A particular RC5 algorithm is designated as RC5-**W/R/B**. The number of bits in a word, W, is a parameter of RC5. Different choices of this parameter result in different RC5 algorithms. RC5 is iterative in structure, with a variable number of rounds. The number of rounds, R, is a second parameter of RC5. RC5 uses a variable-length secret key. The key length **B** (in bytes) is a third parameter of RC5. These parameters are summarized as follows:

W: The word size, in bits. The standard value is ***32bits***; allowable values are 16, 32 and 64. RC5 encrypts two-word blocks so that the plaintext and ciphertext blocks are each 2w bits long. **R**: The number of rounds. Allowable values of R are 0, 1, ..., 255. Also, the expanded key table S contains **T = 2 (r + 1) words**.

B: The number of bytes in the secret key K. Allow able values of b are 0, 1, ..., 255 byte.

K:The b-byte secret key; K[0], K[1], ...,K[b − 1]

RC5 consists of three components: a **key expansion algorithm**, an **encryption algorithm** and a **decryption algorithm**. These algorithms use the following three primitive operations:

- ⊞ Addition of words modulo 2^w
- ⊕ Bit-wise exclusive-OR of words
- <<< Rotation symbol: the rotation of x to the left by y bits is denoted **by x <<< y.**

One design feature of RC5 is its simplicity, which makes RC5 easy to implement. Another feature of RC5 is its heavy use of data-dependent rotations in encryption; this feature is very useful in preventing both differential and linear cryptanalysis.

Key Expansion

The key-expansion algorithm expands the user's key K to fill the expanded key table S, so that S resembles an array of t = 2(r + 1) random binary words determined by K. It uses two word-size magic constants P_w and Q_w defined for arbitrary w as shown below:

$$P_w = \text{Odd}((e - 2)2^w)$$

$$Q_w = \text{Odd}((\varphi - 1)2^w)$$

where e = 2.71828 ... (base of natural logarithms) $\Phi = (1 + \sqrt{5})/2$

$= 1.61803$ ... (golden ratio)

Odd(x) is the odd integer nearest to x.

First algorithmic step of key expansion: This step is to copy the secret key K[0, 1, ..., b − 1]into an array L[0, 1, ..., c − 1]of c =[b/u] words, where u = w/8 is the number of bytes/word.

This first step will be achieved by the following pseudocode operation:

```
for i = b – 1 down to 0 do
        L[i/u] = (L[i/u] <<< 8) + K[i]
```

where all bytes are unsigned and the array L is initially zeroes.

Second algorithmic step of key expansion: This step is to initialize array **S** to a particular fixed pseudo-random bit pattern, using an arithmetic progression modulo 2^w determined by two constants P_w **and** Q_w. S[0] = P_w:

for i = 1 to t − 1 do

S[i] = S[i − 1] + Q_w.

Third algorithmic step of key expansion: This step is to mix in the user's secret key in three passes over the arrays **S** and **L**. More precisely, due to the potentially different sizes of S and L, the larger array is processed three times, and the other array will be handled more after.

i = j = 0; A = B = 0;

do

3∗ max (t,c) times:

A = S[i] = (S[i] + A + B) <<< 3

B = L[j] = (L[j] + A + B) <<<

(A + B); i = (i + 1)(mod t);

j = (j + 1)(mod c).

Note that with the key-expansion function it is not so easy to determine K from S**.**

Example. Since w = 32,r = 12 and b=16

u = w/8 = 32/8 = 4 bytes/word

c =[b/u]=[16/4]= 4 words

t = 2(r + 1) = 2(12 + 1) = 26 words

The plaintext and the user's secret key are given as follows:

Plaintext = eedba521 6d8f4b15

Key = 91 5f 46 19 be 41 b2 51 63 55 a5 01 10 a9 ce 91.

1. **Key expansion**

Two magic constants

P32 = 3084996963 = 0xb7e15163

Q32 = 2654435769 = 0x9e3779b9

Step 1

For i = b − 1 down to 0 do

L[i/u] = (L[i/u] <<< 8) + K[i] where b = 16,u = 4 and L is initially 0.

L[i/4] = L[3] for i = 15, 14, 13 and 12.

L[3] = (L[3] <<< 8) + K[15] = 00 + 91 = 91

L[3] = (L[3] <<< 8) + K[14] = 9100 + ce = 91ce

L[3] = (L[3] <<< 8) + K[13] = 91ce00 + a9 = 91cea9

***L[3] = (L[3] <<< 8) + K[12] = 91cea900 + 10 = 91cea910**

L[i/4] = L[2] for i = 11, 10, 9 and 8.

L[2] = (L[2] <<< 8) + K[11] = 00 + 01 = 01

L[2] = (L[2] <<< 8) + K[10] = 0100 + a5 = 01a5

L[2] = (L[2] <<< 8) + K[9] = 01a500 + 55 = 01a555

***L[2] = (L[2] <<< 8) + K[8] = 01a55500 + 63 = 01a55563**

L[i/4] = L[1] for i = 7, 6, 5 and 4.

L[1] = (L[1] <<< 8) + K[7] = 00 + 51 = 51

L[1] = (L[1] <<< 8) + K[6] = 5100 + b2 = 51b2

L[1] = (L[1] <<< 8) + K[5] = 51b200 + 41 = 51b241

***L[1] = (L[1] <<< 8) + K[4] = 51b24100 + be = 51b241be**

L[i/4] = L[0] for i = 3, 2, 1 and 0.

L[0] = (L[0] <<< 8) + K[3] = 00 + 19 = 19

L[0] = (L[0] <<< 8) + K[2] = 1900 + 46 = 1946

L[0] = (L[0] <<< 8) + K[1] = 194600 + 5f = 19465f

***L[0] = (L[0] <<< 8) + K[0] = 19465f00 + 91 = 19465f91**

Thus, converting the secret key from bytes to words (*) yields:

L[0] = 19465f91

L[1] = 51b241be

L[2] = 01a55563

L[3] = 91cea910

Step 2

S[0] = P_{32}.

For i = 1 to 25 do

S[i] = S[i − 1] + Q_{32}:

S[0] = b7e15163

S[1] = S[0] + Q32 = b7e15163 + 9e3779b9 = 5618cb1c

S[2] = S[1] + Q32 = 5618cb1c + 9e3779b9 = f45044d5

S[3] = S[2] + Q32 = f45044d5 + 9e3779b9 = 9287be8e

When the iterative processes continue up to t − 1 = 2(r + 1) − 1 = 25, we can obtain the expanded key table S as shown below:

S[0] = b7e15163 S[09] = 47d498e4 S[18] = d7c7e065

S[1] = 5618cb1c S[10] = e60c129d S[19] = 75ff5a1e

S[2] = f45044d5 S[11] = 84438c56 S[20] = 1436d3d7

S[3] = 9287be8e S[12] = 227b060f S[21] = b26e4d90

S[4] = 30bf3847 S[13] = c0b27fc8 S[22] = 50a5c749

S[5] = cef6b200 S[14] = 5ee9f981 S[23] = eedd4102.

S[6] = 6d2e2bb9 S[15] = fd21733a S[24] = 8d14babb

S[7] = 0b65a572 S[16] = 9b58ecf3 **S[25]** = 2b4c3474

S[8] = a99d1f2b S[17] = 399066ac

Step 3

i = j = 0;A = B = 0;

3 × max(t, c) = 3 × 26 = 78 times

A = S[i] = (S[i] + A + B) <<< 3

B = L[j] = (L[j] + A + B) <<< (A + B)

i = i + 1(mod 26)

j = j + 1(mod 4)

A = S[0] = (b7e15163 + 0 + 0) <<< 3

= b7e15163 <<< 3 = bf0a8b1d

B = L[0] = (19465f91 + bf0a8b1d) <<< (A + B)

= d850eaae <<< bf0a8b1d = db0a1d55

A = S[1] = (5618cb1c + bf0a8b1d + db0a1d55) <<< 3

= f02d738e <<< 3 = 816b9c77

B = L[1] = (51b241be + 816b9c77 + db0a1d55) <<< (A + B)

= ae27fb8a <<< 5c75b9cc = 7fb8aae2

A = S[2] = (f45044d5 + 816b9c77 + 7fb8aae2) <<< 3

= f5748c2e <<< 3 = aba46177

B = L[2] = (01a55563 + aba46177 + 7fb8aae2) <<< (A + B)

= 2d0261bc <<< 2b5d0c59 = 785a04c3

A = S[3] = (9287be8e + aba46177 + 785a04c3) <<< 3

= b68624c8 <<< 3 = b4312645

B = L[3] = (91cea910 + b4312645 + 785a04c3) <<< (A + B)

= be59d418 <<< 2c8b2b08 = 59d418be

…

A = S[25] = (4e0d4c36 + f66a1aaf + 6d7f672f) <<< 3

= b1f6ce14,<<< 3 = 8fb670a5,

B = L[1] = (cdfc2657 + 8fb670a5 + 6d7f672f) <<< (A + B)

= cb31fe2b <<< fd35d7d4 = e2bcb31f

Encryption

The input block to RC5 consists of two w-bit words given in two registers, A and B. The output is also placed in the registers A and B. Recall that RC5 uses an expanded key table, S[0, 1,...,t − 1], consisting of t = 2(r + 1) words. The key-expansion algorithm initializes S from the user's given secret key parameter K. However, the S table in RC5 encryption is not like an S-box used by DES. The encryption algorithm is given in the pseudo code as shown below:

A = A +

S[0]; B

= B +

S[1]; for

i = 1 to r

do

A = ((A ⊕ B) <<< B) + S[2i];

B = ((B ⊕ A) <<< A) + S[2i + 1];

The output is in the registers A and B.

To encrypt the 64-bit input block, use of the following steps:

- Use the expanded key table S[0, 1, ..., 25]
- Input the plaintext in **two 32-bit registers, A and B.**
- Compute the ciphertext using the RC5 encryption algorithm

Decryption

RC5 decryption is given in the pseudocode as shown below.

For i = r downto1do

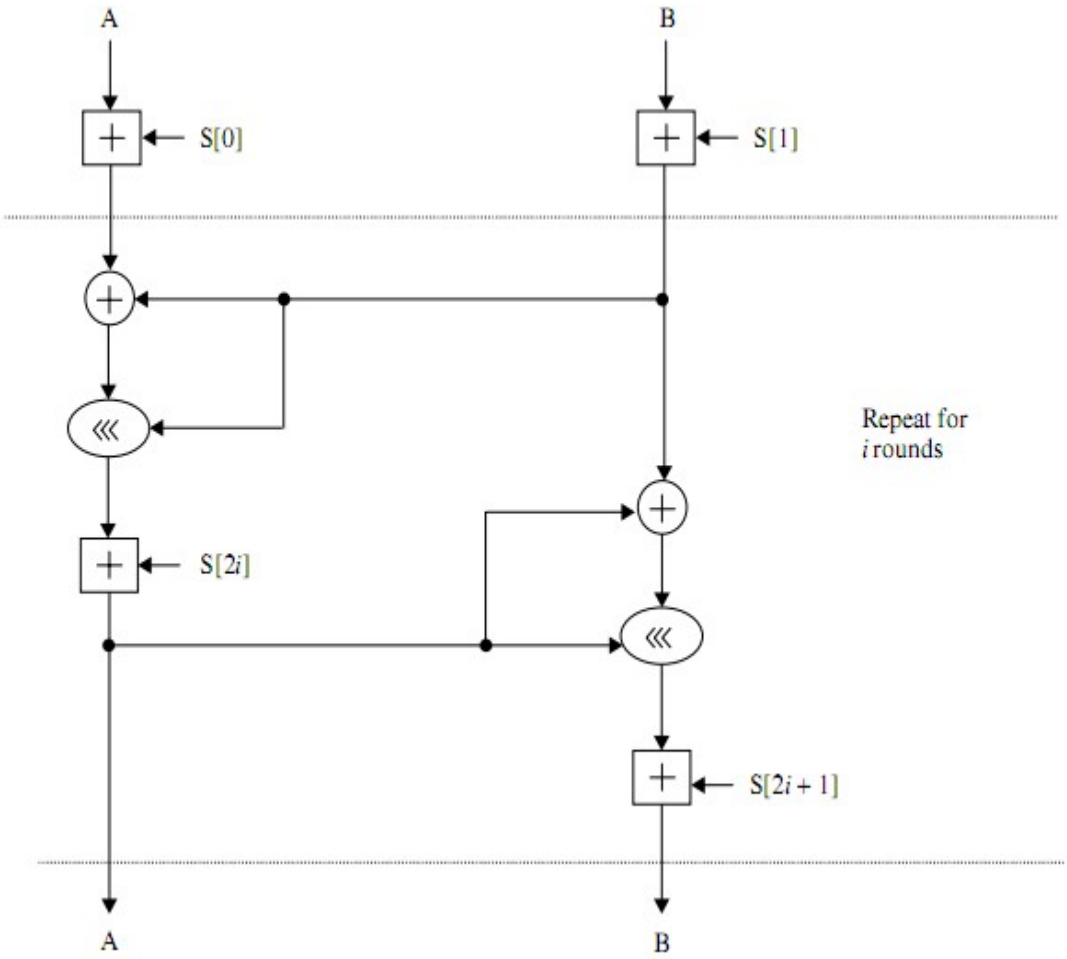
$B = ((B - S[2i + 1]) >>> A) \oplus A$

$A = ((A - S[2i]) >>> B) \oplus B$

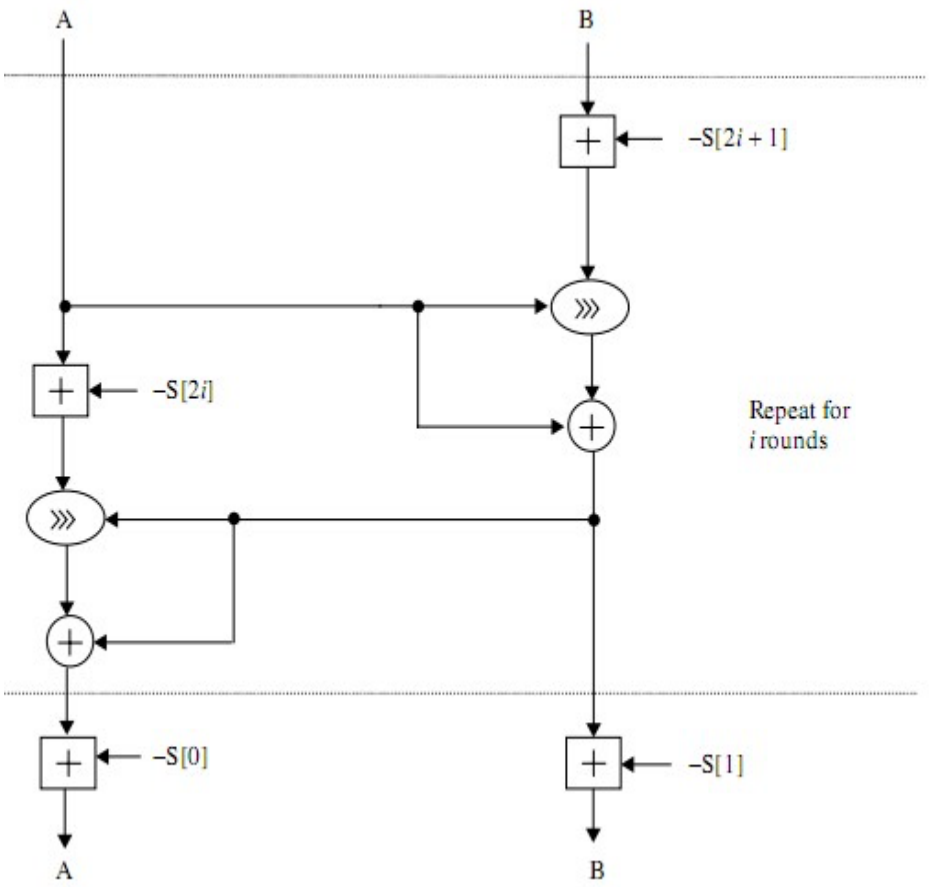
$B = B - S[1]$

$A = A - S[0]$

The decryption routine is easily derived from the encryption routine. The RC5 encryption/decryption algorithms are illustrated as shown in Figures 3.10 and 3.11, respectively.



RC5 encryption algorithm.



RC5 decryption algorithm.

A = ((A ⊕ B) <<< B) + S[2i];

B = ((B ⊕ A) <<< A) + S[2i + 1];