Performance enhancement of Blowfish and CAST-128 algorithms and Security analysis of improved Blowfish algorithm using Avalanche effect

Krishnamurthy G.N[†], Dr. V. Ramaswamy[†], Leela G.H[†] and Ashalatha M.E[†]

[†]Bapuji Institute of Engineering and Technology, Davangere-577004, Karnataka, India

Summary:
There has been a tremendous enhancement in the field of cryptography, which tries to manipulate the plaintext so that it becomes unreadable, less prone to hacker and crackers, and again obtain the plaintext back by manipulating this unreadable text in some way. In this regard, we have modified two secure algorithms Blowfish [1] and CAST-128 [5] which are secret-key block ciphers that enhance performance by modifying their function. We have shown that total time taken for encryption and decryption is reduced for both the algorithms after the modification. We have also made an attempt to show that this improvement will not violate the security when compared to that of existing Blowfish algorithm. For this purpose we have used avalanche effect [3] as the basis of security analysis. Because the change in the total time taken for encryption and decryption cannot be understood on software implementation, we have implemented VHDL application to show the differences in the delay.

Key words:

Plaintext; Ciphertext; Encryption; Decryption; Secret-key; Feistel-network; Avalanche-effect.

Chapter 1: Performance enhancement of Blowfish algorithm and its security analysis using Avalanche effect

1.1 Introduction

Blowfish[1] is a variable-length key[1], 64-bit block cipher. The algorithm 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[3] as shown in Figure 1.1. Each round consists of a key-dependent permutation, a key and data-dependent substitution. All operations are EX-ORs and additions on 32-bit words.

1.2 Subkeys

Blowfish uses a large number of subkeys[3]. These keys must be precomputed before any data encryption or decryption.

The key array also called 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.

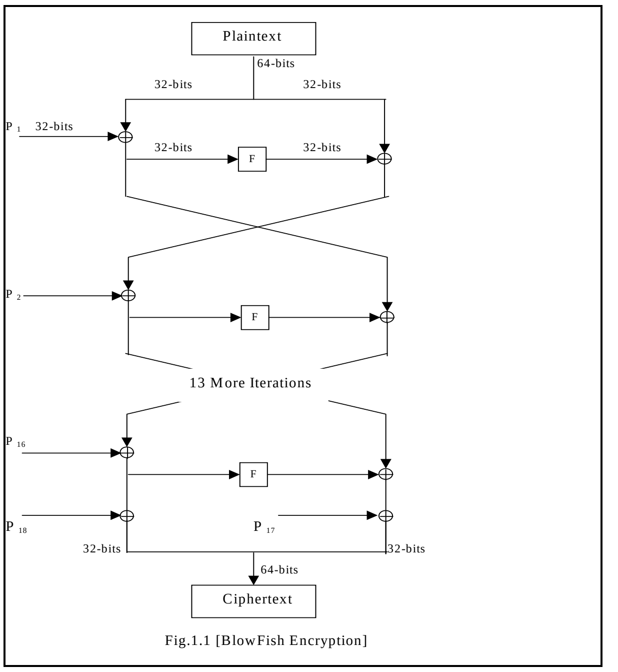


Fig.1.1 [BlowFish Encryption]

Decryption for Blowfish is relatively straightforward. Ironically, decryption works in the same algorithmic direction as encryption beginning with the ciphertext as input. However, as expected, the sub-keys are used in reverse order.

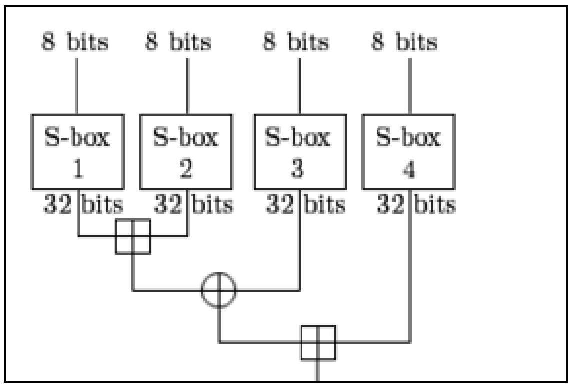
Manuscript received March 5, 2008
Manuscript revised March 20, 2008

Since Function F plays an important role in the algorithm, it was decided to modify function F and determine whether the modified function F` saves the time.

Original function F is defined as follows:-

Divide X_L into four eight-bit quarters: a, b, c, and d

$F(X_L) = ((S1,a + S2,b \bmod 2^{32}) \oplus S3,c) + S4,d \bmod 2^{32}$



32 bits

Fig.2. Existing Blowfish Function F

Thus modified Blowfish function F is:-

$F(X_L)=(S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus (S_{3,c} + S_{4,d} \bmod 2^{32})$

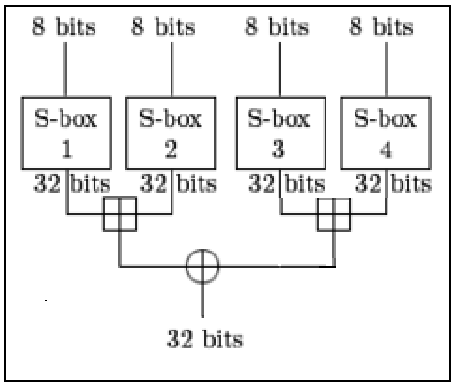
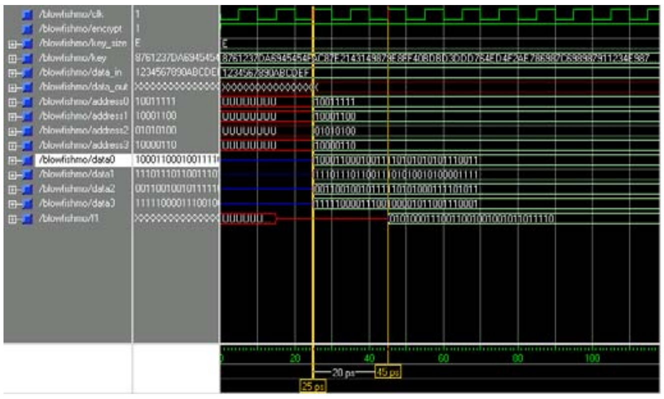


Fig..3. Modified Blowfish Function F

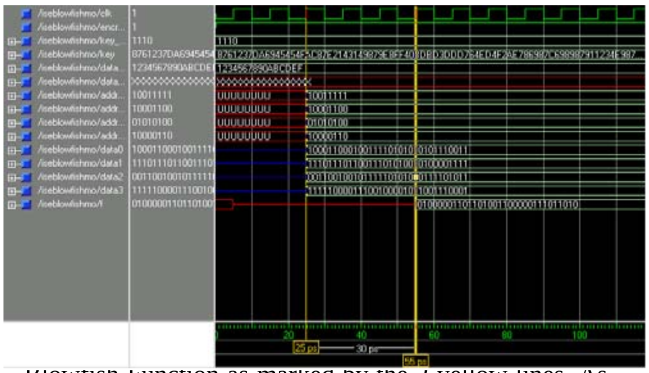
This modification supports the parallel evaluation of two addition operations ($S_{1,a} + S_{2,b} \bmod 2^{32}$) and ($S_{3,c} + S_{4,d} \bmod 2^{32}$) by using threads. But true parallelism cannot be achieved on a uniprocessor system. So this modified function can be best adopted for the hardware implementation of the algorithm. In the hardware implementation of the function F requires only two levels of computation, where as the original function F requires three levels of computation. The parallel evaluation which reduces the time has been experimentally verified using VHDL simulation. As the algorithm uses 16 iterations, this time is saved 16 times for every encryption/decryption. This is a considerable improvement.



```
/blowfishmo/clk        1
/blowfishmo/encrypt    1
/blowfishmo/key_size   E
/blowfishmo/key        8761237DA6945454
/blowfishmo/data_in    1234567890ABCDE
/blowfishmo/data_out   XXXXXXXXXXXXXXXX
/blowfishmo/address0   10011111
/blowfishmo/address1   10001100
/blowfishmo/address2   01010100
/blowfishmo/address3   10000110
/blowfishmo/data0      1000110001001111
/blowfishmo/data1      1110111011001110
/blowfishmo/data2      0011001001011111
/blowfishmo/data3      1111100001110010
/blowfishmo/f1         XXXXXXXXXXXXXXXX
```

Fig. 1.4 Waveform for Existing Blowfish Function F

The above Simulation diagram (Figure 1.4) shows the time required to execute the Function F of the existing Blowfish Function as marked by the 2 yellow lines. As per the result it is taking 55ps - 25ps = 30ps.



```
/iseblowfishmo/clk      1
/iseblowfishmo/encr...  1
/iseblowfishmo/key_...  1110
/iseblowfishmo/key      8761237DA6945454
/iseblowfishmo/data...  1234567890ABCDE
/iseblowfishmo/data...  XXXXXXXXXXXXXXXX
/iseblowfishmo/addr...  10011111
/iseblowfishmo/addr...  10001100
/iseblowfishmo/addr...  01010100
/iseblowfishmo/addr...  10000110
/iseblowfishmo/data0    1000110001001111
/iseblowfishmo/data1    1110111011001110
/iseblowfishmo/data2    0011001001011111
/iseblowfishmo/data3    1111100001110010
/iseblowfishmo/f        0100000110110100
```

Blowfish Function as marked by the 2 yellow lines. As per the result it is taking 45ps - 25ps = 20ps.

1.3 Security Analysis:

A change in one bit of the plain text or one bit of the key should produce a change in many bits of the ciphertext. This change in number of bits in the cipher text whenever there is a change in one bit of the plain text or one bit of key is called **Avalanche effect**.

A desirable feature of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext.

If the changes are small, this might provide a way to reduce the size of the plaintext or key space to be searched and hence makes the cryptanalysis very easy.

So, in order to say that any cryptographic algorithm is secure, it should exhibit strong avalanche effect, and this is the reason why we have considered Avalanche effect for comparing security of our modified algorithm with that of original Blowfish algorithm.

1.4 Implementation

We have taken 300 samples each for the original algorithm and modified algorithm and noted down the Avalanche effect by changing the plain text by one bit between the successive samples. The results observed in security analysis are shown below.

Tabulation of results observed by changing one bit of plaintext in the successive samples is shown in TABLE I

TABLE I : Comparison of avalanche effect for Original and modified Blowfish algorithms

No. of samples	No. of rounds	No. of times the original algorithm gives better avalanche effect	No. of times the modified algorithm gives better avalanche effect	No. of times the original and modified algorithms give same avalanche effect
100	4	41	55	4
100	8	41	46	13
100	16	41	50	9

Chapter 2: Performance enhancement of CAST-128 algorithm

2.1 Introduction

CAST-128[3] is a design procedure for symmetric encryption algorithm developed by Carlisle Adams and Stafford Tavares. CAST has a classical Feistel network with 16 rounds. It operates 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.

Here, we have tried to improve the existing CAST-128 algorithm by modifying its function(F) by parallel evaluation of two operations. The parallel execution is efficient in processing, such that it requires only 66.66% of the time required for the original function. As the algorithm uses 16 iterations, this time is saved 16 times for every encryption /decryption.

CAST-128 employs two sub keys in each round namely a 32-bit masking sub key (Kmi) and a 5-bit rotate sub key (Kri). The function F depends on the round. It has the structure of classical Fiestel network with 16 rounds of operation. It is a variable-length key, 64-bit block cipher. The algorithm consists of two parts: a sub key generation part and a data- encryption part. The algorithm uses four primitive operations**,** Addition (+) and subtraction (-) using modulo 2^{32} arithmetic, Bitwise ex-OR (^) and Left Circular Rotation (<<<).

2.2 Encryption

CAST-128 is a Feistel network consisting of 16 rounds (Fig. 1.6). The input is a 64-bit data element. The plaintext is divided into two 32-bit halves: L0 and R0

We use variables Li and Ri to refer to the left and right of the data after round i is completed. The cipher text is formed by swapping the output of the sixteenth round, in other words, the cipher text is a concatenation of R16 and L16.

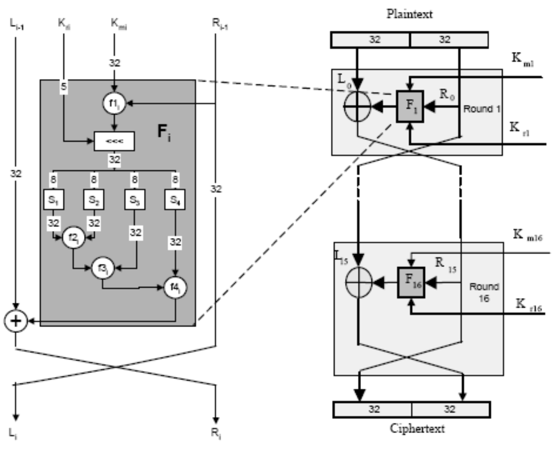


Fig. 1.6. CAST-128 Original Encryption Scheme.

```
L0 || R0 = Plaintext
For i = 1 to 16 do
Li = Ri-1;
Ri = Li-1 XOR Fi[Ri-1,Kmi,Kri];
Ciphertext = R16 || L16
```

The function F includes the use of four 8 * 32 S boxes, the left circular rotation function, and four functions that vary depending on the round number. We label these functions as f1i, f2i, f3i and f4i.

We use I to refer to the intermediate 32-bit value after the left circular rotation function, and the labels Ia, Ib, Ic and Id to refer to the 4 bytes of I, where Ia is the most significant and Id is the least significant byte. With these conventions function F is defined as shown in TABLE II:

TABLE II: Definition of CAST-128 original function

Rounds 1,4,7,10,13,16	I = ((Kmi + Ri-1) <<< Kri)
	F = ((S1[Ia] ^ S2[Ib]) – (S3[Ic]))+ S4[Id]

Rounds 2,5,8,11,14	I = ((Kmi ^ Ri-1) <<< Kri)
	F = ((S1[Ia] - S2[Ib]) + (S3[Ic])) ^ S4[Id]

Rounds 3,6,9,12,15	I = ((Kmi - Ri-1) <<< Kri)
	F = ((S1[Ia] + S2[Ib]) ^ (S3[Ic]))- S4[Id]

2.3 Substitution Boxes

There are eight 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];
S5[0], S5[1],….,S5[255]; S6[0],S6[1],.....,S6[255];
S7[0],S7[1],.….,S7[255]; S8[0],S8[1],.....,S8[255];

Four of these namely S-box1 through S-box4 are used in encryption and decryption process. The remaining four namely S-box5 through S-box8 are used in sub key generation. Each S-box is an array of 32 columns by 256 rows. The 8-bit input selects a row in the array; the 32-bit value in that row is the output. All of the S-boxes contain fixed values.

2.4 Generating the Sub keys

Sub key generation is a complex process. To begin, label the bytes of the 128- bit key as follows:
x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF

Here x0 represents the most significant byte and xF represents the least significant byte.

Also use the following definitions:

- Km1,…..,Km16 Sixteen 32-bit masking sub keys.
- Kr1,…....,Kr16 Sixteen 32-bit rotate sub keys of which only the least significant 5-bits of each are used.
- z0…….zF Intermediate (temporary) bytes.
- K1……K32 Intermediate (temporary) 32-bit words.

The values K1 through K32 are calculated from the key using S-boxes 5 through 8.

Then the sub keys are defined by:

```
for i =1 to 16 do
Kmi = Ki;
  Kri = K16+i;
```

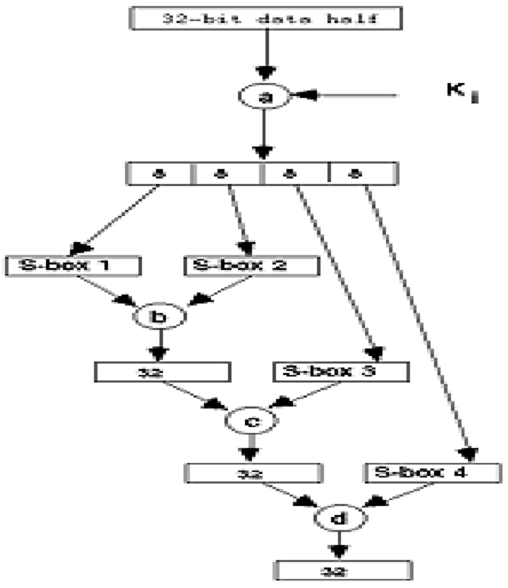


Fig. 1.7. CAST-128 Original F function.

2.5 Decryption

Decryption for CAST-128[4] is relatively straightforward. Ironically, decryption works in the same algorithmic direction as encryption beginning with the ciphertext as input. However, as expected, the sub-keys are used in reverse.

2.6 Function F

The function F(Fig. 1.7) is designed to have good confusion, diffusion and avalanche properties. It uses S-box substitutions, modulo 2^{32} addition and subtraction, exclusive OR operations and key dependent rotation. The strength of the F function is based primarily on the strength of the S boxes, but further use of arithmetic, boolean and rotate operations add to its strength.

2.7 Proposed Modification

Without violating the security requirements, the CAST-128 function F can be modified as shown in TABLE III below.

TABLE III: Definition of original function

Rounds 1,4,7,10,13,16	I = ((Kmi + Ri-1) <<< Kri) F` = (S1[Ia] ^ S2[Ib]) – (S3[Ic] + S4[Id])
Rounds 2,5,8,11,14	I = ((Kmi ^ Ri-1) <<< Kri) F`= (S1[Ia] - S2[Ib]) + (S3[Ic] ^ S4[Id])
Rounds 3,6,9,12,15	I = ((Kmi - Ri-1) <<< Kri) F`= (S1[Ia] + S2[Ib]) ^ (S3[Ic] - S4[Id])

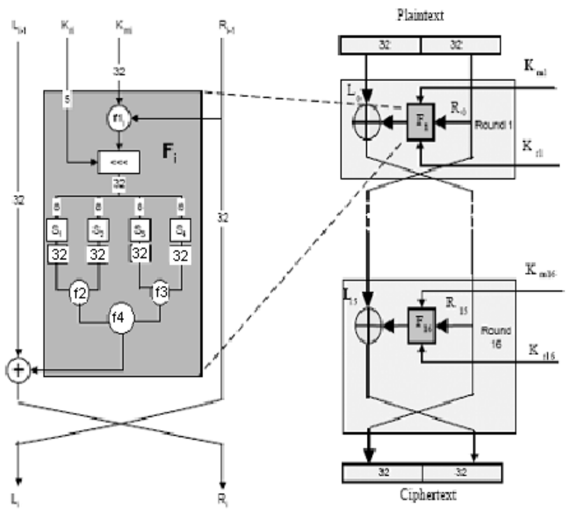


Fig. 1.8 CAST-128 Modified Encryption Scheme.

This modification (Fig. 1.8) supports the parallel evaluation of two operations by using threads. The parallel evaluation reduces the time from two operations to time required for one operation. As the algorithm uses 16 iterations, this time is saved 16 times for every encryption/decryption. This is a considerable improvement. Also, as the security of CAST-128 lies in the fact that it uses variable key, this modification does not make the algorithm vulnerable in any way so that cryptanalysis becomes easy.

But true parallelism cannot be achieved on a uniprocessor system. So the effect of the modification can be seen only in multiprocessor system, with at least two processors. So this modified function can be best adopted for the hardware implementation of the algorithm. In the hardware implementation the modified function F (Fig. 1.9) requires only two levels of computation, where as the original function F (Fig. 1.7) required three levels of computation.

The above modification does not require any change to be made in the original algorithm. The original algorithm works fine with the modified function for both encryption as well as decryption.

The same four primitives namely addition, subtraction, bitwise exclusive OR and left circular rotation operations are used in the modified function.

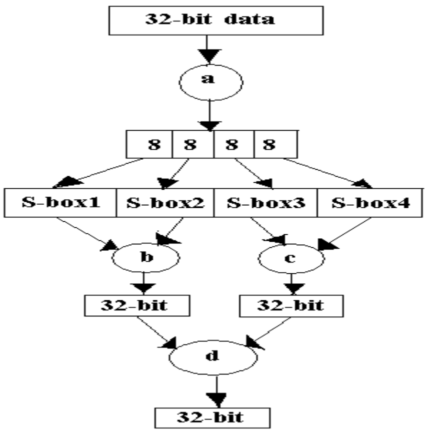


Fig. 1.9. CAST-128 Modified F function.

The generation of sub keys is done in the same way as done in the existing CAST-128 Algorithm. i.e., similar 8 S-boxes with 256 entries each are used to generate the sub keys.

2.8 Sample Waveforms And Results Analysis

Following simulation diagram (Fig. 2.0) shows the time required to execute the Function F of the existing CAST-128 function as marked by the 2 yellow lines. It is taking 55ps - 5ps = 50ps

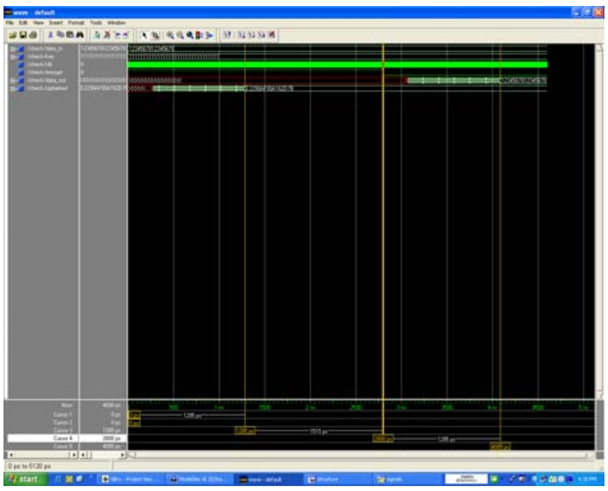


Fig. 2.0. Waveform for CAST-128 Original Function F

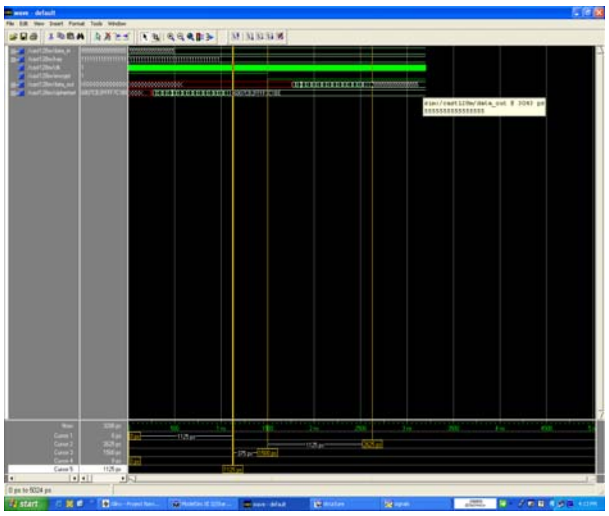


Fig. 2.1. Waveform for CAST-128 Modified Function F

Above simulation diagram (Fig. 2.1) for the time required to execute the modified Function F of the CAST-128 as marked by the 2 yellow lines. It is taking 45ps - 5ps = 40ps.

The ratio of time taken between CAST-128 with modified and existing Function=40/50=0.8; Hence we have 20% improvement in the performance.

Conclusion

The improved modified algorithm has enhanced the performance over existing blowfish algorithm by reducing the number of clock cycles required for the execution of Blowfish function by 33% and hence reducing the overall execution time of the modified Blowfish algorithm by 14%. This is explained in detail in the Appendix along with sample waveforms. We have demonstrated that change in one bit in the plaintext produces strong avalanche effect. Hence security of modified algorithm is at least as strong as the original algorithm. We are now trying to theoretically prove this fact. Also we are studying the effects when one bit of the key is changed.

It is also found that the improved modified algorithm increases the performance of CAST-128 algorithm by reducing total time from 1285 ps to 1125 ps. Using VHDL implementation, it is observed that the reduction in time achieved for encryption and decryption is above 12.5 % compared to the existing algorithm.

References

[1] *B. Schneier*, "Description o f a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)", *Fast Software Encryption, Cambridge Security Workshop roceedings (December 1993)*, Springer-Verlag, 1994, pp. 191-204.

[2] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed., John Wiley & Sons, 1995.

[3] W. Stallings, Cryptography and Network Security: Principles and Practices, 2nd ed., Prentice Hall, 1999.

[4] Kishnamurthy G.N, Dr.V.Ramaswamy and Mrs. Leela.G.H "Performance Enhancement of Blowfish algorithm by modifying its function" Proceedings of International Conference on Computers, Information, System Sciences and Engineering 2006, University of Bridgeport, Bridgeport, CT, USA. pp. 240-244

[5] Adams, C. The CAST-128 Encryption Algorithm. RFC 2144, May 1997.

[6] Anne Canteaut(Editor) "Ongoing Research Areas in Symmetric Cryptography" ECRYPT, 2006.

[7] Dr.V.Ramaswamy, Kishnamurthy.G.N, Mrs. Leela.G.H, Ashalatha M.E, "Performance enhancement of CAST –128 Algorithm by modifying its function" Proceedings of International Conference on Computers, Information, System Sciences and Engineering 2007, University of Bridgeport, Bridgeport, CT, USA.

[8] Lausanne, Statistical Cryptanalysis of Block Ciphers, Doctoral Thesis, EPFL, 2005.

[9] Orr Dunkelman, Techniques for Cryptanalysis of Block Ciphers, Doctoral Thesis, Haifa, 2006

[10] L. Knudsen, "Block Ciphers: A Survey", State of the Art in Applied Cryptography: Course on Computer Security and Industrial Cryptography (Lecture Notes in Computer Science no. 1528), Springer-Verlag, pp. 18-48, 1998.

Krishnamurthy G N received the B.E. degree in Electronics & Communication Engineering from Kuvempu University in 1996 and M.Tech. degree in Computer Science & Engineering from Visveswaraya technological University in 2000. He has registered for his Ph.D and has published papers in national and international conference, journals in the area of Cryptography. After working as a lecturer (from 1997) he has been promoted to Assistant Professor (from 2005), in the Department of Information Science & Engineering, Bapuji Institute of Engineering & Technology, Davangere, affiliated to Visveswaraya Technological University, Bel;gaum. His area of interest includes Design and analysis of Block ciphers, He is a life member of ISTE, India.

Dr. V Ramaswamy Aobtained his Ph.D degree from Madras University, in 1982, He is working as Professor and Head in the Department of Information Science and Engineering. He has more the 25 years of teaching experience including his four yers of service in Malaysia. He is guiding many research scholars and has published many papers in national and international conference and in many international journals. He has visited many universities in USA and Malaysia.

Leela G.H. received the B.E. degree in Electronics & Communication Engineering from Kuvempu University in 1994 and M.E. degree in Digital Electronics from Karnataka University in 1998. After working as a lecturer (from 1994) she has been a Assistant Professor (from 2007), in the Department of Electronics & Communication Engineering, Bapuji Institute of Engineering & Technology, Davangere, affiliated to Visveswaraya Technological University, Bel;gaum. Her area of interest includes VLSI Design, FPGA Design and Cryptography. She is a member of ISTE, India.

M.E.Ashalatha received the B.E. degree in Electronics & Communication Engineering from Mysore University in 1987 and M.Tech degree in Industrial Electronics from Mysore University in 1990. After working as a lecturer (from 1990) and as Assistant Professor (from 1995), she has been a Professor (from 2007) in the Department of Electronics & Communication Engineering, Bapuji Institute of Engineering & Technology, Davangere, affiliated to Visveswaraya Technological University, Bel;gaum. Her area of interest includes VLSI Design, Cryptography and Embedded System Design. She is a member of ISTE and IE, India.