Performance Analysis of RC5, Blowfish and DES Block Cipher Algorithms

Harsh Kumar Verma
Department of Computer Science & Engineering
National Institute of Technology, Jalandhar
Punjab (India)

Ravindra Kumar Singh
Department of Computer Science & Engineering
National Institute of Technology, Jalandhar
Punjab (India)

ABSTRACT
In this paper, Performance analysis of RC5, Blowfish and DES block cipher algorithms have been done on the basis of execution time and resource utilization. CPU utilization and memory utilization both are considered for determining resource utilization. These three algorithms are parameterized algorithm and encrypt two w-bits at a time. Allowable choices for w are 16 bits, 32 bits, and 64 bits. Blowfish and DES have same structure for encryption and decryption while RC5 have different. RC5 has 12 and Blowfish & DES have 16 rounds. These three algorithms have a variable block size and a variable key size in their structure. Performances of RC5 & Blowfish algorithms have been evaluated on key size of 128-bits, 192-bit and 256-bit while key size is fixed 64-bit for DES in this paper.

General Terms
Cryptography, Block cipher, Symmetric encryption, RC5, Blowfish, DES

Keywords
Cryptography, Block cipher, Symmetric encryption, RC5, Blowfish, DES

1. INTRODUCTION
In cryptography, the use of the symmetric key encryption is common to ensure data confidentiality, it uses same key for both encryption of plain text and decryption of cipher text. As illustrated in fig 1.

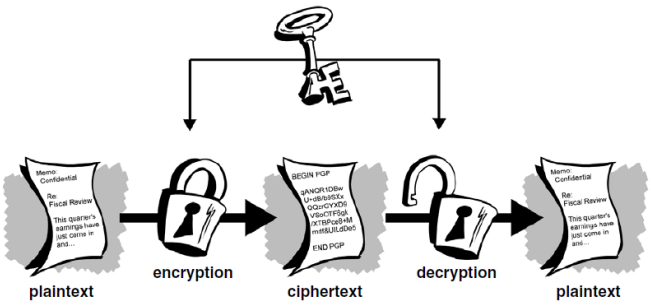


Fig 1 : Symetric encryption

Symmetric key encryption code can be divided into the block cipher and stream one [1]. RC5 [2], Blowfish [3] and DES [4] are symmetric key block ciphers, and explained in further sections.

1.1 RC5
RC5 is a block cipher notable for its simplicity. Designed by Ronald Rivest in1994, *RC* stands for "Rivest Cipher", or alternatively, "Ron's Code" [5, 6]. It is suitable for hardware and software implementation, because it uses only those operations which are available in typical microprocessor [2].

1.2 Blowfish
Blowfish is a variable-length key symmetric block cipher, designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products. Schneier designed Blowfish as a general-purpose algorithm, intended as an alternative to the aging DES [7]. It is significantly faster than DES when implemented on 32-bit microprocessors with large data caches, such as the Pentium and the PowerPC. It is only suitable for applications where the key does not change often, like a communications link or an automatic file encryptor. Notable features of the design Blowfish include key-dependent S-boxes and a highly complex key schedule [3]. Implementations of Blowfish that require the fastest speeds should unroll the loop and ensure that all subkeys are stored in cache; it requires total 521 iterations to generate all required subkeys.

1.3 DES
The Data Encryption Standard (DES) was developed in the 1970s by the National Bureau of Standards with the help of the National Security Agency[8]. Its purpose is to provide a standard method for protecting sensitive commercial and unclassified data. IBM created the first draft of the algorithm, calling it LUCIFER. DES officially became a federal standard in November of 1976 [2]. It is based on a symmetric-key algorithm that uses a 56-bit key for encrypting 64-bits plaintext. DES uses Feistel network and it has 16 rounds in its structure. The notable feature of DES is using S-Box, a table-driven non-linear substitution operation in which input size and output size both can vary either randomly or algorithmically for increasing diffusion.

1.4 Basic Terms
1.4.1 Feistel Networks
A Feistel network is a general method of transforming any function (usually called the F function) into a permutation. It was invented by Horst Feistel [9] in his design of Lucifer and popularized by DES. The fundamental building block of Feistel networks is the F-function: a key-dependent mapping of an input string onto an output string. The alternative substitution and permutation operations of Feistel network is invented from Product Cipher brought up by Claude Shannon [10] in 1949.

1.4.2 Diffusion
"Diffusion" means that any change of bits in a plaintext will affect many bits in cipher text to enhance complexity between the plaintext and the cipher text. In a block encryption / decryption system, diffusion can be achieved by repeatedly implementing a specific permutation and then execute a functional operation.

1.4.3 Confusion

"Confusion" can be achieved by manipulating the relations between cipher text and sub key to be more complicated, leaving no chance of existence of direct linear relationship.

1.4.4 S-boxes

An S-box is a table-driven non-linear substitution operation used in most block ciphers. S-boxes vary in both input size and output size and can be created either randomly or algorithmically. S-boxes were first used in Lucifer, then DES and afterwards in most encryption algorithms.

2. RC5

The RC5 encryption algorithm is a block cipher that converts plain text data blocks of 16, 32, and 64 bits into cipher text blocks of the same length. It uses a key of selectable length b (0, 1, 2, ..., 255) byte. The algorithm is organized as a set of iterations called rounds r that takes values in the range (0, 1, 2, ..., 255) as illustrated in fig. 2.

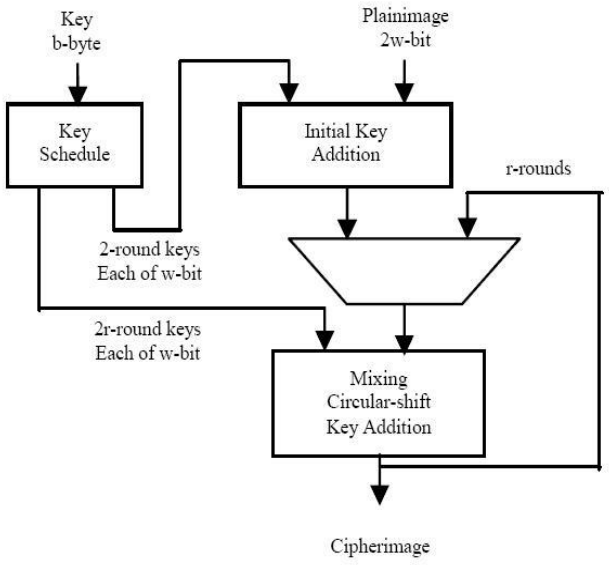


Fig 2: RC5 Encryption Algorithm

The operations performed on the data blocks include bitwise exclusive-OR of words, data-dependent rotations by means of circular left and right rotations and Two's complement addition/subtraction of words, which is modulo-2^w addition/subtraction. RC5 is a fully parameterized family of encryption algorithms, it is more accurately specified as RC5-w/r/b where the word size is w bits, encryption consists of a nonnegative number of rounds r and b denotes the length of the encryption key in bytes. The original suggested choice of parameters were w = 32 bits, r = 12 and b = 16-byte. For all variants, RC5-w/r/b operates on units of two w-bit words using the following basic operations.

The operations used in RC5 are defined as followings.

1. A+B integer addition modulo 2^w
2. A-B integer subtraction modulo 2^w
3. A⊕B bitwise exclusive-or of w-bit words
4. $A^{<<<}B$ rotation of the w-bit word A to the left by the amount given by the least significant lg w bits of B
5. $A^{>>>}B$ rotation of the w-bit word A to the right by the amount given by the least significant lg w bits of B

There are three routines in RC5: key expansion, encryption, and decryption [11]. We discus each of them in next sections, Key-Expansion algorithm is used to generate the round sub keys that will be use in both encryption and decryption algorithms. RC5 has different algorithms for encryption and decryption, in encryption it uses integer addition modulo 2^w but in decryption it uses integer subtraction modulo 2^w. RC5 is a symmetric key encryption so encryption and decryption algorithms uses same key.

2.1 Key-Expansion Algorithm

The user supplies a key of b bytes, copy the secret key K[0...b-1] into an array L[0..c-1] of c = ceil(b/u), where u = w/8 in little-endian order. In other words, we fill up L using u consecutive key bytes of K. Any unfilled byte positions in L are zeroed. In the case that b = c = 0, set c = 1 and L[0] = 0. The number of w-bit words that will be generated for additive round keys is 2(r + 1) and these are stored in the array S[0, ..., 2r + 1].

Magic Constants P_w and Q_w are defined for arbitrary w as follows:

P_w = Odd $((e -1) 2^w)$ … (1)

Q_w = Odd $((v - 1) 2^w)$ … (2)

Where

e is the base of natural logarithms (e = 2.718281828459) and

v is the golden ratio (v =1.618033988749)

Odd (x) is the odd integer nearest to x [2].

Table 1 [2] show these magic constants in hexadecimal using several values of w. Which are calculated by above expressions (1) & (2).

Table 1. Magic Constants values P_w and Q_w

W	**16**	**32**	**64**
P_w	B7E1	B7E15163	B7E151628AED2A6B
Q_w	9E37	9E3779B9	9E3779B97F4A7C15

Key-Expansion with RC5-w/r/b

Input: b byte key that is preloaded into c word array L[0,1,…, c-1], r denotes the no of rounds.

Output: 2r+2 w-bit round keys S[0, 1, …, 2r, 2r+1].

Procedure:

S[0] = P_w,
For i= 1 to 2r+1 do
{
S[i] = S[i – 1] + Q_w
}
X = Y = a = b = 0
Iteration =3 * max(c, 2r+2)
For i = 1 to Iteration do
{
X = S[a] = (S[a] + X + Y) <<< 3
Y =L[b] = (L[b] +X + Y) <<< (X + Y)
i = (a + 1) mod (2r + 2)
j = (b + 1) mod c
}

2.2 Encryption Algorithm

Fig 3 illustrates the encryption procedure of RC5; decryption procedure is just reverse of this structure by converting addition operation to subtraction operations.

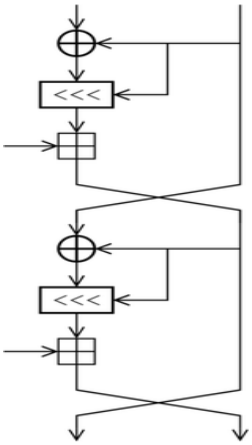


Fig 3: RC5 Block Cipher

RC5 works with two w-bit registers A and B which contain the initial input plain text as well as the output cipher text at the end of encryption. The first byte of plain text or cipher text is placed in the least-significant byte of A, the last byte of plain text or cipher text is placed into the most-significant byte of B. Pseudo code of encryption is given below; at first we load plain text in to registers A and B then apply these operations to encrypt the plain text [5].

Encryption with RC5-w/r/b

Input: Plain text stored in two w-bit input registers A and B. r denotes the no of rounds and 2r+2 w-bit round keys S[0, 1, ..., 2r + 1]

Output: Cipher text will be store in A and B.

Procedure:

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]
}

After applying these operations on registers A and B plain text get converted into the cipher text and we store it in any file that is called encrypted file.

Decryption Algorithm

Pseudo code of decryption is given below; for decryption of cipher text load cipher text into registers A and B then apply these operations to convert cipher text into plain text.

Decryption with RC5-w/r/b

Input: Cipher text stored in two w-bit input registers A and B. r denotes the no of rounds and 2r+2 w-bit round keys S[0, 1, ..., 2r + 1]

Output: Plain text will be store in A and B.

Procedure:

for i = r down to 1 do
{
B = ((B - S[2i + 1]) >>>A) ⊕ A
A = ((A - S[2i]) >>> B) ⊕ B
}
B = B - S[1]
A = A - S[0]

This algorithm uses integer subtraction modulo 2^w and right rotation on registers for getting plain text; it does reverse operations on registers.

3. BLOWFISH

Fig 4 shows the action of Blowfish. Each line represents 32 bits. The algorithm keeps two subkey arrays: the 18-entry P-array and four 256-entry S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries.

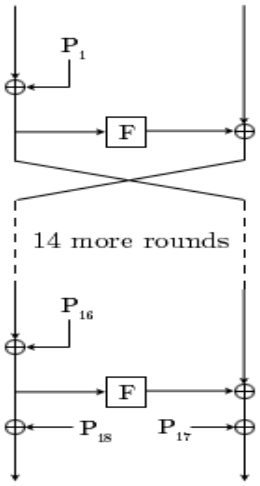


Fig 4 : Blowfish Encryption

Fig 5 shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output.

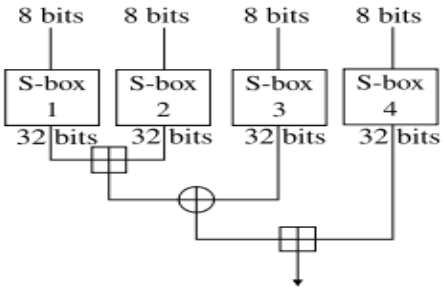


Fig 5 : F-function of Blowfish

Blowfish consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes.

Data encryption occurs via a 16-round Feistel network. Each round consists of a key-dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

3.1 Subkeys Generation:

Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption [7].

3.1.1 SubKeys

1. The P-array consists of 18 32-bit subkeys: P_1, P_2,..., P_{18}.
2. There are four 32-bit S-boxes with 256 entries each:
   $S_{1,0}$, $S_{1,1}$,..., $S_{1,255}$;
   $S_{2,0}$, $S_{2,1}$,..., $S_{2,255}$;
   $S_{3,0}$, $S_{3,1}$,..., $S_{3,255}$;
   $S_{4,0}$, $S_{4,1}$,..., $S_{4,255}$.

3.1.2 Generating the Subkeys:

The subkeys are calculated using the Blowfish algorithm. The exact method is as follows[7]:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3). For example:
   P_1 = 0x243f6a88
   P_2 = 0x85a308d3
   P_3 = 0x13198a2e
   P_4 = 0x03707344
2. XOR P_1 with the first 32 bits of the key, XOR P_2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P_{14}). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)
3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (a) and (b).
4. Replace P_1 and P_2 with the output of step (c).
5. Encrypt the output of step (c) using the Blowfish algorithm with the modified subkeys.
6. Replace P_3 and P_4 with the output of step (e).
7. Continue the process, replacing all entries of the P-array, and then all four S-boxes in order, with the output of the continuously-changing Blowfish algorithm.

3.2 Encryption:

Blowfish is a Feistel network consisting of 16 rounds [7]. The input is a 64-bit data element, X.

Divide X into two 32-bit halves: X_L, X_R. Then, the following operations are performed form r=1 to 16.

$X_L = X_L \oplus P_i$

$X_R = F(X_L) \oplus X_R$

Swap X_L and X_R

After 16 rounds Swap X_L and X_R (Undo the last swap.) and then X_R and X_L are XORed with P_{17} and P_{18}.

$X_R = X_R \oplus P_{17}$

$X_L = X_L \oplus P_{18}$

Lastly recombine X_L and X_R.

Function F (see Figure 3): Divide X_L into four eight-bit quarters: a, b, c, and d.

$F(X_L) = ((S_1,a + S_2,b \bmod 2^{32}) \oplus S_3,c) + S_4,d \bmod 2^{32}$

3.3 Decryption

Decryption is exactly the same as encryption, except that P_1, P_2,..., P_{18} are used in the reverse order [7]. This is not so obvious because XOR is commutative and associative. A common mistake is to use inverse order of encryption as decryption algorithm (i.e. first XORing P_{17} and P_{18} to the ciphertext block, then using the P-entries in reverse order).

4. DES (Data Encryption Standard)

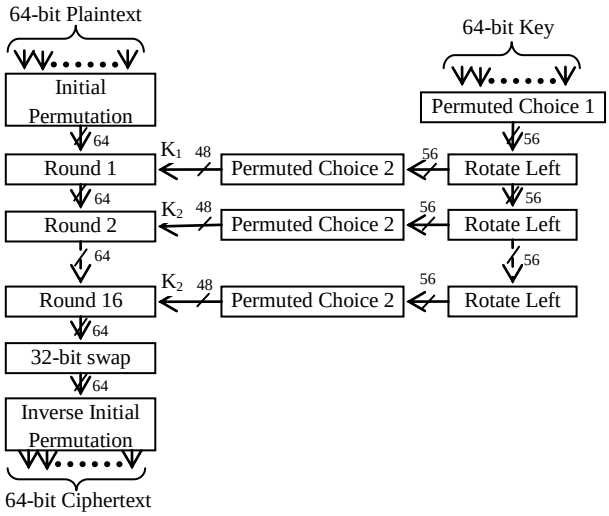


Fig 6: DES Encryption Algorithm

DES encryption process is illustrated in fig 6; there are mainly three algorithms in DES Key Generation algorithm, Encryption algorithm and Decryption algorithm. Each of them is described in further sections.

4.1 Key Generation:

64-bit key is used as input to the key schedule algorithm of DES which produces 16 round keys as shown in fig 2. There are three steps in key schedule algorithm [1].

4.1.1 Permuted Choice One (PC-1): 8×8 matrix is used to store 64-bits key, PC-1 ignores 8th column and read bits column by column from bottom to top and rearrange it in 8×7 matrix row by row from left to right. At first processing starts from column 1st towards 7th but after rearranging 4 rows in new matrix(till 4th values from bottom in column 4th), processing gets start from column 7th towards 4th, at 4th columns 4 values from bottom are already fetched so 5th to 8th values are used to fill new matrix. Thus PC-1 produces 56-bits output.

4.1.2 Rotate Left: The resulting 56-bits output of PC-1 is then treated as two 28-bits quantities, labeled C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to rotate left of 1 or 2 bits according to Table2 [2].

Table 2: Schedule of Left Rotation

Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Rotated Bits	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

4.1.3 Permuted Choice Two (PC-2): After rotate left both 28-bits quantities merge and served as input to PC-2, which produces a 48-bit round key that are used in F function.

4.2 Encryption:

The process of encryption consists of four stages [1].

4.2.1 Initial Permutation (IP): 8×8 matrix is used to store 64-bits Plaintext. Reading even columns first then odd columns (sequence of columns 2, 4, 6, 8, 1, 3, 5& 7) from bottom to top and rearranging it row by row (sequence of rows 1, 2, 3, 4, 5, 6, 7 & 8) from left to right.

4.2.2 Round: there are 16 iterations for substitution in which 64-bits data are divided in two halves (L_i & R_i) of 32-bits. 32-bits of R_i is used for function F and L_i remain unchanged, Fig 7 illustrate the working of one round. Three functions play major role in this stage.

4.2.2.1 Expansion Permutation (E): This function is used to expand 32-bits of data to 48-bits of data by using Left rotation and right rotation. 8×6 matrix is used to store 48-bits; 32-bits are stored in columns 2nd, 3rd, 4th & 5th. Then rotate one bit left to column 2nd and store it in column 6th and rotate one bit right to column 5th and store it in column 1st.

4.2.2.2 S-Box: A set of eight 4×16 S-Boxes are used to convert 48-bits data to 32-bits data. 48-bits divided in 8 parts of 6-bits each. Every S-Box accepts that 6-bits data (2 MSB bits are used to select rows and 4 LSB bits are used to select columns) to select a hexadecimal number (4-bits output). Thus the result is 32-bits data from eight S-Boxes.

4.2.2.3 *Permutation (P)*: 32-Bits data is permutated in this function for getting higher diffusion.

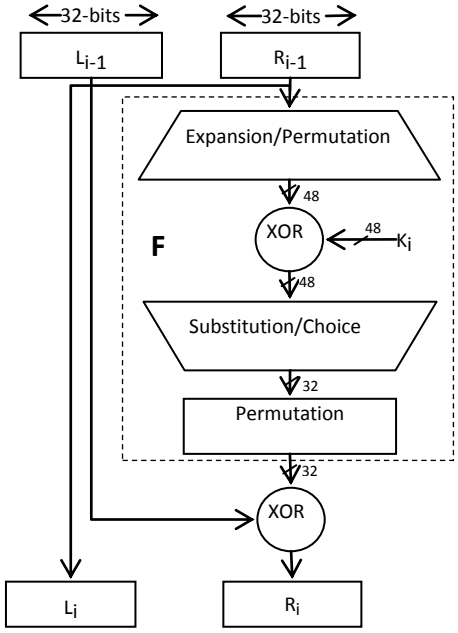


Fig 7: Round Function of DES

The 32-bits left (Li) and right (Ri) parts of every intermediate value for ith iteration swaps their place:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

where K_i is round key.

4.2.3 32-bit Swap: left and right halves of output of last (16^{th}) iteration swaps their places.

$L_{16} = R_{16}$

$R_{16} = L_{16}$

4.2.4 Inverse Initial Permutation (IP^{-1}): it is reverse to initial permutation, that is $M=IP^{-1}(IP(M))$. In this permutation read matrix row by row (sequence of rows 4, 1, 5, 2, 6, 3, 8& 4) form right to left and rearrange it column by column (sequence of columns 1, 2, 3, 4, 5, 6, 7 & 8) from top to bottom.

4.3 Decryption:

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the round keys is reversed [1].

5. COMPARISON AND ANALYSIS

A comparative analysis of RC5, Blowfish & DES is performed to provide some measurements on the encryption and decryption. Effects of several parameters such as number of rounds, block size and the length of secret key on the performance evaluation criteria are investigated.

These three encryption algorithms were implemented in c# in visual studio 2009. Performance was measured on a 3GHz Pentium®4 with 1GB of RAM running Windows XP professional Version 2002, Service pack 3.

5.1 Parametric Comparison

Table 3 summarizes the comparison of RC5, Blowfish & DES for different design parameters such as word size, block size, number of rounds and secret key size.

Table 3: Comparison on the basis of parameters

Parameters	Algorithm Type		
	RC5	**Blowfish**	**DES**
b (key length in bytes)	0 - 255	16, 24 or 32	8
r (no of rounds)	0 – 255 (standard 12)	16	16
No of round keys	2r+2	r+2	r
Block size in words	2w	2w	2w
w (word size in bits)	16, 32, 64 (standard 32)	16, 32, 64 (standard 32)	16, 32, 64 (standard 32)
Block size in bits	32, 64, 128 (standard 64)	32, 64, 128 (standard 64)	32, 64, 128 (standard 64)
Used Function	Does not exist	S-Box	IP, IP^{-1}, E, P, S-Box, PC-1, PC-2
Used Operation	+, -, ⊕, <<<, >>>	+, ⊕, <<<, >>>	⊕, <<<, >>>

5.2 Performance Comparison

In addition, to improve the accuracy of our timing measurements, program was executed 10 times for each input file and we report the average of the times thereby obtained. In this observation key size is 8-bytes for DES while Blowfish & RC5 have three values 16-bytes, 24-bytes and 32-bytes. Number of round(r) was fixed 12 for RC5 and 16 for Blowfish & DES.

5.2.1 On the basis of Execution Time

We compare the execution time of each algorithm on different-2 file types like text file, audio file & video files, for this purpose we mainly used 6 files and recorded their execution (encryption or decryption) times in milliseconds for these three algorithms. List of Input files and their size are given in Table 4.

Fig 8 illustrates the execution time according to their file size for each file using algorithms RC5, Blowfish and DES. Graph shows that RC5 performs faster than Blowfish & DES for every key size. It can also be concluded that increasing key size decreases the performance of Blowfish.

Table 4: Comparison on the basis of execution time

File Name (file type)	File Size (in KB)	RC5-128	RC5-192	RC5-256	Blowfish-128	Blowfish-192	Blowfish-256	DES-64
A.doc	712	125	109.375	109.375	200.8125	205	209.75	364.25
B.pdf	649	109.375	93.75	109.375	178.5	183	187.5	340.5
C.jpg	656	78.125	93.75	93.75	196.45	205.125	209.8125	376.75
D.gif	1396	156.25	156.25	156.25	258.925	263.5	267.5125	427.5
E.mp3	2068	234.375	234.375	234.375	320	337.5	351.25	532.625
F.avi	2800	312.5	312.5	328.125	357.5	370.25	375	574.25

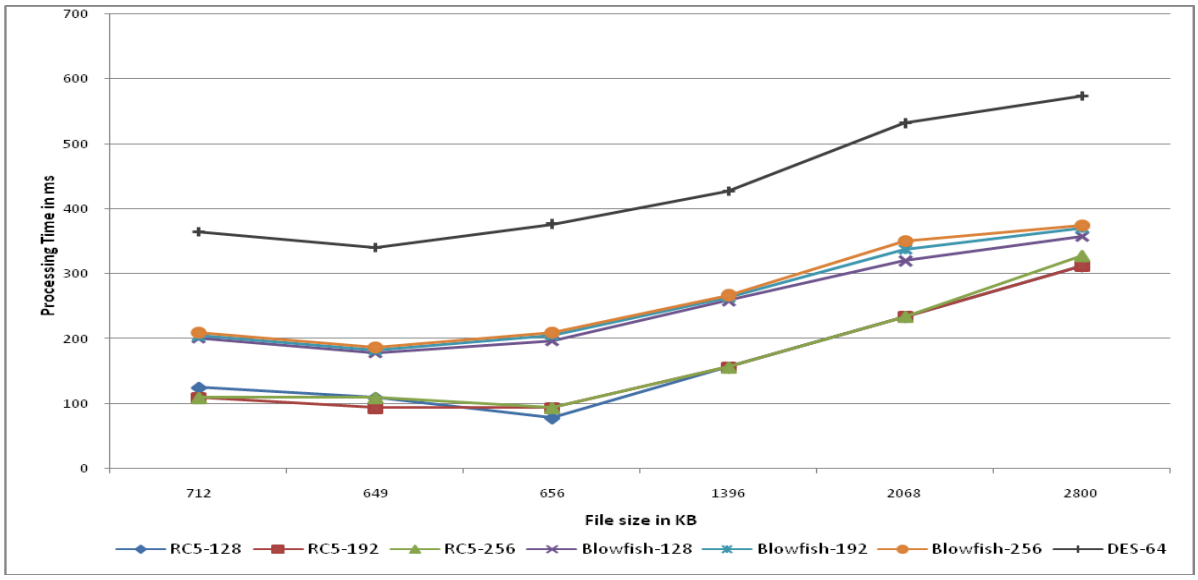


Fig 8: Execution time of RC6, Twofish & AES for 32-bytes key

5.2.2 On the basis of CPU Utilization & Memory Utilization

In this section a video file (.avi) of 2800KB was executed by these three algorithms RC5, Blowfish and DES. CPU utilization and Memory utilization for each algorithm was also captured. For the accuracy point of view we executed that file 5 times and then taken the average of them.

Fig 9 shows the CPU utilization and Memory utilization for RC5 block cipher algorithm. Blue line represents the CPU usage in percentage (0-100 %) and Red line represents the Memory usage in 10MB (40 means 400MB). Average CPU utilization is 51.15 % and average Memory utilization is 499.63 MB for RC5.

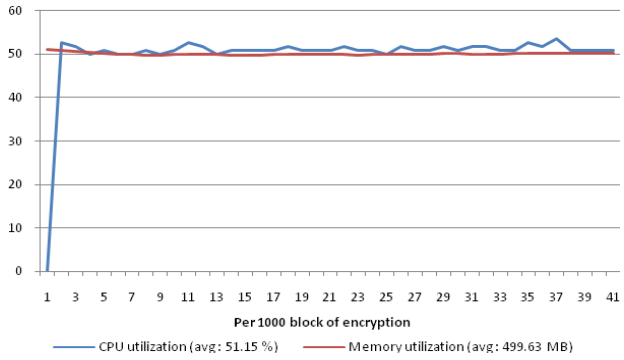


Fig 9: CPU Utilization & Memory Utilization of RC5

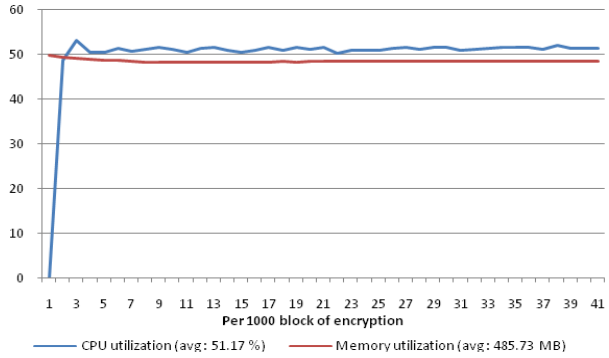


Fig 10: CPU Utilization & Memory Utilization of Blowfish

Fig 10 shows the CPU utilization and Memory utilization for Blowfish algorithm. Blue line represents the CPU usage in percentage (0-100 %) and Red line represents the Memory usage in 10MB (40 means 400MB). Average CPU utilization is 51.17 % and average Memory utilization is 485.73 MB for Blowfish.

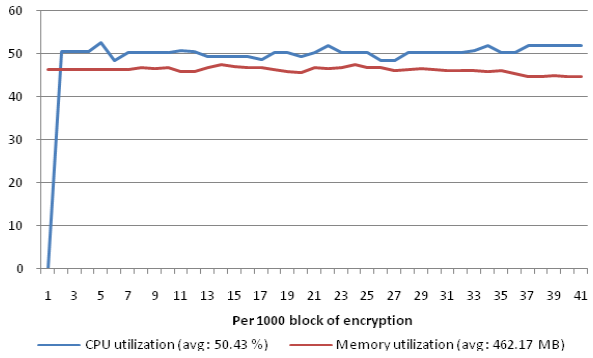
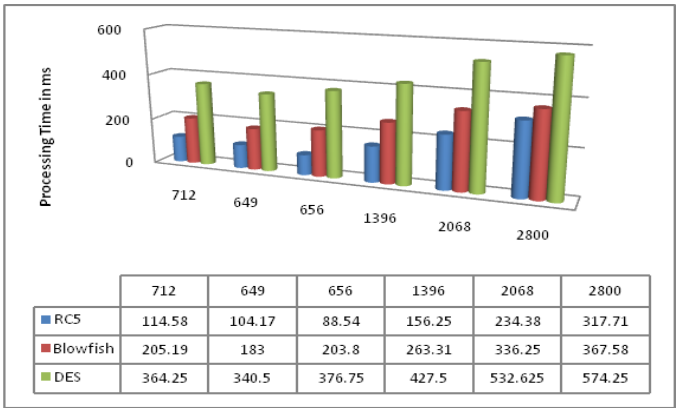


Fig 11: CPU Utilization & Memory Utilization of DES

Fig 11 shows the CPU utilization and Memory utilization for DES algorithm. Blue line represents the CPU usage in percentage (0-100 %) and Red line represents the Memory usage in 10MB (40 means 400MB). Average CPU utilization is 50.43 % and average Memory utilization is 462.17 MB for DES.

5.3 Result Analysis

RC5 performs faster than Blowfish & DES. Fig 12 shows the average execution time for these three algorithms to execute the files mentioned in table 4. According to fig 12 RC5 is 1.54 times faster than Blowfish and 2.57 times faster than DES. Result also concludes that performance of Blowfish algorithm is inversely proportional to keysize, if keysize will increase the performance will decrease and vice-versa.



	712	649	656	1396	2068	2800
RC5	114.58	104.17	88.54	156.25	234.38	317.71
Blowfish	205.19	183	203.8	263.31	336.25	367.58
DES	364.25	340.5	376.75	427.5	532.625	574.25

Fig 12: Average Execution Time in millisecond

In addition, if we consider on resource utilization then we got that RC5 utilize 13.9 MB extra memory compared to Blowfish and 37.46 MB extra memory compared to DES, while CPU utilization is approximately same for all these three algorithms.

6. CONCLUSION

In this research paper RC5, Blowfish and DES block cipher algorithms were compared by using C# program in visual studio 2009. Performance of these three algorithms were measured on a 3GHz Pentium®4 with 1GB of RAM running Windows XP professional Version 2002, Service pack 3. Comparative analysis of RC5, Blowfish and DES have been done with a set of input files and evaluated the encryption & decryption time. Results conclude that RC5 is 1.54 times faster than Blowfish and 2.57 times faster than DES. Result also concludes that performance of Blowfish algorithm is inversely proportional to keysize, if keysize will increase the performance will decrease and vice-versa.

In resource utilization point of view, RC5 utilize 13.9 MB extra memory compared to Blowfish and 37.46 MB extra memory compared to DES, while CPU utilization is approximately same for all these three algorithms. So RC5 block cipher algorithm is faster and simpler than Blowfish & DES block cipher algorithms. Using RC5 is beneficial where high encryption rate is required.

7. REFERENCES

[1] W. Stallings, "Cryptography and Network Security: Principles and Practice", Prentice-Hall, New Jersey, 1999.

[2] "RC5" "wikipedia.org**".** Available at: http://en.wikipedia.org/wiki/RC5

[3] "Blowfish", "wikipedia.org", [online] Available at: http://en.wikipedia.org/wiki/Blowfish_(cipher)

[4] "Data Encryption Standard", "wikipedia.org", [online] Available at: http:// http://en.wikipedia.org/wiki/Data_Encryption_Standard

[5] Ronald L. Rivest, "RC5 Encryption Algorithm", Dr Dobbs Journal, Vol. 226, PP. 146-148, Jan 1995.

[6] Ronald L. Rivest, The RC5 Encryption Algorithm, MIT Laboratory for Computer Science 545 Technology Square, Cambridge, Mass.02139 (Revised March 20, 1997). Available at: httu://theory.lcs.mit.cdu/-rivest/Rivest-rc5rev.pdf

[7] B. Schneier, **"**Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)", [online] Available at: http://www.schneier.com/paper-blowfish-fse.html

[8] NIST FIPS PUB 46-3. "Data Encryption Standard. Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., 1977.

[9] H. Feistel, W.A. Notz, and J.L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications", *Proceedings on the IEEE*, v. 63, n. 11, 1975, pp. 1545 -1554.

[10] C. Shannon, Communication theory of secrecy systems, Bell System Technical Journal, vol 28,pp 656-715, 1949.

[11] "What are RC5 and RC6","rsa.com". Available at: http://www.rsa.com/rsalabs/node.asp?id=2251