Proposed 256 bits RC5 Encryption Algorithm Using Type-3 Feistel Network

Dr. Yossra H. Ali*

Received on:23/7/2009
Accepted on:11/3/2010

Abstract

Proposed 256 bits RC5 is an improvement to RC5, designed to meet the requirements of increased security and better performance. Proposed 256 bits RC5 algorithm makes use of data dependent rotations. One new feature of proposed 256 bits RC5 algorithm is modified its design to use four 64-bit registers rather than two 32-bit registers. The proposal 256 bits RC5 algorithm using Type-3 Feistel network which is iterated simple function 20 times. An Avalanche Effect of RC5 is about 31.372 if we change the same amount of information in key for 256 bits RC5 then the Avalanche Effect is about 142.909. The proposed algorithm is resistant to matching and a dictionary attack which is increased the security of the previous RC5 algorithm by using block size of 256 bits instead of 64 bits.

Keywords: Cryptography, RC5, Type -3 Feistel.

اقتراح خوارزمية تشفير 256 bits RC5 باستخدام شبكة Type -3 Feistel

الخلاصة

خوارزمية تشفير 256 bits RC5 المقترحة هي تطوير الى RC5, صممت لتحقيق متطلبات زيادة الامنية وتحسين الاداء,هذه الخوارزمية تستخدم الابدالات المعتمدة على البيانات والتي هي واحدة من الخواص الجديدة لهذه الخوارزمية حيث تستخدم 64 bits بدلا من 32 bits وهي تستخدم شبكة Type -3 Feistel والتي تكرر دالة بسيطة 20 مرة. الـ Avalanche Effect للـ RC5 حوالي 31.372واذا غيرنا نفس كمية المعلومات لمفتاح الخوارزمية المقترحة فسيكون الـ Avalanche Effect حوالي 142.909 . لذلك فالخوارزمية المقترحة قوية لهجوم ال matching و dictionary والتي زيدت الامنية للخوارزمية السابقة باستخدام حجم كتلة 256 bits بدلا من 64 bits.

1. Introduction

Symmetric encryption system consists of a set of possible plaintext messages (i.e., the plaintext message space M), a set of possible ciphertexts (i.e., the ciphertext space C), a set of possible keys (i.e., the key space K), as well as two families of encryption and decryption functions (or algorithms) that are inverse to each other as shown in figure(1).

• A family E = {Ek : k K} of encryption functions*Ek* :*M→C;*

• *A family D = {Dk : k K} of* decryption functions*Dk* : *C →M.*

For every key k K and every message m M, the functions Dk

*Computer Sciences Department ,University of Technology / Baghdad

https://doi.org/10.30684/etj.28.12.5
2412-0758/University of Technology-Iraq, Baghdad, Iraq

and Ek must be inverse to each other (i.e., Dk(Ek(m)) = Ek(Dk(m)) = m) [1].

Symmetric-key block ciphers have long been used as a fundamental cryptographic element for providing information security. Although they are primarily designed for providing data confidentiality, their versatility allows them to serve as a main component in the construction of many cryptographic systems such as pseudorandom number generators, message authentication protocols, stream ciphers, and hash functions.

There are many symmetric-key block ciphers which offer different levels of security, flexibility, and efficiency. Among the many symmetric-key block ciphers currently available, some (such as DES, RC5, CAST, Blowfish, FEAL, SAFER, and IDEA) have received the greatest practical interest.

Most symmetric-key block ciphers (such as DES, RC5, CAST, and Blowfish) are based on a "Feistel" network construct and a "round function". The round function provides a basic encryption mechanism by composing several simple linear and nonlinear operations such as exclusive-or, substitution, permutation, and modular arithmetic.

Different round functions provide different levels of security, efficiency, and flexibility. The strength of a Feistel cipher depends heavily on the degree of diffusion and non-linearity properties provided by the round function [2].

This paper also present some existing modern cipher such as ABC is a substitution-permutation network comprising 17 rounds with 3 different kinds of round functions. It is derived from MMB and SAFFER block cipher [3] and Unbalanced Feistel Networks and Block-Cipher Design (UFNs) consist of a series of rounds in which one part of the block operates on the rest of the block [4]. And also PRESENT: An Ultra-Lightweight which is block cipher. It is an example of an SP-network and consists of 31 rounds. The block length is 64 bits and two key lengths of 80 and 128 bits are supported [5].

2. RC5 Algorithm

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 is still in progress and is periodically updated to reflect any additional findings [6].

2.1 Description of RC5

RC5 is a symmetric block cipher designed to be suitable for both software and hardware implementation. It is a parameterised 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 summarised as follows:

***w*: The word size, in bits**. The standard value is 32 bits; allowable values are 16, 32 and 64. RC5 encrypts two-word blocks so that the plaintext and ciphertext blocks are each 2*w* 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. Allowable values of *b* are 0, 1, . . . , 255. *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:

1. + Addition of words modulo 2w
2. Bit-wise exclusive-OR of words
3. <<< 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 [6].

2.2 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 *Pw* and *Qw* defined for arbitrary *w* as shown below:

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

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

Where

$e = 2.71828\ldots$ **(base of natural logarithms)**

$\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$ **(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 $c = \lceil b/u \rceil$ 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 initialise array *S* to a particular fixed pseudo-random bit pattern, using an arithmetic progression modulo 2^w determined by two constants *Pw* and *Qw*.

S*[0] = *Pw
for *i* = 1 to *t* − 1 do *S*[*i*] = *S*[*i* − 1] + *Qw*

***- 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*, due to the one-wayness [6].

2.3 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 initialises *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 pseudocode as shown below:

***A* = *A* + *S*[0]**
***B* = *B* + *S*[1]**
for *i* = 1 to *r* do
***A* = ((*A* ⊕ *B*) <<< *B*) + *S*[2*i*]**
***B* = ((*B* ⊕ *A*) <<< *A*) + *S*[2*i* + 1]**
The output is in the registers *A* and *B*

2.4 Decryption

RC5 decryption is given in the pseudocode as shown below.

For *i* = *r* down to 1 do
B* = ((*B* − *S*[2*i* + 1]) >>> *A*) ⊕ *A
A* = ((*A* − *S*[2*i*]) >>> *B*) ⊕ *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 Figure (2) and (3) respectively [6].

3. Type-3 Feistel network

In *Type-3 Feistel network* each block consists of four words as shown in figure (4). Among the various network-structures which are capable of handling four words in a block, it seems that a type-3 Feistel network provides the best tradeoff between speed, strength and suitability for analysis.

A type-3 Feistel network consists of many rounds; where in each round one data word (and a few key words) is used to modify all the other data words. Compared with a type-1 Feistel network (where in each round one data word is used to modify one other data word), this construct provides much better diffusion properties with only a slightly added cost. Hence, fewer rounds can be used to achieve the same strength [2].

A type-3 Feistel network provides the best tradeoff between speed, strength and suitability for analysis. A desirable property of an encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. This is so called Avalanche Effect[2].

4. Proposed 256 bits RC5 Encryption Algorithm

In general the proposed algorithm differs from the previous RC5 algorithm which encrypts and decrypts 64-bit block size. The proposed algorithm could be used to encrypt and decrypt 256 bits block size. Figure (5) shows the structure of proposed algorithm which in fact is a Type-3 Feistel network. It consists of splitting the plaintext into four 64 bits words. A type-3 Feistel network consists of many rounds in each round the output of RC5 is the input to E-function (which is derived from MARS algorithm) then, one data word will be used as the input to the E-function and the three output words from the E-function are added or XORed to the other three data words. In addition, the source word is rotated by 13 positions to the left.

The Proposed algorithm is designed to use a full menu of "strong operations" supported in modern computers to achieve better security properties, high speed, and implementation flexibility. Proposal algorithm will be used primitive operations (add, subtract, multiply, exclusive-or, and data-dependent rotate).

The Algorithm, in Figure (6), shows the encryption operation of proposed algorithm in details. The cipher is working with64 bit words in that all the operations. In this algorithm the following notations are used. The denoted by c^d a bitwise Exclusive-OR of the two words c and d , c+d addition modulo 2^{64}, c-d subtraction modulo 2^{64}, and c×d multiplication modulo 2^{64}. Also, c<<<d and c>>>d, denote cyclic rotations of the 64-bit word c by d positions to the left and right, respectively.

The decryption operation of proposed algorithm is the inverse of the encryption operation as shown in figure (7).

4.1 The E-function

The E-function is derived from Mars Encryption algorithm [2] with some modifications in proposed algorithm. The E-function takes as input one data of 64 bits word and uses two more key words to produce three output words. In this function three temporary variables will be used, denoted below by L, M and R (for left, middle and right). Below it is also refer to these variables as the three "lines" in the function. Initially, R will be set to hold the value of the source word rotated by 13 positions to the left, and M will be set to hold the sum of the source word and the first key word as shown in figure (8).

The second key word is multiplied (constrained to contain an odd integer) into R and then rotate R by 5 positions to the left (so the 5 highest bits of the product becomes the 5 lowest bits of R after the rotation). Then R XORed into L, and also view the five lowest bits of R as a rotation amount between 0 and 31, and rotate M to the left by this amount. Next, R will be rotated R by 5 more positions to the left and XOR it into L. Finally, again the five lowest bits of R will be viewed as a rotation amount and rotate L to the left by this amount. The first output word of the E-function is L, the second is M and the third is R (see figure (6)).

5. Security of the Proposed 256 bits RC5 Algorithm

5.1 Matching ciphertext attack

The block size of 256 bits makes proposed algorithm resistant to the matching ciphertext attack. Where after encryption of 2^{128} blocks, equal ciphertexts can be expected and information is leacked about plaintext but, the previous RC5algorithm with 64 bits block size will be required to 2^{32} ciphertext [7].

5.2 Dictionary Attacks

As the block size is 64 bits, a dictionary will require 2^{64} different plaintexts to allow the attacker to encrypt or decrypt arbitrary message under an unknown key, While the proposed algorithm will require 2^{256} different plaintexts. This

attack applies to any deterministic block cipher with 128-bit blocks regardless of its design [7].

5.3 Avalanche Effect

Horst Feistel referred to the avalanche effect as: "*a small change in the key gives rise to a large change in the ciphertext*" [8].

The following example describe this operation

If we take plaintext in hexadecimal "00000000" and using

Key1 = 00000000000000000000000000000000.

Key2= 00000000000000000000000000000001.

When encrypted by RC5 before improvement ,The plaintext that will be encrypted by key1 and key2 respectively as shown below

Ciphertext1 = *"4b56cf91e4b70955"*

Ciphertext2 = *"3be66a18629dcfce "*

if the ciphertext1 and ciphertext2 converted to binary digit

"01001011010101101100111110010001111001001011011100000100101010101"

"00111011111001100110101000011000011000101001110111001111111001110"

The changing of bits between two ciphertexts equal to 28 which are represents the avalanche effect.

In this section is prepared for making statistical test on the ciphertext that produced from encryption the following plaintext:

"Cryptography, simply defined, is the process of combining some input data, called the plaintext, with a user-specified password to generate an encrypted output, called ciphertext, in such a way that, given the ciphertext, no one can recover the original plaintext without the encryption password in a reasonable amount of time ."

And using a key of length 32 bytes where the key is "**ffffffffffffffffffffffffffffffff**

Then compute the avalanche effect to above plaintext by RC5 before and after improvement, these operations describe in tables (1) and (2).

Table (1) shows the avalanche effect on the plaintext when only one bit is changed in the key by using proposed 256 bits RC5 algorithm.

Table (2) shows the avalanche effect on the plaintext when only one bit is changed in the key by using previous RC5 algorithm.

Table (1) shows that the average of avalanche effect of proposed 256 bits RC5 algorithm is 142.909 while table (2) shows that the average of the avalanche effect of previous 64 bits RC5 algorithm is 31.372.

Tables (1) and (2) are also show that the changing are 24 to 37 bits, (where the 24 represent the smallest avalanche effect and 37 represent the biggest avalanche effect) out of 64 bits and 124 to 157, (where the 124 represent the smallest avalanche effect and 157 represent the biggest avalanche effect) bits out of 256 bits when performing the algorithm before and after improved

respectively which mean that 37.5 to 57.82 of each block of the ciphertext is changed,

Where

24/ 64 *100 = 37.5

and 37/64*100= 57.82

After performing the improved RC5 algorithm the changing of each block of the ciphertext is 48.437 to 61.328,

Where

124/256* 100 = 48.437

and 157/256 *100= 61.328.

6. Time requirement:

In this section the time requirements are computed for the proposed 256 bits RC5 algorithm and the previous 64 bits RC5. Table (3) show this test.

7. The Complexity of Proposed Algorithm

To estimate the complexity of a cryptanalytic attack, one must consider at least the time it takes, the amount of data that is needed, and the storage requirements. For an *n*-bit block cipher the following complexities should be considered.

Data complexity: the amount of data needed as input to an attack. Units are measured in blocks of length*n*.

Processing complexity: the time needed to perform an attack. Time units are measured as the number of encryptions an attacker has to do himself.

Storage complexity: the words of memory needed to do the attack. Units are measured in blocks of length *n*.

The complexity of an attack is often taken as the maximum of the three complexities above; however, in most scenarios the amount of data encrypted with the same secret key is often limited and for most attackers the available storage is small. This demonstrates two important ideas [9].

- The running time of an algorithm is measured in terms of the number of 'basic operations' performed.
- The running time of an algorithm will usually depend on the size of the input.

The proposed algorithm increased the complexity by increasing the block size from 64 bits to 256 bits. The *E*-function uses the combinations of basic operations to achieve a large number of encryption functions, i.e. permutations of binary n-bit vectors, addition, multiplication and rotation then will be produced a high structural complexity.

8. Discussions

The proposed algorithm is a secure, compact and simple block cipher. It offers good performance a considerable exibility.

During the design process, several things can be concluded about cipher design:

1. The design has to remain simple.
2. Re-use the key schedule that was used in RC5, this already had an excellent track as a rock-solid key schedule and had been studied widely.
3. The E-function is used to approximate a pseudo-random function, in the function the three lines will be made of as "independent of each other" as possible. Thus very little interaction between the data in the three lines will be used. This also helps to avoid unwanted cancellations and makes it harder

to obtain a linear approximation of one line in terms of another

4. Still trying to guarantee some measure of "independence" between the data lines, the value of one line will make sure that never completely determines the value of another line. Indeed, the relative entropy of any two lines is at least 9 bits (of lines L, R), and gets as high as 64 bits (of lines R, M).
5. It is based on simple theory principles and simple arithmetic operations and easy to implement. So, it is completely specified, easy to understand the operation steps of the algorithm.

9. Conclusions

1. The block size of 64bits makes previous RC5 algorithm vulnerable to the matching ciphertext attack. Where after encryption of 2^{32} blocks, equal ciphertexts can be expected and information is leacked about plaintext. So that, the proposed algorithm with 256 bits block size is resistant to matching ciphertext attacks. It is required to 2^{128} ciphertext.
2. As the block size is 64 bits, a dictionary attack will require 2^{64} different plaintexts to allow the attacker to encrypt or decrypt arbitrary message under an unknown key. This attack applies to any deterministic block cipher with 128-bit blocks regardless of its design. So that, the proposed algorithm with 256 bits block size is required to 2^{256} different plaintexts.
3. From the results that were obtained in section 4 and after measuring the strength and the complexity of the proposed algorithm. The concluded is that proposed algorithm increases the security and the complexity compared with previous 64 bits RC5 algorithm.

References

[1] Rolf Oppliger "*Contemporary Cryptography*" ,Artech House, INC, 2005.

[2] Burwick, D. Coppersmith E. , D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford and N. Zunic, "*Mars a candidate cipher for AES*', *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[3] Dieter Schmidt, "ABC - A Block Cipher", Wikipedia the free Encycleopedia, May 27, 2002. http://en.wikipedia.org/wiki/ABC_(block_cipher)

[4] Bruce Schneier and John Kelsey," *Unbalanced Feistel Networks and Block Cipher Design", Counterpane* Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419, 2005, *http://f*schneier,kelseyg@counterpane.com

[5] A. Bogdanov1, L.R. Knudsen, G. Leander1, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe," *PRESENT: An Ultra-Lightweight Block Cipher"*, 2007, www.ist-ubisecsens.org

[6] Man Young Rhee "*Internet Security Cryptographic, principles, algorithms and protocols*",John Wiley & Sons Ltd, Englang,2003.

[7] Ross Anderson, Eli Biham, and Lars Knudsen, "Serpent*: A Proposal for the Advanced Encryption Standard*", *An Internet Survey,* 2000. http://citeseer.ist.psu.edu

[8] Shakir M. "*A new feedback symmetric block cipher method*", *Ph. D, Thesis University of Technology, Baghdad,* 1997.

[9] John Talbot & Dominic Welsh "*Complexity and Cryptography An Introduction*", Cambridge University Press, 2006.

Table (1) Avalanche Effect of 256 bits RC5 Algorithm: Change one bit in key

Block No.	Ciphertext 256 bits in Hexadecimal	Avalanche
1	1800645e835a7b76 4172ab2ed98fcb83 73070ec9fe64df85 b9fd0546d53f434a C5df66bb9a572a16 8e96fd000850d7c0 df810151ad4a1da2 5b96af7f1146a3bd	**143**
2	85b9f44b2304916a 19944d7fed0033d6 d07000cc46e44e0f bed9e36451f0834 80b820e5d8ebbd1 bbefff407a17ffa bb568363bbacbf40 eda67b64e5d5c235	**149**
3	3bb9971b5a304ad1 5652e47b9eed42e5 858cb2a064e66d0950070037d848b796 976bbf4914fc1ce9 60edf8561516f072 336fbc42a8d30cb5 23da49fd1dfebe8d	**153**
4	7ca22907e71a4b96 3c3194eb9ffca8ac 68629c037d3a31f7 e45fe240e8027b48 1e875448c6333fd4 f28b121c8e8d1934 a4df953ca6b1064b 96a0cb57c5f4fc93	**157**
5	F5c9366ac0da7ebd b8ea93750d486612 b6e7054fc6f54334 5b636264cf97549a 3503d33c2cc1ae81 f724cdc1baef5ba8 cb7372fee32955a6 ad8f957fb4a243d2	**128**
6	D5ba23c7a3ef03d7 16ef51fc56a630ab ced9fdc5b8078607 370532ae64b22774 4111d7dab612c240 39f0f2337d055c12 2ce61f1643be6880 aa6f73dd8691fec8	**144**
7	fca6872422c5fb62 f17c639e013fd5fb e3127db66936a7d3 dcc6dfbf9d668fc7 df80ec6e3c82abf3 b1fd1afd2e9d2f41 55db1d097e9f5fc3 d14f3e3fff7ccdb1	**142**
8	6b20fe3650e3fd4 617d9e1585d18b47 22ed4e48db893390 d0b66fe7cfed48ce b7e77d6104b264ba 1f391acf3ee8504f 152fcbfb68f941c6 e2cd5cd263b678ae	**146**
9	3695ba4de33c49c7 fcf307576fd68b0a d447863da8f2000f 4d19bc680af798f5 c0159747b7ae0c49 2e29fcf010f194fa 88cb11b166a89a0e f4677b716c8d5e0e	**124**
10	E704f560cade6fb 4319a31941f31b3 6ee3bea38d118621 697ed3a6a0c9af9 b5b65ee23031e5e3 a9932d51da976ebf 4e555168dd410110 b1258eed248c4103	**141**
11	cdca8afee63f7a32 a4a52e45f5f00cb2 ad05709cb01e0a62 29cdae4de794265e 4c686f300b4fc783 1a51fbe5e4bd3347 1cbb6e84372abf06 9b04d34c50a272b0	**139**
Key1	**ffffffffffffffffffffffffffffffff**	
Key2	**fffffffffffffffffffffffffffffffe**	

Table (2) Avalanche Effect RC5 Algorithm: Change one bit in key

Block No.	Avalanche	Block No.	Avalanche	Block No.	Avalanche	Block No.	Avalanche
1	34	12	35	23	32	34	26
2	27	13	32	24	30	35	28
3	33	14	31	25	32	36	36
4	24	15	32	26	35	37	26
5	32	16	31	27	33	38	32
6	32	17	35	28	33	39	29
7	35	18	31	29	28	40	35
8	29	19	29	30	35	41	29
9	27	20	33	31	35	42	30
10	29	21	29	32	28	43	37
11	34	22	32	33	34		
Key1	ffffffffffffffffffffffffffffffff						
Key2	fffffffffffffffffffffffffffffffe						

Table (3) Time Comparisons of Proposed algorithm and previous RC5 algorithm on a Pentium I4

Algorithm	Number of Bytes	Time in Second
Proposed 256 bits RC5	32000	0.031
	320000	0.328
	3200000	3.093
Previous 64 bits RC5	32000	0.046
	320000	0.5
	3200000	4.953

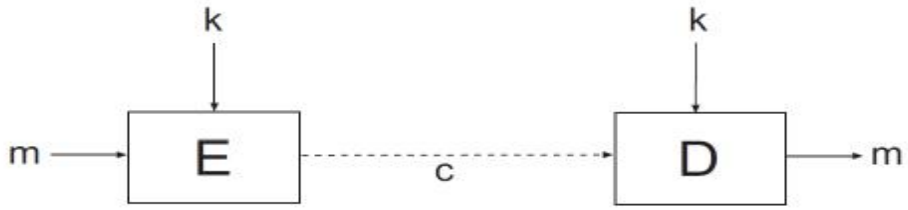

Figure (1) The working principle of a symmetric encryption [1]

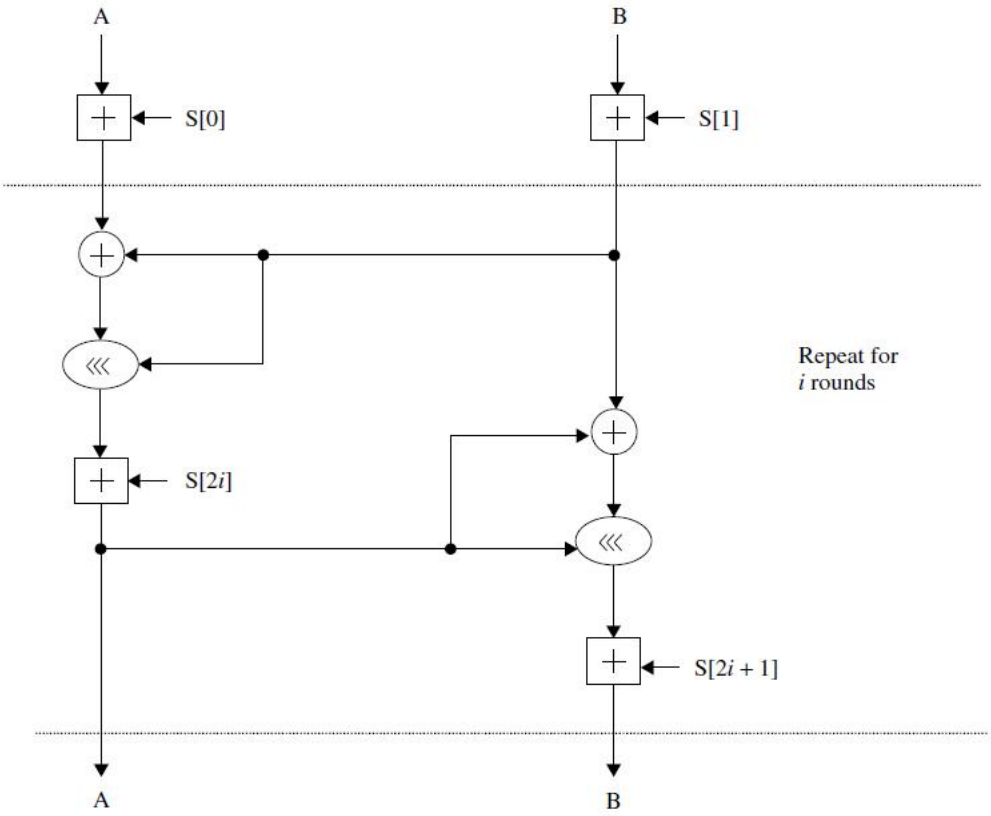

Figure (2) RC5 Encryption Algorithm [6].

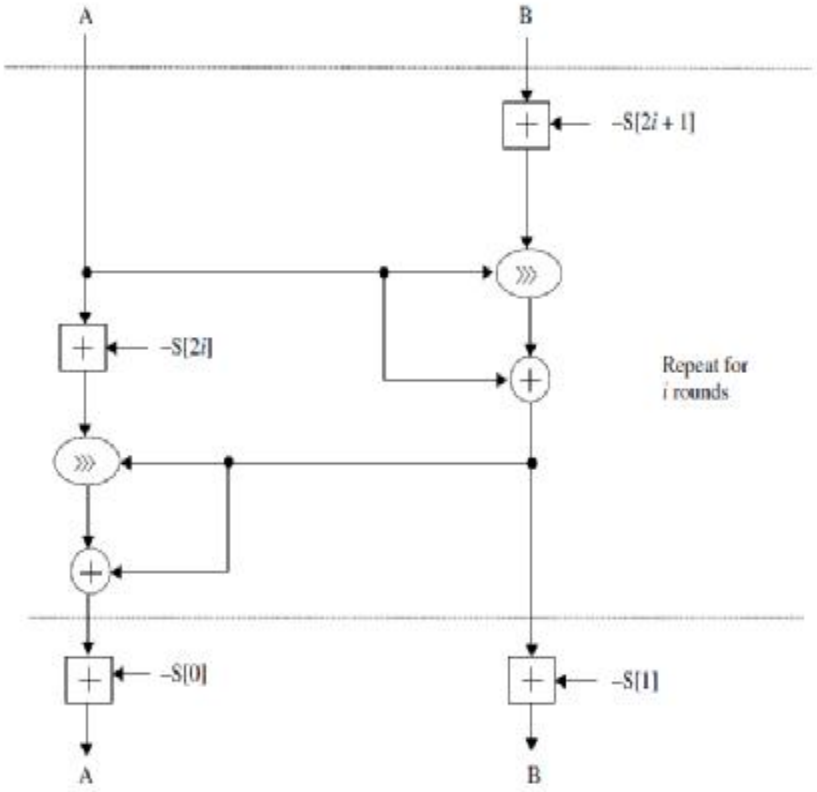

Figure (3) RC5 Decryption Algorithm [6].

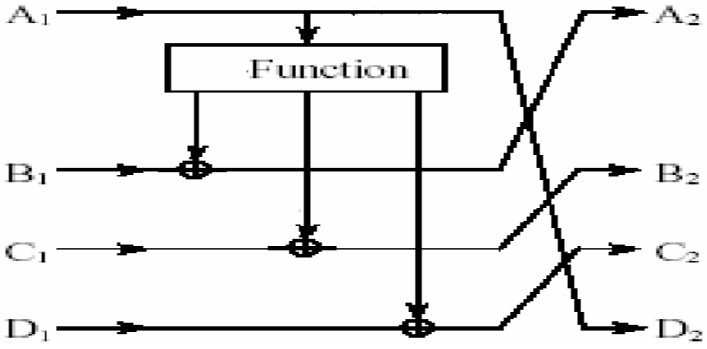

Figure (4) Type-3 Feistel network [2].

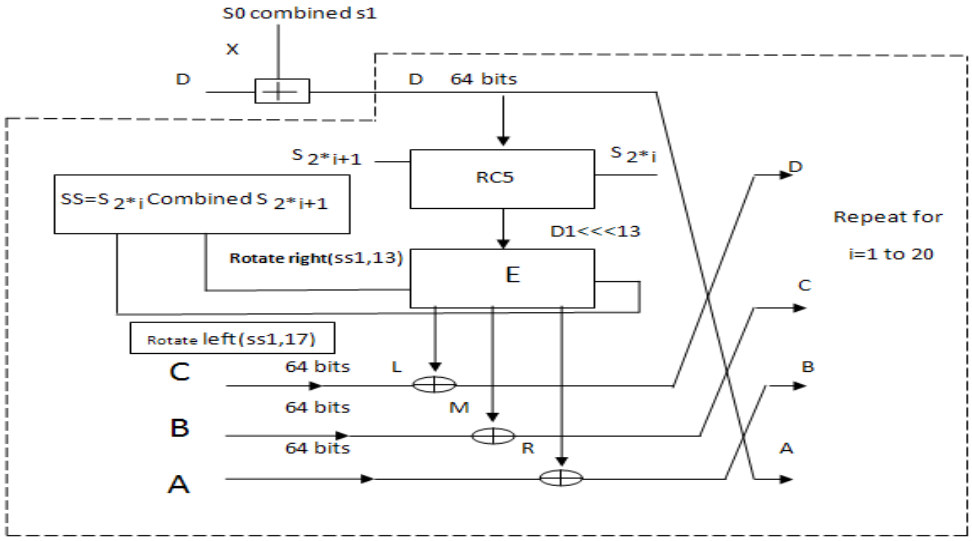

Figure (5) T he Structure of the Proposed Algorithm

Encryption:
Input: Plaintext = (A, B, C, D) // *each of which is 64 bits*
Subkeys S[0…43] of 32 bits
Output: Ciphertext= (A, B, C, D) // *each of which is 64 bits*
Procedure:
X=Combined S[0] and S[1]
D= D+ X
For i = 1 to 20 do
begin
D1=Encryption_RC5 (D, S[2*i],S[2*i+1])
D1=D1<<<13
SS1=combined S[2*i],S[2*i+1]
SS1=Rotate-right(SS1,13)
SS2=Rotate-left(SS1,17)
E (D1, SS1,SS2 L, M, R); // *output (L; M; R)*
C=C^L
B=B^M
A=A^R
(A, B, C, D)= (B, C, D, A)

Figure (6) The Pseudo-code of Encryption for Proposed Algorithm

Decryption:
Input: Plaintext = (A, B, C, D) // *each of which is 64 bits*
Subkeys S[0…43] of 32 bits
Output: Ciphertext= (A, B, C, D) // *each of which is 64 bits*

Procedure:

```
For i = 1 to 20 do
begin
 (A, B, C, D) = (D, A, B, C)
 SS1=combined S[2*i],S[2*i+1]
 SS1=Rotate-right(SS1,13)
 SS2=Rotate-left(SS1,17)
  E(A,SS1,SS2, L, M,R);

A=Decryption_RC5(A,S[2*i],S[2*i+1]);

  B=B^L
  C=C^M
  D=D^R
end
```

Figure (7) The Pseudo-code of Decryption for Proposed Algorithm

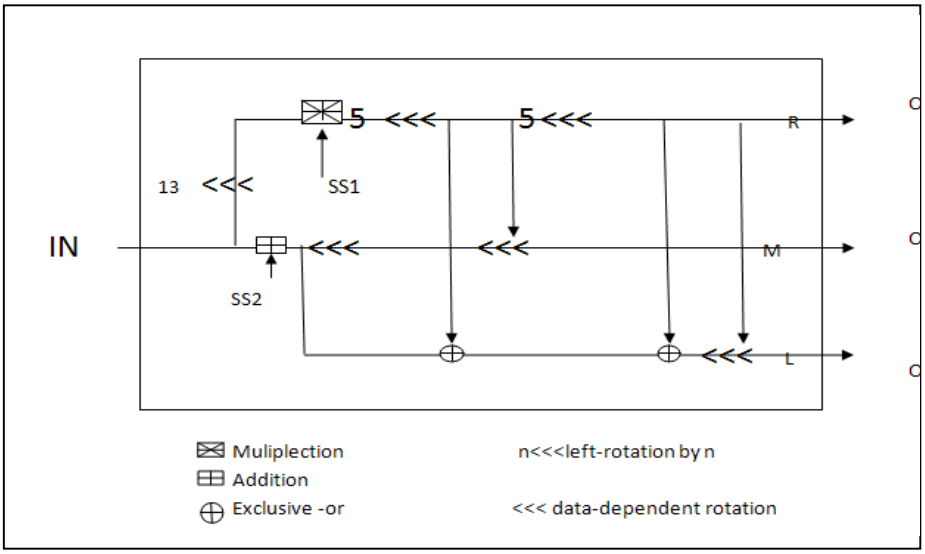

Figure (8) the E-function