Performance Evaluation of RC6, Blowfish, DES, IDEA, CAST-128 Block Ciphers

Kirti Aggarwal
Department of Computer Science and Engineering
National Institute of Technology
Jalandhar, Punjab, India

Jaspal Kaur Saini
Department of Computer Science and Engineering
National Institute of Technology
Jalandhar, Punjab, India

Harsh K. Verma, PhD.
Department of Computer Science and Engineering
National Institute of Technology
Jalandhar, Punjab, India

ABSTRACT

Rapid growth of internet applications fueled the need for securing information and computers. Encryption algorithms play vital role to secure information. This paper provides comparison of most common encryption algorithms namely: DES, Blowfish, CAST-128, RC6, IDEA. Performance evaluation is carried out on the basis of execution time and throughput. These algorithms has different key and block size. DES, IDEA, Blowfish, CAST-128 has block size of 64 bits. DES has key Size of 64 bits while IDEA, Blowfish, CAST-128 has key size of 128 bits. RC6 has Key size and Block size of 128 bits. Simulation results are provided to demonstrate the effectiveness of each algorithm.

General Terms

Cryptography, Symmetric encryption, DES, Blowfish, CAST-128, RC6, IDEA

Keywords

Cryptography, Symmetric encryption, DES, Blowfish, CAST-128, RC6, IDEA

1. INTRODUCTION

The art and science of keeping messages secure is cryptography, and it is practiced by cryptographers[1]. Many encryption algorithms are available which can be categorized as symmetric key encipherment and asymmetric key encipherment. Symmetric key encipherment involves one key which is used for both encryption and decryption. Asymmetric key encipherment uses different keys for encryption and decryption.

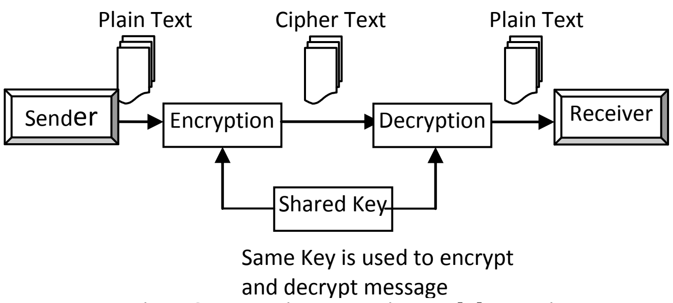

Fig 1. Symmetric encryption and decryption

Symmetric key algorithms can be further divided into: stream and block cipher[1]. DES[2], Blowfish[3], CAST-128[4], RC6[5], IDEA[6] are symmetric key block ciphers, and explained in further sections.

1.1 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 [2]. DES takes 64 bit plaintext which creates 64 bit ciphertext. The heart of DES is the DES function.DES function applies a 48 bit key to the rightmost 32 bits to produce a 32 bit output. This function is made up of four operations: an expansion permutation, a whitener, a group of S boxes and a straight permutation. DES is now considered to be insecure for many applications. This is chiefly due to the 56-bit key size being too small.

1.2 Blowfish

Blowfish is a keyed, symmetric block cipher, designed in 1993 by Bruce Schneier. Schneier designed Blowfish as a general-purpose algorithm, intended as an alternative to the aging DES. Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits. 18 sub-keys are derived from a single initial key [3]. It requires total 521 iterations to generate all required subkeys. It is a 16-round Feistel cipher and uses large key-dependent S-boxes. In structure it resembles CAST-128, which uses fixed S-boxes. Blowfish performs well for applications in which keys does not change often.

1.3 CAST-128

CAST-128 is a 12 or 16-round Feistel network with a 64-bit block size and a key size of between 40 to 128 bits (but only in 8-bit increments) [4]. The full 16 rounds are used when the key size is longer than 80 bits. Components include large 8×32-bit S-boxes based on bent functions, key-dependent rotations, modular addition and subtraction, and XOR operations. There are three alternating types of round function, but they are similar in structure and differ only in the choice of the exact operation (addition, subtraction or XOR) at various points. Although Entrust holds a patent on the CAST design procedure, CAST-128 is available worldwide on a royalty-free basis for commercial and non-commercial uses

1.4 RC6

RC6 is a block cipher based on RC5 and designed by Rivest, Sidney, and Yin for RSA Security [5]. Like RC5, RC6 is a parameterized algorithm where the block size, the key size, and the number of rounds are variable; again, the upper limit on the key size is 2040 bits [7].RC6 was designed to meet the requirements of the Advanced Encryption Standard (AES) competition. RC6 proper has a block size of 128 bits and supports key sizes of 128, 192 and 256 bits, but, like RC5. RC6 can be viewed as interweaving two parallel RC5 encryption processes. It uses an extra multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word.

1.5 IDEA
International Data Encryption Algorithm (IDEA) is a block cipher designed by James Massey of ETH Zurich and Xuejia Lai and was first described in 1991. The algorithm was intended as a replacement for the Data Encryption Standard (DES) [6]. IDEA is a minor revision of an earlier cipher, Proposed Encryption Standard (PES).IDEA was originally called Improved PES (IPES). IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical transformations and an output transformation.

2. DES (Data Encryption Standard)
DES is an algorithm that takes a fixed-length string of plaintext bits and transforms it into ciphertext bit string of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation. The key consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded.

2.1 Key Generation
Initially, 56 bits of the key are selected from the initial 64 by Permuted Choice 1 (PC-1) the remaining eight bits are either discarded or used as parity check bits. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one or two bits (specified for each round), and then 48 subkey bits are selected by Permuted Choice 2 (PC-2) 24 bits from the left half, and 24 from the right.

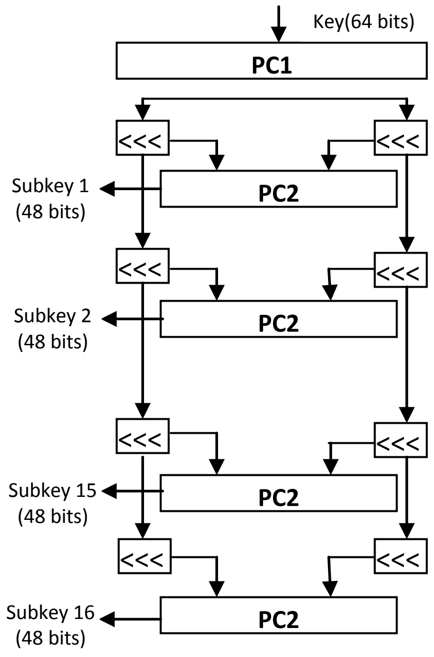

Fig 2. DES Key Generation

3. Encryption and Decryption
The algorithm's overall structure is shown in Figure 1: there are 16 identical stages of processing, termed rounds. There is also an initial and final permutation, termed IP and FP.

Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical.

The ⊕ symbol denotes the exclusive-OR (XOR) operation. The F-function scrambles half a block together with some of the key. The output from the F-function is then combined with the other half of the block, and the halves are swapped before the next round. After the final round, the halves are swapped; this is a feature of the Feistel structure which makes encryption and decryption similar processes.

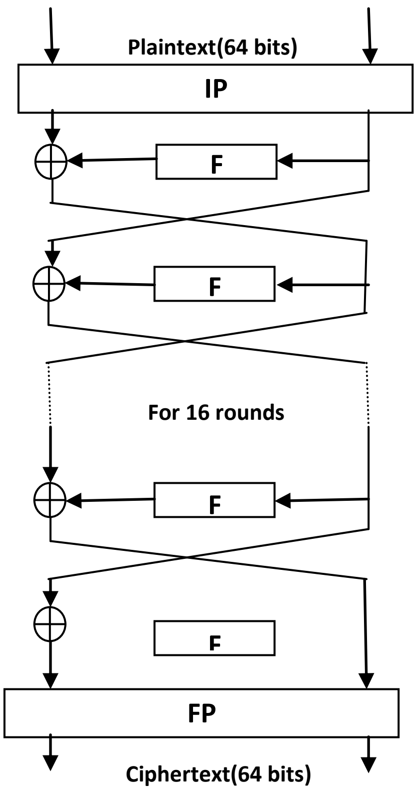

Fig 3. DES Encryption and DecryptionBLOWFISH

It is suitable for applications where the key does not change often, like communication link or an automatic file encryptor. Blowfish symmetric block cipher algorithm encrypts block data of 64-bits at a time. It follows the feistel network and this algorithm is divided into two parts. Key-generation & Data Encryption[8]

3.1 Key Generation
Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption [9].

The P-array consists of 18 32-bit subkeys:

P1, P2,..., P18.

There are four 32-bit S-boxes with 256 entries each:

S1,0, S1,1,..., S1,255;

S2,0, S2,1,..,, S2,255;

S3,0, S3,1,..., S3,255;

S4,0, S4,1,..,, S4,255.

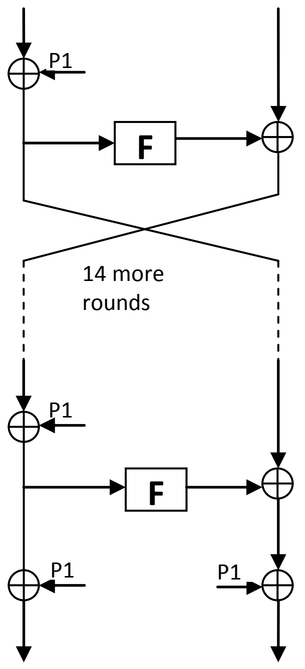

Fig 4: The Feistel structure of Blowfish

ALGORITHM

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).

   $P1 = 0x243f6a88$

   $P2 = 0x85a308d3$

   $P3 = 0x13198a2e$

   $P4 = 0x03707344$ etc.

2. XOR $P1$ with the first 32 bits of the key, XOR $P2$ with the second 32-bits of the key, and so on for all bits of the key (possibly up to $P14$). 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 (1) and (2).
4. Replace $P1$ and $P2$ with the output of step (3).
5. Encrypt the output of step (3) using the *Blowfish* algorithm with the modified subkeys.
6. Replace $P3$ and $P4$ with the output of step (5).
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.

In total 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

3.2 Encryption:-

INPUT

64 bit data element.

OUTPUT

64 bit cipher text.

ALGORITHM

1. Divide x into two 32-bit halves: xL, xR.
2. Then, for $i = 1\ to\ 16$:
3. $xL = xL \oplus Pi$
4. $xR = F(xL) \oplus xR$
5. Swap xL and xR
6. After the sixteenth round, swap xL and xR again to undo the last swap.
7. Then, $xR = xR \oplus P17$ and $xL = xL \oplus P18$.
8. Finally, recombine xL and xR to get the ciphertext.

3.3 Decryption

Decryption is exactly the same as encryption, except that P1, P2…P18 are used exactly in reverse order.

4. CAST-128

CAST has a classical Feistel network consisting of 16 rounds and operating on 64-bit blocks of plaintext to produce 64-bit blocks of cipher text. The key size varies from 40 bits to 128 bits in 8-bit increments.

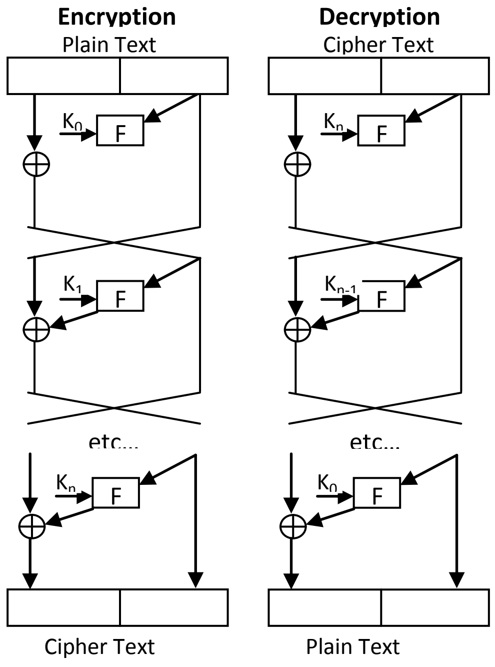

Fig 5. Cast-128 Encryption and Decryption

4.1 Key Generation

Three different round functions are used in CAST-128. The rounds are as follows (where "D" is the data input to the f function and "Ia" - "Id" are the most significant byte through least significant byte of I, respectively). Note that "+" and "-" are addition and subtraction modulo 2^32, "^" is bitwise XOR, and "<<<" is the circular left- shift operation.

Type 1: $I = ((Kmi + D) <<< Kri)$

$f = ((S1[Ia] \wedge S2[Ib]) - S3[Ic]) + S4[Id]$

Type 2: $I = ((Kmi \wedge D) <<< Kri)$

$f = ((S1[Ia] - S2[Ib]) + S3[Ic]) \wedge S4[Id]$

Type 3: $I = ((Kmi - D) <<< Kri)$

$$f = ((S1[Ia] + S2[Ib]) \wedge S3[Ic]) - S4[Id]$$

Rounds 1, 4, 7, 10, 13, and 16 use f function Type 1.

Rounds 2, 5, 8, 11, and 14 use f function Type 2.

Rounds 3, 6, 9, 12, and 15 use f function Type 3.

4.2 Encryption:-

The full encryption algorithm is given in the following four steps.

INPUT: plaintext $m1\ldots m64$; key $K = k1\ldots k128$.

OUTPUT: ciphertext $c1\ldots c64$.

1. Split the plaintext into left and right 32-bit halves $L0 = m1\ldots m32$ and $R0 = m33\ldots m64$.
2. for $i\ from$ 1 to 16, compute Li and Ri as follows:

   $Li = Ri - 1$

   $Ri = Li - 1 \wedge f(Ri - 1, Kmi, Kri)$
3. Exchange final blocks $L16, R16$ and concatenate to form the ciphertext.

4.3 Decryption

Decryption is identical to the encryption algorithm given above, except that the rounds (and therefore the subkey pairs) are used in reverse order to compute $(L0, R0)$ from $(R16, L16)$.

5. RC6

RC6 is a fully parameterized family of encryption algorithms. A version of RC6 is more accurately specified as RC6-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. Since the AES submission is targeted at w = 32 and r = 20, we shall use RC6 as shorthand to refer to such versions. When any other value of w or r is intended in the text, the parameter values will be specified as RC6-w r. Of particular relevance to the AES effort will be the versions of RC6 with 16-, 24-, and 32-byte keys [7].

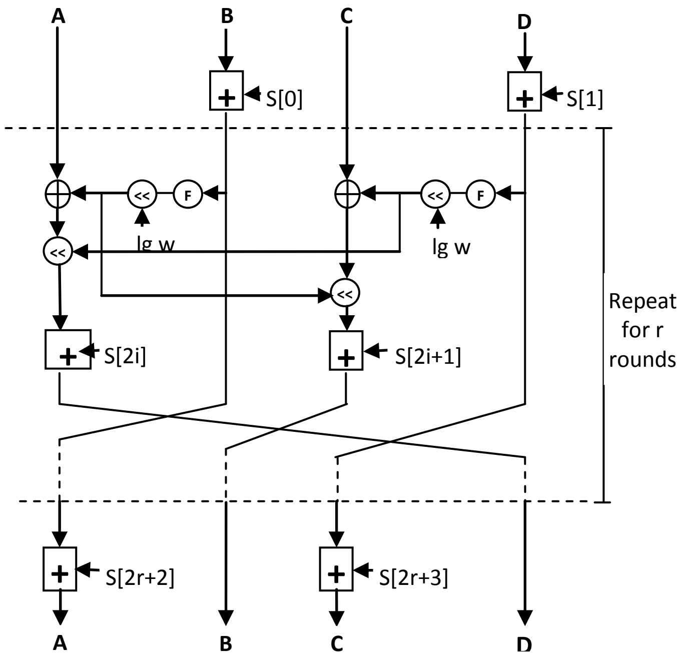

Fig 6. RC6 Encryption

For all variants, RC6-w r b operates on units of four w-bit words using the following six basic operations.

$A + B$	integer addition modulo $2w$
$A - B$	integer subtraction modulo $2w$
$A \oplus B$	bitwise exclusive-or of w-bit words
A x B	integer multiplication modulo $2w$
$A \lll B$	Rotate A to the left by the amount given by the least significant $log\ w$ bits of B
$A \ggg B$	Rotate A to the right, similarly
$(A, B, C, D) = (B, C, D, A)$	parallel assignment

5.1 Key Expansion

Use two magic constants:-

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

$Qw = Odd((t - 1)2^{w)}$

Where:-

$e = 2.718281828459$…….(base of natural logarithm)

$t = 1.618033988749$……..($golden\ ratio = (1 + \sqrt{5})/2$)

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

INPUT

b byte key that is preloaded into c word array $L[0,1, \ldots, c - 1]$

r denotes the no of rounds.

OUTPUT

$(2r + 4)$ w-bit round keys $S[0,1, \ldots, 2r + 2, 2r + 3]$.

ALGORITHM

```
$S[0] = Pw$
$For\ i = 1\ to\ 2r + 3\ do$
$S[i] = S[i - 1] + Qw$
$X = Y = a = b = 0$
$Iteration = 3 * max(c, 2r + 4)$
For $i = 1\ to\ Iteration$ do
$X = S[a] = (S[a] + X + Y) <<< 3$
$Y = L[b] = (L[b] + X + Y) <<< (X + Y)$
$i = (a + 1)\ mod\ (2r + 4)$
$j = (b + 1)\ mod\ c$
```

5.2 Encryption

Four w-bit registers A, B, C, D contain the initial input plain-text as well as the output ciphertext at the end of encryption. The first byte of plaintext is placed in the least significant byte of A, the last byte of plaintext is placed into the most significant byte of D [5].

INPUT

Plaintext stored in four w-bit input registers $A; B; C; D$

Number r of rounds w-bit round keys $S[0; ::: ; 2r + 3]$

OUTPUT

Ciphertext stored in $A; B; C; D$

ALGORITHM

```
1.  $B = B + S[0]$
2.  $D = D + S[1]$
3.  $Repeat\ step\ 4\ to\ 8\ for\ i = 1\ to\ r\ do$
4.  $t = (B \times (2B + 1)) <<< log\ w$
5.  $u = (D \times (2D + 1)) <<< log\ w$
6.  $A = ((A \oplus t) <<< u) + S[2i]$
7.  $C = ((C \oplus u) <<< t) + S[2i + 1]$
8.  $(A, B, C, D) = (B, C, D, A)$
9.  $A = A + S[2r + 2]$
10. $C = C + S[2r + 3]$
```

5.3 Decryption

For decryption of cipher-text load these cipher text into registers A, B, C, D Algorithm uses integer subtraction modulo 2w and right rotation on registers for getting plain text.

INPUT

Ciphertext stored in four w-bit input registers $A; B; C; D$

Number r of rounds

w-bit round keys $S[0; ::: ; 2r + 3]$

OUTPUT

Plaintext stored in $A; B; C; D$

ALGORITHM

```
1.  $C = C - S[2r + 3]$
2.  $A = A - S[2r + 2]$
3.  $Repeat\ step\ 4\ to\ 8\ for\ i = r\ down\ to\ 1\ do$
4.  $(A; B; C; D) = (D; A; B; C)$
5.  $u = (D \times (2D + 1)) <<< log\ w$
6.  $t = (B \times (2B + 1)) <<< log\ w$
7.  $C = ((C \times S[2i + 1]) >>> t) \oplus u$
8.  $A = ((A \times S[2i]) >>> u) \oplus t$
9.  $D = D - S[1]$
10. $B = B - S[0]$
```

6. IDEA

IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical transformations and an output transformation. The processes for encryption and decryption are similar.

6.1 Encryption and Decryption

For each of the eight complete rounds, the 64-bit plaintext block is split into four 16-bit sub-blocks: X1, X2, X3, X4. The 64-bit input block is the concatenation of the subblocks [9]:

X1 || X2 || X3 || X4, where || denotes concatenation. Each complete round requires six subkeys. The 128-bit key is split into eight 16-bit blocks, which become eight subkeys. The first six subkeys are used in round one and the remaining two subkeys are used in round two.

Each round uses each of the three algebraic operations: bitwise XOR, addition modulo 2^{16}, and multiplication modulo $2^{16} + 1$.

ALGORITHM

(Multiply means multiplication modulo $2^{16} + 1$,

Add means addition modulo 2^{16})

1. Multiply X1 and the first subkey Z1.
2. Add X2 and the second subkey Z2.
3. Add X3 and the third subkey Z3.
4. Multiply X4 and the fourth subkey Z4.
5. Bitwise XOR the results of steps 1 and 3.
6. Bitwise XOR the results of steps 2 and 4.
7. Multiply the result of step 5 and the fifth subkey Z5.
8. Add the results of steps 6 and 7.
9. Multiply the result of step 8 and the sixth subkey Z6.
10. Add the results of steps 7 and 9.
11. Bitwise XOR the results of steps 1 and 9.
12. Bitwise XOR the results of steps 3 and 9.
13. Bitwise XOR the results of steps 2 and 10.
14. Bitwise XOR the results of steps 4 and 10.

For every round except the final transformation, a swap occurs, and the input to the next round is: result of step 11 k result of step 13 k result of step 12 k result of step 14, which becomes X1 k X2 k X3 k X4, the input for the next round. After round 8, a ninth "half round" final transformation occurs:

1. Multiply X1 and the first subkey.
2. Add X2 and the second subkey.
3. Add X3 and the third subkey.
4. Multiply X4 and the fourth subkey.

The concatenation of the blocks is the output.

6.2 Key Scheduling

Each of the eight complete rounds requires six subkeys, and the final transformation "half round" requires four subkeys; so, the entire process requires 52 subkeys.

The 128-bit key is split into eight 16-bit subkeys which forms the first 8 subkeys. Then the bits are shifted to the left 25 bits. The resulting 128-bit string is split into eight 16-bit blocks that become the next eight subkeys. The shifting and splitting process is repeated until 52 subkeys are generated.

The shifts of 25 bits ensure that repetition does not occur in the subkeys. Six subkeys are used in each of the 8 rounds.

The final 4 subkeys are used in the ninth "half round" final transformation.

7. PERFORMANCE AND ANALYSIS

Performance analysis of RC6, Blowfish, IDEA, CAST-128 & DES is done to provide some measurement on the encryption and decryption. Various parameters such as number of rounds, file size, key length and key generation time are inquired. A comparative analysis of RC5, Blowfish & DES has already been performed [10] [11]. 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 encryption algorithm were implemented in java using IAIK-JCE library in NetBeans IDE 7.0.1. Performance was measured on Intel(R) Core(TM) i3 CPU M 370 @ 2.40 Ghz 2.39 Ghz 32 bit system with 4 GB of RAM running Windows 7 Ultimate.

7.1 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.

7.1.1 On the basis of execution time

We compare the execution time of each algorithm for encryption, decryption and key generation. Figure 7 shows the key generation time of different cryptographic algorithm in milliseconds. Figure 8 shows the average encryption time of different cryptographic algorithm in milliseconds and figure 9 shows the average decryption time of different cryptographic algorithm in milliseconds.

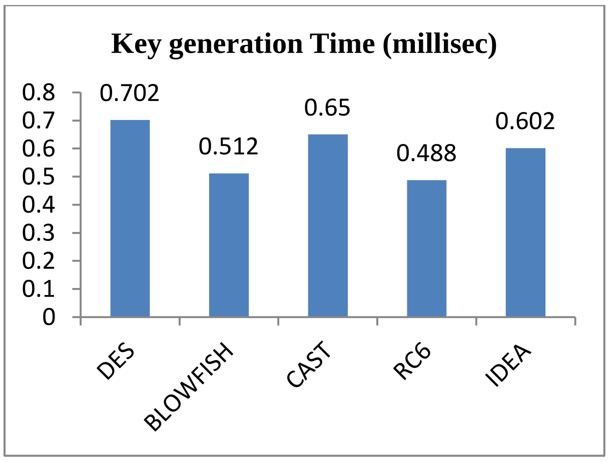

Fig 7. Key Generation time of different algorithms

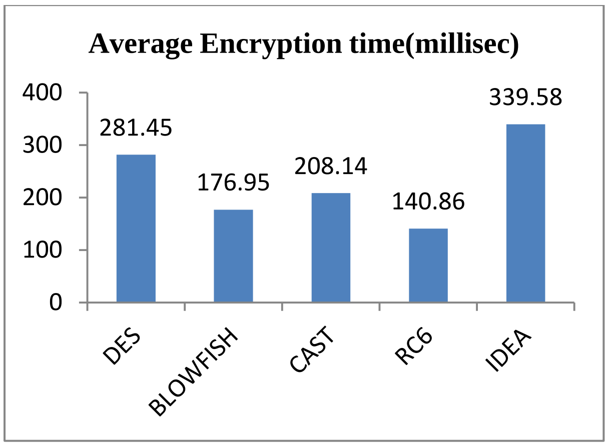

Fig 8. Encryption time of Different algorithms

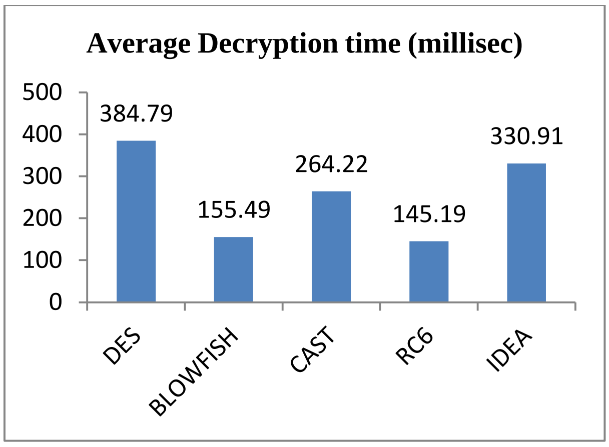

Fig 9. Decryption time of Different algorithms

7.1.2 On the basis of Throughput

Throughput of encryption and decryption of different algorithms in MegaBytes/Sec is shown in figure 10 and 11 respectively.

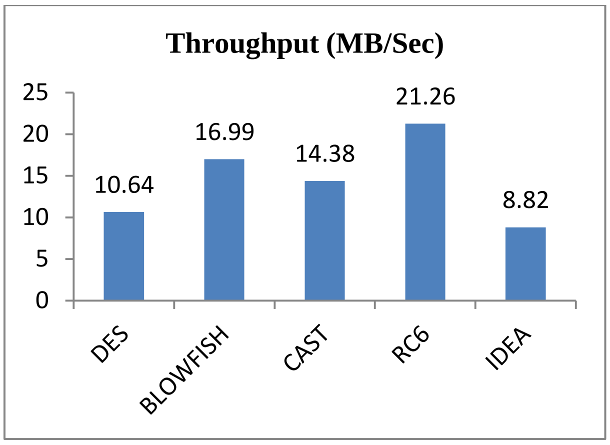

Fig 10.Throughput of Encryption of Different Algorithms

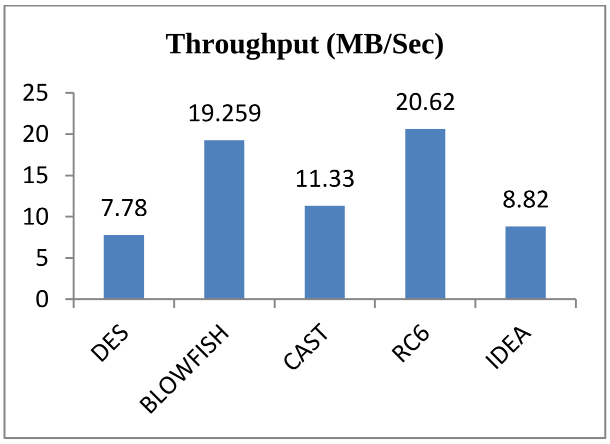

Fig 11. Throughput of Decryption of Different Algorithms

7.2 Result Analysis

Figure 12 and Figure 13 shows the mean execution time of encryption and decryption respectively for the algorithms to execute the files of different size. RC6 has faster execution time than any other algorithm considered here. Throughput of RC6 and Blowfish is almost same. Result also concludes that

Blowfish performs better than IDEA.IDEA has better throughput than DES for decryption but for Encryption DES performs better. Throughput of CAST-128 is better than IDEA but the difference is very small.

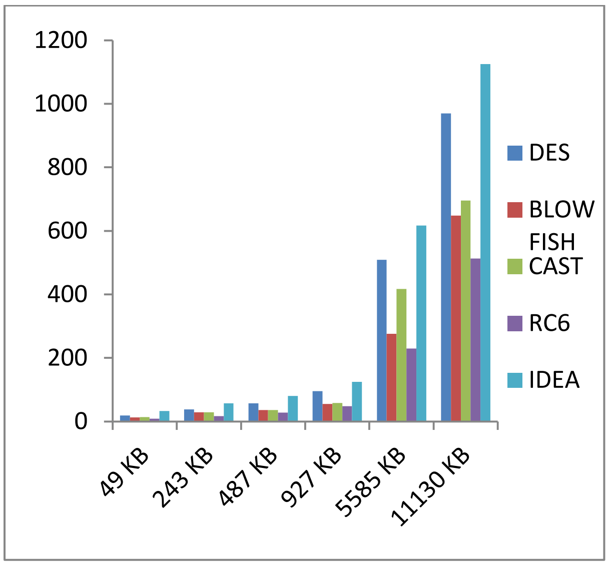

Fig 12. Execution time of Encryption for different file sizes

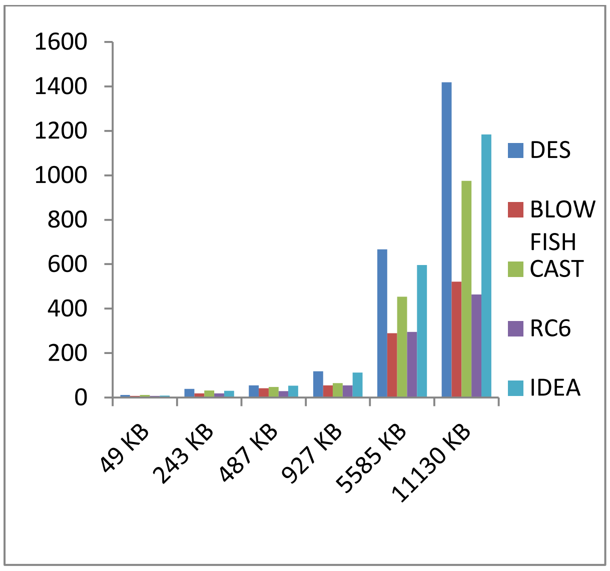

Fig 13. Execution time of Decryption for different file sizes

8. CONCLUSION

In this research paper RC6, Blowfish, IDEA, CAST-128 and DES block cipher algorithms were compared using IAIK-JCE library in java program in NETBEANS 7.0.1. Performance of these five algorithms were measured on Intel(R) Core(TM) i3 CPU M 370 @ 2.40 Ghz 2.39 Ghz 32 bit system with 4 GB of RAM running Windows 7 Ultimate.

Comparative analysis of RC6, Blowfish, IDE, CAST-128 and DES have been done with a set of input files and evaluated the encryption & decryption time. Results conclude that RC6 is faster than Blowfish and which is faster than CAST-128 which is faster than IDEA and DES. Blowfish is suitable for applications where the key does not change often. Using RC6 is beneficial where high encryption rate is required.

9. REFERENCES

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

[2] “Data Encryption Standard”, “wikipedia.org”, [online] Available at: http://en.wikipedia.org/wiki/Data_Encryption_Standard

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

[4] CAST-128 available at http://www.ipa.go.jp/security/rfc/RFC2144EN.html

[5] Ronald L. Rivest,”THE RC6 Block Cipher” RSA Laboratories, 2955 Campus Drive, Suite 400, San Mateo, CA 94403, USA.

[6] IDEA “wikipedia.org**”.** Available at: http://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm

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

[8] 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

[9] B. Schneier, "Applied Cryptography", John Wiley & Sons Inc., 1999

[10] Harsh Kumar Verma, and Ravindra Kumar Singh, “Performance Analysis of RC5, Blowfish and DES Block Cipher Algorithms”, International Journal of Computer Applications (0975 – 8887) March 2012 Volume 42– No.16

[11] Tingyuan Nie, Chuanwang Song, Xulong Zhi “Performance Evaluation of DES and Blowfish Algorithms”,IEEE Biomedical Engineering and Computer Science, 2010. ICBECS 2010. International Conference
23-25 April 2010,pp. 1-4

[12] Vikas Tyagi, Shrinivas Singh, Volume 3, No. 4, April 2012 Journal of Global Research in Computer Science “ENHANCEMENT OF RC6 (RC6_EN) BLOCK CIPHER ALGORITHM AND COMPARISON WITH RC5 & RC6”.

[13] CAST “wikipedia.org**”.** Available at: http://en.wikipedia.org/wiki/CAST-128

[14] RC6 “wikipedia.org**”.** Available at: http://en.wikipedia.org/wiki/RC6

[15] Andreas Sterbenz and Peter Lipp “Performance of the AES Candidate Algorithms in Java”, Institute for Applied Information Processing and Communications Graz, University of Technology Inffeldgasse 16, A-8010 Graz, Austria.

[16] MelekD .Yucel and R.CllneyAt car “COMPARISON OF THE BLOCK CIPHERS DES AND IDEA” EIT,CO'99 INTERNATIONAL CONFERBNCE ON ELECTRICAL AND ELECTRONICS ENGINEERING.