VOL. 10, NO. 17, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

AN EFFICIENT FPGA IMPLEMENTATION OF AES ALGORITHM

Sherin C. George and Anoop Suraj A.
ECE Department, Sahrdaya College of Engineering and Technology, India
E-Mail: celincg22@gmail.com

ABSTRACT

The Advanced Encryption Standard can be programmed in software or built with pure hardware. But Field Programmable Gate Arrays (FPGAS) offer a faster and more customizable solution, since the entire algorithm can be executed in a single tick of clock cycle. This research deals with the implementation of AES algorithm in FPGA using Verilog Language. Software is used for simulation and optimization of the synthesizable Verilog code. All the transformations of both Encryption and Decryption are simulated using an iterative design approach for minimizing the hardware consumption. The design uses an iterative looping approach with block and key size of 128 bits and lookup table implementation of S-box. The FCSR used for key generation make the algorithm more secure, and the usage of Vedic multipliers instead of normal multipliers increases the throughput. This gives low complexity architecture and easily achieves low latency.

Keywords: AES, FPGA, RSA, two fish, S-Box, mix column.

INTRODUCTION

With the fast development and extensive application of computer and communication networks, the information security has roused high attention. Information security is not only applied to the political, military and diplomatic fields, but also applied to the common fields of people's daily lives. With the unceasing development of cryptographic techniques, the long-serving DES algorithm with 56-bit key length has been broken due to the defect of short keys. So AES (Advanced Encryption Standard) alternates DES and has now become the new standard. AES algorithm is now supported by a a small number of international standards at present, and AES algorithm is extensively applied in the financial field, in domestic, such as realizing authenticated encryption in ATM, intelligence card and magnetism card.

Advanced Encryption Standard (AES) is the best secure symmetric encryption technique that has got worldwide approval. The AES centered on the Rijndael Algorithm is a well-organized cryptographic technique that includes creation of ciphers for encryption and inverse ciphers for decryption. Greater security and speed of encryption/decryption is warranted by operations like Sub Bytes (S-box)/Inv. Sub Bytes (Inverse S-box), Mix Columns/Inv. Mix Columns and Key Scheduling.

Two simple techniques for encrypting information are: symmetric encryption (also called secret key encryption) and asymmetric encryption (also called public key encryption). Symmetric algorithms are faster, but the main problem associated with this algorithm is key distribution. On the other hand, asymmetric encryption eliminates key security problem, but these algorithms take too much time for encryption and decryption. Certain systems use asymmetric encryption for secure key exchange joined with symmetric algorithms for rapid data encryption. One of highly regarded symmetric algorithms is AES (Advanced Encryption Standard), AES is encryption standard recognized by the U.S. National Institute of Standards and Technology (NIST) in 2001,based on Rijndael algorithm. The algorithm realized in this paper is Rijndael, so-called after its authors Joan Daemen and Vincent Rijmen, two Belgian cryptographers. Rijndael is a reiterated block cipher with a flexible block length and a flexible key length. The block length and the key length can be fixed to 128, 192 or 256 bits. As it was turned out to be a standard, called AES (Advanced Encryption Standard), the block length was fixed to 128 bits, while the key lengths are as mentioned.

EXISTING ENCRYPTION TECHNIQUES

There are a number of asymmetric algorithms in existence nowadays, including RSA, DSA, El Gamal, and ECC. Currently, the most prevalent is RSA, which stands for Rivest, Shamir, and Adelman, the names of its creators. RSA is based on the difficultly of factoring huge composite numbers into prime factors. RSA can be used for secrecy or symmetric key interchange as well as for digital signatures. DSA, which was suggested by NIST in 1991, is the short form for Digital Signature Algorithm. DSA is to a certain extent less flexible, since it can be used for digital signatures but not for secrecy or symmetric key exchange [1]. The El Gamal algorithm, which was developed by Taher El Gamal, is based on the problem of calculating the discrete logarithm in a finite field. EEC is the abbreviation for Elliptic Curve Cryptography, which was independently recommended in 1985 by Neal Koblitz and V. S. Miller. EEC is not really an algorithm, but an alternate algebraic system for realizing algorithms, such as DSA, using strange mathematical objects known as elliptic curves over finite fields. El Gamal and ECC are not presently supported by .NET out of the box; however, the .NET Framework has been designed to be extensible, making it possible for you or other vendors to provide implementations.

Certain asymmetric algorithms, such as RSA and El Gamal, can be used equally for encryption and digital signatures. Other asymmetric algorithms, such as DSA, are suitable only for realizing digital signatures. The

asymmetric algorithms are much slower and less secure than symmetric algorithms for a similar key size. To get desired result, asymmetric algorithms should be used with a larger key size, and, to achieve acceptable performance, they are mainly applied to small data sizes. Therefore, asymmetric algorithms are generally used to encrypt hash values and symmetric session keys, both of which tend to be rather small in size when compared to plaintext data.

Symmetric-key algorithms are of two types: Stream ciphers and Block ciphers. Stream ciphers encrypts one bit of the message at a time, and block ciphers take a large number of bits and encrypt them as a single unit. Blocks of 64 bits have been normally used. The Advanced Encryption Standard (AES) algorithm accepted by NIST in December 2001 uses 128-bit blocks. Some examples of prevalent and highly regarded symmetric algorithms include AES (Rijndael), Blowfish, CAST5, Two fish, Serpent, RC4, 3DES and IDEA

SALIENT FEATURES OF AES ALGORITHM

Overview of AES Algorithm

AES is a block cipher with a block size of 128 bits. AES uses three different key lengths: 128, 192, or 256 bits. Encryption comprises 10 rounds of processing for 128-bit keys, 12 rounds in the case of 192-bit keys, and 14 rounds for 256-bit keys. Excluding last round in each case, all other rounds are same. Each round of processing includes one single-byte based substitution step, a row-wise shift step, a mix column step, and the adding of the round key [2]. The order in which these four steps are performed is not the same for encryption and decryption. To describe the processing steps used in a single round, it is finest to think of a 128-bit block arranged as a 4×4 matrix of bytes, as follows:

$$\begin{pmatrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte7 & byte_{11} & byte_{15} \end{pmatrix} \quad (1)$$

Therefore, in the beginning four bytes of a 128-bit input block conquer the first column in the 4 × 4 matrix of bytes. The subsequent four bytes takes up the second column, and the next 4 bytes occupy the 3rd column and so on. The 4 × 4 matrix of bytes is referred to as the state array. AES also has the conception of a word. A word consists of 4 bytes, i.e. 32 bits. Hence, each row of the state array is a word, as is each column. For each round of processing, we first works on the input state array and creates an output state array. The output state array formed from the last round is arranged as a 128-bit output block. Unlike DES, the decryption algorithm differs substantially from the encryption algorithm. Although, overall, the same steps are used in encryption and decryption, the order in which the steps are carried out is different, as mentioned previously. NIST announced AES as a standard in 2001, is a slight variant of the Rijndael cipher developed by two Belgian cryptographers Joan Daemen and Vincent Rijmen. Whereas AES needs the block size to stay as 128 bits, the real Rijndael cipher works using all block sizes (and all key sizes) that is a multiple of 32 as long as it goes above 128. The state array for the different block sizes still has only 4 rows in the Rijndael cipher. But, the number of columns depends up on the size of the block.

DES was built on the Feistel network, whereas AES was based on a substitution permutation network. Each round of processing in AES contains byte-level substitutions followed by word-level transformations. Speaking generally, DES also involves substitutions and transformations, except that the transformations are based on Feistel notion of separating the input block into two halves, working on each half separately, and then exchanging the two halves [3].

The overall arrangement of AES encryption/decryption is shown in Figure-1.The number of rounds shown in Figure-1 is 10, is for the case when the encryption key is 128 bit long. (The number of rounds is 12 when the key is 192 bits, and 14 when the key is 256.)Before any round-based processing for encryption can begin, the input state array is XORed with the first four words of the key schedule. The same thing happens during decryption — except that now we XOR the cipher text state array with the last four words of the key schedule. For encryption, each round consists of the following four steps:

- Substitute bytes
- Shift rows
- Mix columns
- Add round key.

The last step consists of XORing the output of the previous three steps with four words from the key schedule. For decryption, each round consists of the following four steps:

- Inverse shift rows
- Inverse substitute bytes
- Add round key
- Inverse mix columns

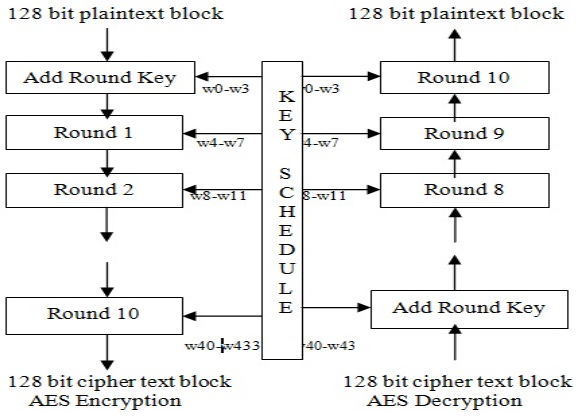

Figure-1. The overall structure of AES encryption for 128 bit encryption key.

The third step involves XORing the output of the previous two steps with four words from the key schedule. Note the changes between the order in which substitution and shifting operations are done in a decryption round and the order in which similar operations are done out in an encryption round. The final round for encryption does not involve the "Mix columns" step. The final round for decryption does not contain the "Inverse mix columns" step.

Four steps in each round of processing

Figure-2 shows the different steps that are carried out in each round except the last one.

STEP 1: (called Sub Bytes for byte-by-byte substitution during the forward process) (The corresponding substitution step used during decryption is called Inverse Sub Bytes.).This step consists of using a 16 × 16 lookup table to find are placement byte for a given byte in the input state array[4]. The entries in the lookup table are created by using the notions of multiplicative inverses in GF(2^8) and bit scrambling to destroy the bit-level correlations inside each byte.

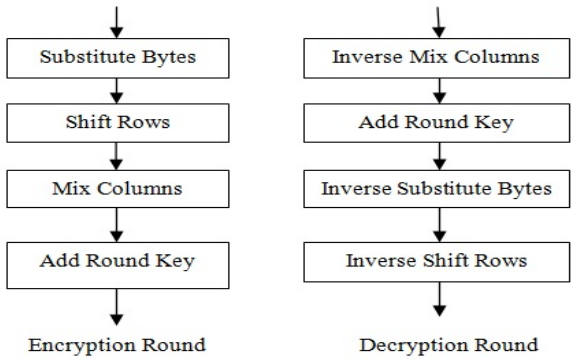

Figure-2. One round of encryption and decryption.

STEP 2: (called Shift Rows for shifting the rows of the state array during the forward process) (The corresponding transformation Add Round Key during decryption is denoted Inverse Shift Rows for Inverse Shift-Row Transformation.)The goal of this transformation is to scramble the byte order inside each 128-bit block.

STEP 3: (called Mix Columns for mixing up of the bytes in each column separately during the forward process) (The corresponding transformation during decryption is denoted Inverse Mix Columns and stands for inverse mix column transformation.) The goal is here is to further scramble up the 128-bit input block. The shift-rows step along with the mix-column step causes each bit of the cipher text to depend on every bit of the plaintext after 10 rounds of processing [5]. In DES, one bit of plaintext affected roughly 31 bits of cipher text. But now we want each bit of the plaintext to affect every bit of the cipher text in a block of 128 bits.

STEP 4: (called Add Round Key for adding the round key to the output of the previous step during the forward process) (The corresponding step during decryption is denoted inverse Add Round Key for inverse add round key transformation.)

PROPOSED WORK

The FCSR used for key generation make the algorithm more secure, and the usage of Vedic multipliers instead of normal multipliers increases the throughput.

Vedic Multiplier

Vedic Mathematics is an early form of Mathematics which existed in most primitive India in 1500 B.C. But was rediscovered by Sri Bharthi KrishnaTirthaji between 1911 and 1918. He divided the whole subject into 16 mathematical Sutras/formulae. These Sutras are simple to understand and quick in terms of computation. Of which, UrdhwaTiryakbhyam Sutra is one of the most accepted sutra used to multiplication. UrdhwaTiryakbhyam technically means "vertically crosswise". Therefore every step of multiplication involves multiplication of the extreme digits. After which the outcome of all the stages are concatenated in order to arrive at the final result of the product.

In a regular Urdhwa sutra multiplier, partial products are got by ANDing the inputs and adding the bi-products. In our approach, we substitute addition with XORing. The product obtained is restricted to 8 bits using usual methods of Galois field restriction. The addition of XOR gates reduces the complexity of the existing Vedic Mathematics Architecture.

Basic Key Generation using FCSR

The idea of FCSRs was put forward by Klapper and Goresky as a substitute to LFSRs for the design of stream ciphers. FCSRs share many of the good properties of LFSRs: sequences with known period and good statistical properties. But unlike LFSRs, they provide an intrinsic resistance to algebraic and correlation attacks because of their quadratic feedback function. However, two recent results have shown weaknesses in stream ciphers using either the Fibonacci or Galois FCSR. Hell and Johansson have exploited the bias in the carries behavior of a Galois FCSR to mount a very powerful attack against the F-FCSR stream cipher. Fisher have considered an equivalent of the F-FCSR stream cipher based upon a Fibonacci FCSR to study the linear behavior of the induced system. We present a new approach for FCSRs which we call the ring representation or ring FCSR. This representation is based on the adjacency matrix of the automaton graph. A ring FCSR can be viewed as a generalization of the Fibonacci and Galois representations. This structure has been widely studied for the LFSR case as shown in and is a building block of the stream cipher Pomaranch when LFSRs are used. However, this paper presents for the first time this structure in the FCSR case.

In a Fibonacci FCSR, we have a single feedback function which depends on multiple inputs. In a Galois FCSR, we have multiple feedback functions with one common input. A ring FCSR can be viewed as a tradeoff between the two previous representations. It has multiple feedback functions with different inputs. An example of a ring FCSR is shown in Figure-1. Ring FCSRs have many advantages over the previous representations. First, they keeps all the good and traditional properties of the FCSRs (known period, large entropy,...). Second, they can also be used to prevent the attack of Hell and Johansson. Third, they have a better diffusion than the Galois or Fibonacci FCSR. Moreover, the ring representation allows the designer to tune the implementation of FCSRs. The above sections gives an overview on FCSRs theory and classical representations. The ring FCSR is presented in the above Section.

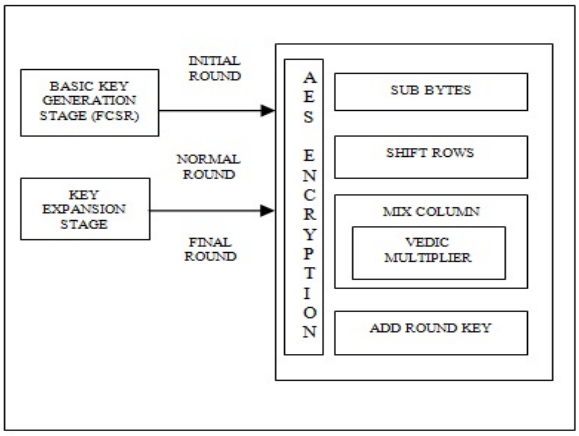

Figure-3. Block diagram of improved AES encryption.

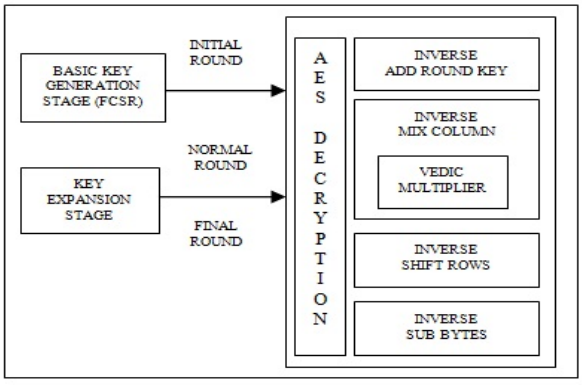

Figure-4. Block diagram of improved AES decryption.

SIMULATION RESULTS

Modelsim SE PLUS 6.3 g software is used for simulation and optimization of the synthesizable Verilog code. Synthesizing and implementation (i.e. Translate, Map and Place and Route) of the code is carried out on Xilinx - Project Navigator, ISE 14.2 Design suite. When we feed 16 bytes (128 bit four 32 bits packets) of data in case of AES encryption into Modelsim simulator, In Figure-5, we can observe that all the 16 bytes of encrypted data for AES, can be observed on the output in terms of 8 bits of chunk. For verification of the working of Decryption we have feed the output of encryption module to the decryption module and observed that the output has regenerated the original text at the output in Figure-6. We have forced 90876543245678909dcfae3456fdea74 as an input to the design and obtained the original data back after the decryption of the encrypted message with the use of same key as seen in Figure-6. We have also monitored that if the single bit of the decryption key is change we are not in a position to retrieve the original data back.

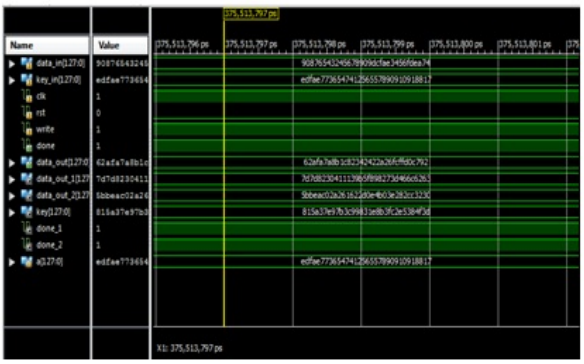

Figure-5. AES encryption.

The AES Encryption simulation result is shown in Figure-5, the encryption parameters are the input plaintext, the key of size 128 bit and the output cipher text. At initial stage, the 16 byte input plain text is mapped in the right order to the 4x4 byte state and then the number of rounds is calculated based on the key Size and then the basic key generated using FCSR was expanded using our key schedule. At round one plaintext and key is XORed and in the remaining nine rounds all four operations are applied: Substitute Bytes, Shift rows, Mix columns, Add round key. In the tenth round, Mix column stage is not included and during each iteration, round key was generated. Here simple XOR of each byte of the key with the respective byte of the state is done to get cipher text of the Encryption algorithm.

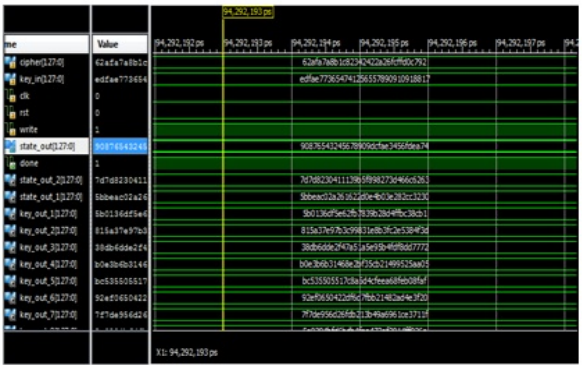

Figure-6. AES decryption.

The AES Decryption simulation result is shown in Figure-6. For AES Decryption, the same encryption process occurs simply in reverse order. The decryption block is as shown in Figure-4, the decryption parameters are the input cipher text, the key of size 128 bit and the output plain text. The output plaintext should be same as encryption input. In decryption the key schedule remains the same. The only operations we need to implement are the Inverse sub Bytes, shift Rows and mix Columns, while add Round Key stays the same.

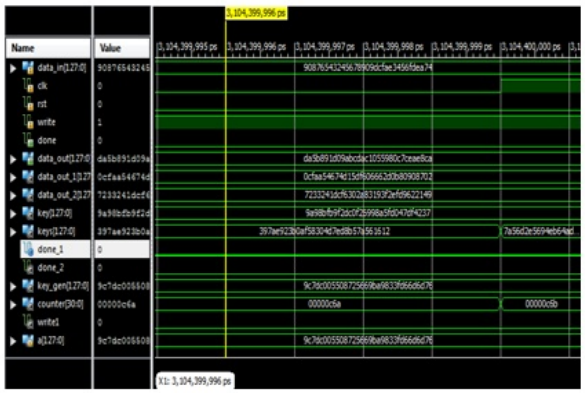

Figure-7. Improved AES encryption.

In the Improved AES Encryption, the basic key for encrypting 128 bit block of input data is generated using a Feedback Carry Shift Register. Thus the chance for breaking the AES algorithm is further reduced and the security is being increased to a greater extent. In the Mix Column stage, Vedic multipliers are used instead of Normal multipliers to improve the speed.

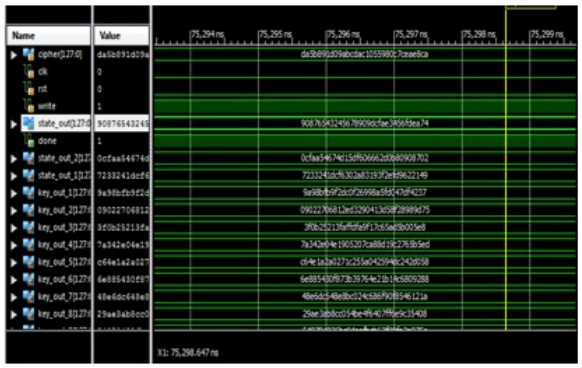

Figure-8. Improved AES decryption.

In the Improved AES Decryption, the basic key for decrypting 128 bit block of cipher text is generated using a Feedback Carry Shift Register. Thus the chance for breaking the AES algorithm is further reduced and the security is being increased to a greater extent. In the Inverse Mix Column stage, Vedic multipliers are used instead of Normal multipliers to improve the speed.

PERFORMANCE ANALYSIS

The comparison with existing architectures is very important for evaluating the efficiency of the proposed design. The comparison is performed on the basis of area requirements, throughput, operating speed, and so forth. The related works show that different architectures are introduced for AES to obtain sufficient area requirements, throughput, and so forth, which are suitable for various applications. This paper introduces an architecture with a FCSR (Feedback Carry Shift Register) for generating the key for encrypting and decrypting data.

In the basic structure of AES implemented here, the required area, in terms of slices and number of registers, is high. Vedic multipliers are used instead of normal multipliers in the basic structure for increasing the speed of operation. It adversely affects the overall area utilized for implementing the idea.

Table-1 shows the comparison of the proposed architecture with the previous work. In the above table 8.1, we have compared the Mix Column stages of previous architecture and new architecture. The total delay is 8.038 ns for previous architecture and it is 4.118 ns for new architecture. From this, we can come to a conclusion that speed is almost doubled. But when we are comparing the number of Slice Registers, LUTs etc used, it is slightly high when compared to previous architecture. It is a fact that we cannot increase speed unless we are ready to compromise area.

Table-1. Comparison between the previous architecture and new architecture.

Parameters	AES Encryption	AES Decryption	Improved AES Encryption	Improved AES Decryption	AES Mix Column	Improved AES Mix Column
No of Slice Registers	188829	14213	19381	18533	0	435
No of LUTs	13769	18428	17959	13690	240	428
No of Fully used LUT-FF pairs	5368	6342	7059	5970	0	302
No of Bonded IOBs	388	388	260	260	258	260
No of BUFG/BUFGCTRLs	1	8	8	8	0	1
Delay	15.559 ns	46.987 ns	36.459 ns	48.649 ns	8.038 ns	4.118 ns

CONCLUSIONS

Due to the increasing needs for secure communications, a more safe and secure cryptographic algorithms has to be proposed and implemented. The Advanced Encryption Standard (AES-128bit) is widely used nowadays in many applications. In this paper, we proposed a new variation of AES and it is compared with the original AES-128 algorithm. A complete hardware implementation for the new AES-128 was also presented

in this paper. After comparing the hardware implementation results, we found that our new design has high throughput when compared with the original AES-128 design. The FCSR used for key generation make the algorithm more secure, and the usage of Vedic multipliers instead of normal multipliers increases the throughput. The extra increase in area can be tolerated and makes the proposed algorithm ideal applications in which high level of security and high throughput are required such as in multimedia communications.

REFERENCES

[1] Hoang Trang and Nguyen Van Loi. 2012. "An efficient FPGA implementation of the Advanced Encryption Standard algorithm", IEEE Paper.

[2] Shylashree N. Nagarjun Bhat and V. Shridhar. 2012. "FPGA Implementation of Advanced Encryption Standard: A Survey", IJAET.

[3] Dr. Prerna Mahajan and Abhishek Sachdeva. 2013. "A Study of Encryption Algorithms AES, DES and RSA for Security", Global Journal of Computer Science and Technology Network, Web & Security.

[4] Hamdan O. Alanazi, B. B. Zaidan, A. A. Zaidan. 2010. "New Comparative Study Between DES, 3DES and AES within Nine Factors", Journal Of Computing, Vol. 2, No. 3.

[5] Amandeep Kaur, Puneet Bhardwaj and Naveen Kumar. 2013. "FPGA Implementation of Efficient Hardware for the Advanced Encryption Standard", IJITEE.

[6] Sumalatha Patil and Mala L. 2013. "Design of High Speed 128 bit AES Algorithm for Data Encryption", International Journal of Current Engineering and Technology, ISSN 2277 – 4106.

[7] Anitha P and Palanisamy V. 2011. "Data Protection Algorithm Using AES", International Journal of Current Research, Vol. 33, No. 6, pp.291-294.

[8] K. Sireesha and S. Madhava Rao. 2013. "A Novel Approach of Area Optimized and pipelined FPGA Implementation of AES Encryption and Decryption", International Journal of Scientific and Research Publications, Vol. 3, No. 9.

[9] Mg Suresh and Nataraj K. R. 2012. "Area Optimized and Pipelined FPGA Implementation of AES Encryption and Decryption", International Journal of Computational Engineering Research, Vol. 2, No. 7.

[10] Subashri T., Arunachalam R., Gokul Vinoth Kumar B. and Vaidehi V. 2010. "Pipelining Architecture of AES Encryption and Key Generation with Search Based Memory", International journal of VLSI design & Communication SystemsVol.1, No.4.

[11] M. Pitchaiah, Philemon Daniel and Praveen. 2012. Implementation of Advanced Encryption Standard Algorithm, International Journal of Scientific & Engineering Research Vol. 3, No. 3.

[12] Ayushi. 2010. A Symmetric Key Cryptographic Algorithm, International Journal of Computer Applications, Vol. 1, No.15.