IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 9, Issue 1, Ver. III (Jan. 2014), PP 22-27
www.iosrjournals.org

Securing M2m Post-Quantum Secret Key Cryptography

Arshiya I [1], Sekar R. [2]
1- M.E, Communication System student, Department of Electronics and Communication Engineering
2- Assistant Professor, Department of Electronics and Communication Engineering Adhiyamaan College, Hosur.

Abstract: *In this paper, an FPGA implementation of a RC6 Block cipher symmetric-key cryptosystems (SKCs) is used. It represents a first step towards securing machine-to-machine (M2M) systems using strong, hardware-assisted SKC. PKC algorithms require a very large amount of CPU cycles to encrypt one block of data. In SKC sender and receiver only have to specify the shared key in the beginning and later they can begin to encrypt and decrypt messages between them using that key. Ex: AES (Advanced Encryption Standard) and Triple DES (Data Encryption Standard). SKC is simple, Fast, Use less computer resources and Prevents widespread message security compromise than PKC. Hence Symmetric key Cryptosystem is used for fast operation. The main goal of this paper is to implement RC6 cipher whose key scheduling is modified by using key selection so that efficiency of the algorithm is improved without modifying the security and to implement the key schedule in a way that it will be able to generate the subkeys faster by pipelining the initial S generate values in the S array. Implementing the modified algorithm in existing real time application will prove the result of the modification.*
Index terms : *RC6 algorithm, cryptography*

I. Introduction

In today mechanism, RC6 Algorithm is known to be the one important and best algorithm for data security. Many works are developed for data security in internet applications and other applications.
RC6 is one of the best AES algorithm and simple cipher usedfor data security. It involves numerous evaluation andadequate security.

II. Related Works

As networked machines become more popular around our living, information security on these devices becomes an important issue. Traditionally, PKC is regarded as too expensive to deploy in M2Msystems. Typical M2M systems only have limited computational power, making deploying strong cryptography on them extremely challenging.

There have been numerous proposals how to secure M2M systems from the academic research community. Most of them use software-based symmetric-key cryptography. For example, TinySec provides link-layer security for sensor networks using software implementation of symmetric-key cryptosystems. In many proposals, more bits will need to be sent over the air for achieving certain level of security, so using hardware accelerators may not necessarily help in these cases. The same functionality would be achieved by PKC in a more communication-efficient way. This is becoming more attractive as computation is getting cheaper in terms of hardware cost and energy consumption, while wireless communication is less so at the same time. As a result, communication is becoming more expensive compared with computation, not to mention the spectrum will become one of the scarcest resources when billions of M2M sensors are deployed and trying to send out their readings over the air. In this case, it is advantageous to use PKC on sensors for the sake of reducing communication cost.
Lastly, there have been several attempts in employing software-based PKC to secure inter-sensor communication. People have demonstrated that it is possible to run PKC on sensors with acceptable performance. We believe that this is the right direction to pursue, and we plan to take it further by hardware acceleration.

III. Rc6 Block Cipher

RC6 is a symmetric key block cipher derived from RC5.

RC6 is more exactly specified as RC6-*w*/*r*/*b*, where the parameters *w*, *r*, and *b* respectively express the word size (in bits), the number of rounds, and the size of the encryption key (in bytes). Since the AES submission is targeted at $w = 32$ and $r = 20$, we implemented this version of RC6 algorithm, using a 32 bits word size, 20 rounds and 16 bytes (128 bits) encryption key lengths. A key schedule generates $2r + 4$ words (*w* bits each) from the *b*-bytes key provided by the user. These values (called round keys) are stored in an array $S[0, 2r+3]$ and are used in both encryption and decryption. RC6 works on a block size of 128 bits and it is very similar to RC5 in structure, using data-dependent rotations, modular addition and XOR operations; in fact, RC6 could be viewed as interweaving two parallel RC5 encryption processes. However, RC6 does use an extra

multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word, and not just the least significant few bits. The computation of $f(X) = (X(2X + 1)) \bmod 2w$ is the most critical arithmetic operation of this block cipher.

Fig.1 - RC6 Cipher block diagram

BASIC OPERATIONS

RC6-*w/r/b* operates on units of four w-bit words using the following six basic operations. The base-two logarithm of w will be denoted by lg w.

- a + b integer addition modulo 2w
- a - b integer subtraction modulo 2w
- a □b bitwise exclusive-or of w-bit words
- a X b integer multiplication modulo 2w
- a<<<b rotate the w-bit word a to the left by the amount given by the least significant lg w bits of b
- a>>>b rotate the w-bit word a to the right by the amount given by the least significant lg w bits of b

IV. Proposed Work

A. Key schedule revisted

The user supplies a key of b bytes. Sufficient zero bytes are appended to give a key length equal to a non-zero integral number of words; these key bytes are then loaded in little-endian fashion into an array of c w-bit (w = 32 bits in our case) words L [0], ... , L [c - 1]. Thus the first byte of key is stored as the low-order byte of L [0], etc., and L [c - 1] is padded with high-order zero bytes if necessary. The number of w- bit (32 bit) words that will be generated for the additive round keys is 2r + 4 and these are stored in the array S [0; ... ;2r + 3]. The constants P32 = B7E15163 and Q32 = 9E3779B9 (hexadecimal) are the same “magic constants" as used in the RC5 key schedule.The main advantage of the proposed system is key selection here we are not going to transfer the key we transfer only key selection and the values of L [0] …. L [c-1] are changed using operations assigned using key selection. This modification increases security and the burden of transferring 128bits of key for each message transfer.

Procedure :

Case(ks)

{

L[0] = L[g] op1 L[h]

L[1] = L[g] op2 L[h]

L[c-1] = L[g] opc-1 L[h];

}

S [0] = P32

S [1] = S [0] + Q32

B. Encryption

As key scheduling takes most of the time . So Encryption is implemented in a way that it will be able to generate the subkeys faster by pipelining the initial S generate values in the S array. So the subkeys are generated that is used for first few rounds and then generating the rest while encryption starting to use these. This will save a significant amount of time and improve the performance of the design

Input: Plain text stored in four w-bit input registers A, B,C, D.

r denotes the no of rounds and 2r+4 w-bit round keys S[0,1, .., 2r + 3]

Output: Cipher text will be store in A, B, C, D

Procedure:

X = Y = i = j = 0

B = B + S[0]

D = D + S[1]

fori = 1 to r do

```
  {
X = S [2i] = (S [2i-2] + X + Y) <<< 3
Y = L [2i] = (L [2i-2] + X + Y) <<< (X + Y)
X = S [2i+1] = (S [2i-1] + X + Y) <<< 3
Y = L [2i+1] = (L [2i-1] + X + Y) <<< (X + Y)
t = (B x (2B + 1)) <<< log2 w
u = (D x (2D + 1)) <<< log2 w
    A = ((A  t) <<< u) + S[2i]
    C = ((C  u) <<< t) + S[2i+ 1]
    (A,B,C,D) = (B,C,D,A)
  }
  A = A + S[2r + 2]
  C = C + S[2r + 3]
```

Initially S [0] and S [1] are generated and then the rest of the subkeys are generated while encryption is running at the same time. As in encryption the values of B and D don't change inside the rounds two core units of our design can run in parallel. The first core generates new values for A and C and the second will generate values for B and D. This will bring the total number of cycle to half.

C. Decryption

```
Key schedule  Procedure with RC6 w/r/b
Case(ks)
{
L[0] = L[g] op1 L[h]
L[1] = L[g] op2 L[h]
L[c-1] = L[g] opc-1 L[h];
}
S [0] = P32
for i = 1 to 2r + 3 do
S [i] = S [i - 1] + Q32
A = B = i = j = 0
v = 3 X max{c, 2r + 4}
for s = 1 to v do
{
A = S [i] = (S [i] + A + B) <<< 3
B = L [j] = (L [j] + A + B) <<< (A + B)
i = (i + 1) mod (2r + 4)
j = (j + 1) mod c
}
Decryption with RC6 w/r/b
Input: Cipher text stored in four w-bit input registers A, B, C, D.
 r denotes the no of rounds and
2r+4 w-bit round keys S[0, 1,.., 2r + 3]
Output: Plain text will be store in A, B, C, D
Procedure:
C = C - S[2r + 3]
  A = A - S[2r + 2]
  fori = r down to 1 do
  {
    (A,B,C,D) = (D,A,B,C)
    u = (D x (2D + 1)) <<< log2 w
    t = (B x (2B + 1)) <<< log2 w
    C = ((C - S[2i + 1]) >>> t) u
    A = ((A - S[2i]) >>> u) t
  }
  D = D - S[1]
  B = B - S[0]
```

V RC6 MAIN MODULE

Input:

- Key Input: Key to be used by ecnr/decr

- Key Avail: Indicates that the key is available to be read
- Data Input: Message/Cipher text is entered into the cipher
- Data Avail: Indicates data is available to be read for enc/dec
- Clock: Master Clock
- Reset: Master Reset
- Enc/Dec:Enc/Dec0/1 encryption/ decryption selection
- Full: Indicates output full and cannot output data

Output:

- Key Read: Indicates the key has been read
- Data Read: Entered into the cipher
- Data Out: Cipher text/ Plaintext is output through this port
- Data Write: Data becomes available on output bus
- Ready: Indicates that the key has been generated and the unit is ready for enc/dec.

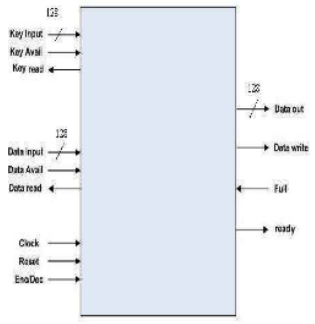

FIG 2 : RC6 Main Module

VI. RC6 CORE MODULE

The RC6 core module is where the function $f(X) = (X\ (2X + 1))$ mod $2w$ is implemented. As we can see the data is first broken down to four words, each 32 bits wide represented by A, B, C and D. Key scheduler prepares two 32 bit words from the S array, one value from the even addresses and one from the odd addresses. In the case of encryption A and C are added with these two values from S. Also u and t are calculated using the function f. u and t are shifted by 5 before they are Xored with output from the barrel shifter.

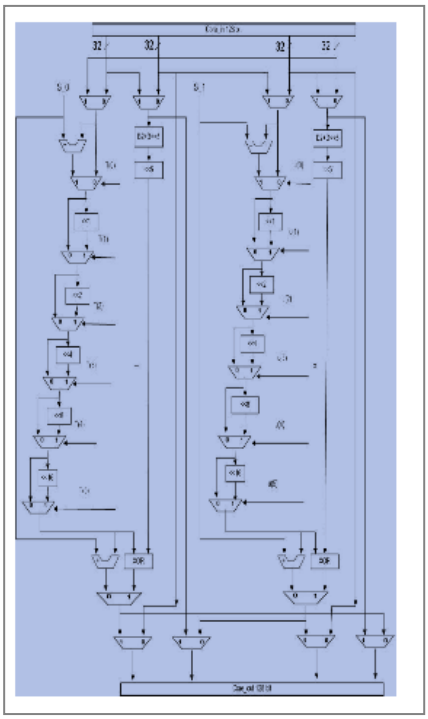

Fig 3 : RC6 Core Module

VII. Results

A Testing

The first step in the design process is to check for the functional correctness of the design using simulations. Then, the FPGA synthesis tool is used to interpret logic components from the Verilog code. The synthesis tool produces a net list, which does not have accurate timing information since placement, and routing of the logic components on the FPGA is not yet determined. The post-synthesis netlist can however be used to check the correct inference of logic components from the Verilog code. The final step in the process is to map the design to the target FPGA. The FPGA implementation tool, which is vendor-specific, generates a net list, which has accurate timing as well as logic information. The final net list is used for simulation to check if the design will actually work when configured physically on the FPGA. This type of simulation is known as timing simulation. Our design did pass timing simulation under the control of test bench. Test benches were written in verilog in order to verify the functionality of the design. The test benches were de- signed to read test vectors from a file and compare the pro- duced output with expected outputs stored in another file. Extensive testing was done for checking both the encryption as well as decryption functionality.

B Waveforms

Key setup: The following waveform shows when signal ready turns to 1 meaning that the key schedule is done and the data is being read. Here we read the input "00000000000000000000000000000000"

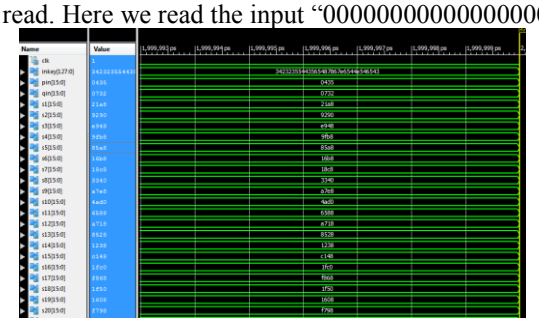

Fig 4 : Key schedule waveform

Encryption: The following waveform shows when the input "00000000000000000000000000000000" is encrypted and the cipher text "A078D51A2816082A155AA150D47AA152" is outputted

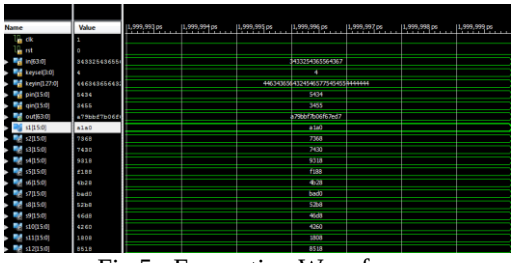

Fig 5: Encryption Waveform

Decryption: The following waveform shows when the cipher "A078D51A2816082A155AA150D47AA152" is decrypted and the original text "00000000000000000000000000000000" is outputted.

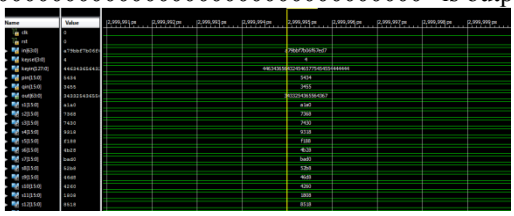

Fig 6: Decryption Waveform

VIII. Conclusion

The aim of this project was to develop a high performance system for data encryption using the RC6 algorithm The proposed algorithm decreases the area to about 1/2 when it implemented to RFID tag including Smart Card and electronic chip.

References

[1] Securing M2M With Post-Quantum Public-Key Cryptographyie-Ren Shih, YongboHu, Ming-Chun Hsiao, Ming-Shing Chen, Wen-Chung Shen, Bo-Yin Yang, An-Yeu Wu, Senior Member, IEEE, and Chen-Mou Cheng IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS, VOL. 3, NO. 1, MARCH 2013

[2] . An improve RC6 algorithm with the same structure of encryption and decryption. Gil-Ho Kim; Jong-Nam Kim;Gyeong-Yeon Cho Advanced Communication Technology,2009. ICACT 2009.11th International Conference on Volume:02 Publication Year : 2010, Page(s):1211-1215

[3] CRIPTOR1.0. VLSI Implementation of the RC6 Block Cipher International Journal of Computer Applications (0975 – 8887) Volume 42– No.16, March 2012

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM J. Comput. vol. 26, no. 5, pp. 1484–1509, Oct. 1997

[5] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," Commun. ACM vol. 47, no. 6, pp. 53–57, Jun. 2004.

[6] Y. Zhou, Y. Fang, and Y. Zhang, "Securing wireless sensor networks: A survey," Commun. Surveys Tuts. vol. 10, no. 3, pp. 6–28, Jul. 2008.

[7] C.Karlof,N.Sastry,andD.Wagner,"TinySec:Alinklayersecurityar-chitecture for wireless sensor networks," in Proc. 2nd Int. Conf. Embed.Netw. Sensor Syst., New York, 2004, pp. 162–175.

[8] R. Watro, D. Kong, S.-F. Cuti, C. Gardiner, C. Lynn, and P. Kruus,"TinyPK: Securing sensor networks with public key technology," in Proc. 2nd ACM Workshop Security ad hoc Sensor Netw., New York, 2004, pp. 59–64.

[9] D. J. Malan, M. Welsh, and M. D. Smith, "Implementing publickey infrastructure for sensor networks," ACM Trans. Sensor Netw. vol. 4, no. 4, pp. 22:1–22:23, Sep. 2008.

[10] FouadRamia, HunarQadir, GMU "RC6 Implementation including key scheduling using FPGA"

[11] Wallace, C. S., "A Suggestion for a Fast Multiplier," IEEE Trans. on Computer, Vol. EC-13, pp.14-17, 1964.

[12] "Active-HDL Getting Started" available at website http://ece.gmu.edu/labs/Active_HDL.pdf (manual)

[13] "XCV-1000 FPGA Board v. 1.0 User Manual," *XessCoporation*, available at website http://www.xess.com/manuals/xcv-1000-manual-v1_0.pdf. 2005.

[14] K. Gaj , Lecture Slides for ECE 545 – Introduction to VHDL, (Fall 2005)