Journal of Current Research in Engineering and Science
JCRES
Bi-Annual Online Journal (ISSN: 2581 - 611X)

VOLUME:8, ISSUE:1, FEBRUARY 2025, PAPER NUMBER:68

Digital Circuit Based Symmetric Key Cryptography using XNOR Gates

Joyita Goswami
BCA Department
Global Institute of Management & Technology (GIMT)
Kolkata, India joyitagoswami@gmail.com

Manas Paul
Department of Computer Science & Technology
Institute of Leadership, Entrepreneurship and Development
Kolkata, India manaspaul@rediffmail.com

Abstract— **A session based symmetric key cryptography technique has been proposed in this paper and it is termed as 'XNOR_H'. In this paper the plain text is considered as a finite number of binary bits and is chopped into blocks with variable length. Binary representation of each bit position of plain text block is XNOR-ed with next bit and LSB bit is XNOR-ed with high logic value (5v) by using the digital circuit. The session key is generated randomly from the chopping information of plain text. Different types of twenty files varying different size are used to compute the result. Results of XNOR_H are compared with standard symmetric key cryptography techniques Triple-DES (168bits) and AES (128bits) with respect to the Encryption and Decryption times, Avalanche and Strict Avalanche values, Bit independence value, and Chi- square values.**

Keywords— XNOR_H, Symmetric key cryptography, Session key, Triple-DES, AES.

I. INTRODUCTION (*HEADING 1*)

To protect our information from outside world is very important to us. Now a day every computer is connected virtually. So data security becomes prime concern of our daily life. Cryptography is an important aspect for secure communication to protect important information. Continuous research works [1, 2, 3, 4, 5, 6, 7] are going on in this field of cryptography to increase data security. Section-II of this paper explains the proposed technique. Section-III deals with the algorithms for encryption, decryption and session key generation. Section-IV explains the proposed technique with an example. Section-V shows the results and analysis on different files and the comparison of the proposed technique with TDES and AES. Conclusions are drawn in Section- VI.

II. TECHNIQUE

XNOR_H considers the input file as a finite number of binary bits. The binary bits are split dynamically into the blocks of length 2^k where k>=3 and k€N, N is the set of natural numbers. The block sizes are written into the file to generate session based key during encryption. The i^{th} position bit of the plain text block is mapped to the j^{th} position of encrypted block. This mapping is bijective in nature. The value of i varies from 0 to (2^k -1), is converted into k-bit binary number and the corresponding binary bits are sent into k-input digital circuit. For each 2k number of combination of inputs, the output of the circuit produces unique 2k number of k-bit binary numbers which are converted to the corresponding decimal to find the value of j. The digital circuit diagrams of encryption and decryption are shown in Fig.1 and Fig.2 respectively.

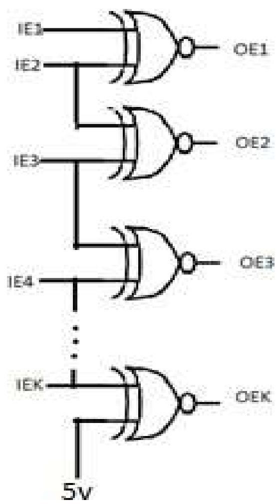

Fig. 1:The logic circuit for Encryption

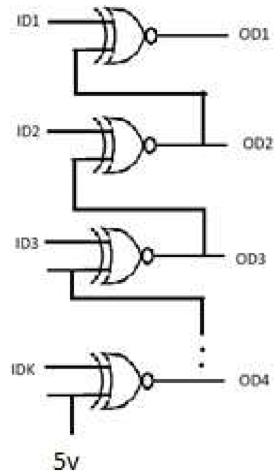

Fig. 2:The logic circuit for Decryption

III. ALGORITHMS

In this section Encryption, Decryption and Session key generation algorithms are explained in details.

A. Encryption Algorithm

Step 1: The plain text i.e input file is considered as a finite number of binary bits.

Step 2: The bits are chopped dynamically into blocks of different lengths like 8 / 16 / 32 / 64 / 128 / 256 / 512 /. [i.e. 2^k for k=3,4,5,6] as follows. First n_1 no. of bits is considered as x_1 no. of blocks with block length y_1 where $n_1 = x_1*y_1$. Next n_2 no. of bits is considered as x_2 no. of blocks with block length y_2 where $n_2 = x_2*y_2$ and so on. Finally n_m no. of bits is considered as x_m no. of blocks with block length y_m (= 8) where $n_m = x_m* y_m$. So no padding is required.

Step 3: The bit position value (say i value) of the plain text block (having block length 2^k), varies from 0 to (2^k - 1), is converted into k-bit binary number as $IE_1IE_2IE_3$ IE_k (where IE_1 is the MSB and IE_k is the LSB) and the corresponding binary bits are sent into k-input digital circuit (shown in Fig.1). The circuit performs XNOR-ed operation with next bit and LSB (i.e IE_k) is XNOR-ed with high logic(5v) to generate the output bits.

Step 4: The output bits of the digital circuit represented as $OE_1OE_2OE_3$ OE_k (where OE_1 is the MSB and OE_k is the LSB) The cipher text is formed by converting the encrypted block to its corresponding characters.

B. *Decryption Algorithm*

Step 1: The cipher text is considered as a finite number of binary bits.

Step 2: Processing the session key the binary bits are sliced into manageable sized blocks.

Step 3: The bit position value (say i value) of the cipher text block having block length 2^k is converted into k-bit binary number as $ID_1ID_2ID_3.....ID_k$ (where ID_1 is the MSB and ID_k is the LSB) and the corresponding binary bits are sent into k-input digital circuit (shown in Fig.2) and generates the output $OD_1OD_2OD_3.....OD_k$ (where OD_1 is the MSB and OD_k is the LSB). The LSB of cipher block (i.e ID_k) is XNORed with positive logic(5v) to generate the LSB (i.e OD_k) of output block. Then this output bit is XNORed with the next bit of cipher block to get corresponding output bit.

The plain text is regenerated by converting the decrypted block to its corresponding characters.

C. *Session Key Generation Algorithm*

XNOR_H generates a session based key for one time use in a particular session. The input bit stream is divided into 16 portions where 1st portion contains 20% of the total file size, 2nd portion contains 20% of the remaining file size and so on. Each portion is divided into x no. of blocks with block length y (=8n) where value of n is selected dynamically for first fifteen portions. Finally last (i.e. 16th) portion is divided into x16 no. of blocks with block length 8 bits (i.e. y16 = 8). So no padding is required. Total length of the input binary stream is

$$= x_1*y_1+x_2*y_2+....... \quad +x_{16}*y_{16}.$$

The value of n for each portion is stored as a character in the key file. So the key file contains sixteen characters.

IV. EXAMPLE

Let consider the word "My". The 8 bit representation of the above characters 'M' and 'y' are '01001101' and '01011001' respectively. The bits are stored into an array from MSB to LSB as 8 (k=3) bit or 16 (k=4) bit block chosen randomly.

Table1 & 2 show how each position of 8 bit block (0 to 7 i.e. 000 to 111) is converted into binary number and following the above logic bits are changed to generate the new position. Table3 & 4 show the same for 16 bit block (0 to 15 i.e. 0000 to 1111).

TABLE1. INPUT FOR 8 BIT BLOCK

Position of input block i.e. 'i' value	Corresponding binary representation of 'i' value	(Circuit Input)		
		IE_1	IE_2	IE_3
0	000	0	0	0
1	001	0	0	1
2	010	0	1	0
3	011	0	1	1
4	100	1	0	0
5	101	1	0	1
6	110	1	1	0
7	111	1	1	1

TABLE2. OUTPUT FOR 8 BIT BLOCK

(Circuit Output)			Corresponding binary representation of 'j' value	Position of encrypted block i.e. 'j' value
OE_1 (=$IE_1 \odot IE_2$)	OE_2 (=$IE_2 \odot IE_3$)	OE_3 (=$IE_3 \odot 1$)		
1	1	0	110	6
1	0	1	101	5
0	0	0	000	0
0	1	1	011	3
0	1	0	010	2
0	0	1	001	1
1	0	0	100	4
1	1	1	111	7

TABLE3. INPUT FOR 16 BIT BLOCK

Position of input block i.e. 'i' value	Corresponding binary representation of 'i' value	(Circuit Input)			
		IE_1	IE_2	IE_3	IE_4
0	0000	0	0	0	0
1	0001	0	0	0	1
2	0010	0	0	1	0
3	0011	0	0	1	1
4	0100	0	1	0	0
5	0101	0	1	0	1
6	0110	0	1	1	0
7	0111	0	1	1	1
8	1000	1	0	0	0
9	1001	1	0	0	1
10	1010	1	0	1	0
11	1011	1	0	1	1
12	1100	1	1	0	0
13	1101	1	1	0	1
14	1110	1	1	1	0
15	1111	1	1	1	1

TABLE4. OUTPUT FOR 16 BIT BLOCK

(Circuit Output)				Corresponding binary representation of 'j' value	Position of encrypted block i.e. 'j' value
OE_1 (=$IE_1 \odot IE_2$)	OE_2 (=$IE_2 \odot IE_3$)	OE_3 (=$IE_3 \odot IE_4$)	OE_4 (=$IE_4 \odot 1$)		
1	1	1	0	1110	14
1	1	0	1	1101	13
1	0	0	0	1000	8
1	0	1	1	1011	11
0	0	1	0	0010	2
0	0	0	1	0001	1
0	1	0	0	0100	4
0	1	1	1	0111	7
0	1	1	0	0110	6
0	1	0	1	0101	5
0	0	0	0	0000	0
0	0	1	1	0011	3
1	0	1	0	1010	10
1	0	0	1	1001	9
1	1	0	0	1100	12
1	1	1	1	1111	15

Case I: If block length is 8 then the encrypted string is '0110010100110101'. Two 8 bit binary numbers are '01100101' (=$[101]_{10}$) and '00110101' (=$[53]_{10}$) is encrypted from binary string and the corresponding characters are 'e' and '5' respectively. So "My" is converted into "e5".

Case II: If block length is 16 then the encrypted string is '0111010100100101'. Two 8 bit binary numbers are

'01110101' (=[117]$_{10}$) and '00100101' (=[37]$_{10}$) is encrypted from binary string and the corresponding characters are 'u' and '%' respectively. So "My" is converted into "u%".

Fig. 3 shows the encryption steps for the above example (for both the cases).

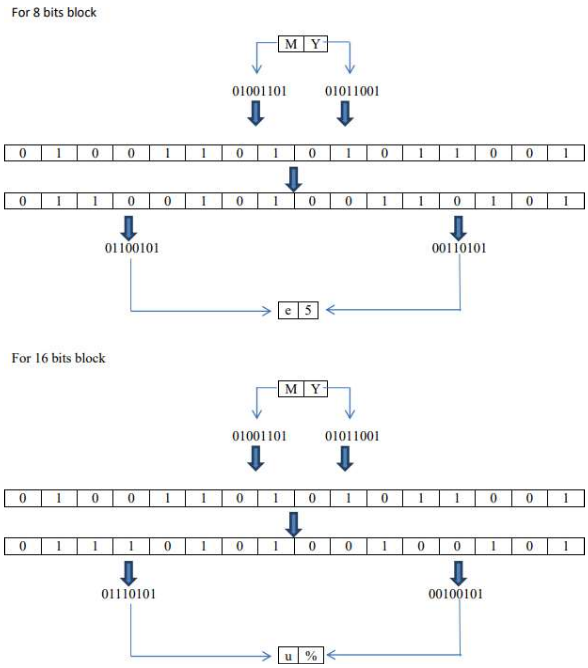

Fig. 3. The encryption steps

V. RESULTS

Using different types of twenty files (varying size), the results are generated. Comparison and extensive analysis has been done among Triple-DES (168bits) and AES (128bits) and the proposed technique XNOR_H with respect to the following parameters.

A. Encryption and Decryption Times

The encryption and decryption times are calculated by taking the differences between processor clock ticks at the starting and ending of execution. The highest speed of execution is determined if it takes the minimum time to execute. Encryption and Decryption times (in milliseconds) of twenty different files are generated for T-DES, AES and XNOR_H. Table 5 & 6 show the different encryption and decryption times of above algorithms for different source files. Files are sorted in ascending order depending on their size.

TABLE5. ENCRYPTION TIME

Serial No.	File type	File size (Bytes)	Encryption time (in m.sec)		
			TDES	AES	XNOR_H
1	txt	51	0	0	0
2	zip	282	0	0	0
3	txt	488	0	0	0
4	txt	2403	0	0	0
5	jpg	5292	16	0	16
6	docx	13108	15	16	30
7	exe	21062	16	15	45
8	jpg	49720	16	16	61
9	rar	113283	30	15	123
10	dll	211107	46	15	199
11	exe	611645	118	31	551
12	docx	1189825	208	29	1053
13	dll	1380843	253	74	1215
14	jpg	3516950	563	89	3099
15	pdf	4357325	711	120	3841
16	avi	7208809	1275	193	6361
17	rtf	15451499	2521	400	13687
18	doc	42606982	6777	1097	37785
19	rar	75701101	12071	1958	66969
20	avi	141278589	22703	3603	124996

TABLE6. DECRYPTION TIME

Serial No.	File type	File size (Bytes)	Decryption time (in m.sec)		
			TDES	AES	XNOR_H
1	txt	51	0	0	0
2	zip	282	0	0	0
3	txt	488	0	0	16
4	txt	2403	0	0	15
5	jpg	5292	0	0	16
6	docx	13108	0	0	30
7	exe	21062	15	16	46
8	jpg	49720	16	15	74
9	rar	113283	29	15	118
10	dll	211107	43	29	178
11	exe	611645	120	59	547
12	docx	1189825	222	74	1053
13	dll	1380843	252	89	1231
14	jpg	3516950	682	192	3085
15	pdf	4357325	859	222	3825
16	avi	7208809	1379	355	6332
17	rtf	15451499	2952	860	13539
18	doc	42606982	8037	2505	37355
19	rar	75701101	14430	4182	66420
20	avi	141278589	26827	8214	123944

B. Avalanche & Strict Avalanche values and Bit Independence Criterion

Avalanche, Strict avalanche and Bit Independence Criterion is used to measure the degree of security of cryptographic technique. The ratio of changed bit in cipher test and total no of bits in cipher text indicates the Avalanche value. It may indicate the high degree of security if the values of Avalanche and Strict Avalanche closer to 1.0. Table 7, 8 & 9 show the Avalanche, Strict Avalanche and Bit Independence values respectively for TDES, AES and XNOR_H which are closer to 1. This comparison indicates that XNOR_H may provide good security.

TABLE7. AVALANCHE VALUES

Serial No.	File type	File size (Bytes)	Avalanche achieved		
			TDES	AES	XNOR_H
1	txt	51	0.951253	0.953911	0.679307
2	zip	282	0.946608	0.958814	0.362125
3	txt	488	0.956220	0.954386	0.443219
4	txt	2403	0.959977	0.959924	0.296429
5	jpg	5292	0.960124	0.959988	0.874308
6	docx	13108	0.960304	0.960255	0.407369
7	exe	21062	0.960066	0.960109	0.895582
8	jpg	49720	0.960368	0.960122	0.959009
9	rar	113283	0.960259	0.960373	0.959480
10	dll	211107	0.960297	0.960343	0.947331
11	exe	611645	0.960362	0.960351	0.938846
12	docx	1189825	0.960377	0.960387	0.947559
13	dll	1380843	0.960382	0.960388	0.920553
14	jpg	3516950	0.960384	0.960382	0.957381
15	pdf	4357325	0.960396	0.960395	0.952163
16	avi	7208809	0.960400	0.960381	0.955293
17	rtf	15451499	0.960361	0.960319	0.913033
18	doc	42606982	0.960340	0.959496	0.930843
19	rar	75701101	0.960384	0.960400	0.929263
20	avi	141278589	0.960382	0.960390	0.788547

TABLE8. STRICT AVALANCHE VALUES

Serial No.	File type	File size (Bytes)	Strict Avalanche achieved		
			TDES	AES	XNOR_H
1	txt	51	0.885077	0.907230	0.645660
2	zip	282	0.925026	0.952614	0.193629
3	txt	488	0.969093	0.968807	0.186214
4	txt	2403	0.977128	0.978467	0.186767
5	jpg	5292	0.978657	0.978676	0.850806
6	docx	13108	0.979198	0.978869	0.194194
7	exe	21062	0.978818	0.979091	0.885153
8	jpg	49720	0.979309	0.979326	0.978146
9	rar	113283	0.979473	0.979286	0.978663
10	dll	211107	0.979526	0.979533	0.963061
11	exe	611645	0.979622	0.979598	0.948446
12	docx	1189825	0.979638	0.979660	0.965693
13	dll	1380843	0.979623	0.979621	0.935006
14	jpg	3516950	0.979665	0.979645	0.976412
15	pdf	4357325	0.960396	0.960395	0.952163
16	avi	7208809	0.960400	0.960381	0.955293
17	rtf	15451499	0.960361	0.960319	0.913033
18	doc	42606982	0.960340	0.959496	0.930843
19	rar	75701101	0.960384	0.960400	0.929263
20	avi	141278589	0.960382	0.960390	0.788547

TABLE9. BIT INDEPENDENCE VALUES

Serial No.	File type	File size (Bytes)	Bit Independence achieved		
			TDES	AES	XNOR_H
1	txt	51	0.153257	0.254511	0.001265
2	zip	282	0.385557	0.349991	0.001968
3	txt	488	0.403445	0.390877	0.003043
4	txt	2403	0.470316	0.474135	0.000047
5	jpg	5292	0.951409	0.955483	0.763968
6	docx	13108	0.955846	0.952357	0.002112
7	exe	21062	0.620802	0.597423	0.658996
8	jpg	49720	0.977266	0.977487	0.970594
9	rar	113283	0.977528	0.977097	0.977890
10	dll	211107	0.738041	0.739243	0.761747
11	exe	611645	0.730894	0.725332	0.755848
12	docx	1189825	0.970751	0.970836	0.963302
13	dll	1380843	0.710765	0.714787	0.838808
14	jpg	3516950	0.974513	0.974523	0.972627
15	pdf	4357325	0.955523	0.943730	0.971040
16	avi	7208809	0.973098	0.971742	0.971272
17	rtf	15451499	0.365998	0.332059	0.350186
18	doc	42606982	0.333407	0.216264	0.422446
19	rar	75701101	0.979501	0.979400	0.948766
20	avi	141278589	0.968364	0.967834	0.777508

C. Chi-Square Values

Component The large Chi-square value compared with tabulated value may indicate a high degree of non-homogeneity among source and encrypted files. Table 6 shows the Chi-square values for Triple-DES (168bits), AES (128bits) and XNOR_H. Average chi-square values of Triple-DES (168bits), AES (128bits) and XNOR_H are 32791046955,31312548782 and 38974112758 respectively. Figure 10 shows the comparison of the Chi-square values of all three techniques against the twenty source files. From the figures it is observed that the degree of non-homogeneity of the encrypted files with respect to source files using the technique XNOR_H is very high. Therefore it may conclude that XNOR_H provides good security.

TABLE10. CHI-SQUARE VALUES

Serial No.	File type	Chi-Square values		
		TDES	AES	XNOR_H
1	txt	110	106	157

2	zip	482	508	342
3	txt	1411	1485	7915
4	txt	23105	20150	119116
5	jpg	899	908	830
6	docx	17607	8973	1025
7	exe	1002978	462119	157206
8	jpg	1318	1249	4833
9	rar	990	997	730
10	dll	509958	454295	746423
11	exe	1946832	1774983	2666985
12	docx	52787	53373	127485
13	dll	3092249	3015235	3177111
14	jpg	75802	76152	102514
15	pdf	397230	354928	797048
16	avi	420854	425348	308142
17	rtf	655520668680	625972131375	778618078873
18	doc	277384331	257108052	845826775
19	rar	58871	58620	7116
20	avi	15282599	15026790	10124537
Average		32791046955	31312548782	38974112758

VI. CONCLUSION

Our proposed technique XNOR_H in this paper is simple to understand. It can be easily implemented by using various high level languages. The performance of XNOR_H is quite satisfactory because of its high processing speed and the measure of the degree of security is compared with Triple-DES and AES. It is applicable in message transmission of any form and any size.

REFERENCES

[1] Sheng Y., Li J., Di X., Man Z., Liu Z.: Bit-level image encryption algorithm based on fully-connected-like network and random modification of edge pixels. IET Image Processing published by John Wiley & Sons Ltd. Vol. 16, Issue 10, Pages 2769-2790, August 2022.

[2] Zhu H., Dai L., Liu Y., Wu L.: A three-dimensional bit-level image encryption algorithm with Rubik's cube method. Mathematics and Computers in Simulation. Vol. 185, Pages 754-770, July 2021.

[3] Nath A., Chakraborty A., Maitral S.: New Bit Level Positional Encryption Algorithm (NBPLEA - Ver 2). International Journal of Scientific Research in Computer Science, Engineering and Information Technology, Vol. 7, Issue 3, Pages 245-257, May-June-2021. doi:10.32628/CSEIT217350.

[4] Wang M., Wang X., Zhao T., Zhang C., Xia Z., Yao N.: Spatiotemporal chaos in improved cross coupled map lattice and its application in a bit-level image encryption scheme. Information Sciences. Vol. 544, Pages 1–24, January 2021.

[5] Karawia A. A., Elmasry Y. A.: New Encryption Algorithm Using Bit-Level Permutation and Non-Invertible Chaotic Map. IEEE Access, Vol. 9, 2021.

[6] Shahna K. U., Mohamed A.: A novel image encryption scheme using both pixel level and bit level permutation with chaotic map. Applied Soft Computing, Vol. 90, May 2020, Art. no. 106162.

[7] Li Z., Peng C., Tan W., Li L.: A Novel Chaos-Based Color Image Encryption Scheme Using Bit-Level Permutation. Symmetry 2020, 12, 1497; doi:10.3390/sym12091497.

[8] Wang J., Li J., Di X., Zhou J., Man Z.: Image Encryption Algorithm Based on Bit-Level Permutation and Dynamic Overlap Diffusion. IEEE Access 2020, Computer Science.

[9] Goswami J., Paul M.: Symmetric Key Cryptography using Digital Circuit based on One Right Shift. International Conference on Advances in Science and Technology (ICAST 2017).MCKV Institute of Engineering, Howrah, West Bengal, India. 17-19 March, 2017.