6th International Conference on Internet Technology and Secured Transactions, 11-14 December 2011, Abu Dhabi, United Arab Emirates

FPGA Implementation of High Speed XTS-AES for Data Storage Devices

Mohamed Elmoghany, Mohamed Diab, Moustafa Kassem, Mustafa Khairallah, Omar El Shahat, Wael Sharkasy
Faculty of Engineering
Alexandria University
Alexandria, Egypt.
Email:{m.osama.elmoghany, mohamed.s.diab, moustafa.m.kassem, mustafa.m.khairallah, omar.s.mohammed, wael.sharkasy}@gmail.com

***Abstract*—This paper presents a novel architecture of XTS-AES mode for data storage devices. An enhanced fully pipelined and area efficient XTS-AES mode design using one AES core is proposed. We propose a design of XTS module to handle the data blocks to be encrypted using a single AES core. Considering previous work in XTS, few designs have been published that use a single AES core, and few efforts have been targeted toward their optimization. This paper describes hardware implementation of XTS-AES design with a throughput of 19.56 Gbps and a maximum achievable frequency of 153.84 MHz. This design is written in Verilog HDL and verified on Altera Cyclone II FPGA.**

I. Introduction

Advanced Encryption Standard (AES) is a symmetric-key algorithm adopted by the U.S. government and it is widely used for electronic data encryption [1]. The IEEE Security in Storage Working Group (SISWG) has developed the XTS - XTS stands for XEX encryption mode with tweak and ciphertext stealing - mode of AES that was defined in the IEEE 1619-2007 standard and was approved by the US National Institute of Standards and Technology [2]. This mode matches the needs of storage devices specially hard disk drives while keeping the security that the AES algorithm provides. The XTS-AES algorithm solves security issues such as copy-and paste attack, while allowing parallelization and pipelining in cipher implementations [3].

FPGA is a good choice for implementing cryptographic algorithms. Its capability to process data in parallel with high speed of processing and its ability to be reconfigured many times and relatively small development cycle compared to ASIC makes it a perfect platform to implement cryptographic algorithms.

Previous designs as in [4] used two AES cores which has lower throughput compared to our design. In [5], a design that uses a single AES core is proposed, but it has higher latency and lower throughput compared to ours.

Our design uses a single AES core but we propose some internal modules that reduce the latency and increase throughput per area about three times higher than that of the design in [5], through a fully pipelined design which enables the generation of a new output every clock cycle. This paper describes the theoretical and architectural model that we propose for the XTS-AES system.

II. AES Algorithm

The AES algorithm is a symmetric block cipher algorithm. It processes data blocks of 128-bits using cipher keys of lengths 128, 196 or 256 bits. It is usually abbreviated as AES-128, AES-196 or AES-256 respectively depending on the key used as defined in [6]. Number of rounds (Nr), depends on the key length. It can be 10, 12 or 14 rounds according to the key length of 128, 192 or 256 bits respectively. Our design uses 128-bit key so the process is divided into 10 rounds. Each round consists of four different transformations: SubBytes, Shift Rows, Mix Columns and AddRoundKey. At the beginning of the operations, the plaintext is XORed with the key. Then the output enters the 10 rounds. The last round doesn't include Mix Columns. More details about AES Algorithm can be found in [1].

A. SubByte Transformation

It is a substitution which is non-linear that operates independently on each byte of a state using a substitution table called S-Box. S-Box is defined as the multiplicative inverse of the finite GF (2^8) with the irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{1}$$

followed by an affine transformation as in [1].

B. Shift Row Transformation

The bytes of the last three rows of the state array are cyclically shifted with different offsets (The offset value depends on the row number, where the offset of the first row is 0, the second row is 1, the third row is 2 and the fourth row is 3) as in [1].

C. Mix Columns Transformation

As illustrated in [1], Mix columns transformation is applied to each column of the state array where each column is treated as a polynomial of degree three over Galois Field (GF) (2^8). The polynomial is multiplied modulo $x^4 + 1$ with the fixed polynomial:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{2}$$

D. Add Round Key

It is a simple bitwise XOR operation where the 128-bit of the state array is XORed with a 128-bit round key.

E. Key Expansion

The initial 128-bit cipher key is expanded to eleven round keys of 128-bit long each. The first round key (RoundKey0) is the input key and subsequent round keys are generated by applying a function to the previously generated ones as in Fig.(1) where a pseudo code for the key expansion process is given. RotWord operation is circular left shift for one byte on a 32-bit word. SubWord operation performs a byte substitution on each byte in its 4-byte input word, using the S-box. More details about key expansion algorithm can be found in [7].

The result of the previous steps is XORed with a round constant, Rcon[j]. Since the round keys are 128-bits long, they are referenced as 4 words of 32-bits each as in the pseudo code.

```
RC[1:10] = ('01','02','04','08','10',
           '20','40','80','1B','36');
Rcon[i] = (RC[i], '00', '00', '00');
W [0:3] = (Key [0], Key [1], Key [2],
           Key [3]);
For (i = 4; i < 44; i++)
{
  If ((i%4) == 0)
   {
    Temp = SubWord (RotWord (W [i-1]))
           XOR Rcon [i/4];
    W[i] = W [i-4] XOR temp;
   }
}
```

Fig. 1. Key Expansion Pseudo-code [8]

III. XTS Algorithm

XTS-AES is a tweakable block cipher that is designed for encryption of sector based storage. It acts on 128-bits data blocks or more and uses AES as a subroutine. Its key material consists of a data encryption key used by AES and a tweak key, which represents the logical position of the data into the encryption.

The XTS mode uses an AES algorithm to encrypt the data number representing the first 128-bit tweak. This encrypted tweak value is fed into a multiplier by a primitive element over GF (2^{128}) whose output is used to tweak the plaintext and ciphertext using two XOR algorithms. The multiplier uses two equations for its operation:

$$a_{i+1}[0] \leftarrow (2(a_i[0] mod 128)) \oplus (135 \lfloor a_i[15]/128 \rfloor) \tag{3}$$

$$a_{i+1}[k] \leftarrow (2(a_i[k] mod 128)) \oplus (\lfloor a_i[k-1]/128 \rfloor) k = 1, 2, .. \tag{4}$$

The XTS-AES encryption procedure used for a single 128-bit block is done as in the following equation (5):

$$C \leftarrow XTS\text{–}AES\text{–}blockEnc(Key, P, i, j) \tag{5}$$

Where:
- Key is the 256-bit XTS-AES key
- P is a block of 128-bits (plaintext)
- i is the value of the 128-bit tweak
- j is the sequential number of the 128-bit block inside the data unit
- C is the block of 128-bits of ciphertext resulting from the operation

This is illustrated in Fig. (2). The key is a concatenation of two keys Key1 and Key2, which are equal in size, such that: Key = Key1 | Key2.

The ciphertext shall then be computed by the following sequence of steps:

1) $T \leftarrow AES\text{–}enc(key_2, i) \otimes \alpha^j$
2) $PP \leftarrow P \oplus T$
3) $CC \leftarrow AES\text{–}enc(key_1, PP)$
4) $C \leftarrow CC \oplus T$

AES-enc (K,P) represents the procedure of encrypting plaintext (P) using AES algorithm with key (K), as in [1]. The multiplication and computation of power in first step is executed in GF (2^{128}), given that α is the primitive element.

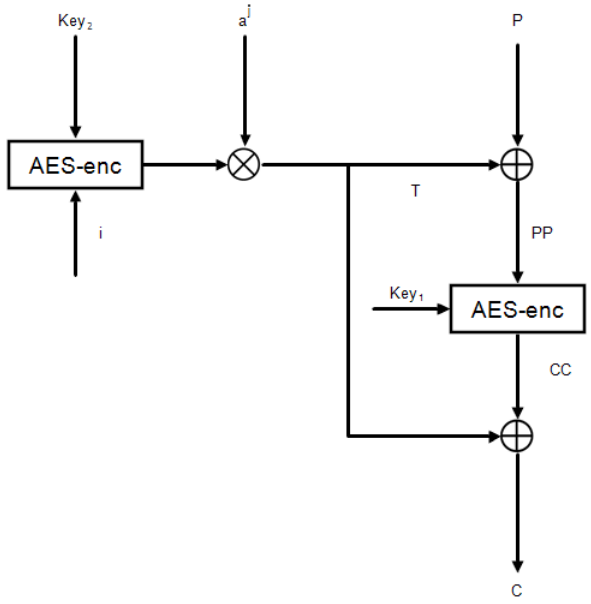

Fig. 2. Diagram of XTS-AES blockEnc procedure

More details about the XTS mode can be found in [3].

IV. XTS-AES Architecture

In this section we present our XTS-AES architecture. We start with explaining AES module and describing XTS-AES module.

A. SubByte

The S-Box is the key point to improve the overall throughput. To implement the S-box, the substitution values have to be pre-calculated and stored in the FPGA Block RAMs like the design in [6], [9]. This is because the values are constant in the original Rijndael specifications. It takes one cycle to finish this substitution.

B. Shift Row

This step involves no logic implementation. The byte shift operation is done by reordering the bytes from the SubByte module.

C. Mix Columns

This operation, in addition to add round key, are performed in one clock cycle. The mix column operation involves three XOR levels and the add round key involves 1 XOR level. In multiplications with '1' no changes happen in the original byte, in multiplication by '2' the byte is lift shifted by '1' bit where the LSB is replaced by '0'. The result is XORed with the irreducible polynomial (in this case ``00011011´´) to generate the result if the MSB is '1' and remains the same if the MSB is '0'. In multiplication by '3' the original byte is simply XORed with the result of multiplication by '2'. More details about mix columns architecture can be found in [10].

D. Key Expansion

In our implementation, round keys are generated on the fly. Each round key takes two clock cycles to be generated: one cycle for S-Box operation and the other cycle for the XOR operations. Therefore, all round keys are generated in 20 clock cycles. The implementation of S-Box operation using combinational logic increases critical path delay and uses significant amount of LUTs. Therefore, we chose to implement S-Box using a single port ROM where the input byte is the address. By using ROM, we are utilizing FPGA dedicated resources efficiently. More details about key expansion architecture can be found in [7].

In Fig. (3), the input key (inkey) is divided into four 32-bit signals inkey[0], inkey[1], inkey[2], inkey[3]. Output key of each round (W) is also divided into four 32-bit signals W[0] ,W[1], W[2] and W[3].

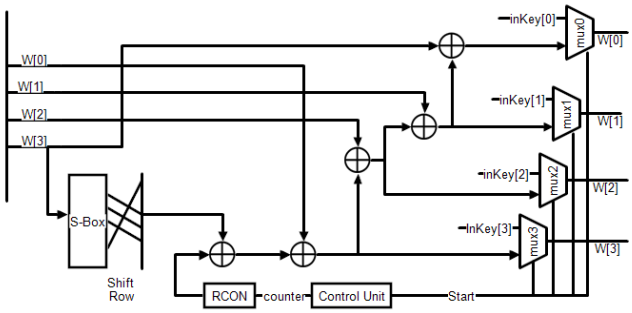

Fig. 3. Key expansion architecture [7]

E. XTS operations

1) GF multipier:

Conceptually, the GF Multiplier is a left shift operation of each byte by one bit with carry propagation from one byte to the other. Also, if the 15th (last) byte shift operation results in a carry, a special value ``10000111´´ is XORed with the first byte. This value is derived from the modulus of the Galois Field (polynomial $x^{128} + x^7 + x^2 + x + 1$).
This step is done in one clock cycle.

More details about multiplication by a primitive element over GF(2^{128}) can be found in [3].

It can be computed using Verilog HDL language.
For the first byte:

$$Byte\_out\_0 <= \{\{Byte\_out\_0[6:0], 1'b0\}\hat{}\ \{8'h87 \,\&\, \{8\{Byte\_out\_15[7]\}\}\};$$

while the rest of bytes involve no logic implementation, they can be processed using a simple concatenation process and can be computed using:

$$Byte\_out\_[i] <= \{Byte\_out\_[i][6:0], Byte\_out\_[i-1][7]\};$$

2) XTS-AES encryption procedure for a single 128-bit block:

a) Previous designs

One of the previous proposed designs for XTS-AES [4] used two AES cores which has lower throughput compared to our design since the encrypted tweak value should be hold till encrypting the plaintext (data). This tweak value is then XORed with the encrypted plaintext to get the ciphertext (encrypted data). Also, the two AES cores consume larger area compared to our design and design in [5].

The latter design [5] as well as our design uses one AES core in order to reduce the area. They depend on using the AES core two times: one for encrypting the tweak value and the other for encrypting the plaintext. These two values are then XORed to get the ciphertext.

b) Our Proposed design using one AES core

A new architecture is proposed in Fig. (4), which is a heavily modified version of the design in [5], and uses one AES core. Our design architecture is fully pipelined and high throughput with latency 22 clock cycles where a new 128-bit output is generated every clock cycle.

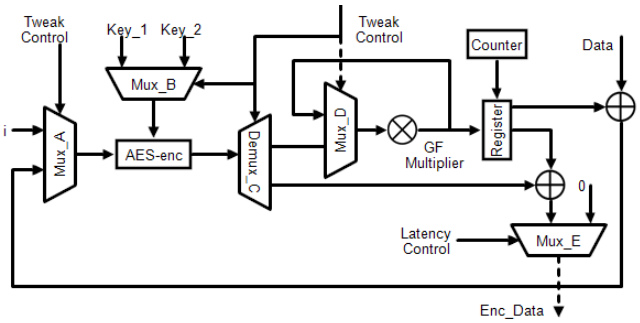

Fig. 4. Diagram of a high throughput fully pipelined XTS-AES single core architecture

As illustrated in Fig. (4) the tweak_control signal is set to logic '1' for the first 22 clock cycles to encrypt tweak value with key2. Demux_C transfers the output of encrypted tweak value to the GF multiplier and saves the first tweak value in the register after 22 clock-cycle latency due to AES pipelining for tweak value. The 128-bit data enters XTS module per clock cycle in the 23rd clock where tweak_control changes to logic

'0' while transferring data. Then the data enters AES core to be encrypted with key1. After another 22 clock cycles of AES pipelining for data, the latency control signal changes to logic '1', the counter begins counting and the ciphertext is ready to be Xored with the saved tweak value in the register every clock cycle.

In comparison with our design, the design in [5] waits for the whole operation of XTS-AES after the input is inserted and then gets the encrypted data before starting the processing on the next input block. Our design, however, is fully pipelined with high throughput of 128-bit per clock cycle. For the GF Multiplier, the design in [5] uses tweak value for every operation while in our design the tweak value is automatically generated by a feedback circuit. This modification allowed the GF Multiplier to generate a valid output every clock cycle.

V. Evaluation and Experimental Results

We used Verilog HDL to develop the hardware model of the XTS-AES core. Behavioral simulation was conducted on Xilinx ISIM while Xilinx ISE 12.2 was used to synthesize the design using Virtex-4 and Virtex-5 FPGA with speed grade of -11 and -3 respectively. The results were verified using IEEE test vectors as in [11] and are shown in Table (I).

TABLE I
Xilinx Results

Work	Family	Frequency MHz	Area Slices	Throughput Gbps	Throughput /Area
[5]	Virtex4	157.0	951	1.8	1.89
[5]	Virtex5	209	808	2.5	3.09
Ours	Virtex4	190.8	6128	24.42	3.98
Ours	Virtex5	297.5	4047	38.077	9.41

As shown in Table (I) where our design is compared with the design in [5], it can be shown that our design has much higher throughput per area.

Our design was then verified using Quartus II and signaltap II logic analyzer on Altera cyclone II FPGA EP2C35F672C6. The results are shown in Table (II).

TABLE II
Altera Results

Family	Logic elements	Frequency (MHz)	Throughput (Gbps)
Cyclone II	12099	153.84	19.56

VI. Conclusion

Few work was conducted until now regarding the implementation of the XTS-AES mode (P1619). In this paper, we introduced a novel architecture of the XTS-AES mode, using a single AES core where the throughput was enhanced using a pipelined XTS module. We also proposed a design for the Galois field multiplier by a primitive element over the field (2^{128}), which only needs the first tweak value, and automatically generates a set of tweak values to be used later.

We implemented an AES algorithm of 10 pipelined rounds where each round operates in 2 clock cycles.

A throughput of 19.56 Gbps is achieved, using 12099 logic elements and a maximum frequency of 153.84 MHz on Altera Cyclone II FPGA.

As for future work, we are considering a low area design with a compression module targeting solid state devices.

VII. Acknowledgment

The authors would like to thank Dr. Hossam El Din Moustafa, Alexandria University, for providing the team with Altera Cyclone II FPGA. Also, we would like to thank Prof. Mohamed Zahran, Dr. Mohamed Abdelsalam, Dr. Ahmed Sultan and Dr. Amr Elsherif for the time they dedicated to help us improve our work.

References

[1] National Institute of Standards and Technology (NIST), Advanced Encryption Standard (AES), FIPS PUB-197, 2001.
[2] L. Hars, "XTS: A Mode of AES for Encrypting Hard Disks", IEEE Security & Privacy, vol.8, issue 3, pp.68-69, 2010.
[3] IEEE Std P1619-2007, The XTS-AES Tweakable Block Cipher, 2008.
[4] E. Hatzidimitriou, A.P. Kakarountas, "Implementation of a P1619 Crypto-Core for Shared Storage Media", in Proc. of IEEE Mediterranean Electrotechnical Conference, 2010.
[5] E. Hatzidimitriou, A.P. Kakarountas, "Exploration and Enhancement of P1619-Based Crypto-Cores for Efficient Performance", IEEE International Conference on Consumer Electronics, 2011.
[6] Yulin Zhang, Xinggang Wang, "Pipelined Implementation of AES Encryption Based on FPGA", IEEE International Conference on Information Theory and Information Security, 2011.
[7] Andreas Brokalakis, Athanasios P. Kakarountas, Costas E. Goutis, "A High-Throughput Area Efficient FPGA Implementation of AES-128 Encryption", IEEE Workshop on Signal Processing Systems, 2005.
[8] William Stallings, "Cryptography and Network Security Principles and Practices", Fourth Edition, 2005.
[9] Alireza Hodjat, Ingrid Verbauwhede, "A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA", Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004.
[10] Swinder Kaur, Renu Vig, "Efficient Implementation of AES Algorithm in FPGA Device", International Conference on Computational Intelligence and Multimedia Applications, 2007.
[11] SISWG, P1619: Standard Architecture for Encrypted Shared Storage-Media, IEEE Project 1619 (P1619), 2007.