www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume – 5 Issue -03 March, 2016 Page No. 16050-16057

File Encryption System Based on Symmetric Key Cryptography

Ajay Kumar, Ankit Kumar
Department of Computer Science
Shivaji College, University of Delhi, Delhi, India
ajay.cs@shivaji.du.ac.in

Department of Computer Science
Shivaji College, University of Delhi, Delhi, India
ankit491995@gmail.com

***Abstract*—In Today's scenario, files are not secure. They are fetch by any means of attack by eavesdropper like cracking the pins, crashing the OS by viruses, malwares, and plenty of ways. We today can't sure that files protection wizards are secure and data can't be reached to the attacker. But if files are encrypted then even files are accessed original data remains confidential. Therefore, this paper represents the File Encryption System based on Symmetric Key Cryptography. I proposed the strategy to encrypt the files/even multiple files can be encrypted by compressing the files into one 'rar' file and it uses Blowfish as encryption/decryption standard and Cipher Block Chain Mode to perform the operations. I implemented compression function for 64-bit Initialization Vector(IV), use CBC mode with Blowfish and RC4 for 256-bit keystream. It is more efficient and secure then other general encryption process.**

***Keywords*—**Blowfish, RC4, SHA1, CBC, Plaintext, Ciphertext, Feistel, eavesdropper, IV

I. INTRODUCTION

Today, files are stored in a digitalized manner by scanning its content and making it secure in person's computer or in other storage devices. But those files can be accessed by theft by any means and the confidential and private data can be explored by the theft. So, in order to make the data more confidential, it is best to encrypt the files and store them in your storage devices, as those files data are useless for the theft even extracted. Only the user having that encrypted file key can decrypt the file and can explore the original content of the file. Original data remains original and even the attacker tries to decrypt the encrypted file, but without its key it is useless, and decryption of file can't be done.

In the File Encryption System, there are four phases – [1] Key Generation, [2] Initialization Vector (IV) generation, [3] Blowfish Encryption Algorithm [4] Cipher Block Chain Mode. In Key Generation, first random seed is generated by the mouse cursor position, process ID and system current time to have a 32-bit key and it is passed as an input for the RC4 (Rivest Cipher 4). The RC4 is a stream cipher, that generates key-stream of 256-bit key. Then, the 256-bit key is provided as an input to the SHA-1 (Secure Hash Algorithm 1) which generates an infeasible 160-bit message digest. The value evaluated is being compressed by a compressed function to bind 160-bit message digest into 64-bit Initialization vector (IV) generation. After the 64-bit IV and 256-bit random key produced, Blowfish Algorithm is placed in Cipher Block Chain (CBC) Mode. Then the IV generated is XORed with the 64-bit block of Plaintext (source file data in bit blocks for generating cipher block) and the value evaluated is passed as a Plaintext to the Blowfish Algorithm. After Algorithm finished its process, it generates 64-bit cipher text (target encrypted file data in bit

Algorithm	**Developed By**	**Key Size (in bits)**	**Number of Rounds**	**Mathematical Operations**	**Applications**	**Block Size**
DES		56	16	XOR, fixed Sboxes	SET, Kerberos	64
Triple DES	Tuchman	112 168	48	XOR, fixed Sboxes	Financial key management, PGP, S/MIME	64
IDEA	Xuejia Lai and James Massey	128	8	XOR, addition, multiplication	PGP	
Blowfish	Bruce Schneir	variable to 448	16	XOR, variable S-boxes, addition		64
RC5	Ron Rivest	variable to 2048	Variable to 255	Addition, subtraction, XOR, rotation		
CAST-128	Carlisle Adamsan Stafford Tavares	40 to 128	16	Addition, subtraction, XOR, rotation, fixed S-boxes	PGP	
AES	Dr. Joan Daemen and Dr. Vincent Rijmen	128, 192, 256	10	Substitution, Shift Row, Mix Column, Add round key		128

Table I

blocks) and the chain repeat itself by replacing IV with cipher text evaluated in previous iteration till all blocks are processed. The Encryption Algorithm used is a Block Cipher Algorithm where data is processed in block of bits. It follows Feistel Cipher structure where Plaintext bits are divided into two equal halves and there are rounds to be followed in which permutation of bits by Substitution, XOR, addition, multiplication etc. are performed with keys of different length as shown in Table I.

In the Application Interface, user can login/register himself and after providing login details he can use Encryption/Decryption of the file.

Key points for the interface to be used are as follows: -

1. User need to secure its secret key by moving the key at secure place from the File Encryption System Folder after encryption is done
2. File Encryption System Folder have the files used by the Application interface need to be copied at D drive due to the operating system protection to the C: Drive where amendment of files is denied for the third party Software
3. Just encrypt your file and go to the File Encryption Folder then move the key and save it in the secure place if want to decrypt your file then copy your key to the File Encryption Folder with the name as Secret_key and do Decryption

II. BLOWFISH ALGORITHM

Bruce Schneier designed blowfish in 1993 . As a fast, free alternative to existing encryption algorithms. Since then it has been analyzed considerably, and it is slowly gaining acceptance as a strong encryption algorithm. Blowfish is a secret-key block cipher algorithm, unlike Rajendial, it is still based on Feistel model. Taking the weakness of the existing encryption algorithm into consideration, blowfish has good resistance to differential attack. The algorithm has the following properties: (1) fast, using 32bit microprocessor, encrypt 1Byte data just take 18 clock cycles; (2) compact, easy to implement and determine the strength of algorithm; (3) variable security level, a variable key length of at most 448 bits, which makes the user can trade-off between security and speed. With high speed and reliable security standard, blowfish has a good performance on the file-encryption application. The algorithm consists mainly of two parts: the key-expansion part and the data-encryption part.

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.

In this Figure 1 64-bit plaintext divided into 32-bits. The algorithm keeps two subkeys arrays: the 18-entry P-array and four 256-entry Substitution boxes. The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used in every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P entries.

The Figure 2 shows Blowfish's Feistel-function. The function splits the 32-bit input into four eight bit quarters, and uses 8-bits as input to the S-boxes. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output.

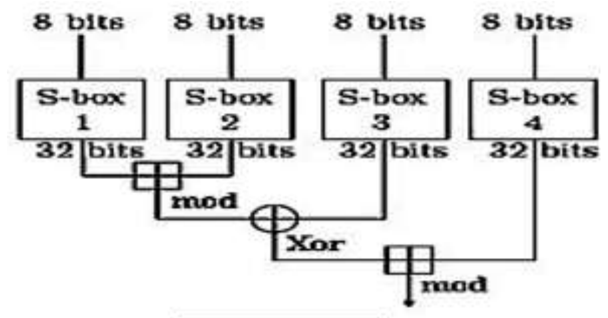


Figure 2

(A) *Blowfish Implementation*

1) *Blowfish Initialization*

Blowfish's key schedule starts by: -

- The P-array and S-boxes are initialized with the hexadecimal digits of π which are non-repeating and non-recurring.
- The secret key is byte by byte, cycling the key XORed with all the P-entries in order. A 64-bit all zero-block is then encrypted with the algorithm.
- The ciphertext evaluated replaces P_1 and P_2. The ciphertext is then encrypted again with new subkeys and new ciphertext replaces P_3 and P_4. This continues, replaces all the entire P-array and the Substitution box entries.
- The Algorithm will run 521 times to generate all the subkeys and about 4KB of data is processed.
- As the P-array is 576 bits long (32 x 18), and the key bytes are XORed through all these 576 bits during initialization, many implementation support key sizes upto 576 bits.

- The 448 bits' limit is present to ensure that every bit of every subkey depends on every bit of the key, as P-array last four values doesn't affect every bit of the ciphertext.

```
for i = 1 to 18
$P_i = P_i \oplus K_{i \bmod keylength}$
Endfor
$X_L = 0$ and $X_R = 0$
for i = 1 to 18
encrypt($X_L$,$X_R$)
```

```
    $P_i = X_L$ $P_{i+1} = X_R$
    Endfor
    for i = 1 to 4
    for j = 1 to 256
    encrypt($X_L$,$X_R$)
    $S_{i, j} = X_L$ $S_{i, j+1} = X_R$
    Endfor
    Endfor
```

2) *Feistel Function*

```
f(x: 32 bit integer unsigned value)
h: unsigned integer value
h = S[0][x >> 24] + S[1][x >> 16 & 0xff]
return (h ⊕ S[2][x >> 8 & 0xff ] ) + S[3][ x & 0xff ]
```

3) *Blowfish Encryption/Decryption*

```
At Encryption
for i = 1 to 16
$X_L = X_L \oplus P_i$
$X_R = X_R \oplus f(X_L)$
$X_R = X_R \oplus P_{i+1}$
$X_L = X_L \oplus f(X_R)$
Endfor
$X_L = X_L \oplus P_{17}$
$X_R = X_R \oplus P_{18}$
Swap($X_L$, $X_R$)
```

In **Decryption,** P_1, P_2, ……, P_{18} are used in the reverse order. Then on following reverse order of P-entries, it follows the same operation as follows in Encryption process.

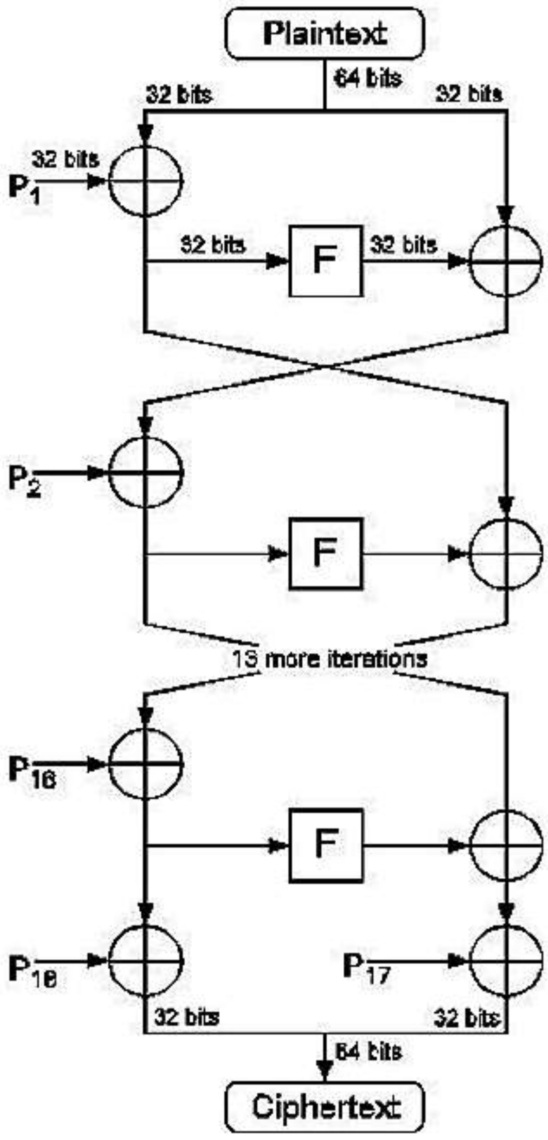


Figure 1

DOI: 10.18535/ijecs/v5i3.35

III. FILE ENCRYPTION SYSTEM DESIGN

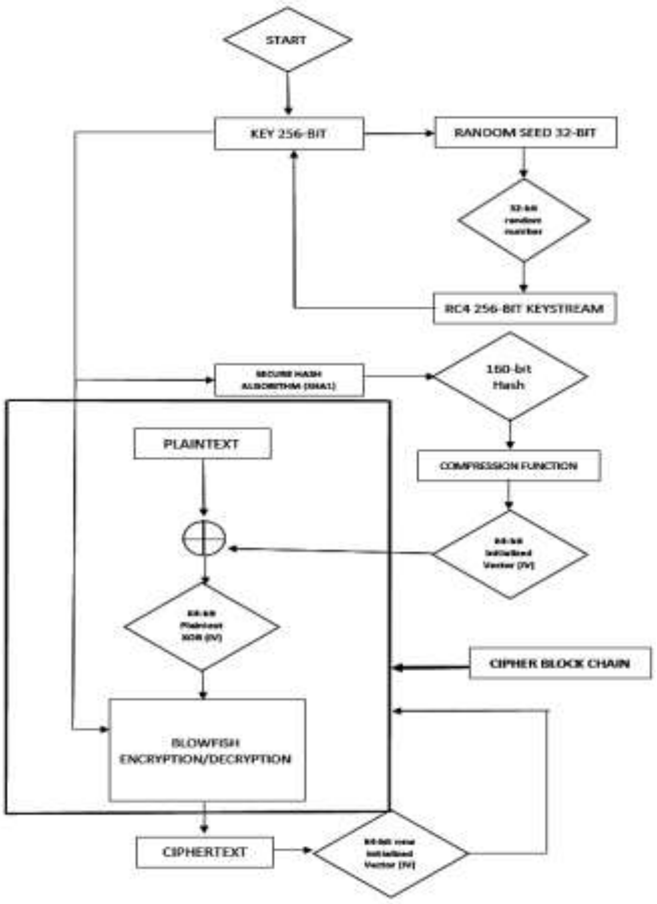

START
KEY 256-BIT
RANDOM SEED 32-BIT
32-bit random number
RC4 256-BIT KEYSTREAM
SECURE HASH ALGORITHM (SHA1)
160-bit Hash
PLAINTEXT
COMPRESSION FUNCTION
64-bit Initialized Vector (IV)
64-bit Plaintext XOR (IV)
CIPHER BLOCK CHAIN
BLOWFISH ENCRYPTION/DECRYPTION
CIPHERTEXT
64-bit new Initialized Vector (IV)

Figure 3 System Structure

(A) *Random Seed (32 bit)*

System Current time, processor id and current mouse cursor position are XORed and input to the srand() function and 32-bit random seed is generated.

This seed is passed as a plaintext to the RC4.

(B) *RC4 Implementation*

1) Key-scheduling algorithm (KSA)

The KSA is used to initialize the permutation in the array "S"-"keylength" is defined as the number of bytes in the key and can be in the range [1,256]. [1] The array "S" is initialized to the identity permutation. S is then processed for 256 iterations in a similar way to the main PRGA.

```
For i from 0 to 255
    S[i] : = i
endfor
j : = 0
for i from 0 to 255
    j : = (j+S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

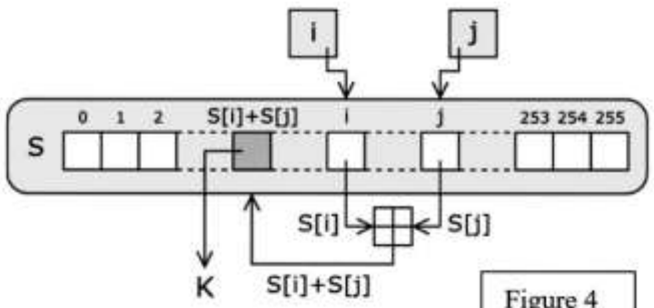


Figure 4

Refer Fig 4 for more details

2) Pseudo-random generation algorithm (PRGA)

The Algorithm modifies the state and outputs a byte of the keystream. In each iteration, (1) the PRGA increments i, looks up the ith element of S, S[i], (2) adds that to j, exchanges the values of S[i] and S[j], (3) uses the sum of S[i] + S[j] (modulo 256) as an index to fetch a third element of S, (the keystream value K below) which is XORed with the next byte of either ciphertext or plaintext. (4) Each element of S is swapped with another element at least once every 256 iterations.

```
i : = 0
j : = 0
while GeneratingOutput:
    i : = (i +1) mod 256
    j : = (j + S[i]) mod 256
    Swap values of S[i] and S[j]
    k : = S[( S[i]+S[j]) mod 256]
    Output k
Endwhile
```

(C) *Secure Hash Algorithm (SHA1) Implementation*

First, the message is padded so that it is a multiple of 512 bits long. It is done as: first append a 1, then as many 0's as necessary to make it 64 bits short of a multiple of 512, and finally a 64-bit representation of the length of the message before padding as shown in figure 5.

Five 32-bit variables are initializing as follows:

A = 67 45 23 01 B = EF CD AB 89

C = 98 BA DC FE

D = C3 D2 E1 F0

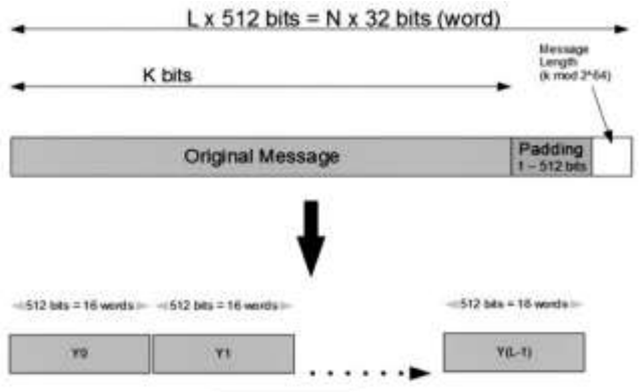


Figure 5

The main loop of the algorithm then begins. It processes the message 512 bits at a time and continues as many 512-bit block as are in the message First the five variables are copied into different variables: A as AA, B as BB, C as CC, D as DD, and E as EE.

The main loop has four rounds of 20 operations each. Each operations performs a nonlinear operation on three of A, B, C, and D and then does shifting and adding. SHA's set of nonlinear function is:

$F_t(X,Y,Z)$ = XY OR (NOT X)Z, for the first 20 operations.

$F_t(X,Y,Z)$ = X XOR Y XOR Z, for the second 20 operations.

$F_t(X,Y,Z)$ = XY OR XZ OR YZ, for the third 20 operations.

$F_t(X,Y,Z)$ = X XOR Y XOR Z, for the fourth 20 operations.

There are also four constants used in the algorithm:

K_t = 5A827999, for the first 20 operations.

K_t = 6ED9EBA1, for the second 20 operations.

K_t = 8F1BBCDC, for the third 20 operations.

K_t = CA62C1D1, for the fourth 20 operations.

The message block is transformed from sixteen 32-bit words (M_0 to M_{15}) to eighty 32-bit words (W_0 to W_{79}) using the following algorithms:

$W_t = M_t$, for t =0 to 15

$W_t = W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR W_{t-16}, for t =16 to 79

If t is the operation number (from 1 to 80), M_j represents the j^th sub-block of the message (from 0 to 15) and <<<s represents a left shift's bits, then the 80 operations look like:

TEMP = (A <<< 5) + Ft(B,C,D) + E + Wt + Kt

E = D

D = C

C = (B <<< 30)

B = A

A = TEMP

After all of this, A, B, C, D and E are added to AA, BB, CC, DD, and EE, respectively, and the algorithm continues with the next block of data. The final output is the concatenation of A, B, C, D, and E as shown in figure 6.

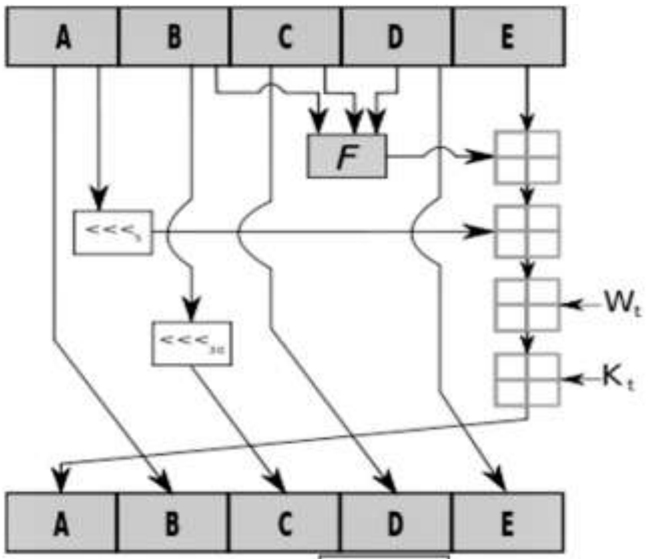


Figure 6

(D) *Compression Function*

160-bit message digest from SHA-1 is divided into two equal 80-bits data in increasing order of bit storage and these x1 and x2 are XORed and data bits stores in x1. Then 64-bit data is being write to the IV from x1.

(E) *Cipher Block Chain Mode (CBC) Implementation*

In the cipher block chaining (CBC) mode, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$Cj = E(K, |Cj-1 \oplus Pj|)$$

where E[K,X] is the encryption of the plaintext X using key K, and $\oplus$ is the exclusive OR operation. Then

$D(K,Cj) = D(K,E(K, |Cj-1 \oplus Pj|)$

$D (K,Cj) = Cj-1 \oplus Pj$

$Cj-1 \oplus D (K,Cj) = Cj-1 \oplus Cj-1 \oplus Pj = Pj$

Initialization Vector must be known to authentic user. For optimal security, the IV should be protected as well as the key. The reason for protecting the IV is as follows: If an opponent is able to recover into using a different value for IV, then the attacker is able to modifies the selected bits in the first block of plaintext. To see this, consider the following:

$C_1 = E(K, |IV \oplus P_1|)$

$P_1 = IV \oplus D(K, C_1)$

Now use the notation that X[j] denotes the jth bit of the bit quantity X. Then

$P_1[i] = IV[i] \oplus D(K, C_1)[i]$

Then, using the properties of XOR, we can state

$P_1[i]' = IV[i]' \oplus D(K, C_1)[i]$

The prime notation signifies bit complementation. This means that if an attacker can predictably modify bits in IV, the bits of the received value of P_1 can be changed.

The process of CBC is showed in Figure 7

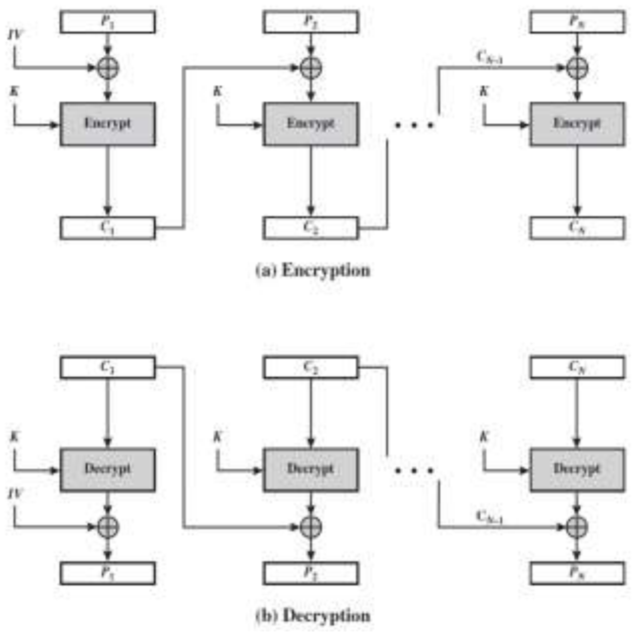


Figure 7

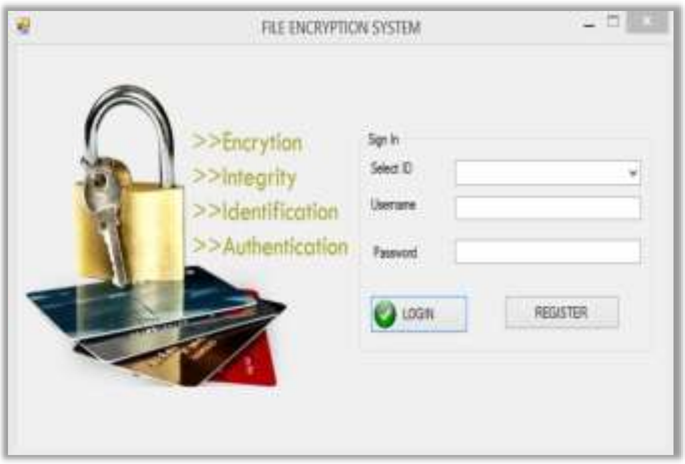
IV. TECHNIQUE/METHOD DEVELOPED

The Cipher Block Chain Mode Technique makes the Blowfish Algorithm more secured and Initialization vector (IV) in CBC mode acts as a secondary key to cipher text for its decryption and it is independent of the plaintext.

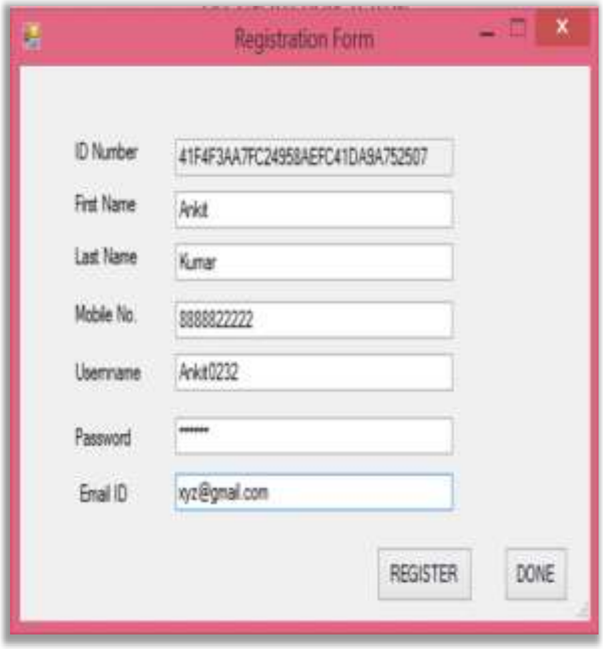
This increases the complexity of Blowfish Algorithm and makes our file more secure and less efficient for attacks.

Message digest generated by SHA1 make it infeasible to get the IV and the compression function to convert 160-bit message digest value into 64-bit IV hides the nature of the generated Initialization vector.

V. RESULT

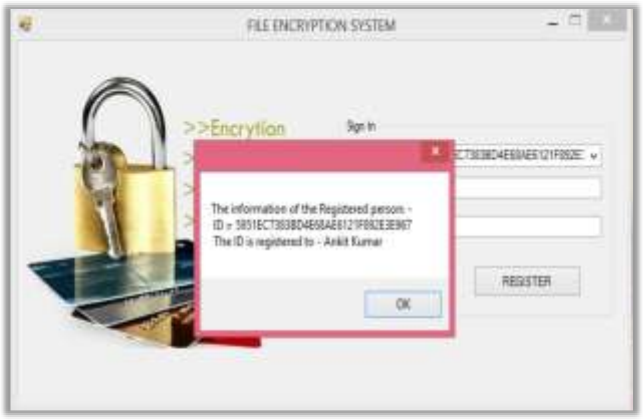


Result IMG 1.

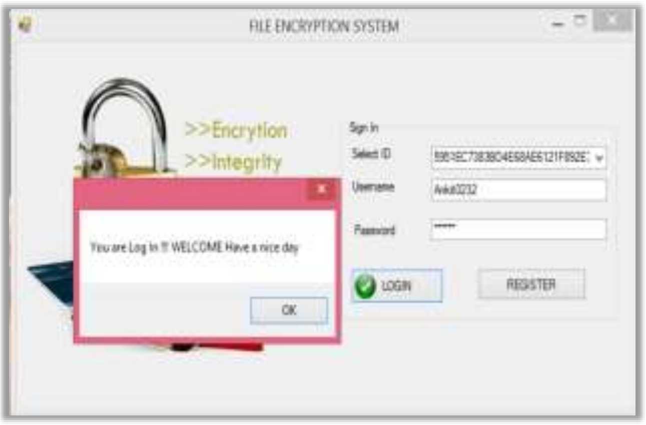

Registration Form

ID Number	41F4F3AA7FC24958AEFC41DA9A752507
First Name	Ankit
Last Name	Kumar
Mobile No.	8888822222
Username	Ankit0232
Password	*******
Email ID	xyz@gmail.com

Result IMG 2.


FILE ENCRYPTION SYSTEM

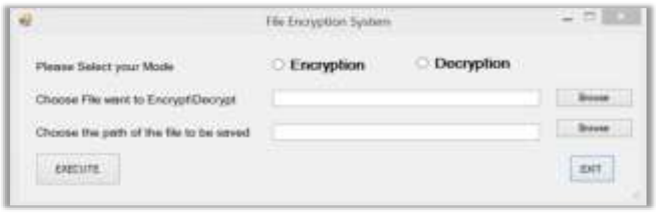
The information of the Registered person -
ID = 5851ECT3838D4E68AE6121F892E3E967
The ID is registered to - Ankit Kumar


FILE ENCRYPTION SYSTEM

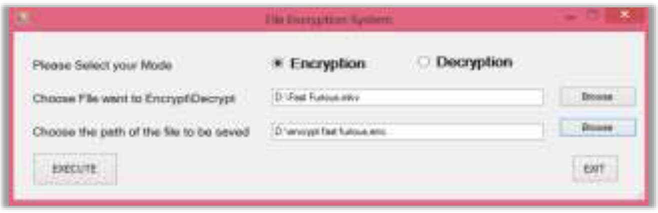
You are Log In !!! WELCOME Have a nice day

Select ID	5851ECT3838D4E68AE6121F892E
Username	Ankit0232
Password	*****

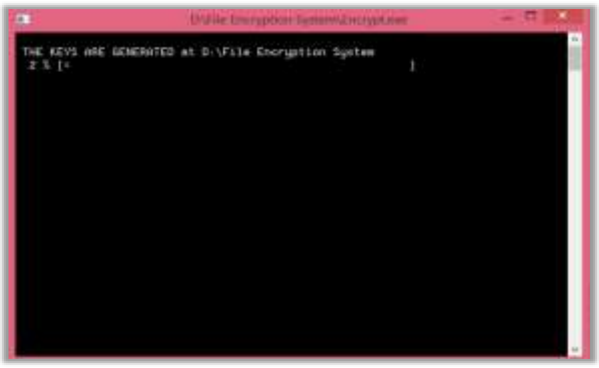
Result IMG 3.


File Encryption System

Please Select your Mode ○ Encryption ○ Decryption
Choose File went to Encrypt/Decrypt
Choose the path of the file to be saved


File Encryption System

Please Select your Mode [✓ Encryption] Decryption
Choose File went to Encrypt/Decrypt: D:\Fast Furious.mkv
Choose the path of the file to be saved: D:\encrypt Fast furious.enc


D:\File Encryption System\Encrypt.exe

```
THE KEYS ARE GENERATED at D:\File Encryption System
 2 % [=                                          ]
```

Result IMG 4.

DATA	*Encryption Time taken (in seconds)*	*Decryption Time Taken (in seconds)*
115 KB	0.63 s	0.65 s
1.04 MB	1.27 s	1.70 s
101 MB	98.42 s	103.75 s

Done in: - Intel i7 4710 HQ at 2.5 GHz

VI. CONCLUSION

The file remains secure till the attacker didn't get the key to decrypt it and the IV information which is appended at the first in the processing of File Encryption/Decryption. Even attacker fetches the algorithm but without the key and IV information, algorithm can't do nothing.

So, the key and IV must be secure enough to any means of attack by the attacker.

This File Encryption Standard can be used with multiple files encryption/decryption by collecting all files in a compressed format for example in rar, zip, iso etc.

DOI: 10.18535/ijecs/v5i3.35

Limitation:

For word documents, files corrupt, but can be recover by Document recovery dialog.

Key and IV must be secured by the authentic user

VII. FUTURE SCOPE

This application can be further deduced by implementing a key management system in order to make keys and Initialization vector secured enough so that files cannot be accessed by any means.

This can be implemented for external storage devices (like Flash Drives) as well as internal storage devices to get security by having protection mode with the device to open, share and view files in protected mode and in this way read and write of encrypted files are useless.

Also data sharing got secured in unsecure network channels and it makes the user to easily share important files and maintaining data integrity.

VIII. REFERENCES

[1] Applied Cryptography by Bruce Schneier
[2] Practical Cryptography by Neils Ferguson, Bruce Schneier
[3] Network Security Essentials: Applications and Standards by William Starlings
[4] Cryptography and Network Security- Principles and Practice, Fifth Edition by William Starlings
[5] Fundamentals of Cryptology by Henk C.A van Tilborg
[6] Cryptanalytic Attacks on Pseudorandom Number Generators by John Kelsey, Bruce Schneier, David Wagner, Chris Hall
[7] Implementing the RC4 Algorithm by Brian Whitley
[8] Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudo Number
[9] Generator by John Kelsey, Bruce Schneier and

Niels F