INTERNATIONAL JOURNAL OF SCIENTIFIC PROGRESS AND RESEARCH (IJSPR) ISSN: 2349-4689
Volume-09, Number - 03, 2015

Implementations of the Hummingbird Cryptographic Algorithm using FPGA

Prof. Anil R. Patil[1], Prof. Ashutosh S. Kale[2]
[1]*Assist. Professor, Dept. of E&TC, GCE, Karad, Maharashtra (India)*
[2]*HOD, Dept. of Comp., S.S.C. POLY, Chalisgaon, Maharashtra (India)*

Abstract- Hummingbird is a recently proposed ultra-lightweight cryptographic algorithm targeted for low-cost smart devices like RFID tags, smart cards etc. It has a hybrid structure of block cipher and stream cipher and was developed with both lightweight software and lightweight hardware implementations for constrained devices in mind. Lightweight cryptography (LWC) is an emerging research area which has to deal with the trade-off among security, cost, and performance. Moreover, Hummingbird has ability resist common attacks to block ciphers and stream ciphers including differential and linear cryptanalysis, birthday attack, structure attacks, algebraic attacks, cube attacks etc The report presents the algorithms for the encryption as well as decryption process and shows some simulation results performed on Xilinx.

Keywords - Lightweight cryptography, resource constrained devices, Hummingbird.

I. INTRODUCTION

The widespread deployment of various wireless networks such as mobile ad-hoc networks, sensor networks, mesh networks, personal area networks and RFID systems is making possible a world of pervasive computing a reality. While the wireless communication technology and devices under development are enabling our march toward the era of pervasive computing, the security and privacy concerns in pervasive computing remains a serious impediment to widespread adoption of emerging technologies. Employing cryptographic primitives to perform strong authentication and encryption and provide other security functionalities is a promising solution to overcome those concerns.

Hummingbird is a recently proposed lightweight cryptographic algorithm targeted for low-cost smart devices like smart cards, RFID tags and wireless sensor nodes. It has a mixed structure of block cipher and stream cipher and was developed with both lightweight software and lightweight hardware implementations for constrained devices in mind. Moreover, it has ability resist common attacks to block ciphers and stream ciphers including differential and linear cryptanalysis, birthday attack, structure attacks, algebraic attacks, cube attacks, etc. In practice, Hummingbird has been implemented across a wide range of different target platforms. Those implementations demonstrate that Hummingbird provides efficient and flexible software solutions for various embedded applications. As a result, our main contribution in is provide the first efficient hardware implementations of Hummingbird encryption/decryption cores on low-cost FPGAs.

Hummingbird's Advantages:

1) It is secure.

2) It is well-suited for resource-constrained environments. The state space of the algorithm requires little memory. Encryption and decryption take few operations to implement, so that Hummingbird requires few gates.

3) It appears to be appropriate for either software or hardware implementation.

4) It is key-agile, that is, it is able to switch keys easily and rapidly

5) There is no cipher-text expansion unless a message authentication code is added to the cipher text.

II. THE HUMMINGBIRD CRYPTOGRAPHIC ALGORITHM

This algorithm is based on rotor machine equipped with novel rotor-stepping rules and design is based on block cipher & stream cipher. The design of algorithm is based combination of a block cipher and stream cipher with 16-bit block size, 256-bit key size, and 80-bit internal state. Figure 1 and Figure 2 shows the initialization and encryption flow of the Hummingbird cryptographic algorithm, respectively. Initialization and encryption has a four 16-bit block ciphers *Eki* (*i* = 1, 2, 3, 4), four 16-bit internal state registers *RSi* (*i* = 1, 2, 3, 4), and a 16-stage Linear Shift Feedback Register (LFSR). Moreover, the 256-bit secret key *K* is divided into four 64-bit sub keys *k*1, *k*2, *k*3 and *k*4 which are used in the four block ciphers respectively.

1.1. Initialization Process

The structure of the Hummingbird initialization algorithm is shown in Figure 1. To use Hummingbird in practice, four 16-bit random nonce's NONCE*i* are first choose to initialize the four internal state registers *RSi* (*i* = 1, 2, 3, 4) respectively followed by four consecutive encryptions on the message *RS*1 xor *RS*3 by Hummingbird running in initialization mode. The final 16-bit cipher text *TV* is used to initialize the LFSR. Moreover, the 13th bit of the LFSR is always set to prevent a zero register. The LFSR is also stepped once before it is used to update the internal state register *RS*3.

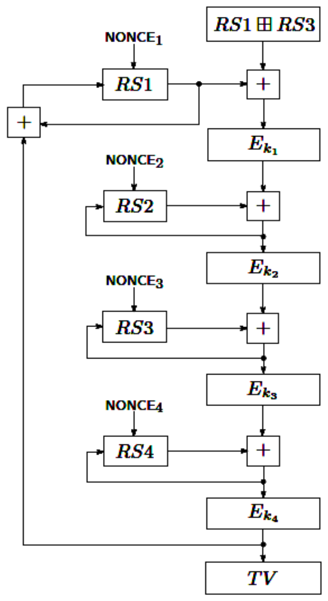

Figure 1: Initialization Process

The initialization process of Hummingbird is described by following algorithm

INPUT: Four 16-bit random nonce NONCEi (i=1,2,3,4)

OUTPUT: Initialization four rotors RSi4(i=1,2,3,4) and LFSR

```
1: RS10 = NONCE1

2:RS10=NONCE1                [NONCE INITIALIZATION]
3: RS10 = NONCE1

4: RS10 = NONCE1

5: For t = 0 to 3 do

6: V12t = Ek1 ((RS1t⊕RS3t) ⊕ RS1t)

7: V23t = Ek2 ((V12t⊕RS2t)

8: V34t = Ek3 ((V23t⊕RS3t)

9: TVt = Ek4 ((V34t⊕RS4t)

10: RS1t+1 = RS1t ⊕ TVt

11: RS2t+1 = RS2t ⊕ V12t

12: RS3t+1 = RS3t ⊕ V23t

13: RS4t+1 = RS4t ⊕ V34t

14: End for

15: LFSR = TV3/ 0x 1000

16: Return RSi4 (i=1,2,3,4) and LFSR [LFSR INITIALIZATION)
```

1.2. Encryption Process

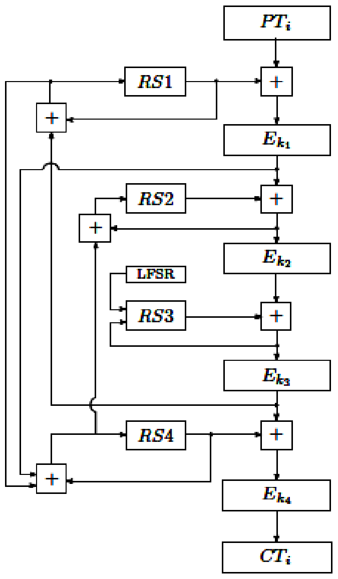

Figure 2: Encryption Process

The overall structure of the Hummingbird encryption algorithm is depicted in Figure 2. Once a system initialization process completed a 16-bit plaintext block *PTi* is encrypted by first executing a modulo 2^{16} addition of *PTi* and the content of the first internal state register *RS*1. The addition result is then encrypted by the first block cipher *Ek*1. This procedure is repeated for another three times and the output of *Ek*4 is the corresponding cipher text *CTi*. Furthermore, the states of the four internal state registers will also be updated in an unpredictable way based on their current states, the state of the LFSR and the outputs of the first three block ciphers, and the encryption process of Hummingbird is described by following algorithm

INPUT: A16-bit plaintext PTi and four RSi (i=1,2,3,4)

OUTPUT: A16-bit cipher text CTi

```
1: V12t = Ek1 (PTi ⊕ RS1t)          [Block Encryption]

2: V23t = Ek2 ((V12t ⊕ RS2t)

3: V34t = Ek3 ((V23t ⊕ RS3t)

4: CTi = Ek4 (V34t ⊕ RS4t)

5: LFSRt+1<= LFSRt                  [Internal State Updating]

6: RS1t+1 = RS1t ⊕ V34t

7: RS3t+1 = RS3t ⊕ V23t ⊕ LFSRt+1

8: RS4t+1 = RS4t ⊕ V12t ⊕ RS1t+1

9: RS2t+1 = RS2t ⊕ V12t ⊕  RS4t+1

10: Return CTi
```

1.3. Decryption Process

The structure of the Hummingbird decryption algorithm is shown in Figure 3.

The decryption process is same as the encryption and a detailed description is shown in the following Algorithm 3.

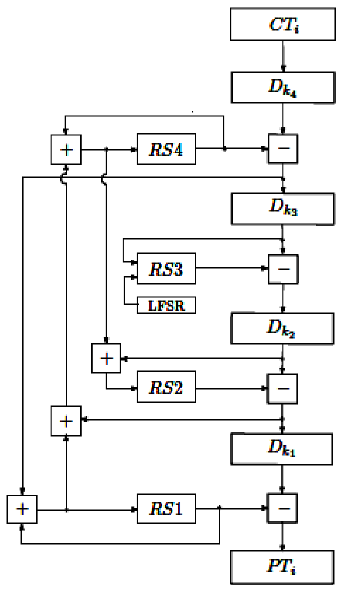

Figure3: Decryption process

1.4. The Decryption process of Hummingbird is described by following algorithm

Input: A 16-bit ciphertext CT_i and four rotors RSi_t $(i = 1, 2, 3, 4)$
Output: A 16-bit plaintext PT_i

```
1: $V34_t = D_{k_4}(CT_i) \boxminus RS4_t$                           [Block Decryption]
2: $V23_t = D_{k_3}(V34_t) \boxminus RS3_t$
3: $V12_t = D_{k_2}(V23_t) \boxminus RS2_t$
4: $PT_i = D_{k_1}(V12_t) \boxminus RS1_t$
5: $\mathrm{LFSR}_{t+1} \leftarrow \mathrm{LFSR}_t$                  [Internal State Updating]
6: $RS1_{t+1} = RS1_t \boxplus V34_t$
7: $RS3_{t+1} = RS3_t \boxplus V23_t \boxplus \mathrm{LFSR}_{t+1}$
8: $RS4_{t+1} = RS4_t \boxplus V12_t \boxplus RS1_{t+1}$
9: $RS2_{t+1} = RS2_t \boxplus V12_t \boxplus RS4_{t+1}$
10: return $PT_i$
```

2.4 Bit Block Cipher

Hummingbird uses four identical block ciphers *Eki* (*i* = 1, 2, 3, 4) in a consecutive Manner consisting of substitution-permutation (SP) network with 16-bit block Size and 64-bit key as shown in the following figure.

The block cipher consists of four regular rounds and a final round. The 64-bit sub key *ki* is split into four 16-bit round keys *K*(*i*)1 ,*K*(*i*)2, *K*(*i*)3 and *K*(*i*)4 that are used in the four regular rounds respectively. Moreover, the final round uses two keys *K*(*i*)5 and *K*(*i*)6 directly derived from the four round keys . While each regular round consist of a key mixing step, a permutation layer and a substitution layer, the final round only includes the key mixing and the S-box substitution steps. For key mixing step simple exclusive-OR operation is used, whereas the substitution layer is composed of four S-boxes with 4-bit inputs and 4-bit outputs as shown in Table 2.

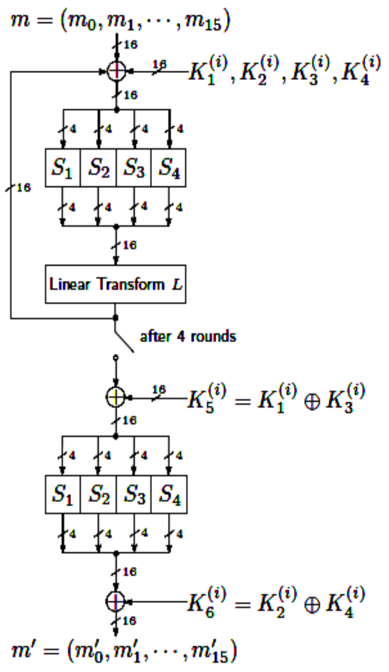
$m = (m_0, m_1, \cdots, m_{15})$
16
$K_1^{(i)}, K_2^{(i)}, K_3^{(i)}, K_4^{(i)}$
4
S_1
S_2
S_3
S_4
Linear Transform L
after 4 rounds
$K_5^{(i)} = K_1^{(i)} \oplus K_3^{(i)}$
$K_6^{(i)} = K_2^{(i)} \oplus K_4^{(i)}$
$m' = (m'_0, m'_1, \cdots, m'_{15})$

Figure4: 16-Bit Block Cipher

Table 2: four s box hexadecimal notation

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
$S_2(x)$	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
$S_3(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_4(x)$	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

The selected four S-boxes denoted by *Si*(*x*) : F42→ F42 *i* = 1, 2, 3, 4, are Serpent-type S boxes with additional properties which can ensure that the 16-bit block cipher is resistant to linear and differential attacks as well as interpolation attack. The permutation layer in the 16-bit block cipher is given by the linear transform L:{0,1}16 {0,1}16 defined as follows:

L (m) =m XOR (m<<6) XOR (m<<10);

Where m = (m0, m1, ------m15) is a 16-bit data block.

2.5 Selection of a "Hardware-Friendly" S-Box

This S-box can be efficiently implemented in the target hardware platform with a small area requirement. Four 4×4 S-boxes *Si*(*x*) : F42→ F42 (*i* = 1, 2, 3, 4) have been carefully selected in Hummingbird according to certain security criteria To implement the compact version of Hummingbird, we need to choose a "hardware-friendly" S-box from four S-boxes listed in Table 2. Let *x* = (*x*3 // *x*2 // x1 // *x*0) be the 4-bit input to the S-box and let *Si*(*x*) = (*S*(3) *i* (*x*) // *S*(2) *i* (*x*) // *S*(1) *i* (*x*) // *S*(0) *i* (*x*)) denote the 4-bit output of the *i*-th S-box (*i* = 1*;* 2*;* 3*;* 4). By using the Boolean minimization tool *Espresso* we can obtain the following minimal Boolean function representations (BFR) for the four S-boxes in Hummingbird as shown in Table 3, where *xi* denotes the inversion of bit *xi*, . denotes a logical AND and + denotes a logical OR. Note that each S-box can be implemented in hardware by using either a look-up table (LUT) or the Boolean function representations (i.e., combinatorial logic). The exact efficiency of the above two approaches significantly depends on specific hardware platforms and synthesis tools. Therefore for each proposed architecture of the 16-bit block cipher we investigate two implementation strategies (i.e., BFR and LUT) for the four S-boxes respectively, and select one that results in the most area-efficient implementation of the 16-bit block cipher.

Table3: Boolean function representations (BFR) for the four S-boxes

S-boxes	Minimal Boolean Function Representations
$S_1(x)$	$S_1^{(0)}(x) = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_1 \cdot x_0$
	$S_1^{(1)}(x) = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot \bar{x}_2 \cdot \bar{x}_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_0 + x_3 \cdot \bar{x}_1 \cdot \bar{x}_0$
	$S_1^{(2)}(x) = \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 + \bar{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_0 + x_3 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0$
	$S_1^{(3)}(x) = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0$
$S_2(x)$	$S_2^{(0)}(x) = \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_0 + x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_3 \cdot \bar{x}_1 \cdot \bar{x}_0$
	$S_2^{(1)}(x) = \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot \bar{x}_1 \cdot \bar{x}_0$
	$S_2^{(2)}(x) = \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_1 \cdot \bar{x}_0 + x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot \bar{x}_1$
	$S_2^{(3)}(x) = \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot \bar{x}_1$
$S_3(x)$	$S_3^{(0)}(x) = x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + x_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot x_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_1$
	$S_3^{(1)}(x) = x_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 + x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_2 \cdot x_1 \cdot \bar{x}_0$
	$S_3^{(2)}(x) = \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot x_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_1$
	$S_3^{(3)}(x) = \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_1 \cdot \bar{x}_0$
$S_4(x)$	$S_4^{(0)}(x) = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot \bar{x}_1 \cdot x_0$
	$S_4^{(1)}(x) = \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + x_3 \cdot x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot x_0 + \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot x_1 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0$
	$S_4^{(2)}(x) = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_0$
	$S_4^{(3)}(x) = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot \bar{x}_0 + x_3 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1 \cdot \bar{x}_0$

2.6 Loop-Unrolled Architecture of 16-bit Block Cipher

The loop-unrolled architecture for the 16-bit block cipher is shown in Figure 5. In below architecture, only one 16-bit data is processed at a time. However, five rounds are cascaded and the whole encryption can be performed using single clock cycle. The architecture consists of 8 XORs, 20 S-boxes, and 4 permutation layers for the data path. After the given 16-bit block is XORed with the first round key, the obtained result is divided into four 4-bit chunks and each of them is then processed by a 4-bit S-box in parallel. The linear transform *L* performs a permutation on the output of the S-box layer for each of four regular rounds. The final round includes only S-box layer and four XOR operations and the output cipher text is stored into a 16-bit flip-flop. To select a “hardware-friendly” S-box for the compact version of Hummingbird, we implement the loop-unrolled architecture of the 16-bit block cipher on the target FPGA and summarize the area requirement when using different S-boxes and implementation strategies.

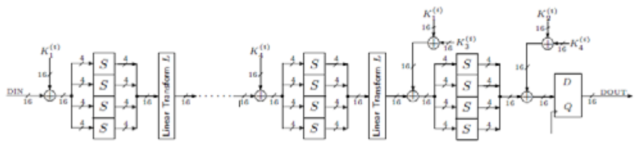

Figure 5: Loop-Unrolled Architecture of 16-bit Block Cipher

When comparing different S-boxes and implementation strategies, Table 4 shows that the loop-unrolled architecture occupies the minimal number of slices provided that the S-box *S*3(*x*) is employed and implemented by a LUT.

III. IMPLEMETATION AND RESULTS

VHDL is used as the hardware description language because of the flexibility to exchange among environments. The coding is VHDL that could easily be implemented on other devices, without any change in design. The software used for this work is Xilinx9.2i. This is used for writing, debugging and optimizing efforts and also for fitting simulating and checking the performance results using the simulation tools available on spartan3e design software

3.1 Implementation of Encryption

3.1.1. Simulation waveform of hummingbird encryption

3.1.2. Simulation result of hummingbird encryption

In this, the initialization process begins and four rotors *RSi* (*i* = 1, 2, 3, 4) are first initialized by four 16-bit random nonce through the interface *RSi* within four clock cycles. Once the initialization process is done after 20 clock cycles, the first 16-bit plaintext block is read from an external register for encryption. With another four clock cycles, the corresponding cipher text is output from the encryption core.

The above result is obtained on Xilinx ISE 9.2i by giving a 64-bit initialization vector 0ABECDAA8ADC6DTF which generates a plaintext of 16- bit 2F3C. Key of length 256 bit is used with value 123470000 and cipher text obtained is 16 bit BDB8. The simulation is kept for 1000ns.

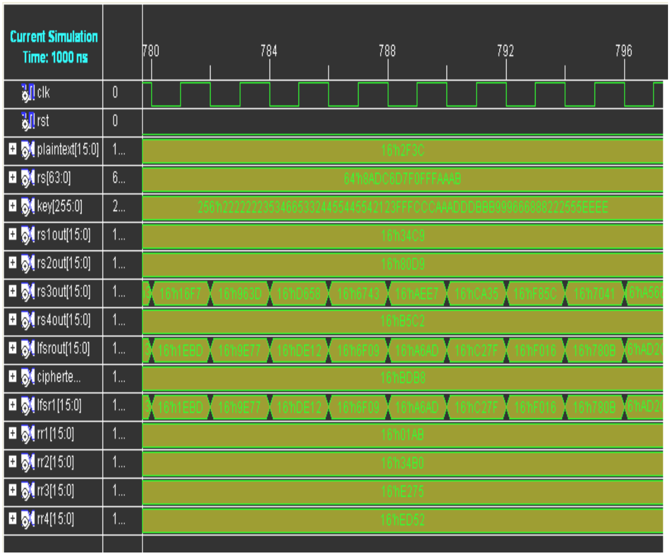
Current Simulation Time: 1000 ns
780
784
788
792
796
clk 0
rst 0
plaintext[15:0] 1... 16'h2F3C
rs[63:0] 6... 64'h8ADC6D7F0FFFAAAB
key[255:0] 2... 256'h22222223534665332445544554212 3FFFCCCAAADDDBBB999666888222555EEEE
rs1out[15:0] 1... 16'h34C9
rs2out[15:0] 1... 16'h80D9
rs3out[15:0] 1... 16'h16F7 16'h963D 16'hD658 16'h6743 16'hAEE7 16'hCA35 16'hF85C 16'h7041
rs4out[15:0] 1... 16'hB5C2
lfsrout[15:0] 1... 16'h1EBD 16'h9E77 16'hDE12 16'h6F09 16'hA6AD 16'hC27F 16'hF016 16'h780B
cipherte... 1... 16'hBDB8
lfsr1[15:0] 1... 16'h1EBD 16'h9E77 16'hDE12 16'h6F09 16'hA6AD 16'hC27F 16'hF016 16'h780B
rr1[15:0] 1... 16'h01AB
rr2[15:0] 1... 16'h34B0
rr3[15:0] 1... 16'hE275
rr4[15:0] 1... 16'hED52

Figure 6: Simulation waveform of hummingbird encryption

3.2. Implementation of Decryption

3.2.1 Simulation waveform of hummingbird decryption

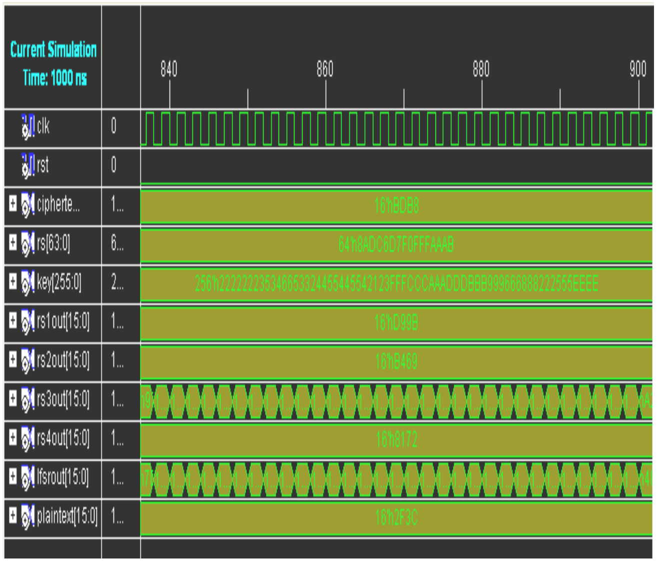
Current Simulation Time: 1000 ns
840
860
880
900
clk 0
rst 0
cipherte... 1... 16'hBDB8
rs[63:0] 6... 64'h8ADC6D7F0FFFAAAB
key[255:0] 2... 256'h22222223534665332445544554212 3FFFCCCAAADDDBBB999666888222555EEEE
rs1out[15:0] 1... 16'hD99B
rs2out[15:0] 1... 16'hB469
rs3out[15:0] 1...
rs4out[15:0] 1... 16'h8172
lfsrout[15:0] 1...
plaintext[15:0] 1... 16'h2F3C

Figure 7: Simulation waveform of hummingbird decryption

3.2.2. Simulation result of hummingbird Decryption

The above result is obtained on Xilinx ISE 9.2i by giving a 64-bit initialization vector 0ABECDAA8ADC6DTF which generates a plaintext of 16- bit 2F3C. Key of length 256 bit is used with value 123470000 and cipher text obtained is 16 bit BDB8. The simulation is kept for 1000ns.

IV. CONCLUSIONS

This paper details about lightweight cryptography and its types and discusses the implementation of ultra lightweight cryptographic algorithm Hummingbird. The security and performance factor is very precisely achieved by the algorithm due to its prominent internal structure. Compared to other lightweight FPGA implementations of block ciphers XTEA, ICEBERG, SEA, AES, Hummingbird can achieve larger throughput with the smaller area requirement. Consequently, Hummingbird can be considered as an ideal cryptographic primitive for resource constrained environment. The efficient FPGA implementation of Hummingbird is possible using the given software algorithms so that it can achieve larger throughput with smaller area requirement. Also, Hummingbird can be used in high-security required devices as it is resistant to most cryptographic attacks

Optimized and Synthesizable VHDL code is developed for the implementation of both 16-bit block cipher and loop unrolled architecture of 16-bit block cipher is verified using ISE 9.2i functional simulator from Xilinx. All the transformations of algorithm are simulated using an iterative design approach in order to minimize the hardware consumption.

REFERENCES:

[1] P. Bulens, F.X. Standaert, J.J. Quisquater and P. Pellegrin, “Implementation of the AES 128 on Virtex-5 FPGAs”, Progress in Cryptology *AFRICACRYPT 2008*, LNCS 5023, pp. 16-26, 2008.

[2] D. Engels, X. Fan, G. Gong, H.Hu, and E. M. Smith, “Hummingbird Ultra-Lightweight Cryptography for Resource Constrained Devices”, to appear in the proceedings of The 14th international conference on Financial Cryptography and data security-FC 2010, 2010.

[3] X. Fan, H.Hu, G.Gong, E.M.Smith and D. Engels,“Lightweight Implementation of Humming bird Cryptographic Algorithm on 4-Bit Microcontrollers”, *The 1st International Workshop on RFID Security and Cryptography 2009 (RISC’09)*, pp. 838-844, 2009

[4] J. Lee and Y. Yeom, “Efficient RFID authentication protocols based on pseudorandom sequence generators”, Des. Codes Cryptogr. 51:195–210, 2009

[5] Yangheng, et al, FPGA / CPLD latest practical technical guidelines, Tsinghua University .

[6] T.Eisenbarth, S. Kumar, C. Paar, A.Poschman, and L. Uhsadel, “ A Survey of Lightweight-Cryptography Implementations”, IEEE Design & Test of Computers, vol. 24, no.6,pp.522-533,2007

[7] Xilinx Inc., “Spartan-3 FPGA Family Data Sheet”, DS099, December 4, 2009, available at http://www.xilinx.com/support/documentation/datasheet/ds099.pdf