Malaysian Journal of Computing and Applied Mathematics 2019, Vol 2(2): 44-59

(Online)

MALAYSIAN JOURNAL OF COMPUTING AND APPLIED MATHEMATICS

Implementation and Statistical Tests of a Block Cipher Algorithm MISTY1*

Moussaoui Sarah[a], Zeghdoud Sabrina[b], Allailou Boufeldja[c]

[a]Laboratoire Centrale de R&D, ECRMT, Algiers, Algeria
[b]Laboratoire de Recherche en Electonique et Ingenieurie des Systemes Electroniques, AMC, Algiers, Algeria
[c]Institue de Recherche et Developpement, ESCH, Algiers, Algeria

*Corresponding Authors: moussaouisarah1105@gmail.com

Received: 17/06/2019, Accepted: 18/11/2019

Abstract

With the development of communication network and new information technologies, the volume of data exchanged is growing, particularly with the of IoTs. There security has become a major concern, especially in sensitive activities. Such security requirements call for efficient cryptographic encryption algorithms, with a small hardware footprint. The current trend is towards light cryptographic algorithms (lightweight). These are designed for power systems with limited storage capacity. This paper proposes the study, hardware implementation and statistical test of block cipher algorithm MISTY1. Its optimized version for a hardware implementation is known as KASUMI, used in the context of 3GPP compliant mobile networks, including 2G (GSM) and 3G (UMTS).

Keywords: Cryptography, Block cipher, MISTY1, KASUMI, Hardware Implementation, FPGA, NIST statistical test.

Introduction

MISTY1 (*Mitsubishi Improved Security Technology*) is a 64bit block cipher with a 128-bit secret key, and a variable number of rounds, based on a Feistel scheme. Its detailed description and specifications were first published in Japan in 1996 (Matsui, 2000) before being presented at "*The international workshop of Fast Software Encryption* "in 1997 (Lai, 1994).

Its optimized version for a hardware implementation is known as KASUMI (Kitsos, et al., 2004), used in the context of 3GPP compliant mobile networks, including 2G (GSM) and 3G (UMTS). As a result, KASUMI and MISTY1 are very similar, "KASUMI" is also the Japanese translation of the word "MISTY" (foggy). In this paper, we describe the *MISTY1* algorithm, the encryption-decryption procedure, the key management process and the component functions.

We suggest a practical application which consists in the implementation of the *MISTY1* algorithm on *FPGA* map, using *VHDL* programming language.

In this context, we propose a purely combinatorial implementation approach. The algorithmic programming of the suggested approach is based on the subdivision of the structure of the *MISTY1* algorithm into three essential parts, namely, the key management process, the management of the encryption sub-keys and the encryption-decryption procedure itself.

Behavioral simulations under *ISIM* were performed to validate the implemented program. Subsequently, a hardware check on the *FPGA* card, through the integrated logic analyzer *Chipscope Pro*, was also performed.

This paper is organized as follows: in the first section, we present the algorithm in question as well as the different equations that compose it. In the second section, we present the adopted implementation approach, the simulations under *ISIM* of the designed program of the *MISTY1* algorithm. In the next section, we show a hardware check on map *FPGA*. Then, we will present the execution results of the algorithm taken from the integrated logic analyzer (*ChipScope*). Finally, we will present the NIST statistical results.

Description of MISTY1

MISTY1 is a Fiestel block cipher with a 64-bits block and a 128-bits key. It is among the final NESSIE portfolio of block ciphers (Preneel, 2011), and has been recommended for Japanese e-Government cipher by the CRYPTREC project (Iman and Yamagishi, 2011).

MISTY1 Structure

1) Encryption Process: Figure 1 presents *MISTY1* encrypting procedure for n tours. The plaintext of 64 bits ($P_{(64)}$) is divided into two parts, 32 bits on the left and 32 on the right, which will be transformed into a 64-bit encrypted text ($C_{(64)}$) by means of exclusive-OR logic operations (Xor), FO_i functions ($1 \le i \le n$) and FL_i ($1 \le i \le n+2$) functions (Matsui, 1997). The FO_i function uses two sub-keys, one of 64 bits (KO_i) and another of 48 bits (KI_i). The FL_i function uses a 32-bit sub-key (KL_i). Subkeys used during the encryption process are generated from the main secret key ($K_{(128)}$)

In the encryption process, the first step is to divide the 64-bit plaintext ($P_{(64)}$) into two 32-bit length parts, ie $P(64) = L_{0(32)} \,||\, R_{0(32)}$. Then the following operations are performed: For odd rounds ($i = 1,3,...,n-1$), we define:

$$R_i = FL_i(L_{i-1}, KL_i) \quad (1)$$

$$L_i = FL_{i+1}(R_{i-1}, KL_{i+1}) \oplus FO_i(R_i, KO_i, KI_i) \quad (2)$$

For pair rounds ($i = 2,4,...,n$), we define: :

$$R_i = L_{i-1} \quad (3)$$

$$L_i = R_{i-1} \oplus FO_i(R_i, KO_i, KI_i) \quad (4)$$

After the last round ($n = 8$), the FL function is applied to the two data L_n and R_n, to get:

$$R_{n+1} = FL_{n+1}(L_n, KL_{n+1}) \quad (5)$$

$$L_{n+1} = FL_{n+2}(R_n, KL_{n+2}) \quad (6)$$

The concatenation of the two resulting data $L_{n+1(32)}$ and $R_{n+1(32)}$ forms the ciphertext output.

$$C_{(64)} = L_{n+1(32)} \,||\, R_{n+1(32)} \quad (7)$$

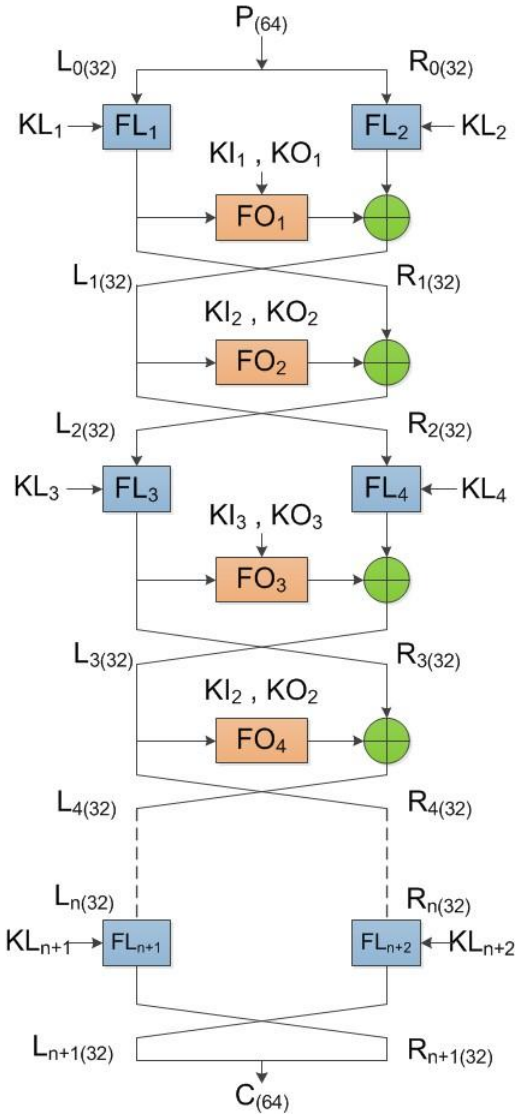

Figure 1. MISTY1 encryption process.

2) Decryption Process: MISTY1 decryption procedure is done in the same way as the encryption process, by inverting the order of the sub-keys and replacing the function *FL* by the function FL^{-1} (Matsui, 1997). Figure 2 illustrates the decryption process of the algorithm MISTY1 with *n* rounds (Matsui, 1997).

The 64-bit encrypted text ($C_{(64)}$ is split into two 32-bit parts, and will be transformed into a 64-bit plaintext ($P_{(64)}$) using the logical operation Xor and the two subfunctions FO_i ($n \leq i \leq 1$) and FL_i^{-1} $(n+2 \leq i \leq 1)$.

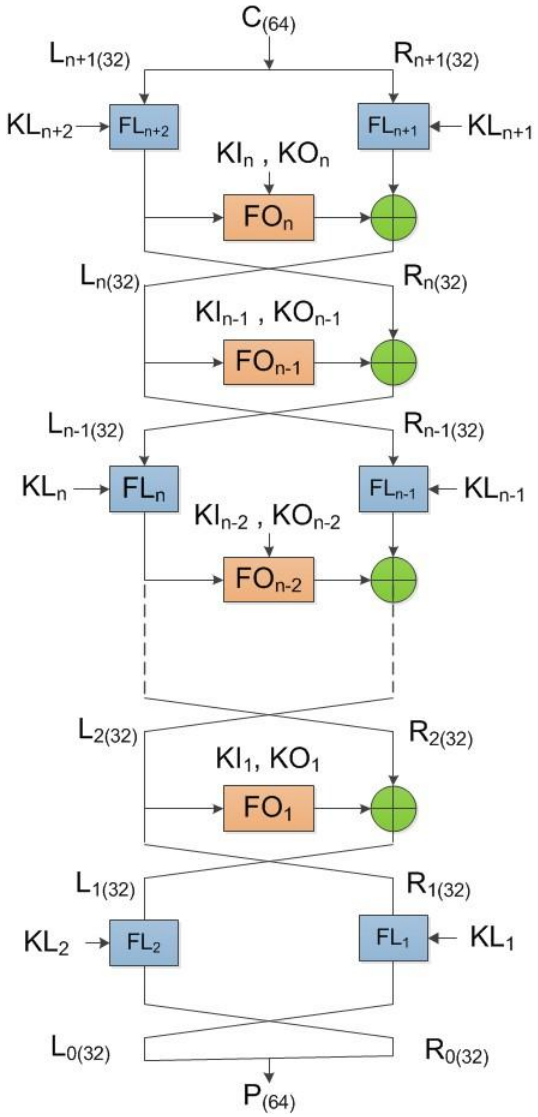

Figure 2. MISTY1 decryption process.

During the decryption process, the ciphertext ($C_{(64)}$) is divided into two 32-bit length parts ($C_{(64)} = L_{n+1(32)} \,||\, R_{n+1(32)}$), and will undergo the operations below:

For odd rounds ($i = n - 1,..., 3.1$), we define:

$$R_i = FL^{-1}_{i+2}\ (Li+1, KI_{i+2}) \quad (8)$$

$$L_i = FL^{-1}_{i+1}\ (R_{i+1}, KL_{i+1}) \oplus FO_i\ (R_i, KO_i, KI_i) \quad (9)$$

For pair rounds ($i = n,...,4,2$), we define :

$$Ri = Li+1 \quad (10)$$

$$L_i = R_{i+1} \oplus FO_i\ (R_i, KO_i, KI_i) \quad (11)$$

After the n^{th} round ($n = 8$), the FL^{-1} function is applied to the data L_1 and R_1, as follows:

$$R_0 = FL_2^{-1}(L_1, KL_2) \quad (12)$$

$$L_0 = FL_1^{-1}(R_1, KL_1) \quad (13)$$

Finally, the two data L_0 and R_0 are concatenated to give as output the plaintext $P_{(64)}$.

$$P_{(64)} = L_{0(32)} \,||\, R_{0(32)} \quad (14)$$

3) Encrypion key management: MISTY1 uses a 128-bit secret key ($K_{(128)}$), subdivided into eight sub-keys of 16 bits ($K_{1(16)},K_{2(16)},...,K_{8(16)}$) as follows

$$K_{(128)} = K_{1(16)} \,||\, K_{2(16)} \,||\, K_{3(16)} \,||\, K_{4(16)} \,||\, K5_{(16)} \,||\, K_{6(16)} \,||\, K7_{(16)} \,||\, K_{8(16)}. \quad (15)$$

The key management procedure yields a 128-bit sub-key ($K^0_{(128)}$), formed by concatenating eight 16-bit words (Matsui, 1997) ($K'_{1(16)}),K'_{2(16)}),...,K'_{8(16)})$). Figure 3 shows the key management procedure.

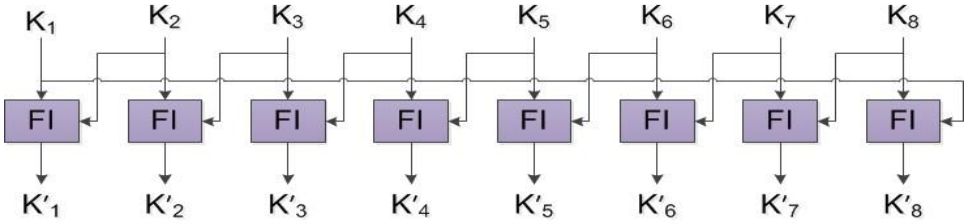

Figure 3. Key Management Scheme.

The K'_i $(1 \le i \le 7)$ are deduced from the following equation:

$$K_i'_{(16)} = FI(K_{i(16)},K_{i+1(16)}) \quad (16)$$

for $i = 8$, $K_8'_{(16)} = FI(K_{8(16)},K_{1(16)})$.

The correspondence between the sub-keys KO_{ij},KI_{ij},KL_{ij} used during the i^{th} round and the sub-keys K_i,K^0_i is given in the Table 1.

Table 1. Encryption Subkeys Management

KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	KL_{iL}	KL_{iR}
K_i	K_{i+2}	K_{i+7}	K_{i+4}	K'_{i+5}	K'_{i+1}	K'_{i3}	$K_{\frac{i+1}{2}}$ (*i* odd) $K'_{\frac{i}{2}+2}$ (*i* pair)	$K'_{\frac{i+1}{2}+6}$ (*i* odd) $K'_{\frac{i}{2}+4}$ (*i* pair)

Components of MISTY1

1) FL function: Figure 4 represents the logic diagram of the FL function, which uses 32-bit input data ($X_{(32)}$) and a 32-bit sub-key ($KL_{i(32)}$) are used (Matsui, 1997). The input data $X_{(32)}$ is divided into two 16-bit words ($X_{L(16)}$ and $X_{R(16)}$), where:

$$X_{(32)} = X_{L(16)} \,||\, X_{R(16)} \quad (17)$$

The sub-key $KL_{i(32)}$ is divided into two sub-keys of 16 bits, $KL_{iL(16)}$ and $KL_{iR(16)}$, where:

$$KL_{i(32)} = KL_{iL(16)} \,||\, KL_{iR(16)} \quad (18)$$

We define :

$$Y_{R(16)} = (X_{L(16)} \cap KL_{iL(16)}) \oplus X_{R(16)} \quad (19)$$

$$Y_{L(16)} = (Y_{R(16)} \cup KL_{iR(16)}) \oplus X_{L(16)} \quad (20)$$

The FL function outputs 32-bit data ($Y_{(32)}$), defined as follows:

$$Y_{(32)} = Y_{L(16)} \,||\, Y_{R(16)} \quad (21)$$

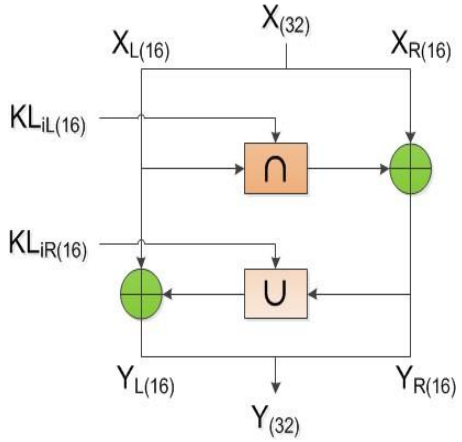

Figure 4. *FL* function.

2) FL^{-1} *function:* Figure 5 represents the logical scheme of the function *FL*$^{-1}$. The *FL*$^{-1}$ function uses a 32-bit input ($Y_{(32)}$) and a 32-bit sub-key ($KL_{i(32)}$) (Matsui, 1997).

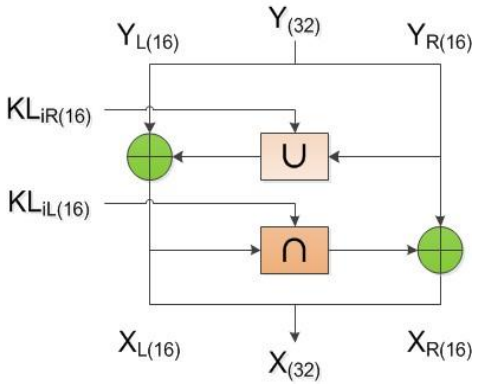

Figure 5. *FL*$^{-1}$ function.

The input data $Y_{(32)}$ is divided into two 16-bit words ($Y_{L(16)}$ and $Y_{R(16)}$), where:

$$Y_{(32)} = Y_{L(16)} \,||\, Y_{R(16)} \tag{22}$$

The sub-key $KL_{i(32)}$ is divided into two sub-keys of 16 bits, $KL_{iL(16)}$ and $KL_{iR(16)}$, where:

$$KL_{i(32)} = KL_{iL(16)} \,||\, KL_{iR(16)} \tag{23}$$

We define :

$$X_{R(16)} = (Y_{R(16)} \cup KL_{iR(16)}) \oplus Y_{L(16)} \tag{24}$$

$$X_{R(16)} = (X_{L(16)} \cap KL_{iL(16)}) \oplus Y_{R(16)} \tag{25}$$

The sub-key $KL_{i(32)}$ is divided into two sub-keys of 16 bits, $KL_{iL(16)}$ and $KL_{iR(16)}$, where:
The *FL*$^{-1}$ function outputs 32-bit data ($Y_{(32)}$), are defined as follows:

$$X_{(32)} = X_{L(16)} \,||\, X_{R(16)} \tag{26}$$

3) FO function: The *FO* function uses 32-bit input data ($Y_{(32)}$) and two sub-keys, one of 64 bits ($KO_{i(64)}$) and another of 48 bits (Matsui, 1997). The logic diagram of the FO function is shown in Figure 6.

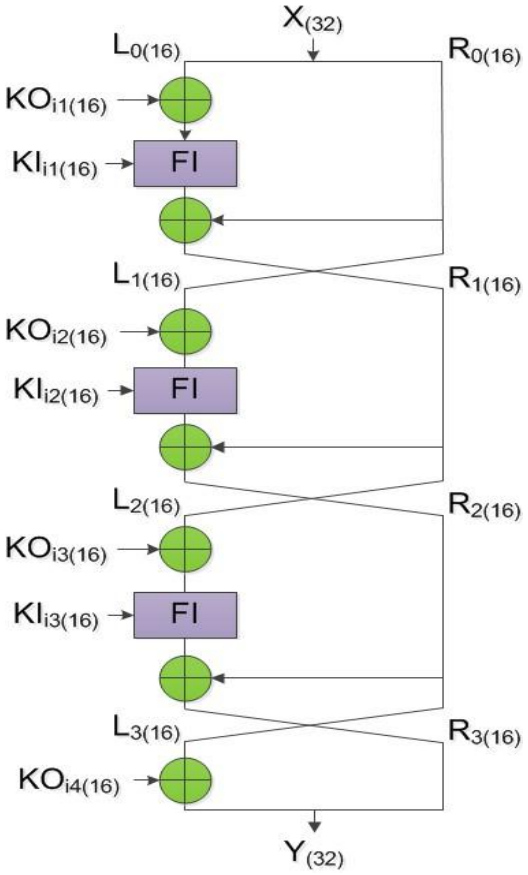

Figure 6. *FO* function.

The input data $X_{(32)}$ is divided into two half-words (16-bits) ($L_{0(16)}$,$R_{0(16)}$), where:

$$X_{(32)} = L_{0(16)} \,||\, R_{0(16)} \quad (27)$$

The two sub-keys are divided into 16-bit sub-keys:

$$KO_{i(64)} = KO_{i1(16)} \,||\, KO_{i2(16)} \,||\, KO_{i3(16)} \,||\, KO_{i4(16)}. \quad (28)$$

$$KI_{i(48)} = KI_{i1(16)} \,||\, KI_{i2(16)} \,||\, KI_{i3(16)}. \quad (29)$$

For an integer j, with (1 *leq j leq* 3) we define:

$$R_j = FI_{ij}(L_{j-1} \oplus KO_{ij}, KI_{ij}) \oplus R_{j-1} \quad (30)$$

$$L_j = R_{j-1} \quad (31)$$

Finally, the FO function outputs 32-bit data ($Y_{i(32)}$), is defined as follows:

$$Y_{i(32)} = (L_{3(16)} \oplus KO_{i4}) \text{ k } R_{3(16)} \quad (32)$$

4) FI function : The function FI_j has a 16-bit word ($X_{j(16)}$ for data input and uses a 16-bit sub-key ($KI_{ij(16)}$) (Matsui, 1997). The input data and the sub-key are each divided into two words of different lengths ($L_{0(9)}$ and $R_{0(7)}$) and ($KI_{ijL(7)}$ and $KI_{ijR(9)}$). With:

$$X_{j(16)} = L_{0(9)} \,||\, R_{0(7)} \quad (33)$$

$$KI_{ij(16)} = KI_{ijL(7)} \,||\, KI_{ijR(9)} \quad (34)$$

The FI function uses two S-Boxes, S7 which links a 7-bit input to a 7-bit output, and S9 which links a 9-bit input to a 9-bit output (Matsui, 1997). In addition, it uses two additional functions designated by *ZE* and *TR*, (Figure 7), defined as:

a) The ZE function converts a 7-bit value ($x_{(7)}$) into a 9-bit value ($y_{(9)}$) by adding two null bits to the left of the most significant bit ($y_{(9)} = ZE(x_{(7)})$).

b) The function TR converts a value of 9 bits ($x_{(9)}$) into a value of 7 bits ($y_{(9)}$) by eliminating the two most significant bits ($y_{(7)} = TR(x_{(9)})$).

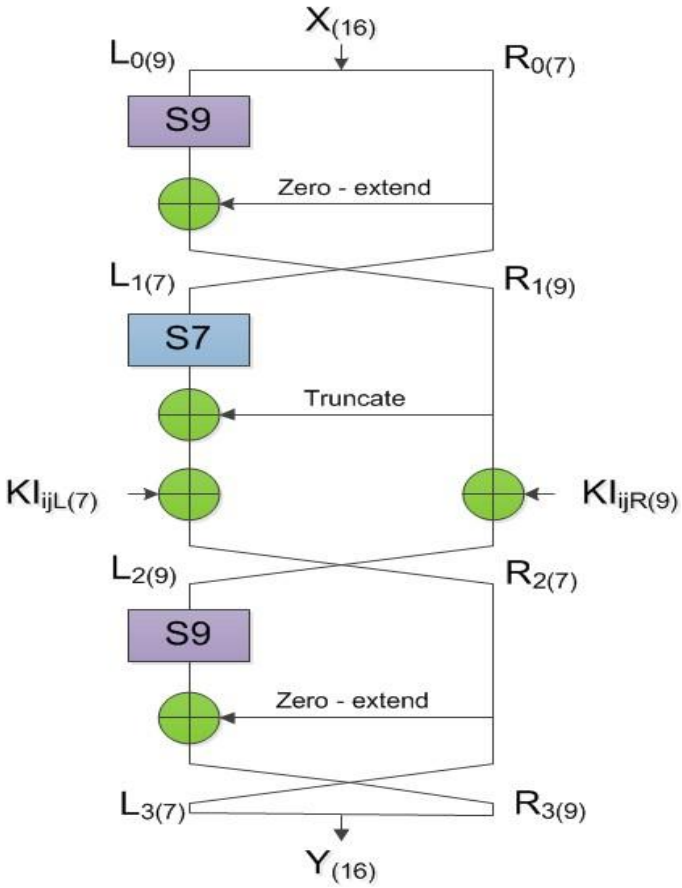

Figure 7. *FI* function.

The output data ($Y_{(16)}$) is given by:

$$Y_{(16)} = L_{3(7)} \,||\, R_{3(9)} \quad (35)$$

The values $L_{3(7)}$ and $R_{3(9)}$ are obtained after the following series of operations:

$$L_{1(7)} = R_{0(7)} \quad (36)$$

$$R_{1(9)} = S9(L_{0(9)}) \oplus ZE(R_{0(7)}) \quad (37)$$

$$L_{2(9)} = R_{1(9)} \oplus KI_{ij}R_{(9)} \quad (38)$$

$$R_{2(7)} = S7(L_{1(7)}) \oplus TR(R_{1(9)}) \oplus KI_{ij}L_{(7)} \quad (39)$$

$$L_{3(7)} = R_{2(7)} \quad (40)$$

$$R_{3(9)} = S9(L_{2(9)}) \oplus ZE(R_{2(7)}) \quad (41)$$

5) S-Boxes: The two S-Boxes are designed in a simple hardware or software implementation, as well as in combinatorial logic or by using a look-up table (Matsui, 1997). What follows will summarize the analytical equations describing the two SBoxes:

a) The logical equations describing S-Boxing S7:

$y0 = x0 \oplus x1x3 \oplus x0x3x4 \oplus x1x5 \oplus x0x2x5 \oplus x4x5 \oplus x0x1x6 \oplus x2x6 \oplus x0x5x6 \oplus x3x5x6 \oplus 1$ $y1 = x0x2 \oplus x0x4 \oplus x3x4 \oplus x1x5 \oplus x2x4x5 \oplus x6 \oplus x0x6 \oplus x3x6 \oplus x2x3x6 \oplus x1x4x6 \oplus x0x5x6 \oplus 1$ $y2 = x1x2 \oplus x0x2x3 \oplus x4 \oplus x1x4 \oplus x0x1x4 \oplus x0x5 \oplus x0x4x5 \oplus x3x4x5 \oplus x1x6x3x6 \oplus x0x3x6 \oplus x4x6 \oplus x2x4x6$ $y3 = x0 \oplus x1 \oplus x0x1x2 \oplus x0x3 \oplus x2x4 \oplus x1x4x5 \oplus x2x6 \oplus x1x3x6 \oplus x0x4x6 \oplus x5x6 \oplus 1$ $y4 = x2x3 \oplus x0x4 \oplus x1x3x4 \oplus x5 \oplus x2x5 \oplus x1x2x5 \oplus$

$x0x3x5 \oplus x1x6 \oplus x1x5x6 \oplus x4x5x6 \oplus 1$ $y5 = x0 \oplus x1 \oplus x2 \oplus x0x1x2 \oplus x0x3 \oplus x1x2x3 \oplus x1x4 \oplus x0x2x4 \oplus x0x5 \oplus x0x1x5 \oplus x3x5 \oplus x0x6 \oplus x2x5x6$ $y6 = x0x1 \oplus x3 \oplus x0x3 \oplus x2x3x4 \oplus x0x5 \oplus x2x5 \oplus x3x5 \oplus x1x3x5 \oplus x1x6 \oplus x1x2x6 \oplus x0x3x6 \oplus x4x6 \oplus x2x5x6$

b) The logical equations describing S-Boxing S9:

$y0 = x0x4 \oplus x0x5 \oplus x1x5 \oplus x1x6 \oplus x2x6 \oplus x2x7 \oplus x3x7 \oplus x3x8 \oplus x4x8 \oplus 1$ $y1 = x0x2 \oplus x3 \oplus x1x3 \oplus x2x3 \oplus x3x4 \oplus x4x5 \oplus x0x6 \oplus x2x6 \oplus x7 \oplus x0x8 \oplus x3x8 \oplus x5x8 \oplus 1$ $y2 = x0x1 \oplus x1x3 \oplus x4 \oplus x0x4 \oplus x2x4 \oplus x3x4 \oplus x4x5 \oplus x0x6 \oplus x5x6 \oplus x1x7 \oplus x3x7 \oplus x8$ $y3 = x0 \oplus x1x2 \oplus x2x4 \oplus x5 \oplus x1x5 \oplus x3x5 \oplus x4x5 \oplus x5x6 \oplus x1x7 \oplus x6x7 \oplus x2x8 \oplus x4x8$ $y4 = x1 \oplus x0x3 \oplus x2x3 \oplus x0x5 \oplus x3x5 \oplus x6 \oplus x2x6 \oplus x4x6 \oplus x5x6 \oplus x6x7 \oplus x2x8 \oplus x7x8$ $y5 = x2 \oplus x0x3 \oplus x1x4 \oplus x3x4 \oplus x1x6 \oplus x4x6 \oplus x7 \oplus x3x7 \oplus x5x7 \oplus x6x7 \oplus x0x8 \oplus x7x8$ $y6 = x0x1 \oplus x3 \oplus x1x4 \oplus x2x5 \oplus x4x5 \oplus x2x7 \oplus x5x7 \oplus x8 \oplus x0x8 \oplus x4x8 \oplus x6x8 \oplus x7x8 \oplus 1$ $y7 = x1 \oplus x0x1 \oplus x1x2 \oplus x2x3 \oplus x0x4 \oplus x5 \oplus x1x6 \oplus x3x6 \oplus x0x7 \oplus x4x7 \oplus x6x7 \oplus x1x8 \oplus 1$ $y8 = x0 \oplus x0x1 \oplus x1x2 \oplus x4 \oplus x0x5 \oplus x2x5 \oplus x3x6 \oplus x5x6 \oplus x0x7 \oplus x0x8 \oplus x3x8 \oplus x6x8 \oplus 1$

Hardware Implementation

Implementation approach

In this section, we present *MISTY1* programming approach, using the *VHDL* language, starting from the previousely studied description.

The first block is responsible for the process of managing the secret key. Starting from the secret key K with a length of 128 bits, we have a sub-key K^0 of 128 bits. the detailed description of the process has been given in section II-A3. Keys K and K^0 are used as input data for the second logical block (management block of encryption sub-keys (Figure 8)). This block will provide the encryption sub-keys KO_{ij}, KI_{ij} and KL_i. Figure 8 illustrates the architecture of the *MISTY1* algorithm as described in *VHDL*.

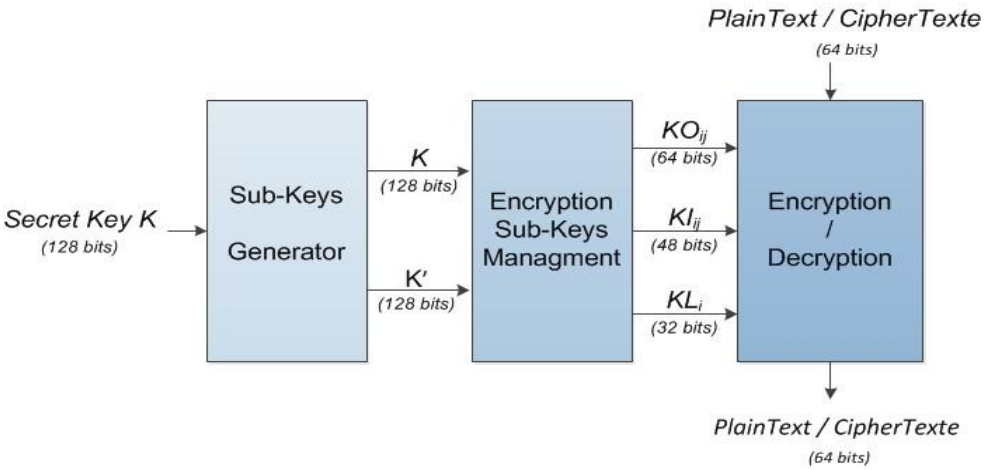

Figure 8. Structure of the implementation program of the algorithm MISTY1

The last block ensures the execution of the 64-bit block encryption-decryption procedure. It is designed from the description given by the Figure 8. The algorithmic of the program is totally inspired by the figure.

Simulation

Before proceeding to the implementation on *FPGA*, we had to verify that our program was working correctly through *ISIM* simulation. The results of the simulations obtained are compared with the test reference data provided in (Matsui, 1997), which are reported in the Table 2.

1) Simulation of the secret key management process: The secret key management program is based on the function *FI* (Figure 3). It uses two S-Boxes S7 and S9. Once programmed in combinatorial logic, the results of the simulations of the two S-Boxes are compared to the two decimal tables reported in (Matsui, 1997).

Table 2. Management of Encryption Sub-Keys

Data	Value (in hexadecimal)
Secret key (K_1 a` K_8) (128 bits)	00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
Sub-key (K^0_1 a` K^0_8) (128 bits)	cf 51 8e 7f 5e 29 67 3a cd bc 07 d6 bf 35 5e 11
Plain Text (64 bits)	01 23 45 67 89 ab cd ef
Cipher Text (64 bits)	8b 1d a5 f5 6a b3 d0 7c

Simulation of the key management program makes it possible to both check the key management process itself and the correct programming of the FI function and the two S7 and S9 S-Boxes. In the Figure 9, the simulation results of the key management module are illustrated. For the same secret reference key *K*, provided in (Matsui, 1997), we get the same sub-key K^0.

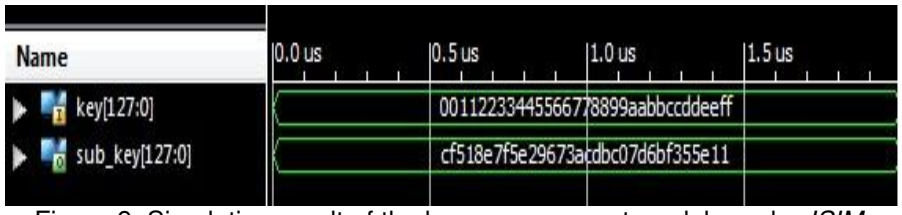

Figure 9. Simulation result of the key management module under *ISIM*.

2) Simulation of the encryption procedure: Once the *FI* function program has been validated, we first programmed the *FO* function which is based on three functions *FI* and some logical operations (Figure 6). Secondly, we programmed the function *FL* which is a series of simple logical operations. Thirdly, we have programmed the encryption sub-key management module.

Finally, and since all the components of MISTY1 have been programmed, we moved to programming the MISTY1 algorithm in encryption mode. Recall that the algorithmic structure of the program has been deduced from the Figure 1.

Using the test reference data published in (Matsui, 1997), we simulated the main program MISTY1 in encryption mode under *ISIM*. The simulation results are shown in Figure 10. Indeed, we introduced the plain text (*Plain-text*) and the secret key (*secretekey*), given in the Table 2, to output the ciphertext (*cipher-text*) as a result.

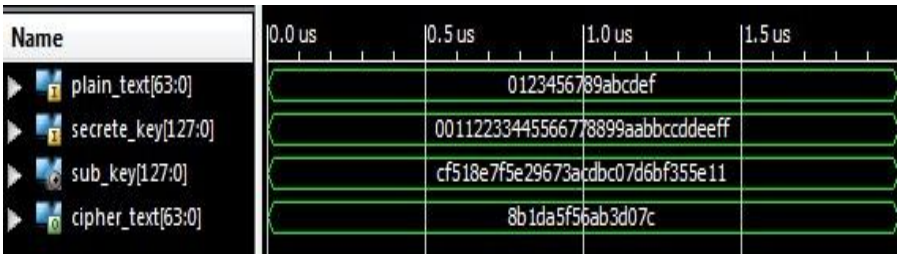

Figure 10. Simulation results of the MISTY1 module in encryption mode.

3) Simulation of the decryption procedure: Once the algorithm MISTY1 is simulated in encryption mode, we switched to simulation of the algorithm in decryption mode. For this we made the following changes:

a) Programming the function FL^{-1};
b) Execute FL^{-1} function instead of *FL* function ;
c) Reverse the order of sub-keys for decryption.

The program of the decryption is based on the flowchart shown in Figure 2. The simulations results in decryption mode are presented in Figure 11. We can conclude that the original plaintext is recovered thanks to the introduction of the ciphertext at the input of the decryption block. we have been able to recover the original plaintext.

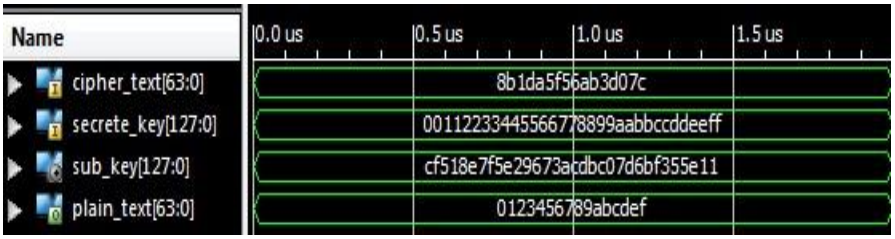

Figure 11. Result of simulation of the MISTY1 module in decryption mode.

FPGA implementation and hardware verification

This section will cover the implementation of MISTY1 based on the development board ML507 which has the Xilinx Virtex 5 xc5vfx70t-1ff1136 FPGA chip.

1) The resource consumption: The resource consumption, after placement and routing of the algorithm give the results presented in Table 3.

Table 3. Resource Consumption of the Misty1 Program on Fpga Xilinx Virtex 5 Type Circuit.

Device Utilization Summary	
Selected device : *XC5vfx70t-1ff1136*	
Slice Logic Utilization	Utilization
Number of Slices Registers	01%
Number of Slices LUTs	15%
Number of fully used LUT-FF pairs	01%
Number of occupied slices	29%
Number of bonded IOBs	40%
Number of BUFG/BUFGCTRLs	01%
Maximum Frequency	20.934 MHz

Table 3 shows that the resources consumed by the studied algorithm are low compared to the resources available in the FPGA circuit, thanks to the simplicity of its architecture. For a frequency of 20.934 MHz, a bit rate of 669.792 Mbps is obtained.

Implementation results

1) Encryption mode: After FPGA implementation, and using the *ChipscopePro* tool, the results are given in Figure 12. The data displayed on the visualization interface is identical to the reference data of the tests.

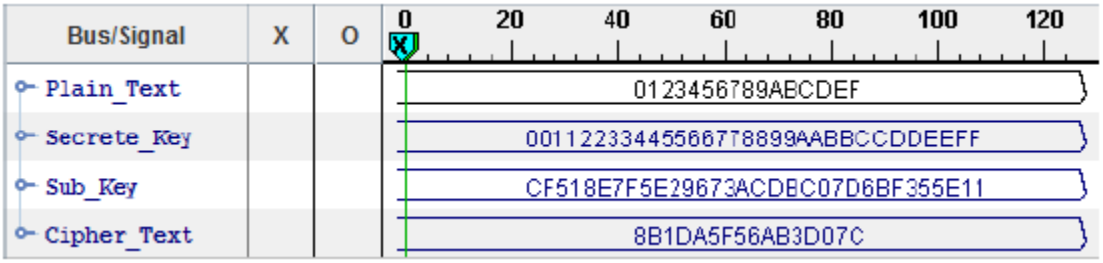

Figure 12. Runtime results (under *ChipScope*) of MISTY1 in encryption mode.

2) Decryption mode: Once the execution of the hardware verification in encryption mode has been completed, we have implemented the decryption mode. Figure 13 shows the results of the implementation. We noticed that we were able to retrieve the plain text from the ciphertext text introduced as input to the decryption module.

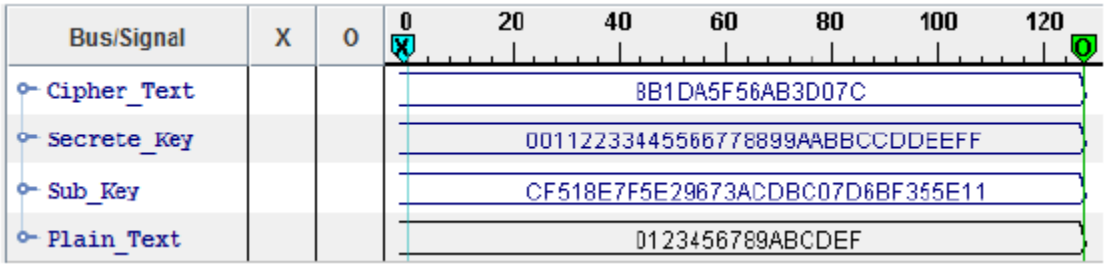

Figure 13. Runtime results (under *ChipScope*) of MISTY1 in decryption mode.

NIST Statistical Test Results

To analyze and interpret the empirical results, we adopt the two approaches presented in (Bassham et al., 2010). The first approach is the evaluation of the proportion of the sequences that have passed the various NIST tests. The second is to evaluate the distribution of *P value* for each test, if one of these two approaches fails, the corresponding null hypothesis should not be rejected.

In the first approach, we calculate the proportion of the sequences that passed the test. For example, if there are 298 sequences that passed the test among the 300 sequences examined (m = 300) and the significance level *alpha* = 0.01, the proportion is equal to $p = \frac{298}{300} = 0.9933$. The confidence interval is determined by the following formula (Bassham et al., 2010):

$$p \pm 3\sqrt{(\frac{p \cdot \alpha}{m})} \quad with \quad p = 1 - \alpha \tag{42}$$

In our case the lower margin of the interval is equal to:

$$0.99 - 3\sqrt{(\frac{0.99 \cdot 0.01}{300})} = 0.972766 \tag{43}$$

The proportion should be above the lower margin set at 0.972766. So we can accept the null hypothesis, because 0.9933 is greater than 0.972766. This approach can be illustrated by a graph representing the proportions of the sequences for each test. The sequences pass a test, if their proportion is above the lower margin represented in the graph 14.

The second approach is to examine the *P–values* distribution of all sequences used for each test, to ensure consistency. It can be illustrated by a histogram, or an interval of 0 to 1 is subdivided into 10 subintervals. The *P–values* are shared in the sub-ranges, for each sub-interval the frequency of *P–values* is shown. Uniformity can also be determined by applying the test χ^2 (Bassham et al., 2010). The distribution χ^2 is illustrated as follows:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - {}^{s}/_{10})^2}{{}^{s}/_{10}} \tag{44}$$

With F_i is the number of *P–values* in the subinterval *i*, and *s* is the number of sequences used.

Generators studied

According to Schneier et al. (1999), Hellekalek and Wegenkittl (2003), an implanted block cipher with a feedback of its output in its input can be considered as a random data generator. This why, we performed the NIST statistical tests on sequences generated from MISTY1.

First approach

The results of the NIST statistical tests obtained by the first approach, for the proposed generator, are presented in the Table 4. The value presented in the table is the proportion of the sequences that pass each test successfully. As a reminder, this must be greater than 0.972766 to consider that the sequence satisfies the criteria of a random sequence. The results obtained for the sequences generated by the algorithm tested by the first approach validate all 15 tests proposed by the NIST battery with proportions higher than 0.972766.

Table 4. Proportion of Sequences that Pass Each Test Successfully

N°	designation of tests	Proportion
1	Frequency Test	0.9933
2	Block Frequency Test	0.9900
3	Cumulative Sums Test Up	0.9900
4	Cumulative Sums Test Up	0.9933
5	Runs Test	0.9867
6	Long Runs of Ones Test	1.0000
7	Rank Test	0.9900
8	Discrete Fourier Transform Test	0.9967
9	Non-overlapping Template Matching Test	0.9900

10	Overlapping Template Matching Test	0.9823
11	Maurer's "Universal Statistical" Test	0.9933
12	Approximate Entropy Test	0.9967
13	Random Excursions Test	1.0000
14	Random Excursions Variant Test	1.0000
15	Serial Test 1	0.9967
16	Serial Test 2	0.9800
17	Linear complexity Test	0.9823

Figure 14 illustrates the proportion of sequences that pass each test with success for the binary sequences formed by the studied generator. We notice that the whole tests of NIST battery are upper at the lower margins for each generator.

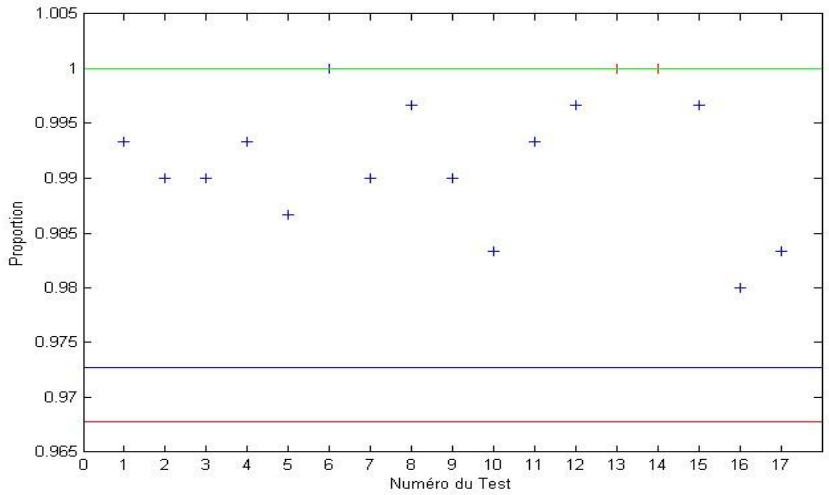

Figure 14. Proportions of the sequences that pass each test successfully (MISTY1)

Second approach

The NIST statistical tests results obtained by the second approach, for the generator studied, are given in Table 5. They show the values of the $P\text{–}values_T$. As a reminder, the $P\text{–}values_T$ allows us to examinate the distribution of the *P–values* of each test, that must be greater than 10^{-4} to consider that the sequence satisfies the uniformity criteria.

The *P–values* distribution of all tests is illustrated by histograms (Figure 15). The histogram contains 10 sticks, each one has a width of 0.1 and a height defined by the number of occurrences of the *P–values* in each interval, defined between 0 and 1 with a step of 0.1. The $P\text{–}values_T$ tells us about this uniformity (Table 5).

Table 5. NIST Statistical Test Results, Proportion of Sequences that Pass Each Test Successfully

N°	Designation of tests	Total P-Value
1	Frequency Test	0.366918
2	Block Frequency Test	0.419021
3	Cumulative Sums Test Up	0.851383
4	Cumulative Sums Test Up	0.060239
5	Runs Test	0.209577

6	Long Runs of Ones Test	0.127148
7	Rank Test	0.055361
8	Discrete Fourier Transform Test	0.035174
9	Non-overlapping Template Matching Test	0.969347
10	Overlapping Template Matching Test	0.171867
11	Maurer's "Universal Statistical" Test	0.935716
12	Approximate Entropy Test	0.942865
13	Random Excursions Test	0.666014
14	Random Excursions Variant Test	0.839124
15	Serial Test 1	0.644060
16	Serial Test 2	0.540878
17	Linear complexity Test	0.129620

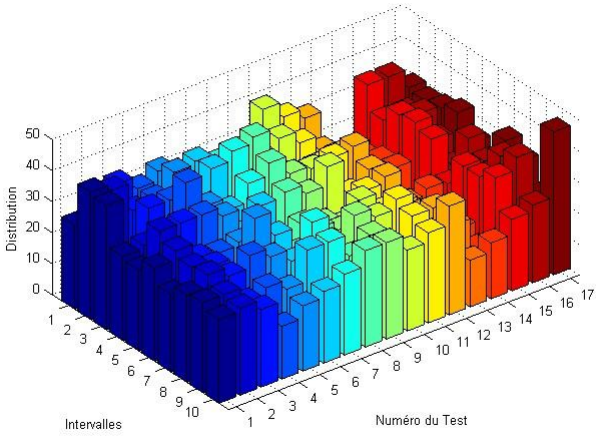

Figure 15. *P–values* distribution of MISTY1.

For our case, the uniformity of the $P\text{–}values_T$ is guaranteed since the number of occurrences in each ten intervals approaches the value "30", since the total of the sequences used for each test is equal to 300.

Conclusion

In this present paper, we presented the structure of the MISTY1 block cipher, and showing its components with a detailed description of the different functions used, either during the process of encryption-decryption or during the subkey management procedure. We also presented the programming approach of the MISTY1 algorithm in VHDL language as well as a verification on the hardware by its implementation on FPGA board. Firstly, we detailed the design approach of the program. Secondly, we coded the algorithm in VHDL language, we performed behavioral simulations under *ISIM*. The results were identical to the data test reference (Matsui, 1997). This results guaranteed that the code worked in both encryption and decryption mode. Thirdly, we moved to the experimental aspect, we implemented the algorithm on an FPGA board, and we were able to perform a hardware check using the built-in tool *Chipscope Pro*. We found that the bit rate is important and the resources consumed are low it offers slight avantages in terms of hardware cost. This makes it more suitable for hardware implementation. Finally, we evaluated the performances of the studied algorithm, in order to validate its security level. The statistical

analysis done by the NIST battery, for the binary sequences formed by the original MISTY1 algorithm keeps the same random character. Therefore, and relevant to our hypothesis, the results obtained for both approaches are consistent.

References

Kitsos, P., M. D. Galanis and O. Koufopavlou, "High-speed hardware implementations of the KASUMI block cipher," 2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512), Vancouver, BC, 2004, pp. II-549

Imai, H., A. Yamagishi. (2011). CRYPTREC (Japanese Cryptographic Algorithm Evaluation Project). Encyclopedia of Cryptography and Security, 285-288.

Preneel, B. (2011). NESSIE project. Encyclopedia of Cryptography and Security, 831-836.

Matsui M. (1997) New block encryption algorithm MISTY. In: Biham E. (eds) Fast Software Encryption. FSE 1997. Lecture Notes in Computer Science, vol 1267. Springer, Berlin, Heidelberg.

Matsui, M. (2000). Supporting Document of MISTY1. version 1.1. Mitsubishi Electric Corporation.

Lai X. (1994) Higher Order Derivatives and Differential Cryptanalysis. In: Blahut R.E., Costello D.J., Maurer U., Mittelholzer T. (eds) Communications and Cryptography. The Springer International Series in Engineering and Computer Science (Communications and Information Theory), vol 276. Springer, Boston, MA.

Bassham, L, E. Rukhin,A. L.Soto, J. Nechvatal, J. R... and Banks, D. L. (2010). Statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST special publication.

Schneier, B. J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson. (1999). Performance comparison of the AES submissions (version 2.0), Retrieved from https://www.schneier.com/academic/paperfiles/paper-aes-performance.pdf.

Hellekalek, P., S. Wegenkittl, S. (2003). Empirical evidence concerning AES. ACM Transactions on Modeling and Computer Simulation (TOMACS), 13(A): 322-333.