An Optimized Key Scheduling Algorithm for CAST -128 using dynamic substitution S-box

Rekha C, Krishnamurthy G.N

Abstract**:** *An optimized key scheduling algorithm for the 64-bit block cipher CAST-128 by using dynamically substituting of S-box. In this regard, an attempt has been made to modify key scheduling algorithm for generating subkeys of a secret-key block cipher which is CAST-128 algorithm so as to enhance performance by modifying the generation of subkeys using dynamic substitution of S-box. The CAST-128 uses four static substitution of S-box to generate subkeys, where as the proposed structure generates subkeys using substitution of S-box dynamically to provides the performance of CAST-128. The approach considers different security aspects and metrics evaluation for verification.*

Keywords **:** ***CAST-128, S-Box, Key scheduling algorithm.***

I. INTRODUCTION

CAST-128 [1] symmetric encryption/decryption algorithm developed as a classical feistel network along with 16 rounds. The cast -128 encryption algorithm operates on plaintext to produce ciphertext with 64-bit blocks using the key where the size of keys varies from 40 to 128 bits in 8-bit increments. In this paper, an attempt has been made to provide improvement over the existing CAST-128 algorithm by altering the key scheduling algorithm with substituting S-box dynamically.

CAST-128 has two subkeys in each round Km_i and Kr_i where Km_i is masking subkey which is a 32-bit length and Kr_i is a rotate subkey of 5-bit length. The CAST has structure of fiestel network operated on 64 bit blocks with 16 rounds and the function F as shown in Figure 1, using key, a variable-length key,. The algorithm has two parts: a data encryption part and a subkey generation part. CAST-128 utilizes four primitive operations**,** Bitwise XOR (^) and Left-Circular-Rotation (<<<), Addition (+) and subtraction (-) operations using mod 232 arithmetic.

Manuscript published on 30 September 2019
* Correspondence Author

Rekha C*, BNM institute of Technology, Dept. of Computer Science and Engg, Bangalore, rekhac1976@gmail.com

Krishnamurthy G.N, BNM institute of Technology, Dept. of Computer Science and Engg, Bangalore, krishnamurthy gn@hotmail.com

The input data element of CAST-128 is a 64-bit plaintext which can be taken into two halves with 32-bit each : L0 and R0 represented as variables Li, represented as the left data, and Ri represented as the right data after the completion of round i.

After the sixteenth round the output is swapped to produce cipher text, means, R16 and L16 is concatenated to produce cipher text.

Input is L0 and R0 of 32 bit each
(L0 || R0) = Plaintext is divided into 2 halves
For each value of i varies from 1 to 16 then
Ri-1 is stored in Li;
(Li-1) is XOR with Function Fi and stored in Ri (where Fi is defined as Fi(Ri-1,Kmi,Kri));
Ciphertext = concatenate output of last round i.e., (R16 || L16)
From above algorithm the function F, as shown in Figure 2, includes four S-boxes with 8 * 32-bit each, four functions, as $f1_i$, $f2_i$, $f3_i$, and $f4_i$, and the left-circular-rotation function that depends on the round number. After the left-circular-rotation function, the intermediate value of 32-bit, is divided into 4 8 bit values a, b, c, d which refer to the 4 bytes of 32-bit, which is taken to S-box for substitution. The function 'F' With these conventions is defined as

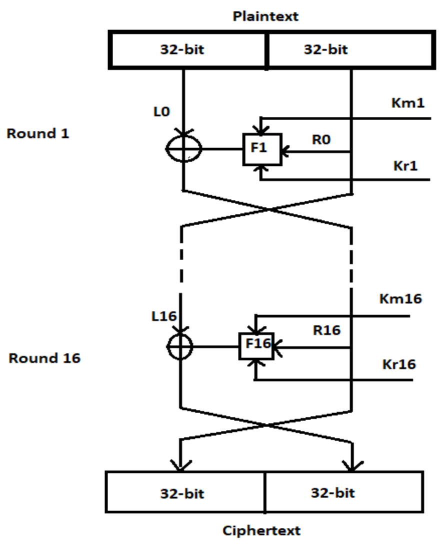

Figure 1 : CAST-128 encryption process.

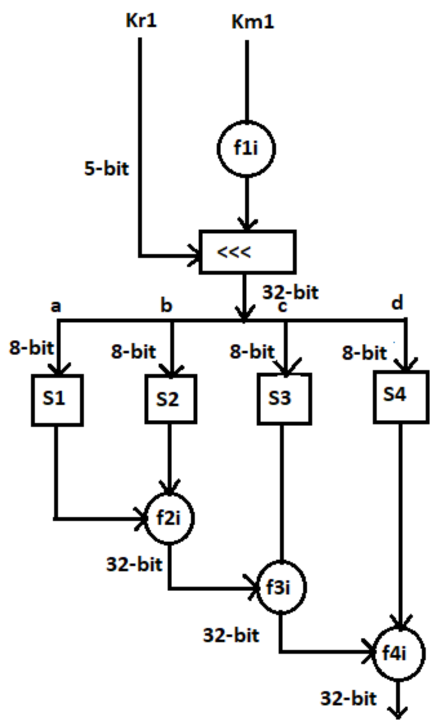

Figure 2 : Block Diagram of function F of CAST-128.

Rounds 1, 4, 7, 10, 13, 16 = ((Km_i + R_i-1) left shift to Kr_i times)

F = ((S1(a) ^ S2(b)) – (S3(c)))+ S4(d)

Rounds 2, 5, 8, 11, 14 = ((Km_i ^ R_i-1) left shift to Kr_i times)

F = ((S1(a) - S2(b)) + (S3(c))) ^ S4(d)

Rounds 3, 6, 9, 12, 15 = ((Km_i - R_i-1) left shift to Kr_i times)

F = ((S1(a) + S2(b)) ^ (S3(c)))- S4(d)

The original CAST-128 uses 8 32-bit substitution S-boxes with entries of 256 each like,

S_1 = {0,...to …., 255);
S_2 = {0,…to …., 255);
S_3 = {0,…to …., 255);
S_4 = {0,…to, …., 255);
S_5 = {0,...to …., 255);
S_6 = {0,…to …., 255);
S_7 = {0,…to …., 255);
S_8 = {0,…to …., 255);

The first four S-boxes from 8 S-boxes, are used in the process of encryption/decryption namely S-box 1 to S-box4, and remaining four S-boxes are used for generating subkeys namely S-box5 to S-box8. Above 8 S-boxes are defines as an array with 256 rows by 32 columns. The input (8-bit) to the S-box selects corresponding row in the array and produces the output value of 32-bit in that row. All of the S-boxes carry fixed values.

The process of generation of the 32 Subkeys is a complicated procedure. To start the process, the key, 128 bit, represented as :

n0n1n2n3n4n5n6n7n8n9nAnBnCnDnEnF

Here n0 and nF represents the most and least significant byte, Along with this there are some definitions are used like :

• Sixteen masking subkeys with each 32-bit represented as Km_1,….., Km_{16}
• Kr_1,……, Kr_{16} a sixteen rotate sub keys with each 32-bit, of which uses only the 5 least significant bits of each Kr value.
• Temporary Intermediate bytes y0, y1, y2…….yF
• Temporary Intermediate 32-bit words K1……K32, These values are used to calculate subkeys from the key using S-box5 through S-box8. Then the subkeys are generated by:

for each value of *i* = 1 to 16 then

$Km_i = K_i$;
$Kr_i = K_{16+i}$;

The subkeys are derived from the following STEPS

y0y1y2y3 = n0n1n2n3 XOR S5[nD] XOR S6[nF] XOR S7[nC] XOR S8[nE] XOR S7[n8]
y4y5y6y7 = n8n9nAnB XOR S5[y0] XOR S6[y2] XOR S7[y1] XOR S8[y3] XOR S8[nA]
y8y9yAyB = nCnDnEnF XOR S5[y7] XOR S6[y6] XOR S7[y5] XOR S8[y4] XOR S5[n9]
yCyDyEyF = n4n5n6n7 XOR S5[yA] XOR S6[y9] XOR S7[yB] XOR S8[y8] XOR S6[nB]
Ki = S5[y8] XOR S6[y9] XOR S7[y7] XOR S8[y6] XOR S5[y2]
Ki+2 = S5[yA] XOR S6[yB] XOR S7[y5] XOR S8[y4] XOR S6[y6]
Ki+3 = S5[yC] XOR S6[yD] XOR S7[y3] XOR S8[y2] XOR S7[y9]
Ki+4 = S5[yE] XOR S6[yF] XOR S7[y1] XOR S8[y0] XOR S8[yC]
x0x1x2x3 = y8y9yAyB XOR S5[y5] XOR S6[y7] XOR S7[y4] XOR S8[y6] XOR S7[y0]
n4n5n6n7 = y0y1y2y3 XOR S5[n0] XOR S6[n2] XOR S7[n1] XOR S8[n3] XOR S8[n2]
n8n9nAnB =y4y5y6y7 XOR S5[n7] XOR S6[n6] XOR S7[n5] XOR S8[n4] XOR S5[y1]
nCnDnEnF =yCyDyEyF XOR S5[nA] XOR S6[n9] XOR S7[nB] XOR S8[n8] XOR S6[y3]

II. LITERATURE SURVEY

. Takeshi Sugawara et al. propose an architecture for compact hardware implementation for 64-bit block cipher CAST-128 which minimizes the number of S-box components and merging of 3 F-function into arithmetic component instead of using all eight S-boxes and three types of F-function that depends on number of rounds [2].

The proposed hardware implementation based CAST-128 was synthesized using standard cell libraries from 0,13µm and 0.18µm CMOS, circuits of 26.4-39.5 Kgates and 189.9-614.7Mbps were obtained.

C. Adams et al. here author has examined the cryptography security of CAST-256 as a candidate for the AES [3]. In this paper, the author designed a 128 bit block size with variable keys upto 256-bits which suits AES requirements that consides cryptanalysis property of diffusion and the cryptanalysis techniques of linear and differential cryptanalysis.

Paris Kitsos et al. proposed comparison of FPGA based 64 bit different block ciphers like Triple-DES, IDEA, CAST-128, **MISTY,** and KHAZAD [4]. Two basic architectures like non-feedback mode for IDEA and Triple DES which uses pipelined architecture between rounds that achieves throughput ranges from 3.0 Gbps to 6.9 Cibps and feedback mode for Triple DES and KIIAZAD uses iterative technique that achieved throughput ranges from 115Mbps to 462 Mbps. Along with the throughput author evaluated latency, throughput per slice and area for all cipher implementations.

K.H. Boey et al, power analysis has been applied to attacks on CAST-128 which is the only algorithm that has not been practically broken either on ASIC or GPGA [5]. Author uses 128 bits of secret key within 300, 500 power traces to reveals the outlines an approach on attacking the registers rather than S-box. Hamming weight power model is evaluated on ASIC device that applied on different widths of register.

Shiho Moriai et al. a new higher order differential attack proposed that improves the complexity for solving the equations for a linear systems [6]. The author has explained with example to show the attack of a CAST cipher with 5 rounds using higher order differential attack by choosing 217 plaintext and round functions with less than 225 times computation. The experimental result in this paper show that in the CAST with 5 rounds the last round key can be recovered in less than 15 sec on an Ultra SPARC station.

Krishnamurthy G.N et al. author attempts to modify the function of CAST 128 algorithm to enhance performance by providing better security and reducing the entire time taken for process of encryption/decryption [7]. In this paper the CAST-128 algorithm improves entire process carrying out without degrading the requirements of memory and security by modifying the execution of the F-function of Fiestel network CAST-128algorithm.

III. PROPOSED METHOD

The original CAST-128 algorithm uses 8 32-bit S-boxes with 256 entries where the first four S-boxes from 8 S-boxes, namely S1 through S4, are used in encryption and decryption process and remaining four S-boxes namely S5 through S8 are used in the generation of subkeys. S5 through S8 S-boxes are used as static s-boxes which knows from which S-box the value is taken for generating subkeys. To avoid this static substitution of S-box, the proposed approach uses dynamic substitution of S-box by changing dimension of S-box from single to double dimension. The first four S-boxes are remain same where as S-boxes from S5 to S8 are modified as

S[0][256] = {1, 2, 3,….255};
S[1][256] = {1, 2, 3,….255};
S[2][256] = {1, 2, 3,….255};
S[3][256] = {1, 2, 3,….255};

And the procedure is modified as

y0y1y2y3 = n0n1n2n3 ^ S[nD%4][nD] ^ S[nF%4][nF] ^ S[nC%4][nC] ^ S[nE%4][nE] ^ S[n8%4][n8]
y4y5y6y7 = n8n9nAnB ^ S[y0%4][y0] ^ S[y2%4][y2] ^ S[y1%4][y1] ^ S[y3%4][y3] ^ S[yA%4][yA]
y8y9yAyB = nCnDnEnF ^ S[y7%4][y7] ^ S[y6%4][y6] ^ S[y5%4][y5] ^ S[y4%4][y4] ^ S[n9%4][n9]
yCyDyEyF = n4n5n6n7 ^ S[yA%4][yA] ^ S[y9%4][y9] ^ S[yB%4][yB] ^ S[y8%4][y8] ^ S[nB%4][nB]
Ki = S[y8%4][y8] ^ S[y9%4][y9] ^ S[y7%4][y7] ^ S[y6%4][y6] ^ S[y2%4][y2]
Ki+2 = S[yA%4][yA] ^ S[yB%4][yB] ^ S[y5%4][y5] ^ S[y4%4][y4] ^ S[y6%4][y6]
Ki+3 = S[yC%4][yC] ^ S[yD%4][yD] ^ S[y3%4][y3] ^ S[y2%4][y2] ^ S[y9%4][y9]
Ki+4 = S[yE%4][yE] ^ S[yF%4][yF] ^ S[y1%4][y1] ^ S[y0%4][y0] ^ S[yC%4][yC]
n0n1n2n3 = y8y9yAyB ^ S[y5%4][y5] ^ S[y7%4][y7] ^ S[y4%4][y4] ^ S[y6%4][y6] ^ S]y0%4][y0]
n4n5n6n7 = y0y1y2y3 ^ S[n0%4][n0] ^ S[n2%4][n2] ^ S[n1%4][n1] ^ S[n3%4][n3] ^ S[y2%4][y2]
n8n9nAnB = y4y5y6y7 ^ S[n7%4][n7] ^ S[n6%4][n6] ^ S[n5%4][n5] ^ S[n4%4][n4] ^ S[y1%4][y1]
nCnDnEnF = yCyDyEyF ^ S[nA%4][nA] ^ S[n9%4][n9] ^ S[nB%4][nB] ^ S[n8%4][n8] ^ S[y3%4][y3].

The original algorithm uses single dimension S-boxes for example S6[xF] means the corresponding value of 'xF' from S6 S-box is considered. From this static substitution we can easily find out from which S-box the value is considered. To avoid this our approach uses two dimensional S-boxes from S5 through S8, for example S[xF%4][xF], here the first index referred to identify the from which S-box the value is considered and the second index 'xF' is used to get the corresponding value of 'xF' in the S-box.

IV. RESULTS AND DISCUSSION

The original CAST-128 algorithm [1] and modified CAST-128 algorithm with the same key are applied on the bitmap image cman.bmp. Then compare both original and modified algorithms by making use of Key Sensitivity , Avalanche Effect, and statistical analysis like horizontal coefficient correlation.

The original cman.bmp bitmap image in Figure 3, is encrypted/decrypted by applying original CAST-128 algorithm, same original image is encrypted/decrypted using modified CAST-128 algorithm. The Figure 4 and Figure 5 shows the result generated by encrypting and decrypting image using original algorithm and Figure 6 and Figure 7 shows the result generated by encrypting and decrypting images by applying modified algorithm.

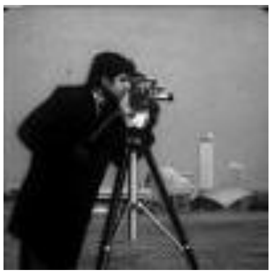

Figure 3 Original image cman.bmp

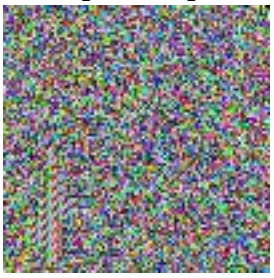

Figure 4 Image cman.bmp Encrypted using Original CAST-128 Algorithm

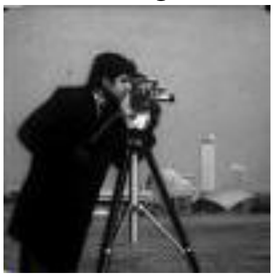

Figure 5. Image cman.bmp Decrypted Original CAST-128 Algorithm

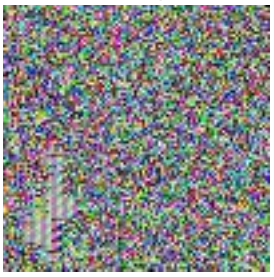

Figure 6. Image cman.bmp Encrypted using Modified CAST-128 Algorithm

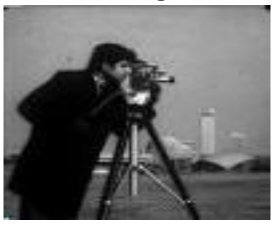

Figure 7. Image cman.bmp Decrypted using Modified CAST-128 Algorithm

A. KEY SENSITIVITY

A 16-character Key with 128 bits is used for encryption/decryption process. The Key sensitive test , [8] has been carried out by encrypt/decrypt the cman.bmp image by applying the 16-character 128-bit key, say Key1, using original and modified CAST-128 algorithm. Then changing any randomly selected one bit from Key1 say key2, then from this modified key, Key2, encrypt and decrypt the same image cman.bmp by applying both original and modified CAST-128 algorithm.

Ex: from key1 DF37A465E262AB1F5D8**C**94A0AF2627, select randomly '**C**' as shown in bold and changed to '**E**' as DF37A465E262AB1F5D8**E**94A0A26B27 say key2. These two keys, Key1 and Key2, which is used to encrypt and decrypt the image cman.bmp and then by applying key sensitivity test the result is compared which is shown in Table I

Table I. The Key Sensitivity Test Result by Using Original And Modified CAST-128-bit algorithm For Various Rounds.

Number of Rounds	**Original AES**	**Modified AES**
2	78.57	78.62
4	78.59	78.64
6	78.59	78.65
8	78.63	78.61
10	78.63	78.62
16	78.57	78.64

B. AVALANCHE EFFECT

The changes in one bit in every block of data in plaintext then there will be enormous changes in the ciphertext this is called avalanche effect in [8] To evaluate the avalanche effect, need to consider 3000 samples of plaintext with one encrypting using two different keys key1 and key2 with original and modified algorithm, two changing one bit in all blocks of data in the plaintext say B1cman.bmp encrypted using both key1 and key2 with both original and modified algorithm, along with four cases. **Case 1**: Avalanche effect is compared using cman.bmp image by encrypting with key1 and key2 using original and modified CAST-128 algorithm is as shown in the table 2. **Case 2**: Avalanche effect is compared using B1cman.bmp image by encrypting with key1and key2 using original and modified CAST-128 algorithm as shown in the table 3. **Case 3**: In this case Avalanche effect is compared using cman.bmp image and B1cman.bmp image by encrypting with same key1 using original and modified CASE-128 algorithm as shown in the table 4.In **Case 4**: the Avalanche is compared using cman.bmp and B1cman.bmp image by encrypting with same key2 using original and modified algorithm.

The result of avalanche effect is compared by considering these cases is shown in table II.

Table- II: CASE 1 : The Result Of Avalanche effect Is Compared Using Original And Modified CAST-128 algorithm For Different Rounds

No of Rounds	No of pairs Of plaintext samples	The Original process provides No of times better Avalanche	The Modified process provides No of times better Avalanche	Both the process provides No of times same Avalanche
2	3000	827	882	1291
4	3000	883	810	1307
6	3000	817	907	1276
8	3000	854	864	1282
10	3000	810	813	1377
16	3000	789	860	1351

Table III. CASE 2 : The Result Of Avalanche effect Is Compared Using Original And Modified CAST-128 algorithm For Different Rounds.

No of Rounds	No of pairs Of plaintext samples	The Original process provides No of times better Avalanche	The Modified process provides No of times better Avalanche	Both the process provides No of times same Avalanche
2	3000	804	1455	741
4	3000	931	1357	712
6	3000	795	1491	714
8	3000	815	1467	718
10	3000	839	1433	728
16	3000	845	1437	718

Table IV. CASE 3 : The Result Of Avalanche effect Is Compared Using Original And Modified CAST-128 algorithm For Different Rounds.

No of Rounds	No of pairs Of plaintext samples	The Original process provides No of times better Avalanche	The Modified process provides No of times better Avalanche	Both the process provides No of times same Avalanche
2	3000	823	1577	600
4	3000	870	1454	676
6	3000	787	1420	793
8	3000	850	1387	763
10	3000	818	1469	713
16	3000	842	1461	697

Table V. CASE 4 : The Result Of Avalanche effect Is Compared Using Original And Modified CAST-128 algorithm For Different Rounds.

No of Rounds	No of pairs Of plaintext samples	The Original process provides No of times better Avalanche	The Modified process provides No of times better Avalanche	Both the process provides No of times same Avalanche
2	3000	875	1558	567
4	3000	750	914	1336
6	3000	762	892	1346
8	3000	883	1403	714
10	3000	872	1439	689
16	3000	844	1459	697

C. CORRELATION COEFFICIENT

The correlation coefficient is determined relationship between horizontally adjacent pixels in an image [9], [10]. The steps for determining the correlation of horizontal adjoining pixels in an image (cman.bmp) is as Follows

1. From the original image cman.bmp and their encrypted image, select N pairs of horizontally adjacent pixels.
2. Select pixels randomly and pixels adjacent to them from the both original image that is cman.bmp shown in Figure 3 and their encrypted images shown in Figure 4 and Figure 6 using both original CAST-128 algorithm as well as modified CAST-128 algorithm.
3. Using rxy formulae to find correlation coefficient, where x and y represents horizontally adjacent pixels with grey scale values in an image.

$$r_{xy} = \frac{cov(x,y)}{\sqrt{D(x)}\sqrt{D(y)}}$$

where D(X) represents variance of x and D(Y) represents variance of y values and COV (X Y) is covariance of x and y and is given by

$$COV(x,y) = \frac{1}{N}\sum_{i=0}^{N}(x_i - E(x))(y_i - E(y))$$

Where N represents number of horizontal adjacent pixels selected randomly, E(X) and E(Y) represents the mean values of x and y values. This test is carried out for about randomly selected horizontally adjacent pixels from the original image cman.BMP and encrypted images. Then using above equations correlation coefficient will be computed and is as shown in below Figure. 8, Figure. 9, and Figure. 10. The correlation coefficient of original image is 1.175418 and for cipher image which is encrypted by blowfish algorithm is 0.515035 and for modified algorithm is 0.586409.

In both original and modified CAST-128 algorithm the correlation coefficients for plaintext image with that of ciphertext images are far apart.

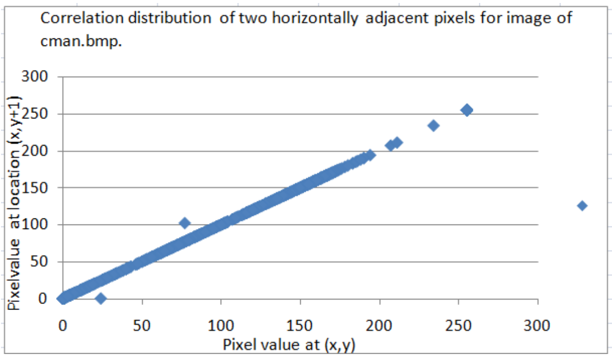

Figure 8. Distribution of Correlated Horizontally Adjoiningt Pixels for the Image cman.bmp

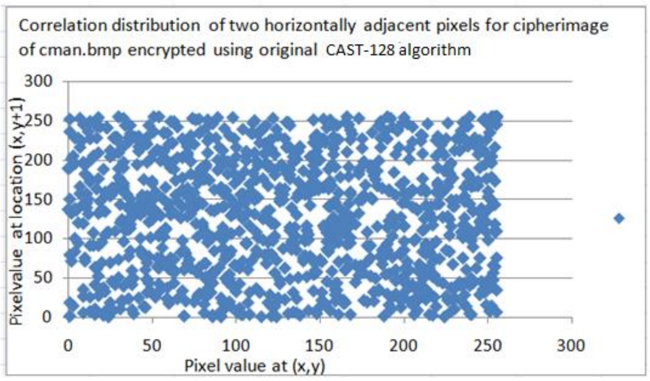

Figure 9. Correlation Distribution of encrypted image using original CAST-128 algorithm by considering Horizontally Adjacent Pixels.

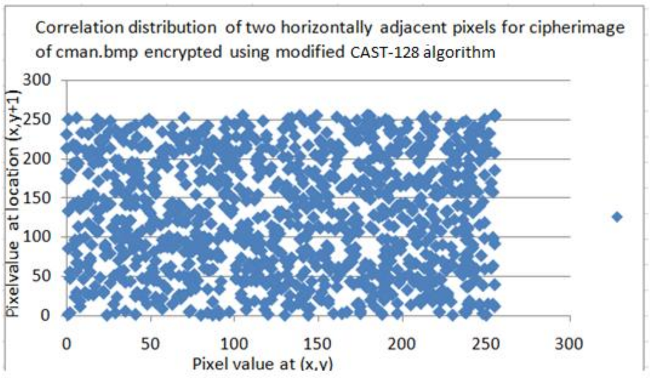

Figure 10. Correlation Distribution of encrypted image using Modified CAST-128 algorithm by considering Horizontally Adjacent Pixels

V. CONCLUSION

In this paper, modified key scheduling algorithm is designed and implemented using dynamic substitution of S-box for generating subkeys for CAST-128 encryption algorithm. With key sensitivity, avalanche effect, horizontal correlation coefficient, higher throughput, our proposed method is obviously provides better security than original CAST-128 encryption algorithm.

REFERENCES

1. Krishnamurthy G N, Dr. V Ramaswamy. "Encryption Quality Analysis and SecurityEvaluation of CAST-128 Algorithm and its Modified Version using Digital images." International Journal of Network Security & Its Applications (IJNSA), Vol.1, No 1, April 2009
2. Takeshi Sugawara, Naofumi Homma, Takafumi Aoki and Satoh A (2007), “A High-Performance ASIC Implementation of the 64-bit Block Cipher CAST-128”,2007 1EEE international symposium on circuits and systems, doi:10.1109/iscas.2007.378277.
3. C. Adams, H.M. Heyst, S.E. TavaresJ, and M. Wiener, “An Analysis of the CAST-256 Cipher”, Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering Shaw Conference Center, Edmonton, Alberta, Canada May 9-12 1999
4. Paris Kitsos, Nicolas Sklavos, Michalis D. Galanis, Odysseas Koufopavlou, “AN FPCA-BASED PERFORMANCE COMPARISON OF THE 64-bit BLOCK CIPHERS”, Proceedings World Automation Congress, 2004.IEEE,2005.
5. K.H. Boey, Y. Lu, M. O’Neill, R. Woods,” Differential Power Analysis of CAST-128”, 2010 IEEE Annual Symposium on VLSI.
6. Shiho Moriai, Takeshi Shimoyama, and Toshinobu Kaneko, “Higher Order Di_erential Attack of a CAST Cipher”, S. Vaudenay (Ed.): Fast Software Encryption FSE'98, LNCS 1372, pp. 17-31, Springer-Verlag Berlin Heidelberg 1998.
7. Krishnamurthy G.N, Ramaswamy V, Leela G.H, Ashalatha M.E, “Performance Enhancement of CAST-128 Algorithm by Modifying Its Function”, T. Sobh (ed.), Advances in Computer and Information Sciences and Engineering, 256–260, Springer Science+Business Media B.V. 2008
8. Shailaja S1, Dr Krishnamurthy G N2, Comparison of Blowfish and Cast-128 Algorithms Using Encryption Quality, Key Sensitivity and Correlation Coefficient Analysis, American Journal of Engineering Research (AJER) e-ISSN : 2320-0847 p-ISSN : 2320-0936 Volume-3, Issue-7, pp-161-166.
9. Sudarshan, TSB; Mir, Rahil Abbas; Vijayalakshmi. S,DRIL a flexible architecture for blowfish encryption using dynamic reconfiguration replication inner-loop pipelining loop folding techniques, Springer, Advances in Computer Systems Architecture, pages 625-639, 2005.
10. Krishnamurthy G.N, Dr. V. Ramaswamy, Leela G.H and Ashalatha M.E, Blow-CAST-Fish: A New 64- bit Block Cipher, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4, April 2008.