Maximum Distance Based Key Hopping (MDBKH) and Associated Implementation on FPGA

Rabie A. Mahmoud

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

Abstract: *Maximum Distance Based Key Hopping (MDBKH) is a process of key hopping that can be used to interchange the user key of a cipher. Maximum distance based key hopping is depending on the maximum difference between the plaintext packet and ciphertext packet to change the user key by computing all distances between the plaintext packet and ciphertext packet then choose the key which is related with the maximum distance. In this work, we discuss various key hopping methods and suggest a procedure of maximum distance based key hopping. Moreover, we provide a Field Programmable Gate Array (FPGA) hardware implementation of the proposed key hopping technique.*

Keywords: Maximum Distance Based Key Hopping, Cryptography, Cyclone II, FPGA.

1. Introduction

Maximum Distance Based Key Hopping (MDBKH) is a key hopping method that is based on the differences between the plaintext packet and ciphertext packet to provide a new procedure of key hopping. MDBKH utilizes four keys; one acts as the authentication key where the maximum distance between the plaintext packet and ciphertext packet will be the standard of changing the key. In the following sections, we provide a review of various key hopping methods, the structure of maximum distance based key hopping and the formal description of maximum distance based key hopping algorithm. Moreover, we provide the details of our hardware implementation, a discussion of the results of the FPGA implementation and finally a summary and our conclusions.

2. Key Hopping Techniques

Key hopping techniques were known through NextComm Key Hopping, State Based Key Hop, Dynamic Re-Keying with Key Hopping, and Power Based Key Hopping.

● "NextComm" [1] has proposed changing the keys frequently to overcome the weakness of Wired Equivalent Privacy (WEP) without incurring possible overheads and other complexities of the network. The key hopping process is designed depending on shared keys, session seeds, and session keys which are used in the same way in encryption/decryption as the shared keys are used in the standard WEP process. Shared keys are not used directly for encryption/decryption but the session keys are derived using a robust hash function for encryption/decryption purposes where the 128-bit MD5 one-way hash function is chosen. Session seeds which are the output of the large counter are changed frequently to ensure that the resulting session keys not to be reused where the size of the counter should also be large enough to ensure the seeds are not reprocessed during the lifetime of the secret shared keys.

● State Based Key Hop (SBKH) protocol [2] is created to provide a strong lightweight encryption scheme for battery operated 802.11 devices. SBKH includes base key pair, key duration, RC4 states, offset and an explicit SBKH sequence counter as the sharing parameters between communicating entities. Base key pair consists of two 128-bit keys which are shared by all entities in a network. The keys can be used as per-session keys that are agreed upon between the two communicating nodes. SBKH uses a 24-bit initialization vector IV within the original 802.11 WEP data frames as a SBKH sequence counter.

● Dynamic Re-keying with Key Hopping (DRKH) protocol [3] uses RC4 encryption technique and secret list of the static keys. One of these keys is an authentication key. The transmitted packets in DRKH between the two communicating nodes take place in several consecutive sessions where the access point (AP) in WLANs will be responsible for determining the sessions' start and end times and maintains the session counter. A one-way hash function such as MD5 or SHA-1 is used to hash the session counter with each of the four secret keys to generate four session keys and then these session keys are used to encrypt/decrypt packets during the lifetime of a session. An initialization vector (IV) is used in DRKH where each session key has a corresponding IV and incremented for every new packet to be encrypted using that particular session key. Non-linear look-up-table based substitution technique to mix a session key with its corresponding IV value is used to re-initialize RC4 state instead of using the key scheduling algorithm where one of the four derived session keys is selected according to a previously agreed upon key hopping sequence for every packet to be encrypted. The key hopping sequence determines the order in which session keys are used to encrypt/decrypt packets where the AP sends the key hopping sequence to the wireless station after the completion of a successful authentication process.

Received 7/19 Reviewed 7/27

● Power Based Key Hopping (PBKH) [4] proposed method utilizes four keys, one of them will be an authentication key and the first key will be used as the default key at each time the communication session is started. The first key will be used to encrypt/decrypt the first plaintext packet then the sender and receiver compute the power of the ciphertext to determine which key will be used to encrypt/decrypt the next packet. When the power of ciphertext packet which is the number of ones equals a certain predefined value known before hand between the communicating entities, the same key is used to encrypt/decrypt the next packet. If the power of ciphertext packet is larger than this value, then the next key from the set of secret keys is to be used. However, the previous key from secret key list is to be used to encrypt/decrypt the next packet when the power of ciphertext is less than this certain value. Initialization vector IV is concatenated to the keys and incremented by one each clock independent of the used key.

3. Maximum Distance Based Key Hopping (MDBKH)

Our proposed method uses the maximum distance concept to implement in key hopping technique. Distance refers to the number of elements that differ between any two distinct vectors. Maximum distance between the plaintext packet and ciphertext packets, which are encrypted by user keys, will be determined the chosen key between communication entities. The proposed method utilities four secret keys, one of them will be an authentication key between the communication entities and the other three secret keys will be used at each time the communication session is started. Each plaintext packet will be encrypted three times by using the three secret keys producing three ciphertext packets, and then the distances will be computed. Four distance values will be extracted from the plaintext packet and one of their related ciphertext packets representing the number of observations of (1, 1), (1, 0), (0, 1), and (0, 0) between the plaintext packet and the ciphertext packet respectively. In other words, twelve distance values will be appearing from each plaintext packet and their related three ciphertext packets. Maximum value of those twelve distances will be the criterion of chosen key between the communication entities. Figure 1 shows the maximum distance based key hopping scheme where each key has four states of distances. Initialization vector IV will be concatenated to the keys and incremented by one each clock independent of the used key where long IV consisting of 64 bits. Although this proposed method in key hopping consuming much time and memory through encrypting/decrypting each plaintext packet three times before chosen the right key, this process provides a higher degree of randomness increasing the entropy and improving the security where it's difficult to expect which user key will be used for each plaintext packet especially each user key related with four states for choosing.

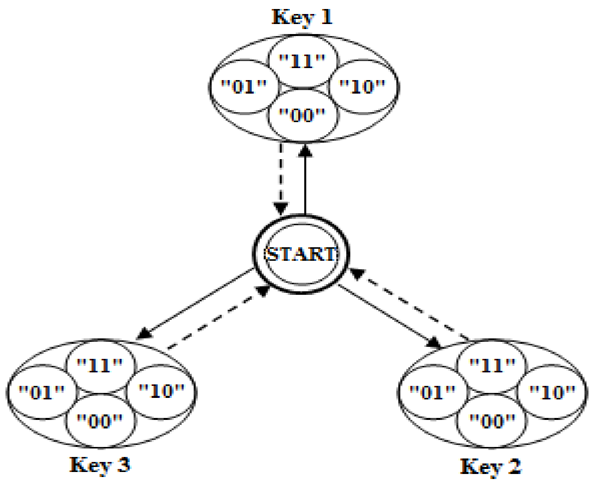


Figure 1: Maximum distance based key hopping scheme

4. The Algorithm

In this section, we provide the formal description of the maximum distance based key hopping algorithm as follows:

```
Algorithm: Maximum Distance Based Key Hopping (MDBKH)

INPUT: Plaintext message (P), User Authentication Key (K),
       User Key (K1), User Key (K2), User Key (K3),
       Initialization Vector (IV).
OUTPUT: Cipher Text (C)

Algorithm body:
Begin
Begin maximum distance based key hopping
1. Read user keys;
2. Authentication key with IV is used for authentication messages.
3. Pre-encrypt the plaintext packet by calling the encrypt function using:
    3.1 the user key K1 concatenating initial vector IV;
    3.2 the user key K2 concatenating initial vector IV;
    3.3 the user key K3 concatenating initial vector IV;
4. Compute all distances between plaintext packet and encrypted packets which are produced in step 3;
5. Specify the key which relate with the maximum value of distance to encrypt/decrypt the packet between the communicating entities;
6. Encrypt the packet calling the encrypt function using the chosen key in step 5;
7. Increment the initial vector IV by one IV = IV + 1;
8. Repeat steps 3 till 7 if message cache is not empty;
9. When the plain text messages are finished then halt;
End maximum distance based key hopping;
End Algorithm;

Function ENCRYPT
Begin
1. Read next message bit;
2. Read next key bit from user key;
3. Use any standard encryption algorithm to encrypt the plaintext; (In the very simple version of this protocol, one can use XOR operation, however generally this is not recommended)
```

```
4. Perform the encryption;
5. Store the resulting cipher;
End;
```

5. FPGA Implementation

The concept of the maximum distance based key hopping applied to choose cipher key leads to a relatively easy to design FPGA-based implementation. We have implemented the proposed technique applying VHDL hardware description language [5], [6], [7] and utilizing Altera design environment Quartus II 9.1 Service Pack 2 Web Edition [8]. The circuit is based on the idea of encrypting 256-bit plaintext blocks using 256-bit user keys that produce 256-bit ciphertext blocks after authenticated connection. Three 192-bit keys are used and a 64-bit initialization vector IV which is incremented by one at each falling-edge of the clock trigger. Three user keys will be interchanged depending on the maximum distance between the plaintext packet and ciphertext packet. The design was implemented using an EP2C70F896C6, Cyclone II family device. The schematic diagram for a demonstrative 256-bit maximum distance based key hopping module is shown in Figure 2. A series of screen-captures of the different design environment output are shown in Figures 3 to 9. Figures 3, 4, 5, 6, and 7 provide the indication of a successful compilation and parts of RTL for maximum distance based key hopping respectively. Figure 8 demonstrates the floor plan. Figure 9 displays the maximum distance based key hopping simulation showing the output encrypted bits.

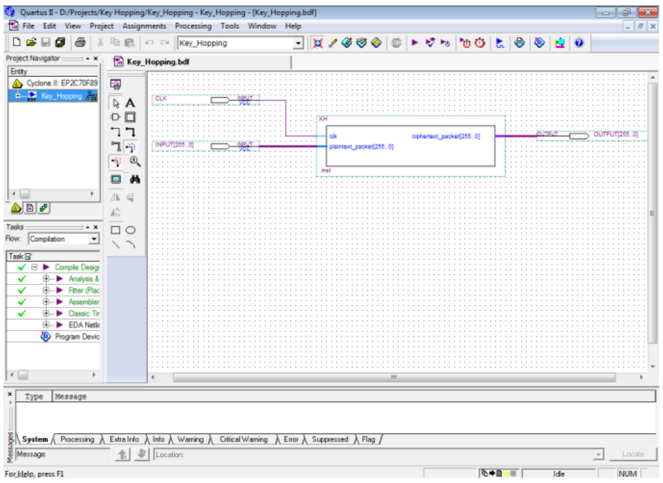
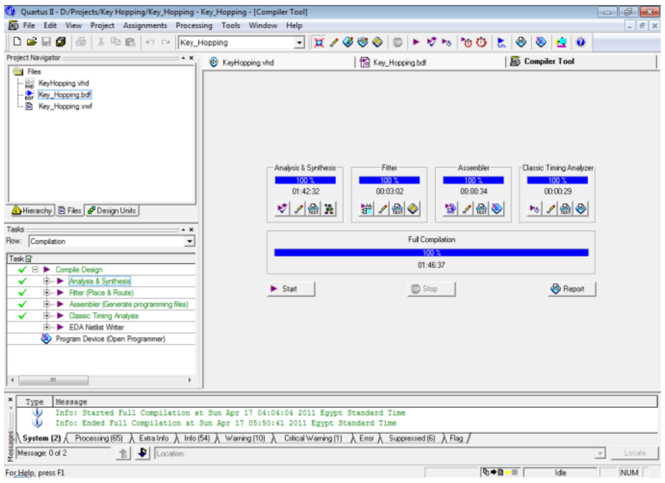
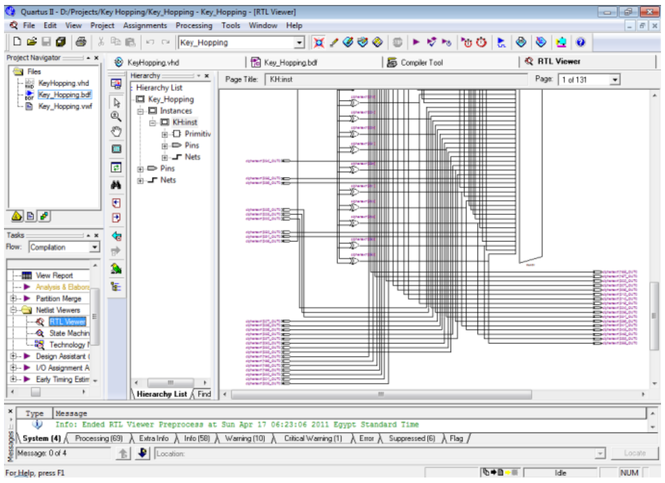

Quartus II - D:/Projects/Key Hopping/Key_Hopping - Key_Hopping - [Key_Hopping.bdf]
Key_Hopping.bdf
Cyclone II: EP2C70F89
Key_Hopping
CLK
INPUT[255..0]
KH
clk
plaintext_packet[255..0]
ciphertext_packet[255..0]
OUTPUT[255..0]

Figure 2: Schematic diagram of maximum distance based key hopping



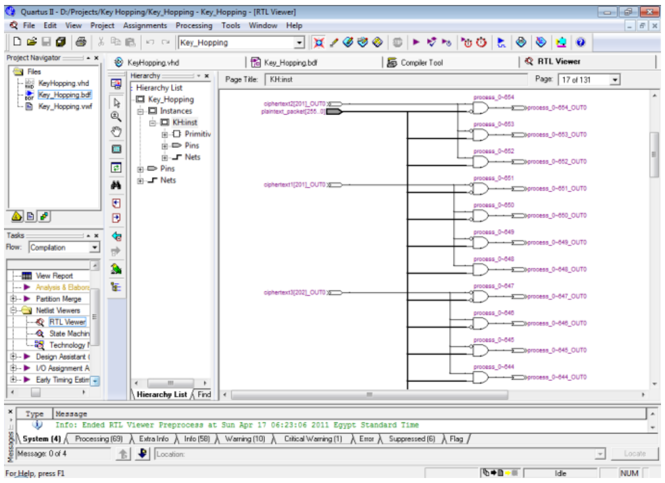
```
Info: Started Full Compilation at Sun Apr 17 04:04:04 2011 Egypt Standard Time
Info: Ended Full Compilation at Sun Apr 17 05:50:41 2011 Egypt Standard Time
```

Figure 3: Compiler tool screen showing correct implementation


Quartus II - D:/Projects/Key Hopping/Key_Hopping - Key_Hopping - [RTL Viewer]
RTL Viewer
Page Title: KH:inst
Page: 1 of 131

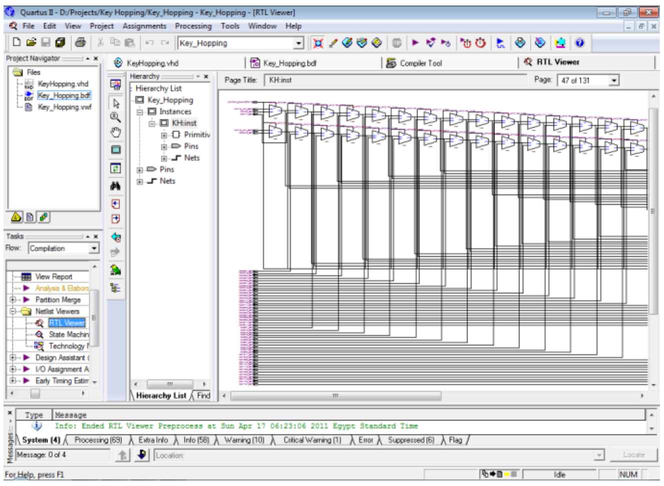
```
Info: Ended RTL Viewer Preprocess at Sun Apr 17 06:23:06 2011 Egypt Standard Time
```

Figure 4: RTL screen for part of maximum distance based key hopping implementation


Quartus II - D:/Projects/Key Hopping/Key_Hopping - Key_Hopping - [RTL Viewer]
RTL Viewer
Page Title: KH:inst
Page: 17 of 131

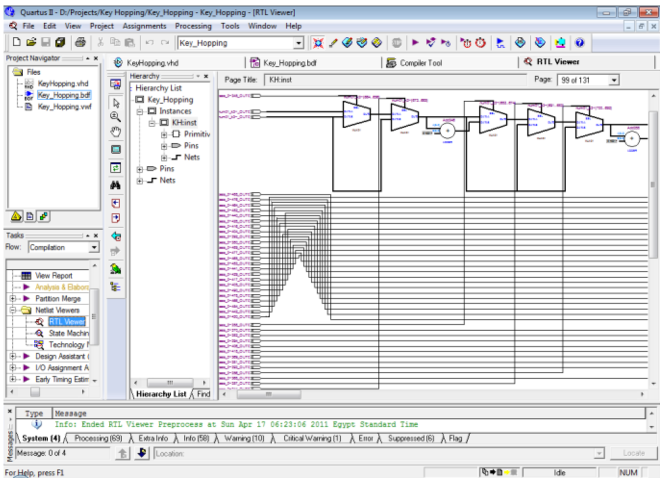
```
Info: Ended RTL Viewer Preprocess at Sun Apr 17 06:23:06 2011 Egypt Standard Time
```

Figure 5: RTL screen for part of the resulting circuit diagram



```
Info: Ended RTL Viewer Preprocess at Sun Apr 17 06:23:06 2011 Egypt Standard Time
```

Figure 6: RTL screen for part of the resulting circuit diagram



```
Info: Ended RTL Viewer Preprocess at Sun Apr 17 06:23:06 2011 Egypt Standard Time
```

Figure 7: RTL screen for part of maximum distance based key hopping implementation

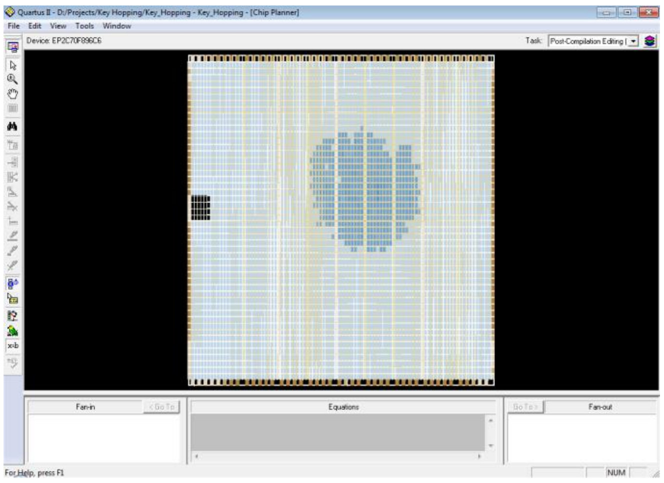
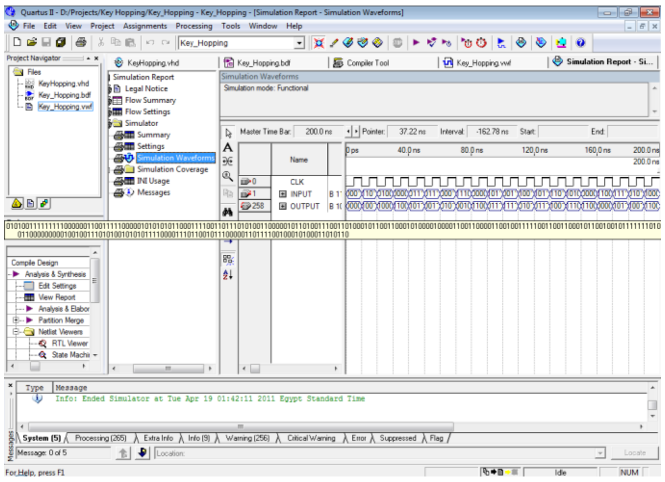


Figure 8: Floor-plan of maximum distance based key hopping implementation



```
Info: Ended Simulator at Tue Apr 19 01:42:11 2011 Egypt Standard Time
```

Figure 9: Simulator screen showing the output encrypted data depending on maximum distance based key hopping

The delays were extracted from the timing reports of implementing balanced, area and speed optimization which is used for synthezing the on maximum distance based key hopping:

- In Balanced optimization technique, clock "CLK" had internal f_{max} of 5.93 MHz (period= 168.748 ns) between source register "KH:inst|IV[62]" and destination register "KH:inst|ciphertext[242]". Longest register to register delay was 168.479 ns. Shortest clock path from clock "CLK" to destination register was 2.832 ns. Longest clock path from clock "CLK" to source register was 2.887 ns. Worst-case of setup time t_{su} was 172.335 ns for register "KH:inst|ciphertext[242]" from data pin "INPUT[254]". Longest pin to register delay was 175.203 ns. Worst-case of clock-to-output time t_{co} was 11.882 ns from clock "CLK" to destination pin "OUTPUT[228]" through register "KH:inst|ciphertext[228]". Longest clock path from clock "CLK" to source register was 2.909 ns. Longest register to pin delay was 8.723 ns. Worst-case of hold time t_h was 0.224 ns for register "KH:inst|ciphertext[80]" from data pin "INPUT[80]". Longest clock path from clock "CLK" to destination register was 2.890 ns. Shortest pin to register delay was 2.932 ns.
- In Area optimization technique, clock "CLK" had internal f_{max} of 5.87 MHz (period= 170.292 ns) between source register "KH:inst|IV[63]" and destination register "KH:inst|ciphertext[236]". Longest register to register delay was 170.031 ns. Shortest clock path from clock "CLK" to destination register was 2.821 ns. Longest clock path from clock "CLK" to source register was 2.868 ns. Worst-case of setup time t_{su} was 173.930 ns for register "KH:inst|ciphertext[236]" from data pin "INPUT[254]". Longest pin to register delay was 176.787 ns. Worst-case of clock-to-output time t_{co} was 11.637 ns from clock "CLK" to destination pin "OUTPUT[194]" through register "KH:inst|ciphertext[194]". Longest clock path from clock "CLK" to source register was 2.900 ns. Longest register to pin delay was 8.487 ns. Worst-case of hold time t_h was -0.226 ns for register "KH:inst|ciphertext[81]" from data pin "INPUT[81]". Longest clock path from clock "CLK" to destination

register was 2.874 ns. Shortest pin to register delay was 3.366 ns.

- In Speed optimization technique, clock "CLK" had internal f_{max} of 6.11 MHz (period= 163.699 ns) between source register "KH:inst|IV[60]" and destination register "KH:inst|ciphertext[219]". Longest register to register delay was 163.490 ns. Shortest clock path from clock "CLK" to destination register was 2.895 ns. Longest clock path from clock "CLK" to source register was 2.890 ns. Worst-case of setup time t_{su} was 166.802 ns for register "KH:inst|ciphertext[219]" from data pin "INPUT[252]". Longest pin to register delay was 169.733 ns. Worst-case of clock-to-output time t_{co} was 11.957 ns from clock "CLK" to destination pin "OUTPUT[81]" through register "KH:inst|ciphertext[81]". Longest clock path from clock "CLK" to source register was 2.862 ns. Longest register to pin delay was 8.845 ns. Worst-case of hold time t_h was 0.251 ns for register "KH:inst|ciphertext[100]" from data pin "INPUT[100]". Longest clock path from clock "CLK" to destination register was 2.868 ns. Shortest pin to register delay was 2.883 ns.

Table 1 shows a comparison between various implementation delays. Figure 10 and 11 show a comparison chart of delays in our design.

Table 1: Delays comparison between optimization technique implementations of maximum distance based key hopping

	Balance	**Area**	**Speed**
Longest register to register delay	168.479	170.031	163.490
Shortest path from clock to destination register	2.832	2.821	2.895
Longest path from clock to source register	2.887	2.868	2.890
Worst-case t_{su}	172.335	173.930	166.802
Longest pin to register delay	175.203	176.787	169.733
Worst-case t_{co}	11.882	11.637	11.957
Longest path from clock to source register	2.909	2.900	2.862
Longest register to pin delay	8.723	8.487	8.845
Worst-case t_h	0.224	-0.226	0.251
Longest path from clock to destination register	2.890	2.874	2.868
Shortest pin to register delay	2.932	3.366	2.883

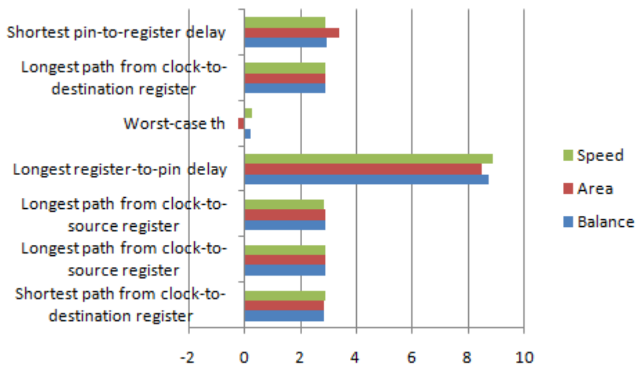


Figure 10: Delays in our design of maximum distance based key hopping implementation

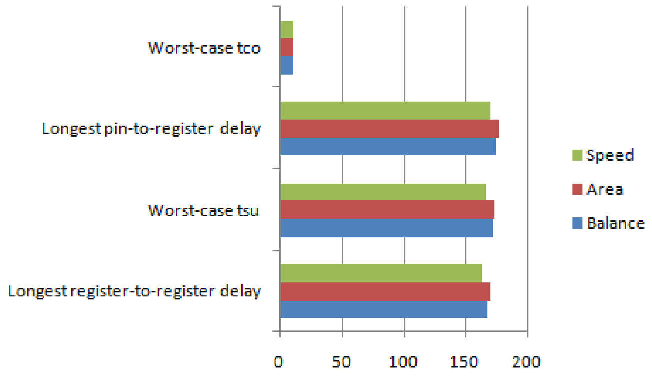


Figure 11: Delays in our design of maximum distance based key hopping implementation

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 Area, Speed, and Balanced optimization technique and Table 3 shows the number of interconnections between logic elements in Area, Speed, and Balanced optimization technique.

Table 2: A synthesis comparison between optimization technique implementations of maximum distance based key hopping

	Balance	**Area**	**Speed**
Total Logic Elements	6321	6316	6391
4 input functions	1888	1984	1991
3 input functions	2091	1946	2017
<=2input functions	2301	2345	2342
Register only	41	41	41
Max Fan-Out	320	320	320
Total Fan-Out	18807	18846	19120
Average Fan-Out	**2.63**	**2.64**	**2.65**

Table 3: A fitter comparison between optimization technique implementations of maximum distance based key hopping

		Balance	**Area**	**Speed**
Interconnects	Block	7832	7809	7925
	C16	602	558	520
	C4	5874	5446	5387
	Local	2210	2252	2214

	R24	865	775	823
	R4	7042	6330	6348
Direct Links		1107	1128	1262
Global Clocks		1	1	1

6. Summary & Conclusion

We have furnished a brief discussion of the hardware implementation of the maximum distance based key hopping by applying VHDL using the schematic editor. The resulting circuit provides a proof-of-concept FPGA implementation. Moreover, area and speed optimization were performed and it is shown that area optimization technique decreasing of usage number of logic elements but speed optimization technique provides average fan-out higher than balanced and area optimization techniques.

Appendix A: The analysis & synthesis and Fitter report details:

Analysis & Synthesis Summary

```
Family: Cyclone II
Device: EP2C70F896C6
Total logic elements: 6,321 out of 68,416 (9 %)
                    -- Combinational with no register: 6,001
                    -- Register only: 41
                    -- Combinational with a register: 279
Total combinational functions: 6,280
    Logic element usage by number of LUT inputs
                -- 4 input functions: 1,888
                -- 3 input functions: 2,091
                -- <=2 input functions: 2,301
                      -- Register only: 41
Total pins: 513 out of 622 (82 %)
Dedicated logic registers: 320 out of 68,416 (< 1 %)
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: 320
Total fan-out: 18807
Average fan-out: 2.63
```

Fitter Summary

```
Block interconnects: 7,832 out of 197,592 (4 %)
C16 interconnects: 602 out of 6,270 (10 %)
C4 interconnects: 5,874 out of 123,120 (5 %)
Direct links: 1,107 out of 197,592 (< 1 %)
Global clocks: 1 out of 16 (6 %)
Local interconnects: 2,210 out of 68,416 (3 %)
R24 interconnects: 865 out of 5,926 (15 %)
R4 interconnects: 7,042 out of 167,484 (4 %)
Nominal Core Voltage: 1.20 V
Low Junction Temperature: 0 °C
High Junction Temperature: 85 °C.
```

Appendix B: Sample VHDL Code:

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

ENTITY MD_KH IS
  port (clk: in std_logic;
     plaintext_packet  : in  std_logic_vector (15 downto 0);
     ciphertext_packet : out std_logic_vector (15 downto 0));
END MD_KH;

ARCHITECTURE behavioral OF MD_KH IS
 signal IV : std_logic_vector(3 downto 0):= "0100" ;
 signal key_1 : std_logic_vector(11 downto 0) :=
                                        "110110100110";
 signal key_2 : std_logic_vector(11 downto 0) :=
                                        "010110010010";
 signal key_3 : std_logic_vector(11 downto 0) :=
                                        "110110010110";
 signal plaintext, ciphertext : std_logic_vector(15 downto 0);
 signal ciphertext1, ciphertext2, ciphertext3 :
                           std_logic_vector (15 downto 0);
 signal key1, key2, key3 : std_logic_vector (15 downto 0);

BEGIN
  key1 <= IV & key_1;
  key2 <= IV & key_2;
  key3 <= IV & key_3;

process (clk, plaintext_packet, key1, key2, key3)
  variable num11_k1, num10_k1, num01_k1, num00_k1:
                                   integer range 0 to 16;
  variable num11_k2, num10_k2, num01_k2, num00_k2:
                                   integer range 0 to 16;
  variable num11_k3, num10_k3, num01_k3, num00_k3:
                                   integer range 0 to 16;
  variable X : integer range 0 to 16;
begin
  plaintext <= plaintext_packet;
  ciphertext1 <= plaintext xor key1;
  ciphertext2 <= plaintext xor key2;
  ciphertext3 <= plaintext xor key3;

if falling_edge (clk) then
    num11_k1:=0;  num10_k1:=0;
    num01_k1:=0;  num00_k1:=0;
    num11_k2:=0;  num10_k2:=0;
    num01_k2:=0;  num00_k2:=0;
    num11_k3:=0;  num10_k3:=0;
    num01_k3:=0;  num00_k3:=0;

for i in 15 downto 0 loop
    if    plaintext(i)='1' and ciphertext1(i)='1' then
                              num11_k1:= num11_k1 + 1;
    elsif plaintext(i)='1' and ciphertext1(i)='0' then
                              num10_k1:= num10_k1 + 1;
    elsif plaintext(i)='0' and ciphertext1(i)='1' then
                              num01_k1:= num01_k1 + 1;
    elsif plaintext(i)='0' and ciphertext1(i)='0' then
                              num00_k1:= num00_k1 + 1;
    end if;

     if    plaintext(i)='1' and ciphertext2(i)='1' then
                              num11_k2 := num11_k2 + 1;
    elsif plaintext(i)='1' and ciphertext2(i)='0' then
                              num10_k2:= num10_k2 + 1;
    elsif plaintext(i)='0' and ciphertext2(i)='1' then
```

```
                                        num01_k2:= num01_k2 + 1;
   elsif plaintext(i)='0' and ciphertext2(i)='0' then
                                        num00_k2:= num00_k2 + 1;
   end if;

    if    plaintext(i)='1' and ciphertext3(i)='1' then
                                        num11_k3:= num11_k3 + 1;
    elsif plaintext(i)='1' and ciphertext3(i)='0' then
                                        num10_k3:= num10_k3 + 1;
    elsif plaintext(i)='0' and ciphertext3(i)='1' then
                                        num01_k3:= num01_k3 + 1;
    elsif plaintext(i)='0' and ciphertext3(i)='0' then
                                        num00_k3:= num00_k3 + 1;
    end if;
  end loop;

  X:= num11_k1;
  if  num10_k1 > X then X:= num10_k1; end if;
  if  num01_k1 > X then X:= num01_k1; end if;
  if  num00_k1 > X then X:= num00_k1; end if;
  if  num11_k2 > X then X:= num11_k2; end if;
  if  num10_k2 > X then X:= num10_k2; end if;
  if  num01_k2 > X then X:= num01_k2; end if;
  if  num00_k2 > X then X:= num00_k2; end if;
  if  num11_k3 > X then X:= num11_k3; end if;
  if  num10_k3 > X then X:= num10_k3; end if;
  if  num01_k3 > X then X:= num01_k3; end if;
  if  num00_k3 > X then X:= num00_k3; end if;

 if    X=num11_k1 or X=num10_k1 or X=num01_k1 or
       X=num00_k1             then ciphertext <= ciphertext1;
 elsif X=num11_k2 or X=num10_k2 or X=num01_k2 or
       X=num00_k2             then ciphertext <= ciphertext2;

 elsif X=num11_k3 or X=num10_k3 or X=num01_k3 or
        X=num00_k3            then ciphertext <= ciphertext3;
 end if;
          IV <= IV + 1;
end if;
end process;
  ciphertext_packet <= ciphertext;

END behavioral;
```

References

[1] W. Ying, “Key Hopping – A Security Enhancement Scheme for IEEE 802.11 WEP Standards,” NextComm Inc, USA, 2002.

[2] K. Srinivasan, and S. Mitchell, “State Based Key Hop (SBKH) Protocol,” Sixteenth International Conference on Wireless Communications Wireless 2004, Alberta, Canada, 2004.

[3] A. M. Kholaif, M. B. Fayek, H. S. Eissa, and H. A. Baraka, “DRKH: Dynamic Re-Keying with Key Hopping,” McMaster University, Hamilton, Canada, 2005.

[4] R. 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.

[5] P. P. Chu, “RTL Hardware Design Using VHDL,” John Wiley & Sons, Inc., New Jersey, 2006.

[6] E. O. Hwang, “Digital Logic and Microprocessor Design with VHDL,” La Sierra University, Riverside, California, USA, 2005.

[7] V. A. Pedroni, “Circuit Design with VHDL,” MIT Press, 2004.

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

Rabie A. Mahmoud received the BS. 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