2016 9th International Conference on Developments in eSystems Engineering

A compact S-Box module for 128/192/256-bit symmetric cryptography hardware

Otávio de Souza Martins Gomes
Instituto Federal de Minas Gerais (IFMG)
Formiga, Brazil
otavio.gomes@ifmg.edu.br

Robson Luiz Moreno
Universidade Federal de Itajubá (UNIFEI)
Itajubá, Brazil
moreno@unifei.edu.br

Abstract— **This article describes a compact implementation in hardware of S-Boxes for Twofish Algorithm in Field Programmable Gate Array - one of the Advanced Encryption Standard (AES) finalists. The core was implemented using Altera Quartus Cyclone board. The code is totally portable and can be used in any FPGA family because it was developed using the VHDL patterns. The function was implemented for 128-bit word and 128, 192 and 256-bit key. The main goal of this work was the implementation of a compact and modular S-Box module for Twofish algorithm that can find a wide range of applications and that can be choosen as an alternative from AES-Rijndael.**

Keywords: Cryptography, Twofish, S-Box, FPGA, AES.

I. INTRODUCTION

In 1997, the NIST (National Institute of Standards and Technology) released a contest to choose a new symmetric cryptograph algorithm that would be called Advanced Encryption Standard (AES) to be used to protect confidential data in the USA.

The algorithm should meet few requirements such as copyright free, faster than the 3DES, cryptography of 128 bit blocks using 128, 192 and 256 bit keys, possibility of hardware and software implementation, among others. The three finalists were Rijndael, Twofish and Serpent. Each one has its own design features to achieve the requirements. After analysis by cryptography experts, the winner was chosen: Rijndael. The algorithm was created by the Belgians Vincent Rijmen and Joan Daemen [1,2].

With the advance of technology and the increment of hardware utilization for cryptography interfaces, it is interesting to analyze Twofish implementation in comparison with the winner, Rijndael. Some researchers are re-evaluating the characteristics of the finalists to achieve the best algorithm for their applications [3-6].

This work was implemented in FPGA due to its flexibility and reconfiguration capability [7]. A reconfigurable device is very convenient for a cryptography algorithm since it allows cheap and quick alterations.

Section II provides a brief introduction of Twofish and its processing phases. Section III describes the chosen FPGA and the circuit implementation. Section IV compares the results of this work with others presented in the literature. Section V presents the conclusions of this work and finally Section VI shows the authors expectations and proposals for future work.

II. TWOFISH ALGORITHM

In order to better understand the Twofish structure, it is based on Blowfish algorithm. The Twofish is a 128-bit block cipher and supports key sizes of 128, 192 and 256-bits. It uses a Feistel function for 16 rounds to provide its strength, that consists of a fundamental block called F function and is key dependent.

This algorithm receives an input block of 128-bits that is divided into 4 blocks of 32-bits. These 4 blocks are processed with the first 4 keys. Like Serpent algorithm, a pair of pre-processing and post-processing blocks are used, called input and output whitening. After the input processing, the lower part will be used as the input of Feistel function, which consists of 4 sequential operations, Primary rotation, g function, Pseudo-Hadamard Transforms (PHT), the combination of the key and the final rotation [8]. The encryption and decryption circuits are almost the same except that the internal keys are applied in a reverse order when decrypting. This structure makes the encryption and decryption units look very similar.

Internally of F function, there are g functions that consist of key dependent S-boxes. Each S-box contains permutations q0 and q1, that are fixed permutations on 8-bit values. Each S-box has its own way to calculate the q functions. Between each calculation of q function it is necessary to execute a XOR operation with the correspondent S Key. For 128-bit key, this is done as follows:

$$y_0 = s_0(x_0) = q_1\,[q_0[q_0[x] \text{ XOR } s_{0,0}] \text{ XOR } s_{1,0}]$$
$$y_1 = s_1(x_1) = q_0\,[q_0[q_1[x] \text{ XOR } s_{0,1}] \text{ XOR } s_{1,1}]$$
$$y_2 = s_2(x_2) = q_1\,[q_1[q_0[x] \text{ XOR } s_{0,2}] \text{ XOR } s_{1,2}]$$
$$y_3 = s_3(x_3) = q_0\,[q_1[q_1[x] \text{ XOR } s_{0,3}] \text{ XOR } s_{1,3}]$$

The results are multiplied by 4X4 MDS (Maximum Distance Separable) using Galois Field- GF (2^8). The primitive polynomial is $x^8+x^6+x^5+x^3+1$ [7].

The Pseudo-Hadamard transform is a simple addition with module 2^{32}. The last operation is an addition that is performed with the output from PHT and the round keys.

A. *Feistel Network*

The g function is executed on Feistel Network and it receives a 32-bit block which is divided into 4 blocks of 8-bit then is inserted into independent S-boxes. Each S-box is a bijective function which takes 8 bits as input and returns an 8-

bit output and use keys S0 and S1 to process the XOR operations between the q functions. Each of the S-boxes contains permutation structure q0 and q1, which works as follows:

$$\begin{aligned}
&a_0, b_0 = [x/16];\ x \bmod 16\\
&a_1 = a_0 \oplus b_0\\
&b_1 = a_0 \oplus \mathrm{ROR4}(b_0; 1) \oplus\ 8a_0 \bmod 16\\
&a_2, b_2 = t_0[a_1], t_1[b_1]\\
&a_3 = a_2 \oplus b_2\\
&b_3 = a_2 \oplus \mathrm{ROR4}(b_2; 1) \oplus 8a_2 \bmod 16\\
&a_4, b_4 = t_2[a_3], t_3[b_3]\\
&y = 16\, b_4 + a_4
\end{aligned}$$

where: ROR indicates rotation right of the first argument by amount of the second argument.

The q function works according Figure 1, that uses tables of values for q0 and q1, for each t*x* function.

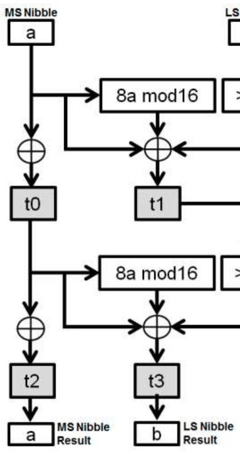


Figure 1 – q Function

Each S-Box will work according to the size of the key: 128, 192 or 256-bit. Using 256-bit key (k=4), the S-Box will calculate all the functions. With 192-bit key, only the last two functions will be used (k=3). The 128-bit key has k=2.

If k = 4 we have

$$\begin{aligned}
y_{3;0} &= q_1[y_{4;0}] \oplus l_{3;0}\\
y_{3;1} &= q_0[y_{4;1}] \oplus l_{3;1}\\
y_{3;2} &= q_0[y_{4;2}] \oplus l_{3;2}\\
y_{3;3} &= q_1[y_{4;3}] \oplus l_{3;3}
\end{aligned}$$

If $k \geq 3$ we have

$$\begin{aligned}
y_{2;0} &= q_1[y_{3;0}] \oplus l_{2;0}\\
y_{2;1} &= q_1[y_{3;1}] \oplus l_{2;1}\\
y_{2;2} &= q_0[y_{3;2}] \oplus l_{2;2}\\
y_{2;3} &= q_0[y_{3;3}] \oplus l_{2;3}
\end{aligned}$$

In all cases we have

$$\begin{aligned}
y_0 &= q_1[q_0[q_0[y_{2;0}] \oplus l_{1;0}] \oplus l_{0;0}]\\
y_1 &= q_0[q_0[q_1[y_{2;1}] \oplus l_{1;1}] \oplus l_{0;1}]\\
y_2 &= q_1[q_1[q_0[y_{2;2}] \oplus l_{1;2}] \oplus l_{0;2}]\\
y_3 &= q_0[q_1[q_1[y_{2;3}] \oplus l_{1;3}] \oplus l_{0;3}]
\end{aligned}$$

B. *Key Expansion*

Each round of Feistel network needs two sub-keys to each round. The key schedule is the function responsible for expand the initial key in 40 sub-keys. The first 8 keys, generated by the key schedule, are used for input and output whitening (K0-K7), other keys are used during the 16 rounds (K8-K39).

The key schedule for keys K0-K39 uses the same primitives as the round function. The S-Boxes keys are generated by Reed Solomon (RS) mapping function. The keys S0, S1 are obtained by matrix multiplication over Galois Field-GF (2^8) using the 128-bit user key. The primitive polynomial is $x^8 + x^6 + x^3 + x^2 + 1$. The RS matrix is a 4x8 matrix derived from the RS code [7], as given:

$$\begin{pmatrix}
01 & A4 & 55 & 87 & 5A & 58 & DB & 9E\\
A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5\\
02 & A1 & FC & C1 & 47 & AE & 3D & 19\\
A4 & 55 & 87 & 5A & 58 & DB & 9E & 03
\end{pmatrix}$$

The keys could be calculated previously or simultaneously with the rounds. The second way is called on-the-fly calculation. Twofish algorithm provides less noise sensitivity due to the lower key sensitivity and plaintext sensitivity [6].

S-Boxes are used on Feistel Network (g function) and Key Expansion (h function).

III. Implementation

The Twofish could be implemented in modules, where all the modules could be independently tested and characterized, and therefore they can be used in any combination, according to its purpose.

In order to conduct tests on S-Box blocks, it was assembled for 128/192/256-bit keys in an Altera Cyclone IV E FPGA board. The test results are presented in Section IV.

The hardware was developed using VHDL to fit any FPGA. The code is totally portable and can be used in any FPGA family because was developed using the HDL patterns.

Figure 2 shows how S-Boxes (g function) work for 128/192/256-bit key.

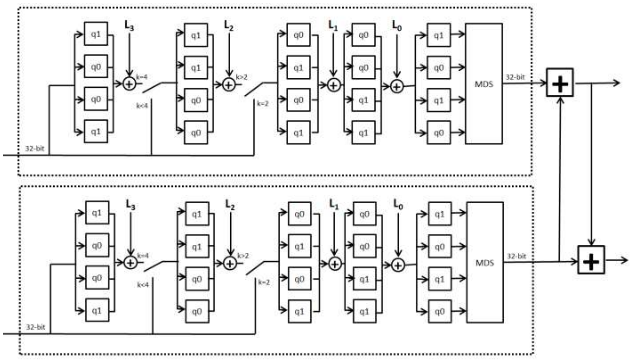


Figure 2 – g Function

The Key Expand block, called h function, is developed with the same basic blocks, but there are some transformations that make the blocks g and h different [7].

Figure 3 shows the entity declaration of block that implements q function. This block has some input ports to control its utilization. A high level input on the signal *load* represent that input signals are ready to be read. The signal *in_q* receives the value that will be used on q table, defining q0 or q1.

```
entity q_unit is
    port(
        clk       : IN std_logic;
        rst       : IN std_logic;
        in_q      : IN std_logic;
        load      : IN std_logic;
        in_value  : IN std_logic_vector(7 DOWNTO 0);
        out_ready: OUT std_logic;
        out_q     : OUT std_logic_vector(7 DOWNTO 0)
    );
end q_unit;
```

Figure 3 – q Unit block entity declaration

The signal *in_value* will receive the input byte that will be processed and *out_q* gives the result. When the output data is ready to be read the signal *out_ready* will be on high level.

Figure 4 shows the inputs and outputs ports of the S-Box hardware developed.

```
entity sBox is
    port(
        clk       : IN std_logic;
        rst       : IN std_logic;
        in_load   : IN std_logic;
        in_data   : IN std_logic_vector (31 DOWNTO 0);
        in_L0     : IN std_logic_vector(31 DOWNTO 0);
        in_L1     : IN std_logic_vector(31 DOWNTO 0);
in_L2             : IN std_logic_vector(31 DOWNTO 0);
in_L3             : IN std_logic_vector(31 DOWNTO 0);
in_selSecLvl: IN std_logic_vector (1 DOWNTO 0);
out_data  : OUT std_logic_vector(31 DOWNTO 0);
        out_ready : OUT std_logic
    );
end sBox;
```

Figure 4 – S-Box block entity declaration

The signals *clk* and *rst* are common to all top blocks. The signal *in_load* will be on high level when the inputs are stable to be read. The signal *in_selSecLvl* is used to select the security level of cryptography (128, 192 or 256) and is loaded with the crypto key and the data. The signals *in_L0* to *in_L3* will be used to load the 32-bit keys that are used during the process. When all the data is ready to be read, the output data signal will be put in *out_data* and the signal *out_ready* will be on high level.

The S-Box top level view could be represented by Figure 5. There are 4 instances of q unit block, one for each level of 32-bit input word.

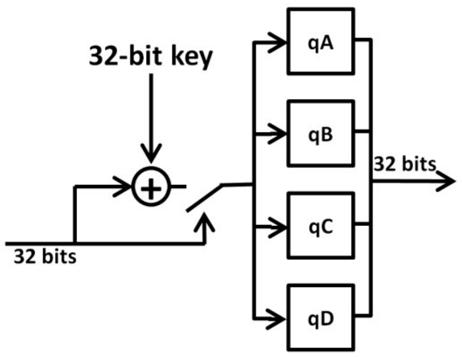


Figure 5 – S-Box compact module

A state machine controls this compact module inserting the correct words and keys for each state, according to the signal *in_selSecLvl*. In this implementation, the change of key-size will not change the size of the module.

IV. Tests and Hardware verification

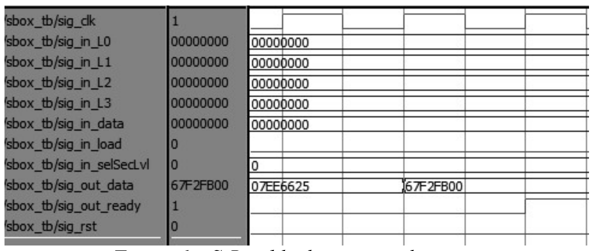
The hardware was tested and the S-Box function was verified according to the standards and test vectors of Twofish documentation design [7,8]. All the results were obtained according to the benchmarks.

Table 1 shows some information about the synthesis. The compact S-Box developed uses less logic than any implementation using the configuration according to the Figure 2.

TABLE 1 – SYNTHESIS RESULTS

S-Boxes	Logic Elements	Key Size (bits)
Compact	1,064	128 / 192 / 256
Standard	1,937	128
	2,627	192
	3,290	256

Figure 6 shows the simulation results of the S-Box with its inputs and outputs. The signal *sig_out_ready* in high level represents that the data on *sig_out_data* can be read as a result (0x67F2FB00).



```
/sbox_tb/sig_clk           1
/sbox_tb/sig_in_L0         00000000   00000000
/sbox_tb/sig_in_L1         00000000   00000000
/sbox_tb/sig_in_L2         00000000   00000000
/sbox_tb/sig_in_L3         00000000   00000000
/sbox_tb/sig_in_data       00000000   00000000
/sbox_tb/sig_in_load       0
/sbox_tb/sig_in_selSecLvl  0          0
/sbox_tb/sig_out_data      67F2FB00   07EE6625   67F2FB00
/sbox_tb/sig_out_ready     1
/sbox_tb/sig_rst           0
```

Figure 6 – S-Box block inputs and outputs

The environment of tests was the Altera Cyclone IV E board (EP4CE115F29C7), Altera Quartus II 64-Bit Version 13.1.0 and Altera ModelSim version 10.1d.

V. Results Comparison

According to the results in AES competition, Twofish is a strong algorithm using hardware implementation [10,11]. The importance of this analysis on one of the finalists of AES competition could be seen through the number of works that is analyzing and using the AES finalists in many applications [3-6].

A significant advantage of this implementation is the possibility to change the key size with the same S-Box hardware to execute the operations. As can be seen on Table 1, compact S-Box has the smallest utilization of Logic Elements for all key sizes, while 128-bit implementation has almost two times the Logic Elements utilization.

The utilization of Twofish in cryptography interfaces could offer some advantages. Using this algorithm with a 128 bits block size, the dictionary attack will require 2^{128} different plaintexts to allow the attack to decrypt arbitrary message under an unknown key [12].

Another advantage of Twofish algorithm is the possibility to execute the expansion key on-the-fly, in parallel with the encryption process, increasing the noise signal against side-channel attacks [13]. The S-Boxes could be implemented in a regular way according to the Figure 2 or in a compact way according to this work (Figure 5). Table 1 shows the use of reconfigurable logic and the advantages of the compact S-Box. Using compact S-Box the implementation of pipelined applications could use less logic for each instance of Twofish.

VI. Conclusions

This article presented a compact S-Box implementation for Twofish 128/192/256 cryptography hardware that can have many applications and uses less logic than regular implementations. The circuit implementation can be customized to a wide range of applications. It can support new applications through Twofish algorithm implementation.

The authors intend to use this work as part of larger projects, including smart metering in power systems and security interface in data communications.

Acknoledgment

The authors acknowledge CAPES and CNPq for their financial support.

References

[1] FIPS FIPS-197, Federal Information Processing Standards Publication FIPS-197, Advanced Encryption Standard (AES), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 1999.

[2] Daemen, J. and Rijmen, V., The design of Rijndael: AES — The Advanced Encryption Standard. Springer-Verlag, 2002.

[3] RIZVI, S.A.M, HUSSAIN, S. Z., WADHWA, N. “Performance Analysis of AES and TwoFish Encryption Schemes”. 2011 International Conference on Communication Systems and Network Technologies.

[4] VERMA, H. K., SINGH, R. K. “Performance Analysis of RC6, Twofish and Rijndael Block Cipher Algorithms”. International Journal of Computer Applications (0975 – 8887), Volume 42– No.16, March 2012.

[5] MUSHTAQUE, A.; DHIMAN, H.; HUSSAIN, S.; MAHESHWARI, S. “Evaluation of DES, TDES, AES, Blowfish and Two fish Encryption Algorithm: Based on Space Complexity”. International Journal of Engineering Research & Technology (IJERT), Vol. 3, Issue 4, April – 2014.

[6] NAEEMABADI, M.; ORDOUBADI, B. S.; DEHNAVI, A. M.; BAHAADINBEIGY, K. “Comparison of Serpent, Twofish and Rijndael encryption algorithms in tele-ophthalmology system”. Advances in Natural and Applied Sciences, Pages: 137-149, April-2015.

[7] SCHNEIER , B.; KELSEY, J.; WHITING, D.; WAGNER, D.; HALL, C.; FERGUSON, N. “Twofish: A 128-bit Block Cipher”; 15 June, 1998. Technical Report available at http://www.counterpane.com/twofish.html

[8] SCHNEIER B., “Twofish- A New Block Cipher”, Technical Report available at http://www.schneier.com/twofish.html

[9] GOMES, O. S. M.; PIMENTA, T. C.; MORENO, R. L., "A Highly Efficient FPGA Implementation", 2nd Latin America Symposium on Circuits and Systems(LASCAS-2011), February 2011.

[10] PESCATORE, J. Using Hardware-Enabled Trusted Crypto to Thwart Advanced Threats. A SANS Whitepaper Written, September/2015

[11] SCHNEIER, B.; WHITING, D. A Performance Comparison of the Five AES Finalists. 2000.

[12] Anderson, R., Biham, E. and Knudsen, L., “Serpent: A Proposal for the Advanced Encryption Standard", NIST AES Proposal. 1998.

[13] Kocher, P. C.; Jaffe, J. and Jun, B. Diferential Power Analysis. In Michael Wiener, editor, Advances in Cryptology - CRYPTO '99 , volume 1666 of Lecture Notes in Computer Science , pages 388-397. Springer, 1999.