A Metamorphic-Enhanced Twofish Block Cipher And Associated FPGA Implementation

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

1. General Organization of Remote Sensing (GORS),
Damascus, Syria.
rabiemah@yahoo.com

2. Computer Engineering Department,
Arab Academy for Science, Tech. & Maritime Transport (AAST),
Alexandria, Egypt.
mail@magdysaeb.net

Abstract: The Metamorphic-Enhanced Twofish Cipher is a metamorphic cipher that uses a variable word size and variable-size user's key. The cipher merged two ciphers by defining a new function using four bit-balanced operations. These operations are: XOR, INV, ROR, NOP for bitwise xor, invert, rotate right and no operation respectively. The new function replaces the h-function, previously used in the Twofish Cipher, and thus creating a Meta h-function. The aim of this alteration is to provide an improvement to the Twofish cipher that introduces high confusion into the enhanced Twofish without disturbing its linear and differential diffusion criteria. In this work, we discuss the Metamorphic-Enhanced Twofish Cipher and provide a Field Programmable Gate Array (FPGA) hardware implementation of the enhanced algorithm.

Keywords: Metamorphic Twofish, Block Cipher, Cryptography, Cryptographic Engineering, FPGA.

1. Introduction

The Metamorphic-Enhanced Twofish Cipher is a metamorphic cipher that improves the Twofish Cipher. In other words, the Metamorphic-Enhanced Twofish Cipher is a tied combination between a Stone Metamorphic Cipher [1], [2] and The Twofish Block Cipher [3], [4], [5]. It has four low-level operations that are all bit-balanced to encrypt the plaintext bit stream. These 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 the plaintext without any change (NOP). The sub-keys of The Metamorphic-Enhanced Twofish Cipher are generated using a combination of the Meta-Twofish encryption function itself (Meta-Twofish Algorithm) and a one-way hash function where the generated sub-keys stream is used to select the various operations. Moreover, the Meta-Twofish encryption function inherits the structure of the Twofish block cipher and uses the four bit-balanced operations in the *h* function of the Twofish to define the function *Meta-h*. This Meta-h is the heart of Meta-Twofish algorithm and is responsible for key expansion of the algorithm. The aim of this alteration is to provide an improvement to the Twofish cipher that introduces high confusion into the enhanced Twofish without disturbing its linear and differential diffusion criteria. In the following sections, we provide the structure of the Metamorphic-Enhanced Twofish Cipher, the structure of Meta-Twofish encryption function by defining the new function called *Meta-h* function, Moreover, we provide the details of a proposed hardware implementation for the function *Meta-h*, a discussion of the results of the FPGA implementation and finally a summary and our conclusions.

2. The Metamorphic Twofish Structure

The Metamorphic Twofish structure has the structure of the stone metamorphic cipher. Figure 1 shows the block diagram of the cipher. The Metamorphic-Enhanced Twofish Cipher is constructed of two basic functions; the Meta-Twofish encryption function and the sub-key generating one-way hash function. The pseudo random number generator is built using the same encryption function and the MDP-384 [6], [7] one-way hash function. Two large numbers (a, b) are used to iteratively generate the sub-keys. The details of the substitution box S-orb can be found in [8].

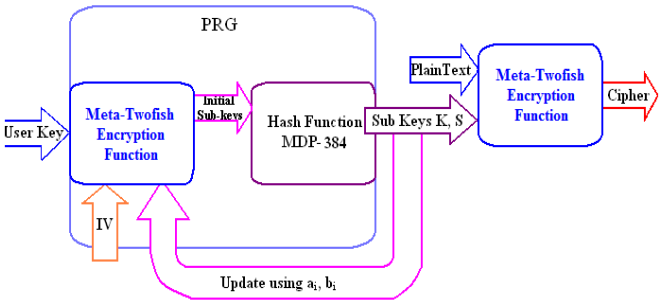

Figure 1: The structure of Metamorphic Twofish Cipher

The user key is first encrypted then the encrypted key is used to generate the sub-keys. The Meta-Twofish encryption

Received 1/2/2012, Reviewed 1/17/2012

function is built using the four low-level operations in Twofish encryption cipher. All operations are at the bit level composing the basic Crypto Logic Unit (CLU). More details of CLU can be found in [1] where the operation selection bits can be chosen from any two sub-key consecutive bits and 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	$C_i = \neg (P_i)$	"01"
ROR	$P_i \leftarrow (P_i, m)$	"10"
NOP	$C_i = P_i$	"11"

3. The Meta-Twofish Encryption Function

The Meta-Twofish encryption function uses the same structure of Twofish algorithm merging with the crypto logic unit in functions *h* in *F*-function. This configuration is used to generate expanded key words. The operation selection bits and the rotation selection bits are chosen from the sub-key bits. Figure 2 shows an overview the Meta-Twofish encryption function structure.

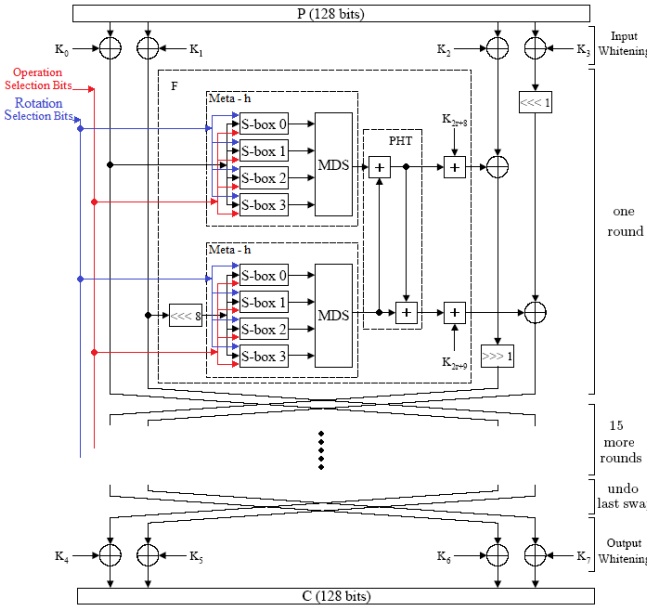

Figure 2: Meta-Twofish encryption function

The formal description of Meta-Twofish algorithm has the formal description of Twofish block cipher expect the function *h* which be modified to *Meta-h* function.

3.1 The Function *Meta-h*

The function *Meta-h* is a function that takes four inputs

- 32-bit word *X*
- List $L = (L_0, \dots, L_{k-1})$ of 32-bit words of length *k*
- 2-bit *operation selection bits*
- 3-bit *rotation selection bits*

and returns one 32-bit word of output where also this function works in *k* stages. In each stage, the four bytes are each passed through a fixed S-box then the basic crypto logic unit (CLU) which is applied one of functions XOR, INV, NOP, or ROR with a byte derived from the list L. The operation selection bits determine the applied function in CLU, while the 3-bit rotation selection bits determine the number of rotations which be provided for byte when ROR function is used. Finally, the bytes are once again passed through a fixed S-box, and the four bytes are multiplied by the MDS matrix. Figure 3 shows an overview of the function *Meta-h* for *k*=2 stage.

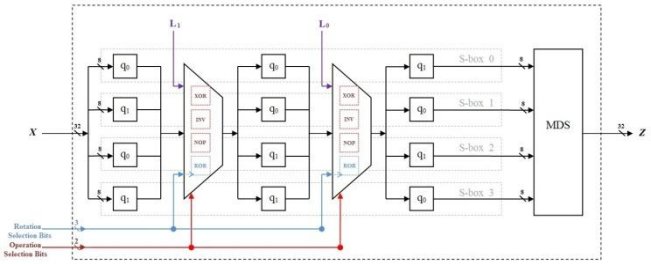

Figure 3: The function *Meta-h* for *k*=2 stages

Formally: The word *X* is splitting into bytes.

$$l_{i,j} = \lfloor L_i / 2^{8j} \rfloor \; mod \; 2^8$$
$$x_{i,j} = \lfloor X / 2^{8j} \rfloor \; mod \; 2^8$$

for $i = 0, \dots, k-1$ and $j = 0, \dots, 3$. Then the sequence of substitutions and CLUs is applied.

$$y_{k,j} = x_j \qquad j = 0, \dots, 3$$

If *operation selection bits* = "00"

If $k = 4$ then

$$y_{3,0} = q_1[y_{4,0}] \oplus l_{3,0}$$
$$y_{3,1} = q_0[y_{4,1}] \oplus l_{3,1}$$
$$y_{3,2} = q_0[y_{4,2}] \oplus l_{3,2}$$
$$y_{3,3} = q_1[y_{4,3}] \oplus l_{3,3}$$

If $k \geq 4$ then

$$y_{2,0} = q_1[y_{3,0}] \oplus l_{2,0}$$
$$y_{2,1} = q_1[y_{3,1}] \oplus l_{2,1}$$
$$y_{2,2} = q_0[y_{3,2}] \oplus l_{2,2}$$
$$y_{2,3} = q_0[y_{3,3}] \oplus l_{2,3}$$

In all cases

$$y_0 = q_1[q_0[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}]$$
$$y_1 = q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1}]$$
$$y_2 = q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}]$$
$$y_3 = q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}]$$

If *operation selection bits* = "01"

If $k = 4$ then

$$y_{3,0} = \neg q_1[y_{4,0}]$$
$$y_{3,1} = \neg q_0[y_{4,1}]$$
$$y_{3,2} = \neg q_0[y_{4,2}]$$
$$y_{3,3} = \neg q_1[y_{4,3}]$$

If $k \geq 4$ then

$$y_{2,0} = \neg q_1[y_{3,0}]$$
$$y_{2,1} = \neg q_1[y_{3,1}]$$

$$y_{2,2} = \neg\, q_0[y_{3,2}]$$
$$y_{2,3} = \neg\, q_0[y_{3,3}]$$

In all cases

$$y_0 = q_1[\neg\, q_0[\neg\, q_0[y_{2,0}]\,]\,]$$
$$y_1 = q_0[\neg\, q_0[\neg\, q_1[y_{2,1}]\,]\,]$$
$$y_2 = q_1[\neg\, q_1[\neg\, q_0[y_{2,2}]\,]\,]$$
$$y_3 = q_0[\neg\, q_1[\neg\, q_1[y_{2,3}]\,]\,]$$

If *operation selection bits* = "10"

If $k = 4$ then

$$y_{3,0} = ROR\,(q_1[y_{4,0}],\ m)$$
$$y_{3,1} = ROR\,(q_0[y_{4,1}],\ m)$$
$$y_{3,2} = ROR\,(q_0[y_{4,2}],\ m)$$
$$y_{3,3} = ROR\,(q_1[y_{4,3}],\ m)$$

If $k \geq 4$ then

$$y_{2,0} = ROR\,(q_1[y_{3,0}],\ m)$$
$$y_{2,1} = ROR\,(q_1[y_{3,1}],\ m)$$
$$y_{2,2} = ROR\,(q_0[y_{3,2}],\ m)$$
$$y_{2,3} = ROR\,(q_0[y_{3,3}],\ m)$$

In all cases

$$y_0 = q_1[ROR\,(q_0[ROR\,(q_0[y_{2,0}], m)], m)]$$
$$y_1 = q_0[ROR\,(q_0[ROR\,(q_1[y_{2,1}], m)], m)]$$
$$y_2 = q_1[ROR\,(q_1[ROR\,(q_0[y_{2,2}], m)], m)]$$
$$y_3 = q_0[ROR\,(q_1[ROR\,(q_1[y_{2,3}], m)], m)]$$

If *operation selection bits* = "11"

If $k = 4$ then

$$y_{3,0} = q_1[y_{4,0}]$$
$$y_{3,1} = q_0[y_{4,1}]$$
$$y_{3,2} = q_0[y_{4,2}]$$
$$y_{3,3} = q_1[y_{4,3}]$$

If $k \geq 4$ then

$$y_{2,0} = q_1[y_{3,0}]$$
$$y_{2,1} = q_1[y_{3,1}]$$
$$y_{2,2} = q_0[y_{3,2}]$$
$$y_{2,3} = q_0[y_{3,3}]$$

In all cases

$$y_0 = q_1[\,q_0[\,q_0[y_{2,0}]\,]\,]$$
$$y_1 = q_0[\,q_0[\,q_1[y_{2,1}]\,]\,]$$
$$y_2 = q_1[\,q_1[\,q_0[y_{2,2}]\,]\,]$$
$$y_3 = q_0[\,q_1[\,q_1[y_{2,3}]\,]\,]$$

where q_0 and q_1 are fixed permutation on 8-bit values, and *m* represents the integer number of "*rotation selection bits"*. The resulting vector ofy_i's is multiplied by the MDS matrix

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$Z = \sum_{i=3}^{3} z_i \,.\, 2^{8i}$$

Where: *Z* is the result of *Meta-h*.

4. The Algorithm

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

Algorithm: METAMORPHIC TWOFISH BLOCK CIPHER
INPUT: *Plain text message P, User Key K, Block Size B*
OUTPUT: *Cipher Text C*
Algorithm body:
Begin
Begin key schedule
1. Read user key;
2. Encrypt user key by calling Meta-Twofish encryption function and using the initial agreed-upon values as the random input to this function;
3. Read the values of the large numbers a and b from the encrypted key;
4. Generate a sub-key by calling the hash one-way function;
5. Store the generated value of the sub-key;
6. Repeat steps 5 and 6 to generate the required number of sub-keys;
End key schedule;

Begin Encryption
7. Read a block B of the message P into the message cache;
8. Use the next generated 128-bit key from the 384-bit key to bit-wise encrypt the plain text bits by calling the Meta-Twofish encryption function;
9. If message cache is not empty, Goto step 8;
10. Else if message cache is empty:
If message not finished
10.1 Load next block into message cache;
10.2 Goto 8;
Else if message is finished then halt;
End Encryption;
End Algorithm.

Function Meta-Twofish Encryption
Begin
1. Read next message bit;
2. Read next key bit from sub-key;
3. Read selection bits from sub-key;
4. Read rotation selection bits from sub-key;
5. Use selection & rotation bits to select and perform operation: XOR, INV, ROR, NOP in Meta-h functions in Meta-Twofish Algorithm;
6. Perform the encryption operation using plaintext bit and sub-key bit to get a cipher bit;
7. Store the resulting cipher bit;
End;

5. FPGA Implementation

The function *Meta-h is* applied to the *F*-function in various rounds of Meta-Twofish encryption function that leads to the FPGA-based implementation. We have implemented the function *Meta-h* applying the VHDL hardware description language [9], [10], [11] and utilizing Altera design environment Quartus II 9.1 Service Pack 2 Web Edition [12].

The function *Meta-h* circuit has 32-bit input which is splitting into four bytes, 32-bit L_i words, 2-bit operation-selection bits, and 3-bit rotation-selection bits. Thus, it produces a 32-bit output. Each byte of input is run through its own S-box and applying the metamorphic operations through crypto logic unit (CLU) with byte derived from the *L* list. The design was implemented using an EP2C70F896C6, Cyclone II family device. The schematic diagram for *Meta-h* function is shown in Figure 4. A series of screen-captures of the different design environment output are shown in Figures 5 to 12. Figures 5, 6, 7, 8, and 9 provide the indication of a successful compilation and parts of RTL for *Meta-h* function respectively. Figure 10 shows the technology map viewer of *Meta-h* function. Figure 11 demonstrates the floor plan. Figure 12 displays the simulator screen showing the output of *Meta-h* function for all operation selection states and rotation-selection bits equal to 101". The details of the analysis and synthesis report are shown in appendix A. The details of timing comparison between Meta-h function and h function is shown in appendix B.

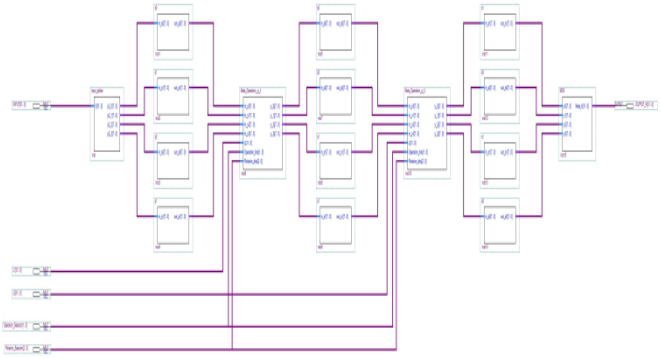

Figure 4: Schematic diagram of *Meta-h* function

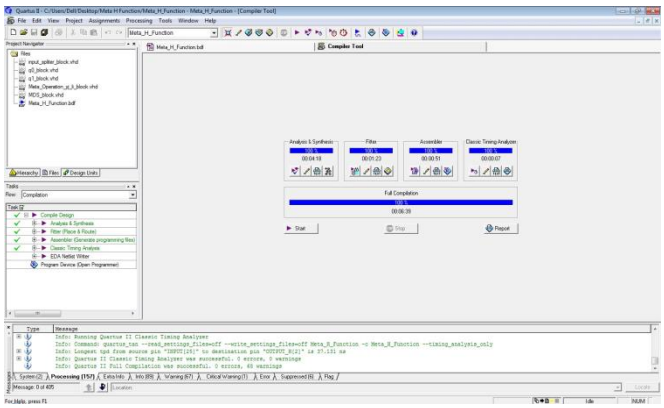

Figure 5: Compiler tool screen showing correct implementation

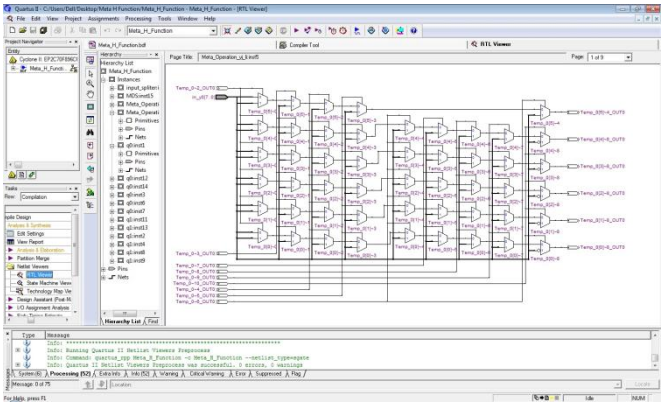

Figure 6: RTL screen for part of *Meta-h* function

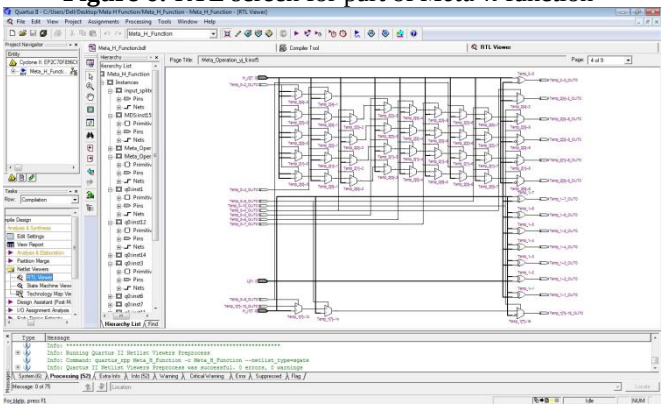

Figure 7: RTL screen for part of *Meta-h* function

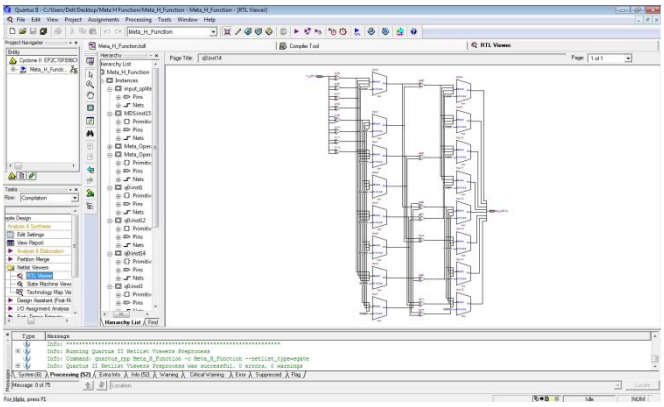

Figure 8: RTL screen for part of *Meta-h* function

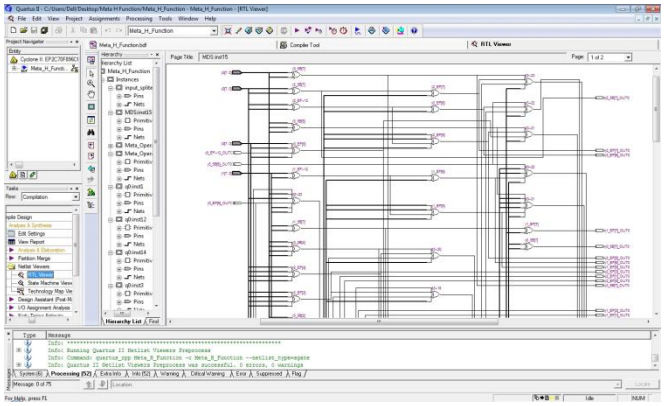

Figure 9: RTL screen for part of *Meta-h* function

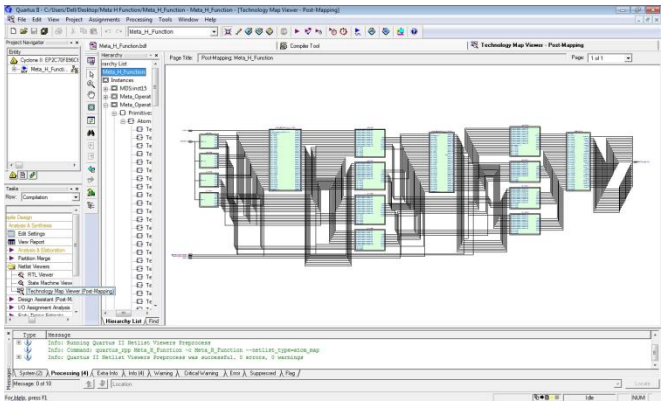

Figure 10: Technology Map Viewer of *Meta-h* function

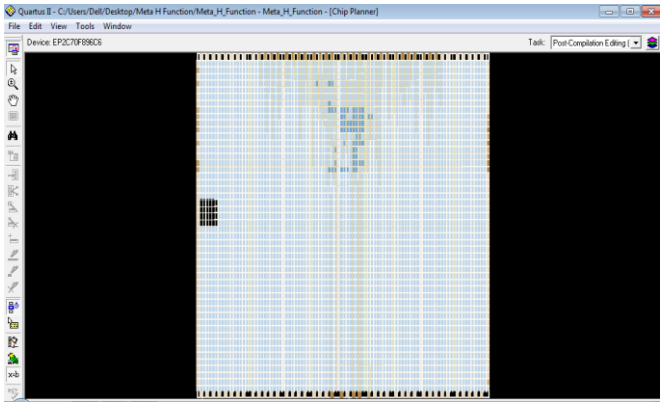

Figure 11: Floor-plan of *Meta-h* function

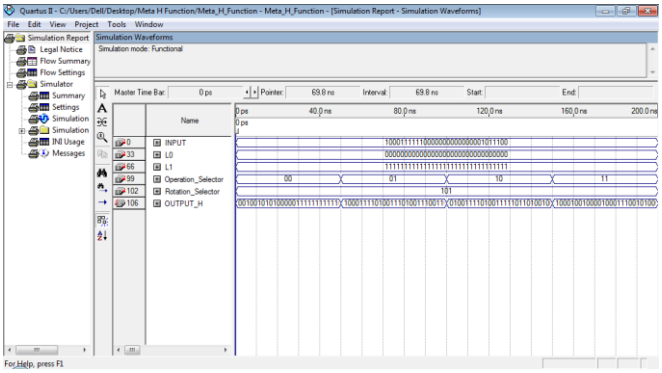

Figure 12: Simulator screen showing the output of *Meta-h* function for all operation selection states and rotation-selection bits ="101"

6. Summary and Conclusions

We have furnished a cipher that combines a metamorphic cipher and the well-known Twofish block cipher. Moreover, the modified Twofish algorithm, called Metamorphic-Enhanced Twofish Block Cipher, uses four bit-balanced operations in the core of the algorithm. This is the *Meta-h* function. This alteration provides an improvement to the Twofish Cipher by introducing high confusion into the enhanced Twofish without disturbing its linear and differential diffusion criteria. In addition, we have presented a hardware implementation of the function *Meta-h* by applying VHDL using the schematic editor, and the resulting circuit provides a proof-of-concept FPGA implementation. Balanced, area, and speed optimization techniques were performed and it was shown that the worst case pin-to-pin delay is equal to 37.131 ns in the case of balanced optimization, 39.831 ns in the case of area optimization and 39.055 ns in speed optimization. Speed optimization technique provides maximum Fan-Out although consumes worst case pin-to-pin delay, and area optimization provides minimum consuming of total logic elements. While the *Meta-h* function consumes more time as compared to by the *h* function, still the Metamorphic-Enhanced Twofish algorithm will appreciably increase the entropy and provide higher degree of randomness and conjectural security.

References

[1] 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.

[2] R. A. Mahmoud, M. Saeb, "Hardware Implementation of the Stone Metamorphic Cipher," International Journal of Computer Science and Network Security (IJCSNS), Vol.10, No.8, pp.54-60, 2010.

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

[4] B. Schneier's Twofish-support site: http://www.schneier.com/twofish.htm

[5] P. Chodowiec, K. Gaj, "Implementation of the Twofish Cipher Using FPGA Devices," Electrical and Computer Engineering, George Mason University, 1999.

[6] M. Saeb, "Design & Implementation of the Message Digest Procedures MDP-192 and MDP-384," ICCCIS 2009, International Conference on Cryptography, Coding & Information Security, Paris, June 24-26, 2009.
[7] R. A. Mahmoud, M. Saeb, "Hardware Implementation of The Message Digest Procedure MDP-384," International Journal of Computer Science and Network Security (IJCSNS), Vol.10, No.12, pp. 5-14, Dec., 2010.
[8] M. Saeb, "The Chameleon Cipher-192: A Polymorphic Cipher," SECRYPT 2009, International Conference on Security & Cryptography, Milan, Italy, July, 2009.
[9] P. P. Chu, "RTL Hardware Design Using VHDL," John Wiley & Sons, Inc., New Jersey, 2006.
[10] E. O. Hwang, "Digital Logic and Microprocessor Design with VHDL," La Sierra University, Riverside, California, USA, 2005.
[11] V. A. Pedroni, "Circuit Design with VHDL," MIT Press, 2004.
[12] Altera's user-support site: http://www.altera.com/support/examples/vhdl/vhdl.html

Appendix A: The analysis & synthesis and Fitter report details

Analysis & Synthesis Summary

Family: Cyclone II
Device: EP2C70F896C6
Total logic elements: 801 out of 68,416 (1 %)
-- Combinational with no register: 801
-- Register only: 0
-- Combinational with a register: 0
Total combinational functions: 801
Logic element usage by number of LUT inputs
-- 4 input functions: 554
-- 3 input functions: 112
-- <=2 input functions: 135
-- Register only: 0
Total pins: 133 out of 622 (21 %)
Total memory bits: 0 out of 1,152,000 (0 %)
Embedded Multiplier 9-bit elements: 0 out of 300 (0 %)
Total PLLs: 0 out of 4 (0 %)
Optimization Technique: Balanced
Maximum fan-out: 124
Total fan-out: 2854
Average fan-out: 3.05

Fitter Summary

Block interconnects: 905 out of 197,592 (< 1 %)
C16 interconnects: 105 out of 6,270 (2 %)
C4 interconnects: 455 out of 123,120 (< 1 %)
Direct links: 181 out of 197,592 (< 1 %)
Global clocks: 0 out of 16 (0 %)
Local interconnects: 384 out of 68,416 (< 1 %)
R24 interconnects: 110 out of 5,926 (2 %)
R4 interconnects: 511 out of 167,484 (< 1 %)
Nominal Core Voltage: 1.20 V
Low Junction Temperature: 0 °C
High Junction Temperature: 85 °C.

Also, the usage number of logic elements and their connections in the device can be changed depending on the optimization technique. Table 2 shows the number of usage logic elements in balanced, area, and speed optimization technique and Table 3 shows the number of interconnections between logic elements in balanced, area, and speed optimization technique.

Table 2: A synthesis comparison between optimization technique implementations of *Meta-h* Function

	Balance	Area	Speed
Total Logic Elements	801	780	817
4 input functions	554	514	540
3 input functions	112	124	155
<=2input functions	135	142	122
Total Fan-Out	2854	2744	2901
Average Fan-Out	3.05	3.00	3.04
Max Fan-Out	**124**	**126**	**129**

Table 3: A fitter comparison between optimization technique implementations of *Meta-h* Function

		Balance	Area	Speed
Interconnects	Block	905	898	931
	C16	105	142	109
	C4	455	515	491
	Local	384	382	402
	R24	110	87	96
	R4	511	519	497
Direct Links		181	181	198
Global Clocks		0	0	0

Pin-to-pin delays (T_{PD}) delays, which are the time required for a signal from an input pin to propagate through combinational logic and appear at an external output pin, were extracted from the timing reports of implementing balanced, area and speed optimization for synthezing the *Meta-h* function:

- In Balanced optimization technique, longest T_{PD} from source pin "INPUT[25]" to destination pin "OUTPUT_H[2]" was 37.131 ns.
- In Area optimization technique, longest T_{PD} from source pin "INPUT[0]" to destination pin "OUTPUT_H[31]" is 39.831 ns.
- In Speed optimization technique, longest T_{PD} from source pin "INPUT[21]" to destination pin "OUTPUT_H[18]" is 39.055 ns.

Appendix B: Timing comparison between *Meta-h* function and *h* function

Meta-h function consumes more time comparing by ordinary *h* function in Twofish algorithm where these delay approximately equal to 12 ns by implementing in the same EP2C70F896C6, Cyclone II device. Table 4 shows a comparison between *Meta-h* function and *h* function delays. Figure 13 shows a comparison chart of delays in our design.

Table 4: Delays comparison between *Meta-h* function and *h* function

	Balance	Area	Speed
Longest pin to pin delay for *Meta-h* Function	37.131	39.831	39.055
Longest pin to pin delay for *h* Function	25.868	26.707	27.713

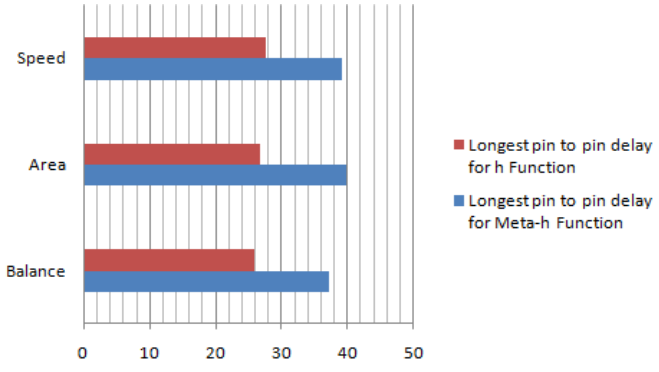

Figure 13: Delays in our design for *Meta-h* function and

h function

The four *Meta-h* functions to generate the expanded keys and *g* functions in each round of Meta-Twofish algorithm can be done in parallel. So, each round in Meta-Twofish algorithm consumes 12 ns and 192 ns for 16 rounds more than Twofish algorithm to encrypt 128-bit packet.

Appendix C: Sample VHDL code
For CLU in *Meta-h* function

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Meta_Operation IS
   port (in_y0, in_y1 :     in  std_logic_vector(7   downto 0);
         in_y2, in_y3 :     in  std_logic_vector(7   downto 0);
          L :               in  std_logic_vector(15 downto 0);
         Operation_bits :  in  std_logic_vector(1   downto 0);
          Rotation_bits   :  in  std_logic_vector(2   downto 0);
         out_y0, out_y1 : out std_logic_vector(7  downto 0);
         out_y2, out_y1 : out std_logic_vector(7  downto 0));
END Meta_Operation;

ARCHITECTURE behavioral OF Meta_Operation IS
  signal Temp_0, Temp_1 : std_logic_vector(7 downto 0);
  signal Temp_2, Temp_3 : std_logic_vector(7 downto 0);

BEGIN

Temp_0 <=  in_y0 xor L(31 downto 24)
    when   Operation_bits="00"   else     not in_y0
    when   Operation_bits="01"   else          in_y0
    when   Operation_bits="11"    else

              in_y0
    when Operation_bits="10" and Rotation_bits="000"  else
              in_y0(0) & in_y0(7 downto 1)
    when Operation_bits="10" and Rotation_bits="001"  else
              in_y0(1 downto 0) & in_y0(7 downto 2)
    when Operation_bits="10" and Rotation_bits="010"  else
              in_y0(2 downto 0) & in_y0(7 downto 3)
    when Operation_bits="10" and Rotation_bits="011"  else
              in_y0(3 downto 0) & in_y0(7 downto 4)
    when Operation_bits="10" and Rotation_bits="100"  else
              in_y0(4 downto 0) & in_y0(7 downto 5)
    when Operation_bits="10" and Rotation_bits="101"  else
              in_y0(5 downto 0) & in_y0(7 downto 6)
    when Operation_bits="10" and Rotation_bits="110"  else
              in_y0(6 downto 0) & in_y0(7)
    when Operation_bits="10" and Rotation_bits="111";
--------------------------------

Temp_1 <=  in_y1 xor L(23 downto 16)
    when   Operation_bits="00"   else
              not in_y0
      :
      :
      ------------------------------

Temp_2 <=  in_y2 xor L(15 downto 8)
    when   Operation_bits="00"   else
              not in_y2
      :
      :
      --------------------------------

Temp_3 <=  in_y3 xor L(7 downto 0)
    when   Operation_bits="00"   else
              not in_y3
      :
      :
      --------------------------------

   out_y0 <= Temp_0;
   out_y1 <= Temp_1;
   out_y2 <= Temp_2;
   out_y3 <= Temp_3;

END behavioral;
```

Magdy Saeb received the BSEE., School of Engineering, Cairo University, in 1974; the MSEE. and Ph.D. 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, Magdy Saeb is a professor in the Department of Computer Engineering, Arab Academy for Science, Technology & Maritime Transport, Alexandria, Egypt, He was on leave to Malaysian Institute of Microelectronic Systems (MIMOS), Kuala Lumpur, Malaysia. His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography Data Security Techniques, Mobile Agent Security, Bioinformatics. 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 Image processing, Cryptography, FPGA Implementations of Cryptography and Data Security Techniques. rabiemah@yahoo.com