Light weight security coding using PRESENT algorithm for cryptography application

B. Akhil[1], Md Muzammil Shareef[2], B. Shalini[3],Dr.SK.Fairooz[4], Shaik Mohammed Rafi[5]

[123]*UG Scholar , Sreyas Institute of Engineering & Technology, Hyderabad.*

[4] *Dr. SK.Fairooz , Associate Professor , Dept of ECE, Sreyas Institute of Engineering & Technology, Hyderabad.*

[5] Shaik Mohammed Rafi, *Assistant Professor , Dept of ECE, Sreyas Institute of Engineering & Technology, Hyderabad.*

Abstract: *In this article, we are developing a security approach using a lightweight algorithm called "PRESNT". This security encryption is a round update process with 31 iterations and updates. Hardware requirements to develop multiple iterations of security are resource limiting, and PRESENT has a significant reduction in resource use. In this work, the focus is on implementing using Xilinx FPGA installation. It is proposed to implement the PRESENT algorithm using VHDL, timing process testing, coding verification and decoding.*

Keywords: Present, PFGA ,VHDL ,Cryptography ,Xilinx.

I. INTRODUCTION

The need to send secure data poses different scenarios and features in each type of application. In each case, the two main components that go into design and determine reliability and durability are hardware and software. In wireless applications with lower data transmission (performance) rates, devices tend to be custom and inexpensive.

At this point, it would be reasonable to ask why we might want to design a new block cipher. After all, it has become a "accepted" fact that flow blades may be more compact. Indeed, with the eSTREAM project [15], renewed efforts are made to understand the design of embedded flow codes, and many promising proposals offer attractive performance features. But we look at a few reasons why we think about coding merged blocks. First of all, primitive block encoding is very versatile and by operating block encoding in counter mode (for example) we get current encoding. But second, and perhaps most importantly, the art of block coding seems to be a little better understood than stream encoding. For example, although there is a rich theory supporting the use of linear linear shift offset records, it is not easy to combine these building blocks to provide a safe rendering. We suspect that carefully designed block coding may be a less risky task than coding the newly designed stream.

Block-based encodings [10, 11] are one of the most widely used coding systems [12], where you can find endless algorithms with different properties. Most of them were implemented in programmable logic devices [9, 11, 13, 14]. One of these algorithms is AES (Advanced Encryption Standard), which is a mandatory reference because it is a block-based encryption standard [5, 13, 15], as well as many others listed in many works that compare its characteristics. And scales [1,11]. Another mandatory reference is the elliptic curve coding [17, 18] due to its background and the need to give the project a solid mathematical foundation. With that in mind and knowing that the implementation of these algorithms can be done using different techniques and devices, this research aims to implement block-based coding that meets a lightweight philosophy and can be easily achieved in terms of hardware. . Therefore, the current algorithm [4, 8, 9,] is chosen to be implemented in devices, then studies will be conducted to determine the main metrics.

Hardware actions provide quick solutions for applications where data traffic is higher and require real-time encryption [14].The organization of paper is section II Existing work, section III Proposed work, section IV as Simulation results and section V as conclusions.

II. Existing Work

Although work on low-cost encryption is increasing, the number of documents related to very light encryption is surprisingly limited. While we focus on designing the algorithm, we will not refer to work on low-cost authentication and communication protocols. Some of the most comprehensive work on integrated implementation is currently being carried out within the eSTREAM project. As part of this initiative, new current blades have been proposed that are suitable for

efficient deployment of devices. While this work continues, some promising candidates
Emerging [7,]. While the offsets are complex, implementation documents indicate that approximately 1300-2600 Gateway equivalent (GE) will be required for the largest codes embedded in the eSTREAM project.
For modern blades, the White Paper [15] provides a very comprehensive analysis of a low-cost AES application. However, the resources required for this coding are around 3600 GE, which is an indirect result of the fact that Rijndael is designed for software efficiency on 8 and 32 bit processors. The requirements for implementing the small tea cipher algorithm [13, 14] are unknown, but the initial estimate is that the tea needs 2100GE and needs at least xtea 2, 2000 GE.

III. PROPOSED WORK

PRESENT is one of the most popular block-based lightweight encryption algorithms due to its specific design that has an easy application, both in hardware and software [4, 8]. The hardware can be implemented in some of the smallest FPGAs on the market, with remarkably high performance [9].

A) Structure of the PRESENT algorithm

Figure 1 describes the basic structure of the current algorithm where its blocks are displayed and how each of its 31 rounds is executed [4].

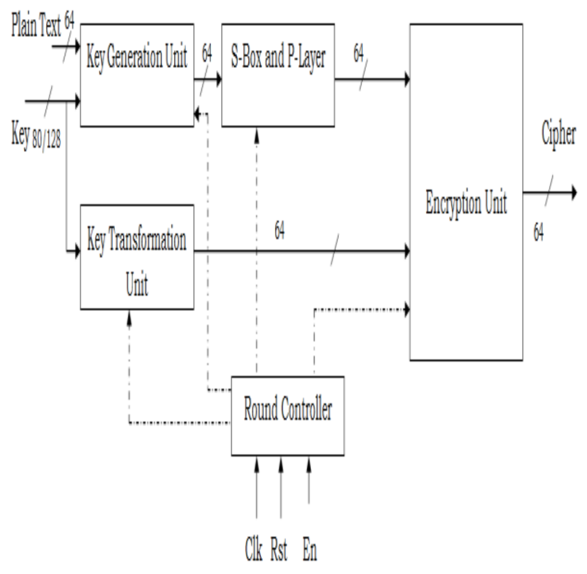


Figure 1. Block diagram of Encoding the cipher text using Present algorithm

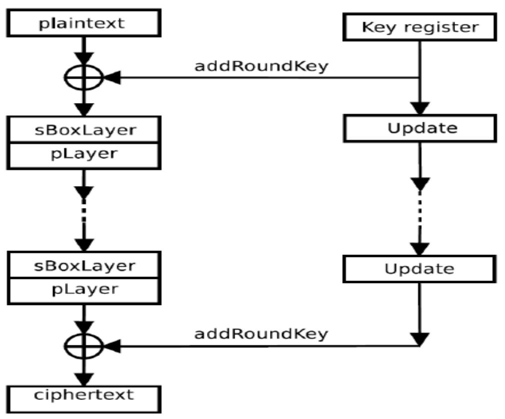


Figure 2. Round of the PRESENT algorithm

B) Byte substitution layer: Sbox Layer

It consists of a non-linear substitution that is applied independently to each bite of the state matrix that generates a new bite. This shift consists of replacing each bite as a result of applying the S-Box Replacement Table [4,8]. This replacement block applies to the 16 bites that add up to 64 bits of information, which is the standard size for cipher blocks.

Table 1. Substitution box S-box of the PRESENT algorithm

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

When designing the cipher algorithm, the highest entropy is searched for within the data being replaced. In this case, the word size is 4 bits, since it is a completely simple algorithm that can be implemented in a single LUT in FPGA.

C) Bits permutation: p Layer

It is a mixture layer where bit-by-bit substitution is performed in a 64-bit information block where bit i is moved from round to position P (i), and the substitution order is shown in Table 2.

Table 2. Substitution order

Í	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

D) Password expansion function: add Round Key

PRESENT may contain passwords ranging from 80 to 128 bits. However, this design and implementation will only consider 80 bits that will be stored in a record K of this size and will be numbered K79, K78 ... K0. In each round, the most important 64 bits of the new calculated password will only be confused after applying the password expansion function so that the new password Ki = K79, K78 .... K0 is determined by rotating the following bit:

Ki =K63 K62 ....K0 =K79 K78 ....K0 (1)

After performing this spin in the input block, the following operations must be performed for each new Ki-generated sub-password:

Rotate bits for the input password:

[K79 K78 ....K0] =[K18 K17 ....K20K19] (2)

Substitution using S-Box for the nibble from $k78$ to $k76$ of the password:

[K79 K78 K77K76]=S [K79 K78 K77K76] (3)

Add or combine nibbles k19 to k15 of password with pie counter, by adding finite field GF (21) or XOR process:

[K19 K18 K17K16K15]=

[K19 K18K17K16K15] $\oplus Round\ Counter$ (4)

This function allows the generation of blocks of useful information as secondary passwords from the system password K. The first Nk words from this array contain the password used for encryption, where the user password is set to the W array while the rest of the words are generated from these first Nk words [4, 8]. This function takes consecutive bytes of the sequence derived from the password expansion function and assigns it to each Ki sub-password, to form blocks of the same size as the array of status. This means that it takes $Nb*4$ bytes for each round, here Nb is 16.

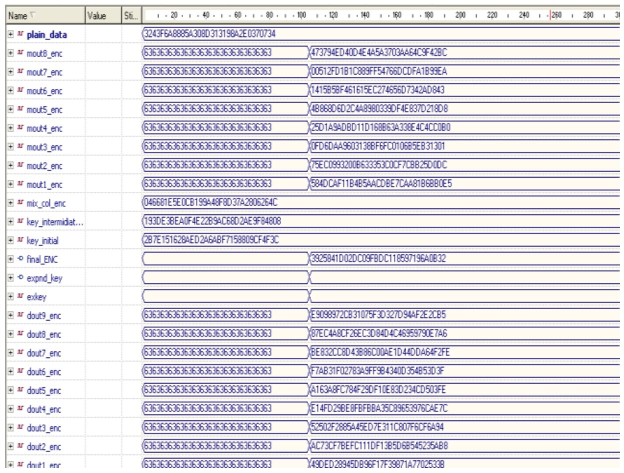
The password (password expansion) for the decryption process is created in the same way as for the decryption process. The difference is in the password selection function. In the decryption process, the password list blocks are taken from the final values to the initial values, which is the user's personal password. This means that the last sub-password that Ki used to decode will be the first to decode [4,8]. Therefore, the encryption process must include all rounds of password creation to start from this last Ki password, and perform the reverse process until reaching the original password. Therefore, the circular counter should be descending and mix in each round with the previously indicated sting.

IV. Simulation Results

Upon completion of implementation, the devices are measured on each functional block of the description and compared to the reference, which was created by the people who uploaded the algorithm. It has been determined that there has been a significant improvement in hardware usage.

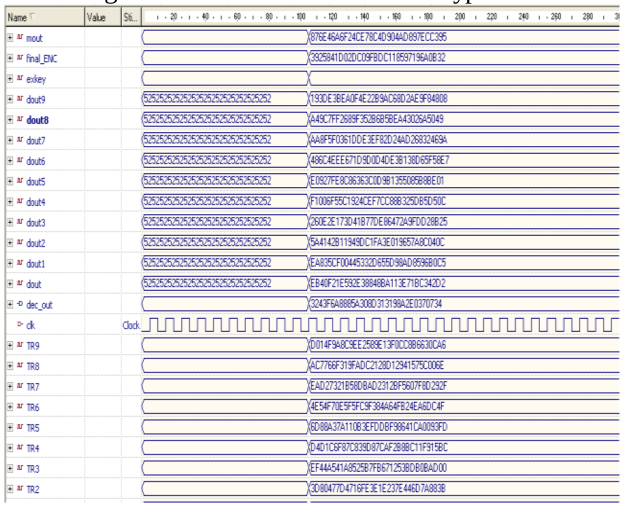
A 128-bit block of data is considered for the, Designed system given as

Plain Text : "3243f6a8885a308d313198a2e0370734". The Initial Key considered is "2b7e151628aed2a6abf7158809cf4f3c" and the encrypted output is "3925841d02dc09fbdc118597196a0b32". The encrypted output is a input to the decryption. After the Whole Process The Obtained Decrypted data is given as DEC_output: "3243f6a8885a308d313198a2e0370734". This Shows That the Data is Recovered exactly as Input (Plaintext) . The Intermediate Keys ,subbytes output ,shift rows Output and mixcolumn outputs .



```
Name               Value  Sti...
plain_data                       3243F6A8885A308D313198A2E0370734
mout8_enc                        63636363636363636363636363636363  473794ED40D4E4A5A3703AA64C9F42BC
mout7_enc                        63636363636363636363636363636363  00512FD1B1C889FF54766DCDFA1B99EA
mout6_enc                        63636363636363636363636363636363  1415B5BF461615EC274656D7342AD843
mout5_enc                        63636363636363636363636363636363  4B868D6D2C4A8980339DF4E837D218D8
mout4_enc                        63636363636363636363636363636363  25D1A9ADBD11D168B63A338E4C4CC0B0
mout3_enc                        63636363636363636363636363636363  0FD6DAA9603138BF6FC0106B5EB31301
mout2_enc                        63636363636363636363636363636363  75EC0993200B633353C0CF7CBB25D0DC
mout1_enc                        63636363636363636363636363636363  584DCAF11B4B5AACDBE7CAA81B6BB0E5
mix_col_enc                      046681E5E0CB199A48F8D37A2806264C
key_intermidiat...               193DE3BEA0F4E22B9AC68D2AE9F84808
key_initial                      2B7E151628AED2A6ABF7158809CF4F3C
final_ENC                                                          3925841D02DC09FBDC118597196A0B32
expnd_key
exkey
dout9_enc                        63636363636363636363636363636363  E9098972CB31075F3D327D94AF2E2CB5
dout8_enc                        63636363636363636363636363636363  87EC4A8CF26EC3D84D4C46959790E7A6
dout7_enc                        63636363636363636363636363636363  BE832CC8D43B86C00AE1D44DDA64F2FE
dout6_enc                        63636363636363636363636363636363  F7AB31F02783A9FF9B4340D354B53D3F
dout5_enc                        63636363636363636363636363636363  A163A8FC784F29DF10E83D234CD503FE
dout4_enc                        63636363636363636363636363636363  E14FD29BE8FBFBBA35C89653976CAE7C
dout3_enc                        63636363636363636363636363636363  52502F2885A45ED7E311C807F6CF6A94
dout2_enc                        63636363636363636363636363636363  AC73CF7BEFC111DF13B5D6B545235AB8
dout1_enc                        63636363636363636363636363636363  49DED28945DB96F17F39871A7702533B
```

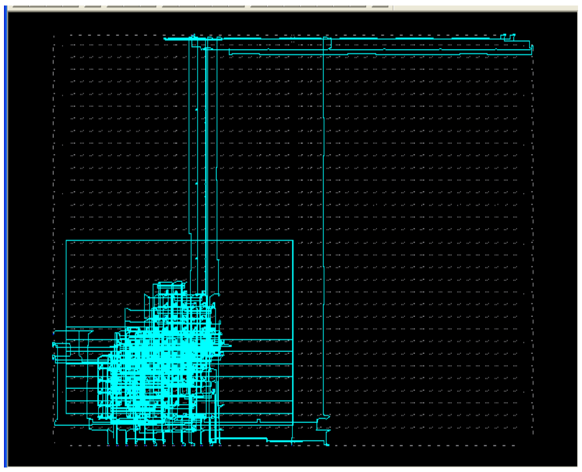
Fig 4.1 Simulation Results of Encryption



```
Name      Value  Sti...
mout                                               876E46A6F24CE78C4D904AD897ECC395
final_ENC                                          3925841D02DC09FBDC118597196A0B32
exkey
dout9            52525252525252525252525252525252  193DE3BEA0F4E22B9AC68D2AE9F84808
dout8            52525252525252525252525252525252  A49C7FF2689F352B6B5BEA43026A5049
dout7            52525252525252525252525252525252  AA8F5F0361DDE3EF82D24AD26832469A
dout6            52525252525252525252525252525252  486C4EEE671D9D0D4DE3B138D65F58E7
dout5            52525252525252525252525252525252  E0927FE8C86363C0D9B1355085B88E01
dout4            52525252525252525252525252525252  F1006F55C1924CEF7CC888325DB5D50C
dout3            52525252525252525252525252525252  260E2E173D41B77DE86472A9FDD28B25
dout2            52525252525252525252525252525252  5A41428119490C1FA3E019657A8C040C
dout1            52525252525252525252525252525252  EA835CF00445332D655D98AD8596B0C5
dout             52525252525252525252525252525252  EB40F21E592E38848BA113E71BC342D2
dec_out                                            3243F6A8885A308D313198A2E0370734
clk       Clock
TR9                                                D014F9A8C9EE2589E13F0CC8B6630CA6
TR8                                                AC7766F319FADC2128D12941575C006E
TR7                                                EAD27321B58DBAD2312BF5607F8D292F
TR6                                                4E54F70E5F5FC9F384A64FB24EA6DC4F
TR5                                                6D88A37A110B3EFDDBF98641CA0093FD
TR4                                                D4D1C6F87C839D87CAF2B8BC11F915BC
TR3                                                EF44A541A8525B7FB671253BDB0BAD00
TR2                                                3D80477D4716FE3E1E237E446D7A883B
```

Figure 4.2 Simulation Results of Decryption

The figure shows the simulation result showing light security coding using PRESENT security algorithm with 32 iterations as the update value.



.

Fig 4.3 Showing the implementation of proposed design on to the targeted FPGA

Figure depicts resource use for FPGA (Xcv300-6bg432) and its guidance for light security encryption implemented using the PRESENT algorithm. The directive is implemented using the FPGA editor. Note that the proposed lightweight security coding uses a relatively smaller resource using the PRESENT algorithm compared to the secure algorithms in the target FPGA.

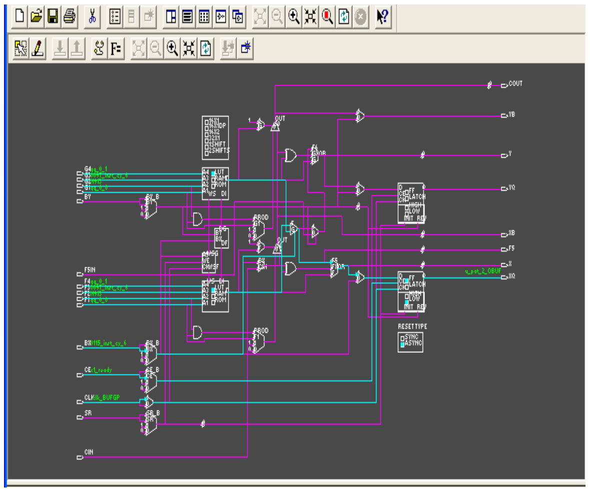


Fig. 4.4 logical placement for the implemented Light weight security coding

Figure Showing the Logical Block implementation in a CLB taken from the proposed design as shown above, targeting (Xcv300-6bg432) FPGA generated on FPGA Editor tool.

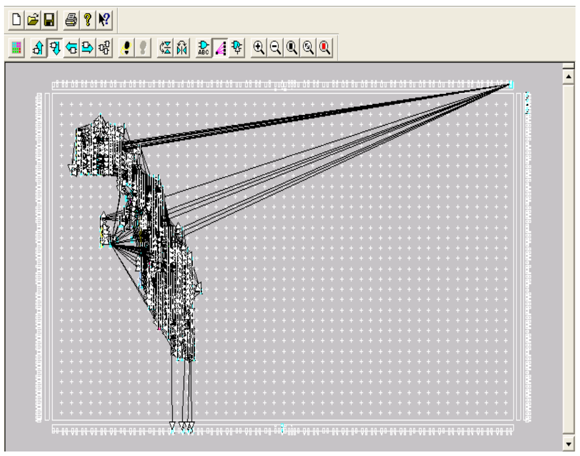


Fig4.5 Floor plan for the proposed design developed on FPGA Floor planner tool.

Figure shows the floor planning of the implemented system for Light weight security coding using PRESENT algorithm onto the targeted FPGA. The floor planning gives the net list interconnects between each two node in a given programmed FPGA architecture.

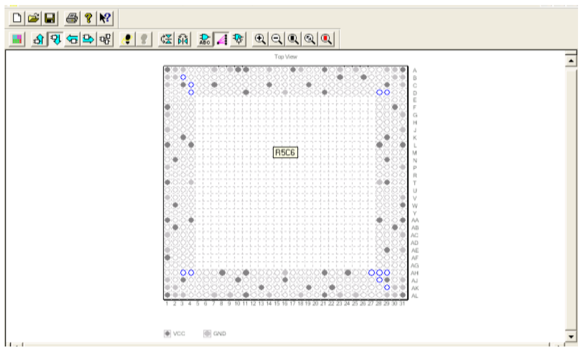


Fig 4.6 chip view for the proposed design developed on FPGA floor Planner tool.

Figure shows the virtual chip configuration for the implemented Light weight security coding using PRESENT algorithm . The figure shows the dedicated pins, VCC and GND pins for the implemented design.

V. Conclusions

Reduced hardware use by 30% compared to previous FPGA applications. Hardware implementation using the standard VHDL programming language decreased 26.89% in resource use on Xilinx Spartan 3 FPGA. It is worth noting that the reference application uses the VHDL standard and can be copied to any third-party FPGA, thus we expect equivalent reduction on it.

VI. REFERENCES

[1] J. Attridge, *"An overview of hardware security modules*", SANS Institute, Info Sec Reading Room, **1** (2002), no. 1, 1-10.

[2] H. A. Alkhzaímí and M. M. Lauridsen, "*Cryptanalysis of the Simon family of block ciphers*", Technical University of Denmark, Vol. 1, 2013, no. 1, 1-26.

[3] P. Valla and J. Kaps, "*Lightweight cryptography for FPGAs", *Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on, 2009, 225- 230.

[4] A. Bogdanov, L. Knudsen, G. Leander, and C. Paarl, A. Poschmann, M. J. B. Robshaw, Y. Seurin, C. Vikkelsoe, "*PRESENT: An Ultra-Lightweight Block Cipher, Chapter in Cryptographic Hardware and Embedded Systems* - *CHES 2007*, Springer-Verlag Berlin Heidelberg, 2007, Ch. 5, 450-466.

[5] R. Azuero, E. Jacinto and J. Castano, "*A low-memory implementation of 128 aes for 32 bits architectures*", in *En Congreso Argentino de Sistemas Embebidos CASE 2012*, 2012, 67-73.

[6] M. Kumar and A. Singhal, "*Efficient implementation of advanced encryption standard (AES) for ARM based platforms"*, 1st International Conference on Recent Advances in Information Technology (RAIT), 2012, 23-27.

[7] K. M. Abdellatif, R. Chotin-Avot and H. Mehrez, "Lightweight and compact solutions for secure reconfiguration of FPGAs", 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2013, 1-4.

[8] J. Pospiil and M. Novotny, "*Evaluating cryptanalytical strength of lightweight cipher PRESENT on reconfigurable hardware*", 15th Euromicro Conference on Digital System Design (DSD), 2012, 560-567.

[9] E. Kavun and T. Yalcin, "*RAM-based ultra-lightweight FPGA implementation of PRESENT"*, 2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011, 280-285.

[10] A. Aysu, E. Gulcan and P. Schaumont, "*Simon says: Break area records of block ciphers on FPGAs*," IEEE Embedded Systems Letters, **6** (2014), no. 2, 37-40.

[11] S. Feizi, A. Ahmadi and A. Nemati, "*A hardware implementation of Simon cryptography algorithm"*, 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE), 2014, 245-250.

[12] Shuangqing Wei, J. Wang, R Yin, and J. Yuan, "*Trade-off between security and performance in block ciphered systems with erroneous ciphertexts*", **8** (2013), no. 4, 636-645.

[13] C.-P. Fan and J.-K. Hwang, *"Implementations of high throughput sequential and fully pipelined AES processors on FPGA*", International Symposium on Intelligent Signal Processing and Communication Systems, 2007. ISPACS 2007, 353-356.

[14] Chih-Peng Fan and Jun-Kui Hwang, *"Implementations of high throughput sequential and fully pipelined AES processors on FPGA"*, International Symposium on Intelligent Signal Processinq and Comunication Systems ISPACS 2007, 2007, 353-356.

[15] J. J. Tay, M. M. Wong and I. Hijazin, *"Compact and low power AES block cipher using lightweight key expansion mechanism and optimal number of s- boxes"*, 2014 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), 2014, 108-114.