Performance Analysis of RC6, Twofish and Rijndael 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 RC6, Twofish and Rijndael 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 algorithms are parameterized algorithm and were designed to meet the requirements of the Advanced Encryption Standard (AES) competition and selected among five finalists of that competition. These three algorithms have a variable block size and a variable key size in their structure and encrypt four w-bits at a time. Allowable choices for w are 16 bits, 32 bits, and 64 bits. Twofish and Rijndael have same structure for encryption and decryption while RC6 have different. RC6, Twofish and Rijndael have 20, 16 and 10 rounds respectively. Performances of these three algorithms have been evaluated on key size of 128-bits, 192-bit and 256-bit in this paper.

General Terms
Cryptography, Block cipher, Symmetric encryption, RC6, Twofish, Rijndael

Keywords
Cryptography, Block cipher, Symmetric encryption, RC6, Twofish, Rijndael

1. INTRODUCTION
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.

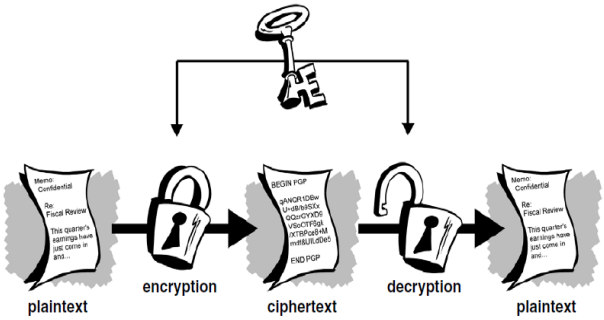


Fig 1 : Symetric encryption

In cryptography, the Advanced Encryption Standard (AES) [1] is an encryption standard adopted by the U.S. government. Back in 1997 the National Institute of Standards and Technology (NIST) made a public call for new cipher algorithms that could replace the DES. A rough summary of the requirements made by NIST for the new AES were the following:

- ✓ Symmetric-key cipher
- ✓ Block cipher
- ✓ Support for 128-bit block sizes
- ✓ Support for 128-, 192-, and 256-bit key lengths

A combination of factors such as security, performance, efficiency, ease of implementation and flexibility contributed to the selection of this algorithm as the AES. RC6 [2], Twofish [3] and Rijndael [4] were designed to meet the requirements of the Advanced Encryption Standard (AES) competition and selected among five finalists of that competition, and explained in further sections.

1.1 RC6
RC6 is derived from RC5 [5]. There are two main new features in RC6 compared to RC5: the inclusion of integer multiplication and the use of four w-bit working registers instead of two w-bit registers as in RC5. It was designed by Ron Rivest, Matt Robshaw, Ray Sidney and Yiqun Lisa Yin. The algorithm was also submitted to the NESSIE and CRYPTREC projects. It is a proprietary algorithm, patented by RSA Security [6].

1.2 Twofish
Twofish is a 128-bit symmetric key block cipher proposed by B.Schneier [3]. Twofish accepts a variable-length key up to 256 bits. The cipher is a 16-round Feistel network which adapted components from the ciphers Khufu [7], Square [8] and SAFER [9]. Notable features of the design Twofish include key-dependent S-boxes, maximum distance separable (MDS) matrice, pseudo-Hadamard transform (PHT) and a highly complex key schedule. Basic Terms used in Twofish:

1.2.1 Feistel Networks
A Feistel network [3] is a general method of transforming any function (usually called the F function) into a permutation. It was invented by Horst Feistel [10] 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 [11] in 1949.

1.2.2 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 [12].

1.2.3 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 [12].

1.2.4 S-boxes
An S-box [3] 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.

1.2.5 MDS Matrices
A maximum distance separable (MDS) [3] code over a field is a linear mapping from *a* field elements to *b* field elements, producing a composite vector of $a + b$ elements, with the property that the minimum number of non-zero elements in any non-zero vector is at least $b + 1$ [13]. Reed-Solomon (RS) error-correcting codes are known to be MDS. A necessary and sufficient condition for $a \times b$ matrix to be MDS is that all possible square sub matrices, obtained by discarding rows or columns, are non-singular.

1.2.6 Pseudo-Hadamard Transforms
A pseudo-Hadamard transform (PHT) [3] is a simple mixing operation that runs quickly in software. It is used for diffusion. For given two inputs *a* & *b*, the 32-bit PHT is defined as:

$$a_0 = a + b \bmod 2^{32} \quad \ldots (1)$$

$$b_0 = a + 2b \bmod 2^{32} \quad \ldots (2)$$

1.2.7 Whitening
Whitening [3], the technique of XORing key material before the first round and after the last round, it was shown that whitening substantially increases the difficulty of key search attacks against the remainder of the cipher.

1.3 Rijndael
The Rijndael algorithm is a block cipher of 10, 12 ,14 rounds that encrypts blocks of 128, 192, or 256 bits respectively using symmetric keys of 128, 192 or 256 bits. In October 2000, the Rijndael (pronounced Rain Doll) algorithm was chosen as the basis for the new standard encryption algorithm and now it is known as AES algorithm [14]. Specifically, Rijndael appears to perform consistently well in both hardware and software platforms under a wide range of environments. Brute force attack is the only effective attack known against it, in which the attacker tries to test all the characters combinations to unlock the encryption. Rijndael is not a Feistel structure but it also makes use of S-Boxes for increasing diffusion.

2. RC6
RC6 is very similar to RC5 in structure, using data-dependent rotations [5], addition modulo 2^w and XOR operations; in fact, RC6 could be viewed as interweaving two parallel RC5 encryption processes. However, RC6 does use an extra multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word and not just the least significant few bits. Integer multiplication is used to increase the diffusion achieved per round so that fewer rounds are needed and the speed of the cipher can be increased. The base-two logarithm of w will be denoted by lg w.

Like RC5, 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. For all variants, RC6-w/r/b operates on units of four w-bit words using the following basic operations [15].

The operations used in RC6 are defined as followings.

A+B integer addition modulo 2^w

A-B integer subtraction modulo 2^w

A⊕B bitwise exclusive-or of w-bit words

A*B integer multiplication modulo 2^w

$A^{<<<}B$ rotation of the w-bit word A to the left by the amount given by the least significant lg w bits of B

$A^{>>>}B$ rotation of the w-bit word A to the right by the amount given by the least significant lg w bits of B

$f(x) = x(2x+1) \bmod 2^w$

There are three routines in RC5: key expansion, encryption, and decryption. We discus each of them in next sections, Key-Expansion algorithm is used to generate the round sub keys that will be use in encryption and decryption algorithms. RC6 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. RC6 is a symmetric key encryption so encryption and decryption algorithms uses same key.

2.1 Key-Expansion Algorithm
Key-Expansion Algorithm of RC6 is similar as RC5, only difference is that RC6 will be generate 2r+4 additive round keys rather that 2(r + 1) used in RC5 [15].

Key-Expansion with RC6-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+4 w-bit round keys S[0,1, …, 2r + 2,2r+3].
Procedure:

```
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
}
```

2.2 Encryption Algorithm
Fig 2 illustrates the encryption procedure of RC6; decryption procedure is just reverse of this structure by converting addition operation to subtraction operations.

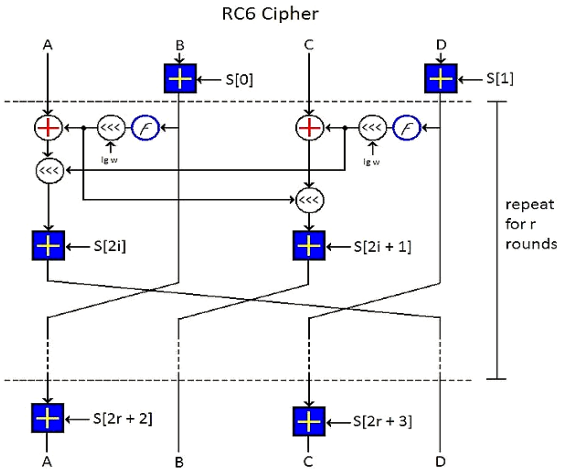


Fig 2: RC6 Block Cipher

RC6 works with four w-bit registers A, B, C, D 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 D. Pseudo code of encryption [15] is given below; at first we load plain text in to registers A, B, C, D and then apply these operations to encrypt the plain text.

Encryption with RC6-w/r/b

Input: Plain text stored in four w-bit input registers A, B, C, D. r denotes the no of rounds and 2r+4 w-bit round keys S[0,1, .., 2r + 3]
Output: Cipher text will be store in A, B, C, D
Procedure:
B = B + S[0]
D = D + S[1]
for i = 1 to r do
{
t = (B * (2B + 1)) <<< lg w
u = (D * (2D + 1)) <<< lg w
A = ((A ⊕ t) <<< u) + S[2i]
C = ((C ⊕ u) <<< t) + S[2i+ 1]
(A, B, C, D) = (B, C, D, A)
}
A = A + S[2r + 2]
C = C + S[2r + 3]

Operation (A, B, C, D) = (B, C, D, A) means the parallel assignment of values on the right to registers on the left. After applying these operations on registers A, B, C, D plain text gets converted into the cipher text.

2.3 Decryption Algorithm

Pseudo code of decryption [15] is given below; for decryption of cipher text load these cipher text into registers A, B, C, D and then apply these operations to convert cipher text into plain text.

Decryption with RC6-w/r/b

Input: Cipher text stored in four w-bit input registers A, B, C, D. r denotes the no of rounds and 2r+4 w-bit round keys S[0, 1,.., 2r + 3]
Output: Plain text will be store in A, B, C, D
Procedure:
C = C - S[2r + 3]
A = A - S[2r + 2]
for i = r down to 1 do
{
(A, B, C, D) = (D, A, B, C)
u = (D * (2D + 1)) <<< lg w
t = (B * (2B + 1)) <<< lg w
C = ((C - S[2i + 1]) >>> t) ⊕ u
A = ((A - S[2i]) >>> u) ⊕ t
}
D = D - S[1]
B = B - S[0]

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

3. TWOFISH

Fig 3 shows an overview of the Twofish block cipher. Twofish uses a 16-round Feistel-like structure with additional whitening of the input and output. The only non-Feistel elements are the 1-bit rotates. The rotations can be moved into the *F* function to create a pure Feistel structure, but this requires an additional rotation of the words just before the output whitening step.

The plaintext is split into four 32-bit words, these are XORed with four key words in input whitening step. This is followed by sixteen rounds. In each round, the two words on the left are used as input to the *g* functions. (One of them is rotated by 8 bits first.) The *g* function consists of four byte-wide key-dependent S-boxes, followed by a linear mixing step based on an MDS matrix. The results of the two *g* functions are combined using a Pseudo-Hadamard Transform (PHT) and two keywords are added. These two results are then XORed into the words on the right (one of which is rotated left by 1 bit first and other one is rotated right afterwards). The left and right halves are then swapped for the next round except the last round and the four words are XORed with four more key words to produce the cipher text [3].

Operations used in Enhanced Twofish

- ✓ A+B integer addition modulo 2^w
- ✓ A⊕B bitwise exclusive-or of w-bit words.
- ✓ $^{<<<}$n rotation to the left by n-bit.
- ✓ $^{>>>}$n rotation to the right by n-bit.
- ✓ (A, B, C, D) = (C, D, A, B) parallel assignment

Notations used in Enhanced Twofish

- ✓ 0x hexadecimal representation
- ✓ $^{r}X_{Li}$ the i-th left-half data of X in the round r (i=1~2,r=0~16)
- ✓ $^{r}X_{Ri}$ the i-th right-half data of X in the round r (i=1~2,r=0~16)
- ✓ M the master key consists of 32 bytes m0,…,m31
- ✓ Ki the input/output whitening sub key and round sub key (i=0~39)
- ✓ Li the S-box key(i=0~3)

3.1 Encryption

Twofish block cipher has same structure for encryption and decryption both; it is a significant advantage of Feistel network. To encrypt a 128-bit input plaintext P, we first divide it into four 32-bit data $^{0}X_{L1,}$ $^{0}X_{L2,}$ $^{0}X_{R1,}$ $^{0}X_{R2}$ and XORed with four 32-bit sub key K_0, K_1, K_2, K_3. Then, the following operations are performed form r=0 to 15.

$(T_0, T_1) = F(^{r}X_{L1}, (^{r}X_{L2}<<<8), L)$

$^{r+1}X_{L1} = ((T_0+K_{4r+8}) \oplus {}^{r}X_{R1}) <<< 1$

$^{r+1}X_{L2} = (T_1+K_{4r+9}) \oplus ({}^{r}X_{R2} >>> 1)$

$^{r+1}X_{R1} = {}^{r}X_{L1}$

$^{r+1}X_{R2} = {}^{r}X_{L2}$

After 16 rounds undo last swap and then $^{15}X_{L1,}$ $^{15}X_{L2,}$ $^{15}X_{R1,}$ $^{15}X_{R2}$ are concatenated and XORed with K_4, K_5, K_6, K_7. The resultant output is the 128-bit cipher text.

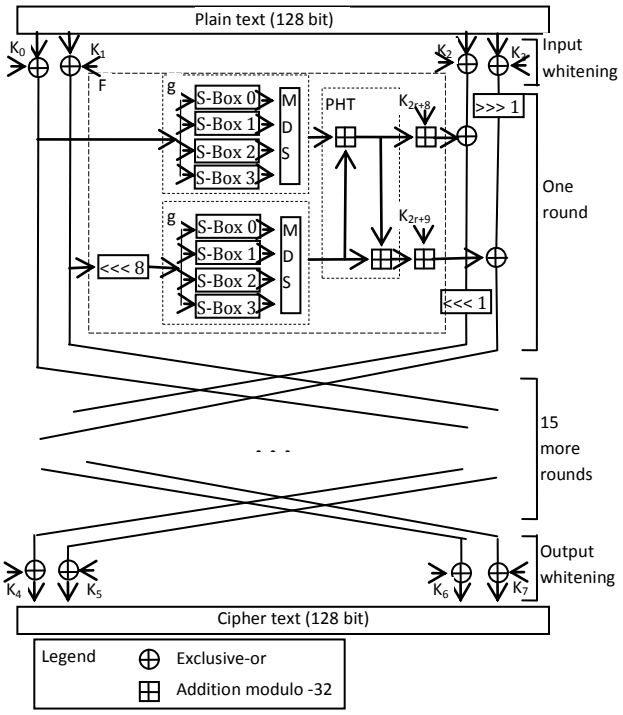


Fig 3: Twofish Block Cipher

3.2 Decryption

The decryption procedure of Twofish can be done in the same way as the encryption procedure by reversing the order of the sub keys, which is one of merits of Feistel networks [3].

4. RIJNDAEL

Rijndael encrypts blocks of 128, 192, or 256 bits respectively using symmetric keys of 128, 192 or 256 bits. Operations in Rijndael algorithm are performed on a two-dimensional byte array of four rows and four columns or State that contain 128 bits [16]. Rijndael algorithm can be better understood in three parts, KeyExpansion algorithm, Encryption algorithm and Decryption algorithm.

4.1 KeyExpansion Algorithm:

Round keys K_i are derived from the 128-bit user key by means of the key expansion algorithm. The total number of round keys required is equal to r + 1(where r = Number of rounds) because one extra key is needed in the Initial round. KeyExpansion algorithm takes a 4-word (16-byte) as input and produces a linear array of (r + 1) * 4 words ((r + 1) * 16 bytes). The following pseudocode describes the KeyExpansion algorithm [4].

*4.1.1 KeyExpansion (byte key [16], word w [(r+1)*4])*

```
{
word temp;
for (i = 0; i < 4; i++)
   {
w [i] = (key [4 * i], key [4 * i + 1], key [4 * i + 2],
key [4 * i + 3]);
   }
for (i = 4; i < (r + 1) * 4; i++)
   {
temp = w [i - 1];
if (i % 4 = 0)
      {
temp = SubWord ( RotWord (temp)) ⊕ Rcon[i / 4];
      }
w [i] = w [i - 4] ≈ temp;
   }
}
```

4.1.2 KeyExpansion Algorithm uses these terms [4],

4.1.2.1 RotWord: it performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].

4.1.2.2 SubWord: it performs a byte substitution on each byte of its input word, using the S-Box.

4.1.2.3 *Rcon[x]:* is a round constant.

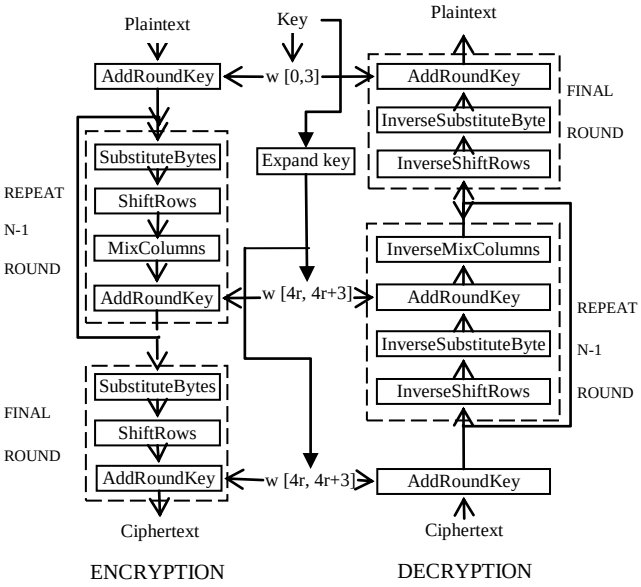


Fig 4: Rijndael Encryption Algorithm

4.2 Encryption Algorithm:

The process of Rijndael encryption is illustrated in fig 4. The algorithm consists of Initial round, (r-1) uniform rounds, followed by Final round. The Initial round performs AddRoundKey function (as explained below). The reason that the rounds have been listed as "(r-1) uniform rounds followed by Final round" is because the Final round involves a slightly different manipulation from the others. The (r-1) uniform rounds consist of four functions [17]:

1. SubstituteBytes (SB): a non-linear substitution step where each byte is replaced with another according to a lookup table (S-Box).
2. ShiftRows (SR): a transposition step where each row of the state is shifted cyclically a certain number of steps. Row 1 is circular left shift by one place, Row 2 by two, Row 3 by three places whereas, Row 0 remains unchanged.
3. MixColumns (MC): a mixing operation which operates on the columns of the state combining the four bytes in

each column. Each column is considered a polynomial over GF(2^8) and multiplied modulo X^4+ 1 with a fixed polynomial C(x), where C(x) = '03'x^3 + '01'x^2 + '01'x + '02'

4. AddRoundKey (ARK): each byte of the state is XORed with the round key.

The Final round only performs SubstituteBytes, ShiftRows and AddRoundKey transformations. Output of the final round is treated as the cipher text.

4.3 Decryption Algorithm:

In decryption method, the sequence of the transformations differs from that of the encryption approach. It comprises of an inverse of the final round, inverses of the (r-1) rounds, followed by the initial round. The inverse of the round is found by inverting each of the transformations in the round.

1. InverseSubstituteBytes (SB^{-1}): it is obtained by applying the inverse of the affine transformation and taking the multiplicative inverse in GF (2^8) of the result.
2. InverseShiftRows (SR^{-1}): In this transformation, row 0 is not shifted, row 1 is shifted left by three places, row 2 by two places and row 3 by one places.
3. InverseMixColumns (MC^{-1}): The polynomial, C(x), used to transform the state columns in the InverseMixColumns is given by, C(x) = 'B'x^3 + 'OD'x^2 + '09'x + 'OE'

The same set of keys is used in encryption and decryption process of Rijndael but is used in reverse order.

5. COMPARISON AND ANALYSIS

A comparative analysis of RC6, Twofish & Rijndael 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 1 summarizes the comparison of RC6, Twofish & Rijndael for different design parameters such as word size, block size, number of rounds and secret key size.

Table 1: Comparison on the basis of parameters

Parameters	Algorithm Type		
	RC6	**Twofish**	**Rijndael**
b (key length in bytes)	0 - 255	16, 24 or 32	16, 24 or 32
r (no of rounds)	0 – 255 (standard 20)	16	10, 12, 14
No of round keys	2r+4	2r+8	r+1
Block size in words	4w	4w	4w
w (word size in bits)	16, 32, 64 (standard 32)	16, 32, 64 (standard 32)	16, 32, 64 (standard 32)
Block size in bits	64, 128, 256 (standard 128)	64, 128, 256 (standard 128)	64, 128, 256 (standard 128)
Used Function	$F(x) = x(2x+1) \bmod 2^w$	S-Box, MDS, PHT	SB, SR, MC, ARK, SB^{-1}, SR^{-1}, MC^{-1}, S-Box
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 have three values 16-bytes, 24-bytes and 32-bytes while number of round(r) was fixed 20 for RC6, 16 for Twofish and 10 for Rijndael.

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 2, 3 and 4.

Table 2: Comparison for 16-bytes key

File Name (file type)	File Size (in KB)	RC6	Twofish	Rijndael
A.doc	712	200.8125	232.5	229.65
B.pdf	649	178.5	205	214.625
C.jpg	656	196.45	224.5	232.75
D.gif	1396	258.925	314.75	309.5
E.mp3	2068	320	383.5	380.45
F.avi	2800	357.5	426.735	435

Table 3: Comparison for 24-bytes key

File Name (file type)	File Size (in KB)	RC6	Twofish	Rijndael
A.doc	712	205	241.25	246.125
B.pdf	649	183	209.85	223.25
C.jpg	656	205.125	238.75	241.325
D.gif	1396	263.5	321.45	327.847
E.mp3	2068	337.5	397.125	398.575
F.avi	2800	370.25	443.95	454.65

Table 4: Comparison for 32-bytes key

File Name (file type)	File Size (in KB)	RC6	Twofish	Rijndael
A.doc	712	209.75	256.95	268.125
B.pdf	649	187.5	213.75	228.75
C.jpg	656	209.8125	241.5	253.45
D.gif	1396	267.5125	329.75	333.85
E.mp3	2068	351.25	415.025	421.25
F.avi	2800	375	455.75	473.925

Fig 5, 6 and 7 illustrate the execution time according to their file size for each files using algorithms RC6, Twofish and Rijndael for 16-bytes, 24-bytes and 32-bytes key respectively. Execution time of RC6 is represented by blue color; Twofish is represented by red color and Rijndael is represented by green color.

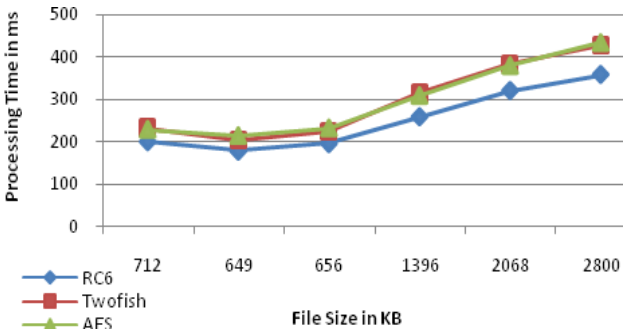


Fig 5: Execution time for 16-bytes key

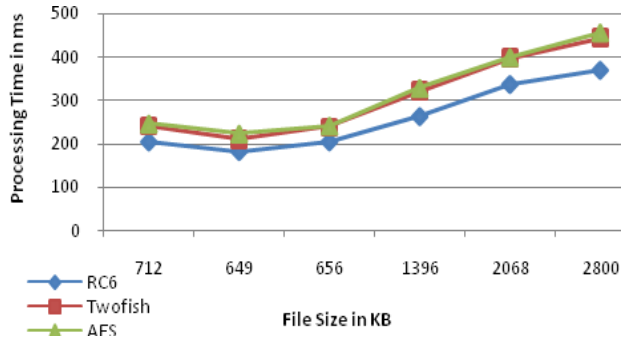


Fig 6: Execution time for 24-bytes key

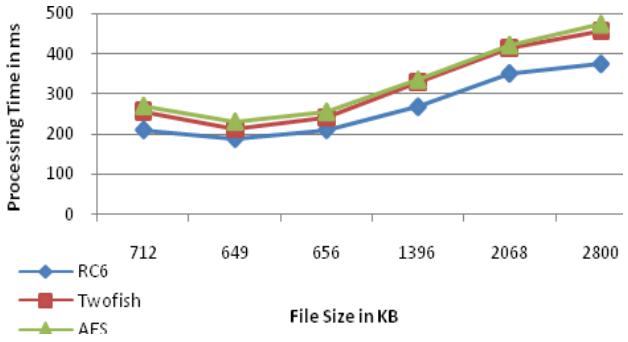


Fig 7: Execution time for 32-bytes key

Graph shows that RC6 performs faster than Twofish & Rijndael for every key size. It can also be concluded that increasing key size decreases the performance.

5.2.2 On the basis of Resource Utilization (CPU Utilization & Memory Utilization)

In this section a video file (.avi) of 2800KB was executed by these three algorithms RC6, Twofish and Rijndael. 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.

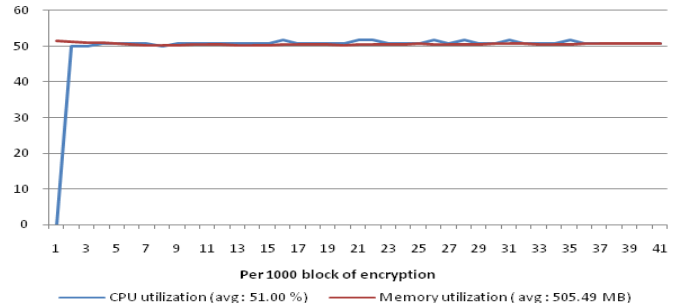


Fig 8: CPU Utilization & Memory Utilization of RC6

Fig 8 shows the CPU utilization and Memory utilization for RC6 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 51.00 % and average Memory utilization is 505.49 MB for RC6.

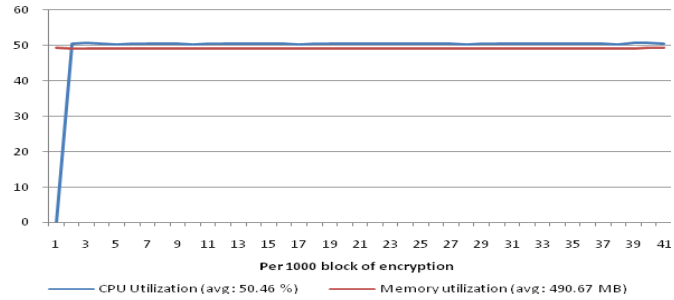


Fig 9: CPU Utilization & Memory Utilization of Twofish

Fig 9 shows the CPU utilization and Memory utilization for Twofish 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 50.46 % and average Memory utilization is 490.67 MB for Twofish.

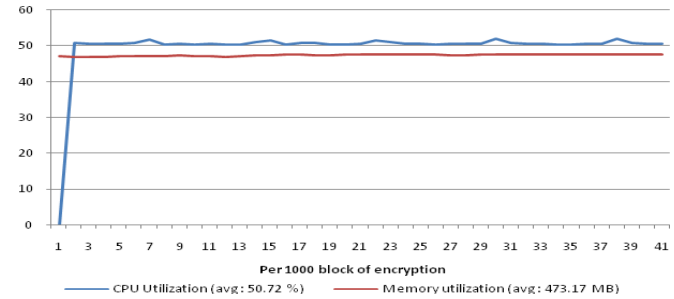


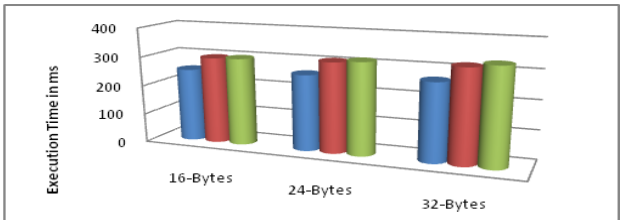
Fig 10: CPU Utilization & Memory Utilization of Rijndael

Fig 9 shows the CPU utilization and Memory utilization for Rijndael 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 50.72 % and average Memory utilization is 473.17 MB for Rijndael.

5.3 Result Analysis

RC6 performs faster than Twofish & Rijndael. Fig 11 shows the average execution time for these three algorithms to execute the files mentioned in table 2, 3 and 4. According to fig 11 RC6 is 1.182 times faster than Twofish and 1.191 times faster than Rijndael for 16-bytes key, 1.184 times faster than Twofish and 1.209 times faster than Rijndael for 24-bytes key, 1.195 times faster than Twofish and 1.236 times faster

than Rijndael for 32-bytes key. Result also concludes that performance of all these three algorithms are inversely proportional to keysize, if keysize will increase the performance will decrease and vice-versa.



	16-Bytes	24-Bytes	32-Bytes
RC6	252.03	260.73	266.8
Twofish	297.83	308.73	318.79
AES	300.33	315.29	329.89

Fig 11: Average Execution Time in millisecond

In addition, if we consider on resource utilization then we got that RC6 utilize 14.82 MB extra memory compared to Twofish and 32.32 MB extra memory compared to Rijndael, while CPU utilization is approximately same for all these three algorithms.

6. CONCLUSION

In this research paper RC6, Twofish and Rijndael 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 RC6, Twofish and Rijndael have been done with a set of input files and evaluated the encryption & decryption time. Results conclude that RC6 is 1.182 times faster than Twofish and 1.191 times faster than Rijndael for 16-bytes key, 1.184 times faster than Twofish and 1.209 times faster than Rijndael for 24-bytes key, 1.195 times faster than Twofish and 1.236 times faster than Rijndael for 32-bytes key. Result also concludes that performance of all these three algorithms is inversely proportional to keysize, if keysize will increase the performance will decrease and vice-versa.

In resource utilization point of view, RC6 utilize 14.82 MB extra memory compared to Twofish and 32.32 MB extra memory compared to Rijndael, while CPU utilization is approximately same for all these three algorithms. So RC6 block cipher algorithm is faster and simpler than Twofish & Rijndael block cipher algorithms. Using RC6 is beneficial where high encryption rate is required while Rijndael is beneficial where memory is much concern.

7. REFERENCES

[1] "Report on the Development of the Advanced Encryption Standard (AES).", "csrc.net". Available at: http://csrc.nist.gov/encryption/aes/round2/r2report.pdf

[2] Ronald L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L.Yin, The RC6 ™ Block Cipher , M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, Version 1.1 - August 20, 1998. Available at: http://people.csail.mit.edu/rivest/Rc6.pdf

[3] Bruce Schneier, John Kelsey, Doug Whiting ,David Wagner, Chris Hall, Niels Ferguson, "Twofish: A 128-Bit Block Cipher", 1998, [online] Available at: http://www.certainkey.com/resources/article/twofish.pdf

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

[5] "RC6® Block Cipher", "rsa.com". Available at: http://www.rsa.com/rsalabs/node.asp?id=2512

[6] "RC6", "wikipedia.org". Available at: http://en.wikipedia.org/wiki/RC6

[7] R. Merkle." Fast software encryption functions". In A.J. Menezes and S.A.Vanstone, editors, Advances in Cryptology - CRYPTO'90, LNCS 537, pp. 476~501. Springer Verlag, 1991.

[8] J. Daemen, L. Knudsen, and V. Rijmen. "The block cipher Square". In E. Bi-ham, editor, Fast Software Encryption, Fourth International Workshop, Haifa, Israel, January 1997, LNCS 1267, pp. 149~165. Springer Verlag, 1997.

[9] J.L. Massey." SAFER K-64: A byte-oriented block-ciphering algorithm". In R. Anderson, editor, Fast Software Encryption - Proc. Cambridge Security Workshop, Cambridge, U.K., LNCS 809, pp. 1~17. Springer Verlag, 1994.

[10] 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, pp. 1545-1554, 1975.

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

[12] Shun-Lung Su, Lih-Chyau Wuu, and Jhih-Wei Jhang, "A New 256-bits Block Cipher –Twofish256", ISBN: 978-1-4244-1365-2, 07 February 2008. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4447043

[13] F.J. MacWilliams and N.J.A. Sloane, "The Theory of Error-Correcting Codes", North-Holland, Amsterdam, 1977.

[14] "Advanced Encryption Standard", "wikipedia.org", [online] Available at: http:// http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[15] Abdul Hamid M. Ragab, Nabil A. Ismail, Senior Member IEEE, and Osama S. Farag Allah, "Enhancements and Implementation of RC6™ Block Cipher for Data Security", IEEE Catalogue No. 01 CH37239-0-7803-7101-1/01 © 2001 IEEE.

[16] Fei Shao, Zinan Chang, Yi Zhang, "AES Encryption Algorithm Based on the High Performance Computing of GPU", IEEE, ISBN: 978-1-4244-5726-7, February 26-28, 2010.

[17] Parikh C., Patel P., "Performance Evaluation of AES Algorithm on Various Development Platforms", IEEE, ISBN: 978-1-4244-1109-2, June 22-23, 2007.