A Metamorphic-Key-Hopping GOST Cipher and Its FPGA Implementation

Rabie A. Mahmoud[1], Magdy Saeb[2]

1. General Organization of Remote Sensing (GORS),
Damascus, Syria.

2. Computer Engineering Department,
Arab Academy of Science, Technology & Maritime Transport (AAST),
Alexandria, Egypt.

mail@magdysaeb.net

Abstract: The Metamorphic-Key-Hopping GOST Cipher is a metamorphic cipher with a key hopping technique applying a function that uses four bit-balanced operations. It utilizes a key hopping that depends on the packet power t to alternate the keys. The function operations depend on what is known as the Crypto Logic Unit (CLU). These operations are: XOR, INV, ROR, NOP for bitwise xor, invert, rotate right and no operation respectively. The key hopping technique is called Power Based Key Hopping (PBKH). The Crypto logic unit replaces the adder CM_2, previously used in the well-known GOST Cipher, and the power based key hopping replaces the key scheduling process. In this work, we present the Metamorphic-Key-Hopping GOST Cipher. In addition, we provide a Field Programmable Gate Array (FPGA) implementation of the modified algorithm.

Keywords: Metamorphic, GOST, Cipher, Power Based Key Hopping, Cryptography, FPGA.

1. Introduction

GOST encryption algorithm [1], [2], [3] was considered as the Soviet, now Russian, State Encryption Standard. GOST consisted of 32 iterations to encrypt 64-bit packets using 256-bit key stored in Key Storage Unit (KSU) as a sequence of eight words called Partial Keys where each iteration used a specified partial key as a kind of key scheduling. Each iteration included two adders between the packet and the partial key (CM_1 as added modulo 2^{32} and CM_2 as XOR operation), eight 4-bit secret definition of S-boxes, and shift register to rotate the contents 11-bit left which provide avalanche effect after the 32 iteration at most. The Metamorphic-Key-Hopping Modified GOST Cipher is a metamorphic cipher that modified GOST cipher to encrypt 256-bit packets using 1024-bit key with 128 iterations, key hopping technique, and four low-level operations that are all bit-balanced to encrypt the plaintext bit stream instead of the adder CM_2. The four bit-balanced operations were introduced in The Stone Metamorphic cipher [4], [5] through Crypto Logic Unit (CLU) where those bit-balanced operations are: XORing a key bit with a plaintext bit (XOR), inverting a plaintext bit (INV), exchanging one plaintext bit with another one in a given plaintext word using a right rotation operation (ROR), and producing a plaintext bit without any change (NOP). Power Based Key Hopping (PBKH) [6] is used as key hopping technique to choose the partial key for iterations, and to choose the whitening partial key for whitening technique in the cipher. PBKH depended on the power of the output packet of iteration to determine which partial key will be used in the next iteration, and the power of ciphertext packet to determine which whitening partial key will be used for encrypting next plaintext packet. The power of a packet is the sum of the number of ones in this packet.

In addition, the definitions of certain S-boxes [7], [8] are expanded and rearranged to be thirty two four-bit S-boxes in the Metamorphic-Key-Hopping GOST cipher. A shift register rotates left the 43-bit contents to provide the fast avalanche effect after 128 iterations at most.

In the following sections, we provide the structure of the Metamorphic-Key-Hopping GOST Cipher through describing the key storage unit, the crypto logic unit, the deployed S-boxes, the cyclic shift register and the whitening technique. Finally, we provide the details of a proposed hardware implementation for the cipher, a discussion of the results of the FPGA implementation, a summary and our conclusions.

2. The Proposed Cipher Structure

Metamorphic-Key-Hopping GOST algorithm consists of 128 iterations to encrypt 256-bit plaintext packet into 256-bit ciphertext packet using 1024-bit keys. Figure 1 shows the block diagram of the cipher. The basic components of Metamorphic-Key-Hopping GOST algorithm are:

- 1024-bit cryptographic key *K*,
- 128-bit Initialization Vector (*IV*),
- 2-bit operation-selection-bits,
- 7-bit rotation-selection-bits.

The proposed algorithm uses XOR operation between the 256-bit plaintext packet and the whitening partial key WK_i as Input-Whitening step before the first iteration, then splits the packet into two 128-bit parts which are placed in two registers R_2 and R_1. The contents of register R_1 is added modulo 2^{128} to the partial key K_i (the adder CM_1)

Received 10/12, Reviewed 10/25

$$R_1 + K_i \pmod{2^{128}} \qquad (1)$$

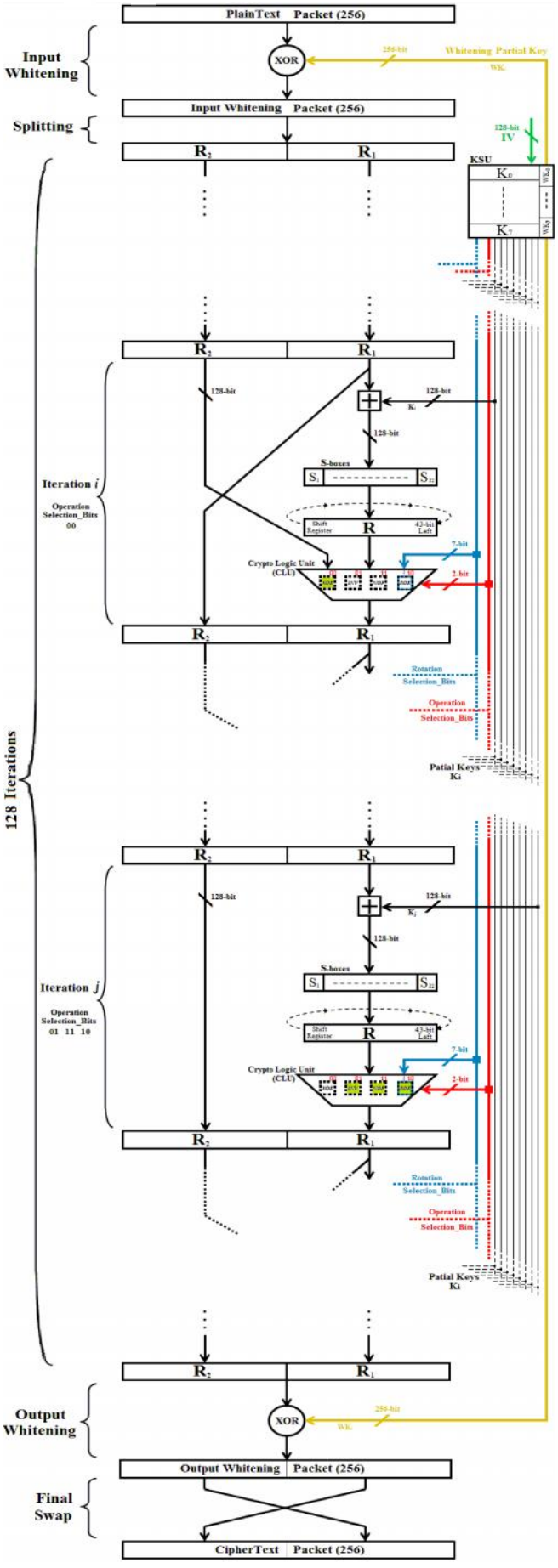


Figure 1: The structure of Metamorphic-Hopped GOST Cipher

then the 128-bit resulting *S* divided into 32 four-bit blocks where those blocks are used as the inputs of the 32 corresponding S-box entries. The outputs of 32 four-bit S-boxes are stored in the shift register R where the content is rotated 43-bit to the left. The Crypto Logic Unit (CLU) which is considered the second adder CM_2 in the proposed algorithm determines the operation for the output of the shift register R and the register R_2. The register R_1 of the next iteration stores output of CLU while the register R_2 stores the old contents of register R_1 from the previous iteration when XOR operation is used in CLU, and stores the old content of register R_2 from previous iteration when other operation is used in CLU. After 128 iterations, the output whitening step is performed by XORing the output of all iterations with the same whitening partial key WK_i in the input-whitening step. Finally, the 256-bit packet of the output-whitening will be split into two parts and swapped to the left and right halves to produce the ciphertext packet.

2.1 The Key Storage Unit (KSU)

The Key Storage Unit (KSU) stores all of the basic components of the proposed algorithm. The 1024-bit cryptographic key K is stored in KSU as a sequence of eight 128-bit words (K_0, …, K_7). Each word is called the Partial Key K_i. The 128-bit Initialization Vector IV is also stored in the KSU where this unit is responsible to increment the IV by one after encrypting each plaintext packet to use the new IV for encrypting the next plaintext packet. Other partial keys are created in the Metamorphic-Hopping GOST algorithm for input/output whitening steps and called Whitening Partial Keys WK_i. KSU stores each whitening partial key as a sequence of 256-bit word (WK_0, …, WK_7) which is consisted of 128-bit partial key K_i concatenated with the 128-bit IV

$$WK_i = K_i || IV \quad \text{for } i = 0, \ldots, 7. \qquad (2)$$

he Power Based Key Hopping (*PBKH*) process which is used to interchange the partial keys and the whitening partial keys in the KSU depending on the power of the output packet. This process determines which partial key will be used in the next iteration, and the power of ciphertext packet determines which whitening partial key will be used for the encrypting next plaintext packet where the power of a packet is known by the number of ones. It is compared with the threshold power that is a certain predefined value. The power threshold can be a secret component but the modified Metamorphic-Key-Hopping GOST algorithm uses it as equal to 128. The whitening partial key WK_7 and the partial key K_0 are default keys at beginning of each message encryption. In other words, the whitening partial key WK_7 will be used in input/output whitening steps for encrypting the first packet of the message then the power of ciphertext packet will determine the next whitening partial key for next packet in the message. Also, the partial key K_0 will be used in the first iteration of encrypting the packet then the power of output of the iteration will determine the next partial key for next iteration in the algorithm. The same whitening partial key WK_i / partial key K_i will be used when the power of ciphertext packet / output of iteration packet equals to the power threshold, the next whitening partial key WK_{i+1} / partial key K_{i+1} will be used when the power of ciphertext packet / output of iteration packet is greater than the power threshold, and the previous whitening partial key WK_{i-1} / partial key K_{i-1} will be used when the power of ciphertext packet / output of iteration

packet is less than the power threshold. Figure 2 shows the scheme of power based key hopping of Metamorphic-Hopped GOST.

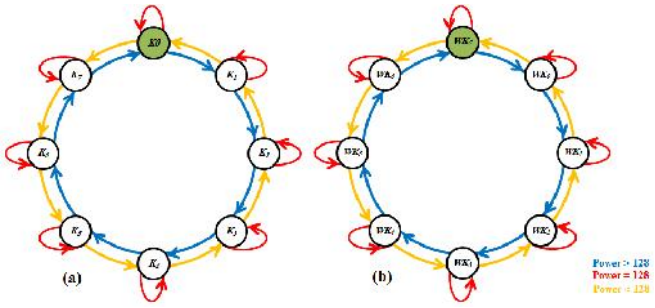


Figure 2. Power based key hopping scheme
(a) Partial keys K_i (b) Whitening partial keys WK_i

2.2 The Crypto Logic Unit (CLU)

The Crypto Logic Unit (CLU) is the encrypting function giving a metamorphic concept for our modifying GOST algorithm where four low-level operations are composing the basic crypto logic unit. The basic crypto logic unit (CLU) is shown in Figure 3 and more details can be found in [4]. Crypto logic unit (CLU) applied one of bit-balanced operations:

- (XOR) by XORing a key bit with a plaintext bit,
- (INV) by inverting a plaintext bit,
- (NOP) by producing the plaintext without any change,
- (ROR) by exchanging one plaintext bit with another one in a given plaintext word using a right rotation operation.

Table1 demonstrates the details of each one of these operations.

Table 1: CLU operations

Mnemonic	*Operation*	*Select Operation code*
XOR	$C_i = K_i \oplus P_i$	"00"
INV	$C_i = \dot{E}(P_i)$	"01"
NOP	$C_i = P_i$	"11"
ROR	$P_i \leftarrow (P_i, m)$	"10"

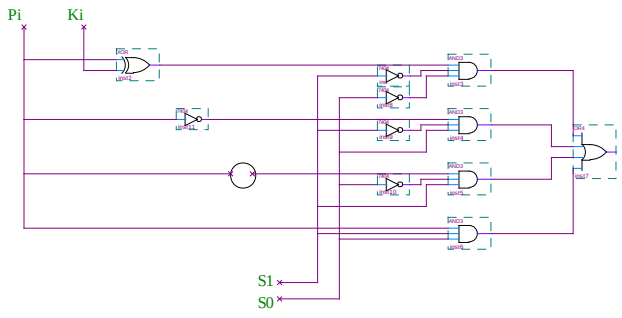


Figure 3. The basic crypto logic unit

The operation selection bits determine the applied operation in CLU and the rotation selection bits determine the number of rotations which are provided when ROR operation is used. In Metamorphic-Hopped GOST algorithm, 2-bit for operation selection bits and 7-bit for rotation selection bits will be chosen from the used whitening partial key WK_i. Relationship between the 128 iterations of algorithm and the 256-bit of whitening partial key WK_i is used to feed each iteration with their operation selection bits through allowing a unique 2-bit from the whitening partial key WK_i for each iteration where first bit of a unique 2-bit extracted from the partial key part of whitening partial key and the second one from the *IV* part of whitening partial key. Figure 4 shows the chosen of operation selection bits to feed the iterations from any whitening partial key. Also, non-serial 7-bit from the *IV* part of whitening partial key will be used as rotation selection bits.

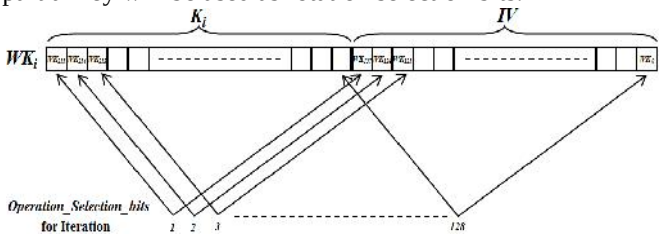


Figure 4. The proposed key format where the location of the operation selection bits is shown

2.3 The Modified S-boxes

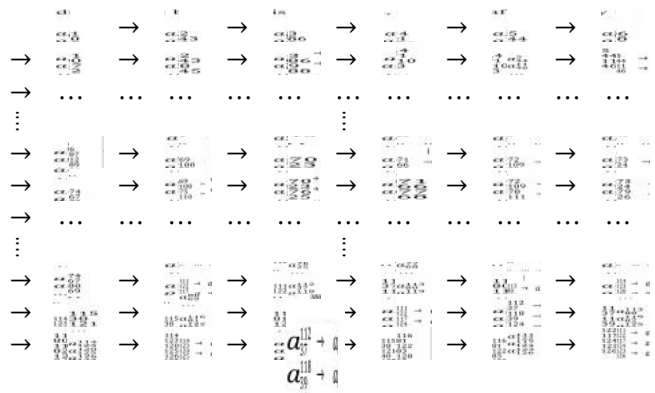
Metamorphic-Hopped GOST algorithm provides a non-secret 32 4-bit S-boxes (S_i, i = 1…32) to substitute 128-bit block where those 32 S_i S-boxes will be alternated every an eight iterations until end of iterations in the algorithm. The Twofish S-boxes which are defined in q_0 and q_1 permutations will be applied in Metamorphic-Hopped GOST algorithm in other arrangement. The iterations 1 to 8, 33 to 40, 65 to 72, and 97 to 104 apply t_0 S-box of permutation q_0 in Twofish algorithm for each S-box in the first sixteen S-boxes and t_0 S-box of permutation q_1 in Twofish algorithm for each S-box in the last sixteen S-boxes. In other words, the iterations 1 to 8, 33 to 40, 65 to 72, and 97 to 104 apply the $t_{0\,q_0}$ S-box for each S-box in the first sixteen S-boxes $S_{1...16}$ and the $t_{0\,q_1}$ S-box for each S-box in the last sixteen S-boxes $S_{17...32}$. The iterations 9 to 16, 41 to 48, 73 to 80, and 105 to 112 apply $t_{1\,q_0}$ for each S-box in $S_{1...16}$ and $t_{1\,q_1}$ for each S-box in $S_{17...32}$. The iterations 17 to 24, 49 to 56, 81 to 88, and 113 to 120 apply $t_{2\,q_0}$ for each S-box in $S_{1...16}$ and $t_{2\,q_1}$ for each S-box in $S_{17...32}$. The iterations 25 to 32, 57 to 64, 89 to 96, and 121 to 128 apply $t_{3\,q_0}$ for each S-box in $S_{1...16}$ and $t_{3\,q_1}$ for each S-box in $S_{17...32}$. The 4-bit S-boxes values have been written as hexadecimal values and given by:

$t_{0\,q_0}$ = [8 1 7 D 6 F 3 2 0 B 5 9 E C A 4],
$t_{0\,q_1}$ = [2 8 B D F 7 6 E 3 1 9 4 0 A C 5],
$t_{1\,q_0}$ = [E C B 8 1 2 3 5 F 4 A 6 7 0 9 D],
$t_{1\,q_1}$ = [1 E 2 B 4 C 3 7 6 D A 5 F 9 0 8],
$t_{2\,q_0}$ = [B A 5 E 6 D 9 0 C 8 F 3 2 4 7 1],
$t_{2\,q_1}$ = [4 C 7 5 1 6 9 A 0 E D 8 2 B 3 F],
$t_{3\,q_0}$ = [D 7 F 4 1 2 6 E 9 B 3 0 8 5 C A],
$t_{3\,q_1}$ = [B 9 5 1 C 3 D E 6 4 7 F 2 0 8 A].

2.4 The Cyclic Shift Register R

The cyclic shift register *R* in Metamorphic-Hopped GOST algorithm re-designed to rotate the output of S-boxes 43-bit the left and provide a similar diffusion of GOST algorithm for 128-bit as right-half algorithm where each individual input bit

occupies every other bit of the right-half of algorithm exactly once after 128 iterations at most. Let R_1^i denote the input to the right-half of the algorithm at iteration i and $a_0^i, \cdots a_{127}^i$ the individual input bits to this iteration, the bits affected by a_0^1 as:



Hence after 128 iterations, $a_0^{[illegible]}$ occupies every other bit of the right-half exactly once.

2.5 Whitening

The Whitening technique is used in modified GOST before the first iteration. This is called Input-whitening step and after the last iteration that is called Output-Whitening step through XORing the packet with the whitening partial key where the whitening is a technique used to increase the difficulty of key-search attacks against the remainder of the cipher.

3. The Algorithm

In this section, we provide the formal description of the Metamorphic-Hopped GOST block cipher algorithm as follows:

ALGORITHM: METAMORPHIC-HOPPED GOST BLOCK CIPHER

INPUT: *Plaintext message P, User Cryptographic Key K, Block B, Initialization Vector IV*

OUTPUT: *Ciphertext C*

Algorithm body:

Begin

Begin Key Storage Unit (KSU)

1. *Read a 1024-bit user cryptographic key K;*

2. *Generate eight 128-bit partial keys K_i by splitting the user cryptographic key K= (K_0, …, K_7);*

3. *Read a 128-bit Initialization Vector IV;*

4. *Generate eight 256-bit whitening partial keys WK_i by concatenating each partial key with initialization vector*

$WK_i \leftarrow K_i$ & IV for i=0,…, 7;

5. *Choose Non-serial 7-bit rotation_selection_bits from IV;*

6. *Choose 128 unique pair's operation_selection_bits from whitening partial key WK_i for 128 iterations, each unique pair contains bit from partial key K_i and bit from initialization vector IV;*

7. *If the block B is the first block in the plaintext message P then use the whitening partial key WK_7*

Else use power based key hopping function to determine the whitening partial key WK_i for encrypting block B;

8. *Use partial key K_0 for first iteration at each encryption;*

End Key Storage Unit;

Begin Encryption

9. *Read a 256-bit block B of the message P into the message cache;*

10. *Use the whitening partial key WK_i for input-whitening step by XORing the block B with whitening partial key*

Input-Whitening ← B XOR WK_i;

11. *Split the input-whitening into two parts and store the parts in 128-bit register R_1 and 128-bit register R_2*

Input-Whitening ← R_2& R_1;

12. *Add the register R_1 to partial key K_i and store the addition in 128-bit register S*

S ← $R_1 + K_i \pmod{2^{128}}$;

13. *Substitute the register S by calling S-boxes function and store the substitution in 128-bit shift register R*

R ← S-boxes(S);

14. *Rotate the shift register R 43-bit left*

R ← ROL(R, 43);

15. *Apply crypto logic unit on register R and register R_2;*

16. *If XOR operation is used in crypto logic unit then store the register R_1 in register R_2 and store the output of crypto logic unit in register R_1*

$R_2 \leftarrow R_1$ AND $R_1 \leftarrow$ CLU(R, R_2) when operation_selection_bits="00"

Else just store the output of crypto logic unit in register R_1

$R_1 \leftarrow$ CLU(R, R_2) when operation_selection_bits="01" OR operation_selection_bits="11" OR operation_selection_bits="10";

17. *Use power based key hopping function to determine the partial key K_i for next iteration;*

18. *Repeat steps 12 to 17 127 times more;*

19. *Use the same whitening partial key WK_i in input-whitening step for output-whitening step by XORing the output of all iterations with whitening partial key*

Output-Whitening ← final (R_2& R_1) XOR WK_i;

20. *Split the output-whitening into two 128-bit halves;*

21. *Swap the halves to produce the ciphertext C;*

22. *Increment the initialization vector IV by one*

IV ← IV + 1;

23. *Repeat steps 3 to 22 if message cache is not empty;*

24. *When the plaintext messages are finished then halt;*

End Encryption;

End Algorithm.

Function Crypto Logic Unit (CLU)

Begin

1. *Read a 128-bit register R_2;*

2. *Read a 128-bit shift register R;*

3. *Read 7-bit rotation_selection_bits from IV which are chosen in key storage unit;*

4. *Read 2-bit operation_selection_bits form whitening partial key WK_i which are specified for the iteration and chosen in key storage unit;*

5. *Use operation selection & rotation selection bits to select and perform operation:*

XOR when operation_selection_bits="00"
INV when operation_selection_bits="01"
NOP when operation_selection_bits="11"
ROR when operation_selection_bits="10";

6. *Perform the crypto operation using register R_2 bit and shift register R bit to get a crypto bit;*

7. *Store the output of crypto logic unit in register R_1;*

End;

Function Power Based Key Hopping (PBKH)
Begin
1. *Read eight 256-bit whitening partial keys / eight 128-bit partial keys;*
2. *Specify which whitening partial key WK_j / partial key K_j is used;*
3. *Read ciphertext packet / iteration's output packet;*
4. *Compute the power Pr of packet in step 3;*
5. *Compare the value of power Pr with power threshold* α *which is equal to 128:*
5.1. *If the power Pr = 128 then use the same key WK_j / K_j for next plaintext packet / iteration,*
5.2. *If the power Pr > 128 then use the next key WK_{j+1} / K_{j+1} for next plaintext packet / iteration,*
5.3. *If the power Pr < 128 then use the previous key WK_{j-1} / K_{j-1} for next plaintext packet / iteration;*
6. *Assign the new selected key for encrypting next packet / next iteration;*
End;

Function S-boxes
Begin
1. *Read a 128-bit register S;*
2. *Split S into 32 4-bit boxes;*
3. *Substitute each 4-bit box by defined S-box:*
3.1. *If the iteration is one of iterations 1 to 8, 33 to 40, 65 to 72, and 97 to 104 then use $t_{0\,q_0}$ S-box for each S-box in $S_{1...16}$ and $t_{0\,q_1}$ S-box for $S_{17...32}$*
$t_{0\,q_0}$ = [8 1 7 D 6 F 3 2 0 B 5 9 E C A 4]
$t_{0\,q_1}$ = [2 8 B D F 7 6 E 3 1 9 4 0 A C 5],
3.2. *If the iteration is one of iterations 9 to 16, 41 to 48, 73 to 80, and 105 to 112 then use $t_{1\,q_0}$ S-box for each S-box in $S_{1...16}$ and $t_{1\,q_1}$ S-box for $S_{17...32}$*
$t_{1\,q_0}$ = [E C B 8 1 2 3 5 F 4 A 6 7 0 9 D]
$t_{1\,q_1}$ = [1 E 2 B 4 C 3 7 6 D A 5 F 9 0 8],
3.3. *If the iteration is one of iterations 17 to 24, 49 to 56, 81 to 88, and 113 to 120 then use $t_{2\,q_0}$ S-box for each S-box in $S_{1...16}$ and $t_{2\,q_1}$ S-box for $S_{17...32}$*
$t_{2\,q_0}$ = [B A 5 E 6 D 9 0 C 8 F 3 2 4 7 1]
$t_{2\,q_1}$ = [4 C 7 5 1 6 9 A 0 E D 8 2 B 3 F],
3.4. *If the iteration is one of iterations 25 to 32, 57 to 64, 89 to 96, and 121 to 128 then use $t_{3\,q_0}$ S-box for each S-box in $S_{1...16}$ and $t_{3\,q_1}$ S-box for $S_{17...32}$*
$t_{3\,q_0}$ = [D 7 F 4 1 2 6 E 9 B 3 0 8 5 C A]
$t_{3\,q_1}$ = [B 9 5 1 C 3 D E 6 4 7 F 2 0 8 A],
4. *Store the output of S-boxes in 128-bit shift register R;*
End;

4. The FPGA Implementation

A Prototype FPGA-based implementation of the proposed modified GOST cipher is applied to encrypt 256-bit plaintext packet using 1024-bit user key and 128-bit initialization vector focusing on crypto logic unit and power based key hopping concept for 128 iterations. We have implemented the cipher applying the VHDL hardware description language and utilizing Altera design environment Quartus II 13.0 Service Pack 1 Web Edition [9]. The design was implemented using two EP4CGX150DF31I7AD, Cyclone IV GX family devices connected sequentially on the board. The first chip (Chip1) is responsible for reading the 256-bit plaintext packet and extracting 256-bit whitening partial key from key storage unit which includes the user key and *IV* as inputs of chip, then splitting the input-whitening packet into two 128-bit sub-packets R_1 and R_2 which are the inputs for the first iteration. 64-iterations are included in Chip1 where the output of Chip1 will be the input of second chip (Chip2). Chip2 contains the other 64-iteration where the output of all iterations will be XORed with the whitening partial key as output-whitening then final swap produces 256-bit ciphertext packet. Correct implementation and part of schematic diagram for Chip1 and Chip2 of Metamorphic-Hopped GOST Cipher are shown in Figures 5 and 6 respectively. Also, RTL screen then technology map viewer for Chip1 and Chip2 are shown in Figures 7, 8, 9 and 10 respectively. Figures 11 and 12 demonstrate the floor plan for Chip1 and Chip2 respectively. The details of the analysis and synthesis summary and timing analyzer are shown in appendix B.

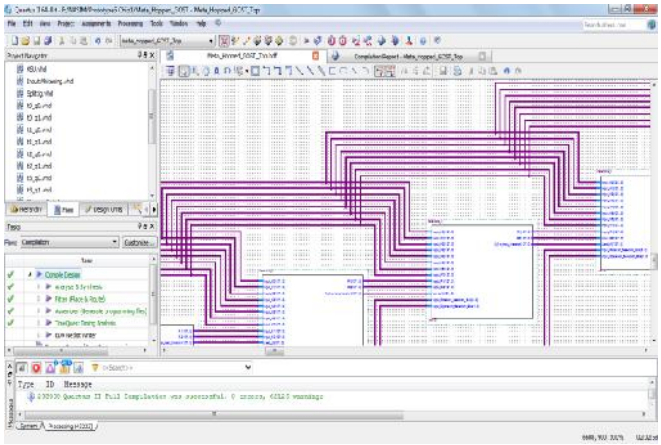


Figure 5: Correct implementation and part from schematic diagram of Chip1 of Metamorphic-Hopped GOST Cipher

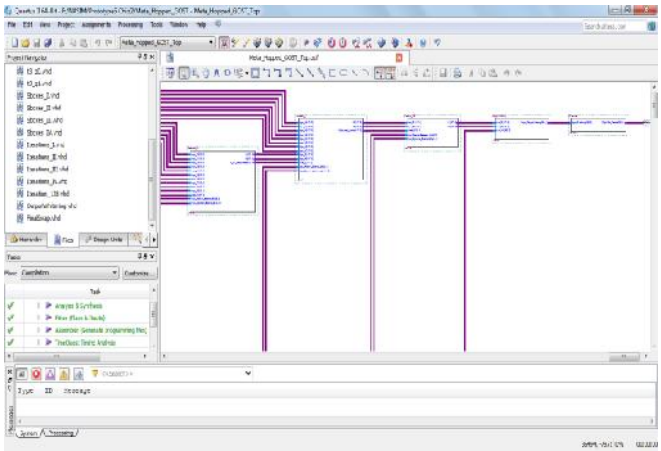


Figure 6: Correct implementation and part from schematic diagram of Chip2 of Metamorphic-Hopped GOST Cipher

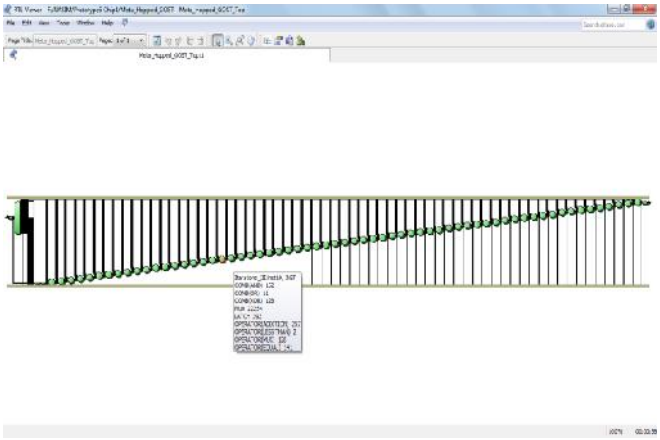


Figure 7: RTL screen for Chip1 of Metamorphic-Hopped GOST Cipher

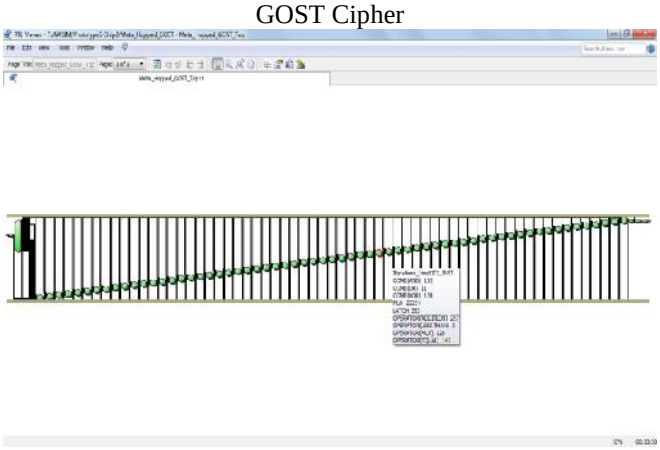


Figure 8: RTL screen for Chip2 of Metamorphic-Hopped GOST Cipher

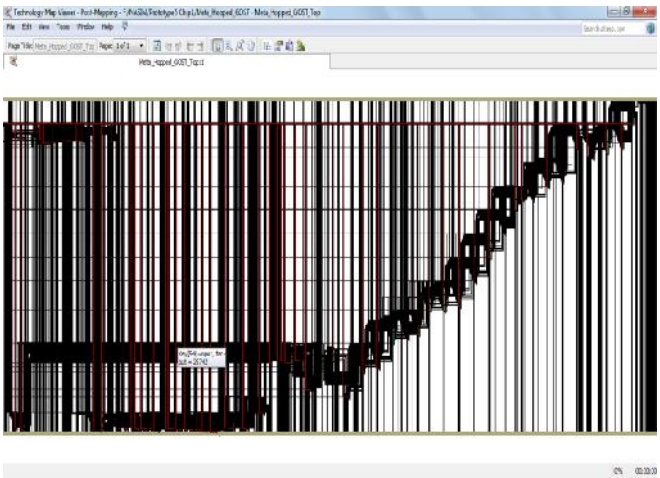


Figure 9: Technology Map viewer of Chip1 of Metamorphic-Hopped GOST Cipher

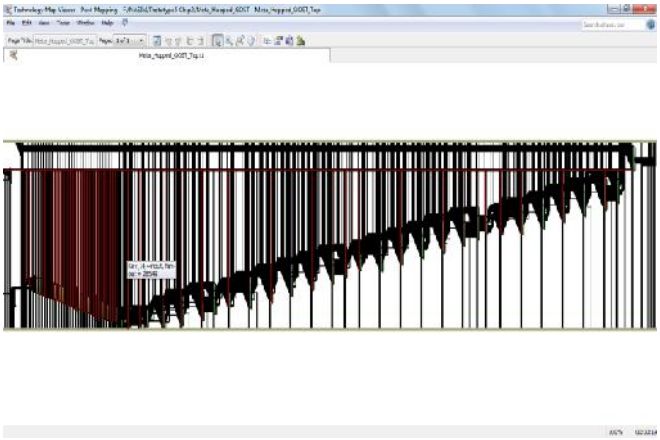


Figure 10: Technology Map viewer of Chip2 of Metamorphic-Hopped GOST Cipher

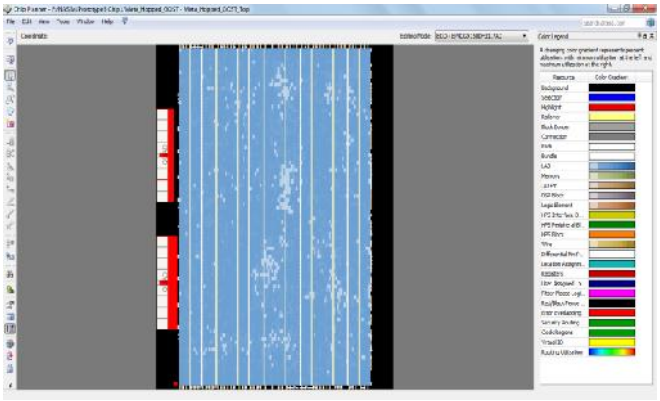


Figure 11: Floor-plan of Chip1 of Metamorphic-Hopped GOST Cipher

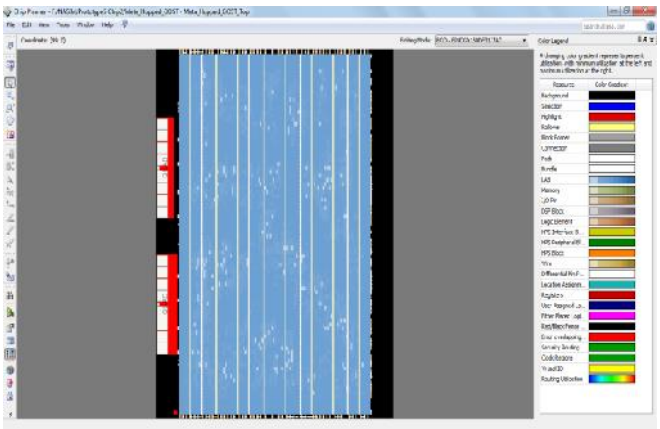


Figure 12: Floor-plan of Chip2 of Metamorphic-Hopped GOST Cipher

5. Summary and Conclusions

We have furnished a cipher that aggregates a multiple of unexampled concepts to modify the GOST cipher. This modified cipher is called the Metamorphic-Key-Hoppying GOST cipher. A Crypto Logic unit that is based on four bit-balanced operations is used in the various iterations of the algorithm. A power-based key hopping technique is used to choose a partial whitening and iteration keys. In addition, a new arrangement modified version of S-boxes are used in this

cipher. We have also included a proof-of-concept FPGA hardware implementation. The Metamorphic-Key-Hoppying GOST algorithm will appreciably increase the entropy and provide higher degree of randomness and thus a conjectural security.

References

[1] National Soviet Bureau of Standards, Information Processing Systems, Cryptographic Protection, Cryptographic Algorithm, GOST 28147-89, 1989.

[2] J. Pieprzyk, L. Tombak, "Soviet Encryption Algorithm," Australian Research Council under the reference number A49131885, Department of Computer Science, University of Wollongong, Wollongong, Australia, June, 1994.

[3] C. Charnes, L. O'Connor, J. Pieprzyk, R. Safavi-Naini, Y. Zheng, "Further Comments on the Soviet Encryption Algorithm," Department of Computer Science, University of Wollongong, Wollongong, Australia, June, 1994.

[4] M. Saeb, "The Stone Cipher-192 (SC-192): A Metamorphic Cipher," The International Journal on Computers and Network Security (IJCNS), Vol.1 No.2, pp. 1-7, Nov., 2009.

[5] Rabie A. Mahmoud, M. Saeb, "Hardware Implementationof the Stone Metamorphic Cipher," International Journal ofComputer Science and Network Security(IJCSNS), Vol.10, No.8, pp.54-60, 2010.

[6] Rabie A. Mahmoud, M. Saeb, "Power-based Key Hopping (PBKH) and Associated Hardware Implementation," International Journal of Computer Science & Information Security (IJCSIS), VOL.8, No.9, Dec., 2010.

[7] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, "Twofish: A 128-Bit Block Cipher," Counterpane Systems, Minneapolis, USA, 1998.

[8] Rabie A. Mahmoud, M. Saeb, "A Metamorphic-Enhanced Twofish Block Cipher And Associated FPGA Implementation," International Journal of Computer and Network Security (IJCNS), Vol.2, No.1, Jan., 2012.

[9] Altera's user-support site: http://www.altera.com/support/examples/vhdl/vhdl.html

Appendix A: Major differences between GOST and Metamorphic-Hopped GOST

Table2: Differences between GOST and Metamorphic-Hopped GOST

GOST	Metamorphic-Hopped GOST
Encrypt 64-bit packet	Encrypt 256-bit packet
256-bit user key	1024-bit user key
32 iterations	128 iterations
Key schedule	Power based key hopping
No initialization vector	128-bit initialization vector *IV*
No whitening technique	Whitening technique
Secret *S*-boxes	Non-secret *S*-boxes
XOR operation as CM_2	Crypto logic unit
Shift register *R* to rotate 11-bit left	Shift register *R* to rotate 43-bit left
Secret components: • User key • *S*-boxes	Secret components: • User key • *IV* • *Operation_selection_bits* • *Rotation_selection_bits*

Appendix B: The analysis & synthesis and fitter report details

Analysis & Synthesis and Fitter Summary for Chip1

• Family: Cyclone IV GX
• Device: EP4CGX150DF31I7AD
• Nominal Core Voltage: 1.20 V
• Minimum Core Junction Temperature: -40 °C
• Maximum Core Junction Temperature: 100 °C.

• Optimization Technique: Balanced
• Total logic elements: 123,092 out of 149,760 (82 %)
-- Combinational with no register: 123,092
-- Register only: 0
-- Combinational with a register: 0
Logic element usage by number of LUT inputs
-- 4 input functions: 108,724
-- 3 input functions: 13,245
-- <=2 input functions: 1,123
-- Register only: 0
Logic elements by mode
-- Normal mode: 114,964
-- Arithmetic mode: 8,128
• Total memory bits: 0 out of 6,635,520 (0 %)
• Total LABs: 8,898 out of 9,360 (95 %)
• Embedded Multiplier 9-bit elements: 0 out of 720 (0 %)
• Total PLLs: 0 out of 8 (0 %)
• Total fan-out: 477,549
• Average fan-out: 3.85
• Highest non-global fan-out: 12,254
• Maximum fan-out: 12,254

• Average interconnect usage (total/H/V): 44% / 41% / 49%
• Peak interconnect usage (total/H/V): 54% / 52% / 59%
• Block interconnects: 203,420 out of 445,464 (46 %)
• C16 interconnects: 6,446 out of 12,402 (52 %)
• C4 interconnects: 123,647 out of 263,952 (47 %)
• Direct links: 18,560 out of 445,464 (4 %)
• GXB block output buffers: 0 out of 3,600 (0 %)
• Global clocks: 0 out of 30 (0 %)
• Interquad Reference Clock Outputs: 0 out of 2 (0 %)
• Interquad TXRX Clocks: 0 out of 16 (0 %)
• Interquad TXRX PCSRX outputs: 0 out of 8 (0 %)
• Interquad TXRX PCSTX outputs: 0 out of 8 (0 %)
• Local interconnects: 55,309 out of 149,760 (37 %)
• R24 interconnects: 6,476 out of 12,690 (51 %)
• R4 interconnects: 139,899 out of 370,260 (38 %)

Timing Analyzer Summary for Chip1

• Longest propagation delay RR which is measured from rising edge to rising edge was 1371.711 ns from input port "IV[3]" to output port "R1_64[88]". Also, longest delay RF which is measured from rising edge to falling edge was 1371.653 ns, longest delay FR which is measured from falling edge to rising edge was 1371.893 ns, and longest delay FF

which is measured from falling edge to falling edge was 1371.835 ns.
• Longest minimum propagation delay was from input port "PlainText[15]" to output port "R2_64[66]" where RR was 41.923 ns, RF was 42.328 ns, FR was 42.614 ns, and FF was 43.019 ns.

Analysis & Synthesis and Fitter Summary for Chip2

• Family: Cyclone IV GX
• Device: EP4CGX150DF31I7AD
• Nominal Core Voltage: 1.20 V
• Minimum Core Junction Temperature: -40 °C
• Maximum Core Junction Temperature: 100 °C.

• Optimization Technique: Balanced
• Total logic elements: 127,689 out of 149,760 (85 %)
-- Combinational with no register: 127,689
-- Register only: 0
-- Combinational with a register: 0
Logic element usage by number of LUT inputs
-- 4 input functions: 109,325
-- 3 input functions: 16,758
-- <=2 input functions: 1,606
-- Register only: 0
Logic elements by mode
-- Normal mode: 119,561
-- Arithmetic mode: 8,128
• Total memory bits: 0 out of 6,635,520 (0 %)
• Total LABs: 9,174 out of 9,360 (98 %)
• Embedded Multiplier 9-bit elements: 0 out of 720 (0 %)
• Total PLLs: 0 out of 8 (0 %)
• Total fan-out: 491,464
• Average fan-out: 3.82
• Highest non-global fan-out: 12,237
• Maximum fan-out: 12,237
• Average interconnect usage (total/H/V): 54% / 51% / 59%
• Peak interconnect usage (total/H/V): 66% / 62% / 72%
• Block interconnects: 225,032 out of 445,464 (51 %)
• C16 interconnects: 6,791 out of 12,402 (55 %)
• C4 interconnects: 150,783 out of 263,952 (57 %)
• Direct links: 17,376 out of 445,464 (4 %)
• GXB block output buffers: 0 out of 3,600 (0 %)
• Global clocks: 1 out of 30 (3 %)
• Interquad Reference Clock Outputs: 0 out of 2 (0 %)
• Interquad TXRX Clocks: 0 out of 16 (0 %)
• Interquad TXRX PCSRX outputs: 0 out of 8 (0 %)
• Interquad TXRX PCSTX outputs: 0 out of 8 (0 %)
• Local interconnects: 55,621 out of 149,760 (37 %)
• R24 interconnects: 6,784 out of 12,690 (53 %)
• R4 interconnects: 179,417 out of 370,260 (48 %)

Timing Analyzer Summary for Chip2

• Longest propagation delay was from input port "IV[10]" to output port "CipherText[224]" where RR was 1460.198 ns, RF was 1460.136 ns, FR was 1460.373 ns, and FF was 1460.311 ns.
• Longest minimum propagation delay was from input port "PlainText[7]" to output port "CipherText[152]" where RR was 44.274 ns, RF was 44.699 ns, FR was 44.970 ns, and FF was 45.395 ns.

Appendix C: Sample VHDL code for iteration in Metamorphic-Hopped GOST cipher

Hierarchy design is used to implement Metamorphic-Hopped GOST cipher through programming the substitutions t_0q_0, .., t_3q_1 individually, then using component configuration of generated statements to program S-boxes_I, S-boxes_II, S-boxes_III, and S-boxes_IV which are related with specified iterations. Finally, 128 iteration blocks which are divided into four typs Iterations_I, Iterations_II, Iterations_III, Iterations_IV also programmed related with S-boxes then connected together in block diagram/schematic file. Sample VHDL codes are:

VHDL Code for S-box $t_0\,q_0$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY t0_q0 IS
PORT(A : in   std_logic_vector(3 downto 0);
       B : out std_logic_vector(3 downto 0));
END t0_q0;

ARCHITECTURE behavioral OF t0_q0 IS
BEGIN
   with A select
      B <= "1000" when "0000", -- 8
           "0001" when "0001", -- 1
           "0111" when "0010", -- 7
           "1101" when "0011", -- D
           "0110" when "0100", -- 6
           "1111" when "0101", -- F
           "0011" when "0110", -- 3
           "0010" when "0111", -- 2
           "0000" when "1000", -- 0
           "1011" when "1001", -- B
           "0101" when "1010", -- 5
           "1001" when "1011", -- 9
           "1110" when "1100", -- E
           "1100" when "1101", -- C
           "1010" when "1110", -- A
           "0100" when "1111"; -- 4
END behavioral;
```

VHDL Code for S-boxes_I

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Sboxes_I IS
PORT (Input_Sboxes : in   std_logic_vector(127 downto 0);
       Output_Sboxes : out std_logic_vector(127 downto 0));
END Sboxes_I;

ARCHITECTURE behavioral OF Sboxes_I IS
  COMPONENT t0_q0
    PORT(A : in   std_logic_vector(3 downto 0);
```

```
            B : out std_logic_vector(3 downto 0));
  END COMPONENT;
    ------------
  COMPONENT t0_q1
    PORT(A : in  std_logic_vector(3 downto 0);
            B : out std_logic_vector(3 downto 0));
  END COMPONENT;

BEGIN
S1: t0_q0 PORT MAP (A=>Input_Sboxes(127 downto 124),
                    B=>Output_Sboxes(127downto124));
 :
 :
S16: t0_q0 PORT MAP (A=>Input_Sboxes(67 downto 64),
                     B=>Output_Sboxes(67 downto 64));
S17: t0_q1 PORT MAP (A=>Input_Sboxes(63 downto 60),
                     B=>Output_Sboxes(63 downto 60));
 :
 :
S32: t0_q1 PORT MAP (A=> Input_Sboxes(3  downto  0),
                     B=>Output_Sboxes(3 downto  0));
END behavioral;
```

VHDL Code for Iterations_I

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Iterations_I IS
PORT (Input_K0, Input_K1, Input_K2, Input_K3,
      Input_K4, Input_K5, Input_K6,
      Input_K7      : in  std_logic_vector(127 downto 0);
  Input_R1, Input_R2: in  std_logic_vector(127 downto 0);
            Used_Ki : in  std_logic_vector(127 downto 0);
Input_Rotation_Selection_Bits :
                      in std_logic_vector(6 downto 0);
Input_Operation_Selection_Bits :
                      in std_logic_vector(1 downto 0);
           R1, R2 : out std_logic_vector(127 downto 0);
Ki_for_next_Iteration: out std_logic_vector(127 downto 0));
END Iterations_I;

ARCHITECTURE behavioral OF Iterations_I IS
SIGNAL Pre_Sboxes      : std_logic_vector(127 downto 0);
SIGNAL Pre_43LeftRotation
                       : std_logic_vector(127 downto 0);
SIGNAL Output_43LeftRotation
                       : std_logic_vector(127 downto 0);
SIGNAL Pre_Output_R1 : std_logic_vector(127 downto 0);
SIGNAL Pre_Output_R2 : std_logic_vector(127 downto 0);
SIGNAL X             : std_logic_vector(255 downto 0);
SHARED VARIABLE Power: integer RANGE 0 to 256:=0;
------------------
  COMPONENT Sboxes_I
    PORT(Input_Sboxes:in std_logic_vector(127downto0);
         Output_Sboxes:out std_logic_vector(127downto0));
  END COMPONENT;

BEGIN
Pre_Sboxes <= Input_R1 + Used_Ki;
S: Sboxes_I PORT MAP (Input_Sboxes => Pre_Sboxes,
                      Output_Sboxes => Pre_43LeftRotation);
Output_43LeftRotation <= Pre_43LeftRotation(84downto 0)
                       & Pre_43LeftRotation(127downto85);

Pre_Output_R1 <=   Output_43LeftRotation XOR Input_R2
WHEN               Input_Operation_Selection_Bits ="00"
ELSE NOT Output_43LeftRotation                 WHEN
                   Input_Operation_Selection_Bits ="01"
ELSE       Output_43LeftRotation               WHEN
                   Input_Operation_Selection_Bits ="11"
ELSE       Output_43LeftRotation               WHEN
                   Input_Operation_Selection_Bits="10"
         AND Input_Rotation_Selection_Bits ="0000000"
ELSE       Output_43LeftRotation(0)   &
           Output_43LeftRotation(127 downto1)  WHEN
                   Input_Operation_Selection_Bits ="10"
         AND Input_Rotation_Selection_Bits ="0000001"
ELSE
   :
   :
   :
ELSE       Output_43LeftRotation(125downto0)  &
           Output_43LeftRotation(127downto126) WHEN
                   Input_Operation_Selection_Bits ="10"
         AND Input_Rotation_Selection_Bits ="1111110"
ELSE       Output_43LeftRotation(126 downto 0) &
           Output_43LeftRotation(127)          WHEN
                   Input_Operation_Selection_Bits ="10"
         AND Input_Rotation_Selection_Bits ="1111111";

Pre_Output_R2 <=  Input_R1        WHEN
        Input_Operation_Selection_Bits ="00"
ELSE    Input_R2                  WHEN
        Input_Operation_Selection_Bits ="01" OR
        Input_Operation_Selection_Bits ="10" OR
        Input_Operation_Selection_Bits ="11";

PROCESS (X)
    BEGIN
        FOR j in 255 downto 0 LOOP
            IF X(j) = '1' THEN Power := Power + 1;
            END IF;
        END LOOP;
END PROCESS;

Ki_for_next_Iteration <=  Input_K0          WHEN
    (Power > 128 AND Used_Ki = Input_K7) OR
    (Power = 128 AND Used_Ki = Input_K0) OR
    (Power < 128 AND Used_Ki = Input_K1)
ELSE                  Input_K1          WHEN
    (Power > 128 AND Used_Ki = Input_K0) OR
    (Power = 128 AND Used_Ki = Input_K1) OR
    (Power < 128 AND Used_Ki = Input_K2)
ELSE
   :
   :
ELSE                  Input_K7          WHEN
     (Power > 128 AND Used_Ki = Input_K6) OR
     (Power = 128 AND Used_Ki = Input_K7) OR
     (Power < 128 AND Used_Ki = Input_K0);

R1 <= Pre_Output_R1;
R2 <= Pre_Output_R2;
END behavioral;
```

Magdy Saeb received the BSEE, School of Engineering, Cairo University, in 1974, the MSEE, and Ph.D. degrees in Electrical & Computer Engineering, University of California, Irvine, in 1981 and 1985, respectively. He was with Kaiser Aerospace and Electronics, Irvine California, and The Atomic Energy Establishment, Anshas, Egypt. Currently, he is a professor and head of the Department of Computer Engineering, Arab Academy of Science, Technology & Maritime Transport, Alexandria, Egypt; He was on-leave working as a principal researcher in the Malaysian Institute of Microelectronic Systems (MIMOS). He holds five International Patents in Cryptography. His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography Data Security Techniques, Encryption Processors, Mobile Agent Security.www.magdysaeb.net

Rabie A. Mahmoud received the B.Sc. Degree, Faculty of Science, Tishreen University, Latakia-Syria, in 2001, the MS. and Ph.D. in Computational Science, Faculty of Science, Cairo University, Egypt, in 2007 and 2011 respectively. Currently, he is working in General Organization of Remote Sensing (GORS), Damascus, Syria. His current interests include Cryptography, FPGA Implementations of Cryptography and Data Security Techniques. rabiemah@yahoo.com