Contents lists available at ScienceDirect

Journal of King Saud University – Engineering Sciences

journal homepage: www.sciencedirect.com

Original article

An efficient AES implementation using FPGA with enhanced security features

Harshali Zodpe *, Ashok Sapkal

Department of Electronics and Telecommunication, College of Engineering, Pune, India

A R T I C L E I N F O

Article history:
Received 25 April 2018
Accepted 23 July 2018
Available online 24 July 2018

Keywords:
Advanced Encryption Standard (AES)
Field Programmable Gate Array (FPGA)
PN Sequence Generator

A B S T R A C T

Data transferred in an electronic way is vulnerable to attacks. With an aim to protect data for secure communication, a new Hybrid non pipelined Advanced Encryption Standard (AES) algorithm based on traditional AES algorithm with enhanced security features is proposed in this work. Abysmal analysis of the AES algorithm implies that the security of AES lies in the S-box operations. This paper presents a new approach for generating S-box values (modified S-box) and initial key required for encryption/encryption (improved key generation) using PN Sequence Generator. The AES algorithm with proposed modifications shows significant improvement in the encryption quality as compared to traditional AES algorithm. The traditional AES algorithm equipped with proposed novel modified S-box technique and improved key generation technique gives an avalanche effect of 60% making it invulnerable to attacks. The proposed design is synthesized on various Field Programmable Gate Array (FPGA) devices and compared to the existing designs resulting in significant improvement in throughput. The proposed design is implemented on Spartan6 FPGA device.

1. Introduction

With the expansion of data communications and its applications, there is a greater demand for increasing security systems and devices to guard individual information sent over the transmission channel. One of the most important techniques for securing the information is data encryption. Two kinds of cryptographic techniques viz. symmetric and asymmetric cryptosystems have already been created and are widely used (Mohammed et al., 2011). Symmetric cryptography techniques, such as the Data Encryption Standard (DES), Triple DES and Advanced Encryption Standard (AES), utilize a key that is identical to the transmitter as well as a receiver, for encrypting and decrypting the data transferred. The Asymmetric cryptography techniques, such as the Rivest-Shamir-Adleman algorithm (RSA), Elliptic Curve Cryptography (ECC) and Digital Signature Algorithm (DSA) make use of different keys for encrypting and decrypting the data transferred (Ebrahim et al., 2014). For securing large amount of data symmetric cryptography is much more appropriate. The AES algorithm, identified by the National Institute of Standards and Technology (NIST) of the United States of America is approved to displace DES (Fips-197, 2001).

Analysis of cipher strength is an essential part of security assessment of any corporate or academic entity. Cipher analysts have indicated that, out of 10 rounds of AES, about 8 rounds can be brute forced successfully on today's modern day hardware systems. The remaining 2 rounds cannot be broken in sufficient duration so as to make the attack on the system meaningful to the attacker (Bogdanov et al., 2011). Due to the current trend of increase in computational power, it may not be long that the entire AES cipher would be de-ciphered under a given duration, and hence compromise the system under test. Consequently extensive research is being currently carried out to identify techniques to further secure AES algorithm.

In this work, a PN Sequence Generator is used generation of S-box values and the initial key required for Encryption/Decryption. A PN Sequence Generator gives distinct values which satisfies the criterion for S-box values.

These techniques result in enhancing the security of the AES algorithm as the feedback taps and seed value of the PN Sequence Generator are not known to an attacker and will make the

* Corresponding author.
E-mail addresses: harshali.zodpe@mitpune.edu.in (H. Zodpe), ams.extc@coep.ac.in (A. Sapkal).

Peer review under responsibility of King Saud University.

https://doi.org/10.1016/j.jksues.2018.07.002
1018-3639/© 2018 The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University.

algorithm invulnerable to brute force attack. The key contributions of this work are as follows:

- A new approach for generation of S-box values using PN Sequence Generator is presented. A PN sequence generator generates a distinct sequence of random numbers based on the initial seed value and feedback taps. This property of a PN Sequence Generator is used for generating dynamic S-box which enhances the strength of the cryptosystem.
- This work presents a PN Sequence Generator based design of a Key Generation Block for generating initial key internally. A new secure initial key generation technique eliminates the need to apply the key externally and hence strengthens the modified AES algorithm against attack as compared to traditional AES algorithm.
- The techniques proposed in this paper are synchronized at both encryption and decryption ends for a truly secure AES algorithm.
- The AES algorithm with modified S-box values and improved key generation technique is tested on Strict Avalanche Criterion for key sensitivity and an avalanche effect of 60% is achieved.
- Also the proposed design is optimized for speed and area and compared with existing FPGA implementations. The implementation of AES algorithm with modified S-box values using Spartan6 XC6SLX150-3FGG900 FPGA device achieves a throughput of 3.039 Gbps with latency of 10 clock cycles.

Organization of this paper is as follows: Section 2 presents the related work reported by various authors in the literature. Section 3 presents the description of AES algorithm. In Section 4, we propose the techniques and explain in details the need and advantages of generation of S-box values and initial key using PN Sequence Generator for enhancing the security of AES algorithm. Section 5 presents simulation results and comparison of the results with previous works reported in the literature. The conclusion of this work is stated in Section 6.

2. Related work

AES implementations are categorized into software and hardware implementations. Hardware implementation offers faster speed, more security and consumes less power and thus is an attractive choice as compared to software implementation (Karthigai Kumar and Baskaran, 2010). Implementation of algorithms on hardware can be achieved using either Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA) devices (Gaj and Chodowiec, 2009). The authors presented ASIC implementation of the cryptographic algorithm (Chih-Pin et al., 2003; Liu and Luke, 2003). ASIC is an integrated circuit designed for specific application and lacks flexibility. Moreover, increased Non Recurring Engineering cost makes small volume production unaffordable. As compared to ASIC, an FPGA device is reconfigurable, efficient, offers more flexibility and requires less time to market and hence FPGA is a popular choice for hardware implementation (Alexandru and Fratila, 2011; Elbirt et al., 2001). FPGAs also achieve much better performance than multi-core CPUs for mathematical computations.

Significant amount of research has been done on hardware implementation of AES algorithm using FPGA. These hardware implementations aims at achieving increased throughput increased operating frequency, decreased latency, lesser area and lower power dissipation. The existing AES algorithm implementations are based on iterative structure and loop unrolled structures to minimize area and maximize operating frequency or obtain a trade-off between area and operating frequency. In iterative structures used for AES implementation only one cipher round is unrolled and data is iteratively looped until the entire encryption/decryption is completed (Chodowiec and Gaj, 2003; Farooq and Aslam, 2017; Hongge et al., 2012; Iyer et al., 2011). This results in reduced area utilization. In loop unrolled structures, all the cipher rounds are unrolled and data is looped through the cipher rounds sequentially until the entire encryption/decryption is completed (Ali et al., 2011; Standaert et al., 2003; Zhang and Parhi, 2004). The loop unrolled structure implementation, results in increased speed and increased area utilization. The AES algorithm can be implemented using non-pipelined and pipelined or sub-pipelined techniques. The non-pipelined implementations result in optimization of area at the cost of speed (Hussain and Jamal, 2012; Rais and Qasim, 2009a, 2009b). Whereas pipelined or sub-pipelined implementations result in enhanced throughput with increased area utilization as pipelining enables processing of multiple blocks simultaneously (Hammad et al., 2010; Qiang et al., 2015; Qu et al., 2009; Wang and Ha, 2013).

Abysmal analysis of the AES algorithm implies that the S-box substitution is the key component in creating confusion in the encryption process and hence attempts are being made to improve the quality of S-box. AES algorithm with S-box using GF(2^8) Galois Field inversions based on polynomial basis, using composite field arithmetic has been implemented by the authors, where the critical path delays are reduced using multiple stages of pipelining (Liu and Parhi, 2008). The authors presented S-box generation using combinational logic based truth table (Ahmad et al., 2010; Rashidi and Rashidi, 2013). In these, techniques such as Positive Polarity Reed-Muller structure or its variance, Sum of Product, Product of Sum, Twisted Binary Decision Diagram and Binary Decision Diagram are used. These techniques lead to high throughput but very poor area cost ratio. Various memory based S-box implementations use BRAMs, ROMs and LUTs to generate S-box. The memory based S-box generation results in reduced resource utilization but due to unbreakable delay reduces the throughput (Granado-Criado et al., 2010; Zhang and Wang, 2010). Chih Peng and Hwang (2008) proposed a CAM-based SubBytes scheme to achieve increased throughput AES architecture. The authors have presented design of S-box using second-order reversible Cellular Automata and the strength of S-box is evaluated using methods such as Bit Independence Criterion (BIC), non-linearity, entropy and correlation immunity bias (Gangadari and Rafi Ahamed, 2016).

3. Overview of AES algorithm

The AES algorithm performs operations on 128-bit plaintext and uses identical key for encryption as well as decryption. The AES algorithm processes facts obstruct of 128-bit parts and performs 10, 12 and 14 rounds of operations employing a cipher secret of duration 128-bits, 192-bits and 256-bits respectively. The algorithm operates on data block comprised of a 4×4 byte matrix known as the state. The essential procedures of AES algorithm are carried out on the state. The operations of AES Encryption algorithm with 128-bit key size are show in Fig. 1.

The AES-128 algorithm can be divided in three stages viz. adding initial round key, rounds 1–9 and the final round. In the initial round, the 128-bit plaintext is Exclusive-ORed with 128-bit initial key. In each cipher round, SubBytes( ), ShiftRows( ), MixColumns( ) and AddRoundKeys( ) transformations are performed on a two-dimensional 4×4 array of bytes called the states. In the final round, SubBytes( ), ShiftRows( ), and AddRoundKeys( ) operations are performed on the states.

3.1. SubBytes transformation

The SubBytes transformation is the only non-linear and an invertible bytes transformation. It makes AES potent enough

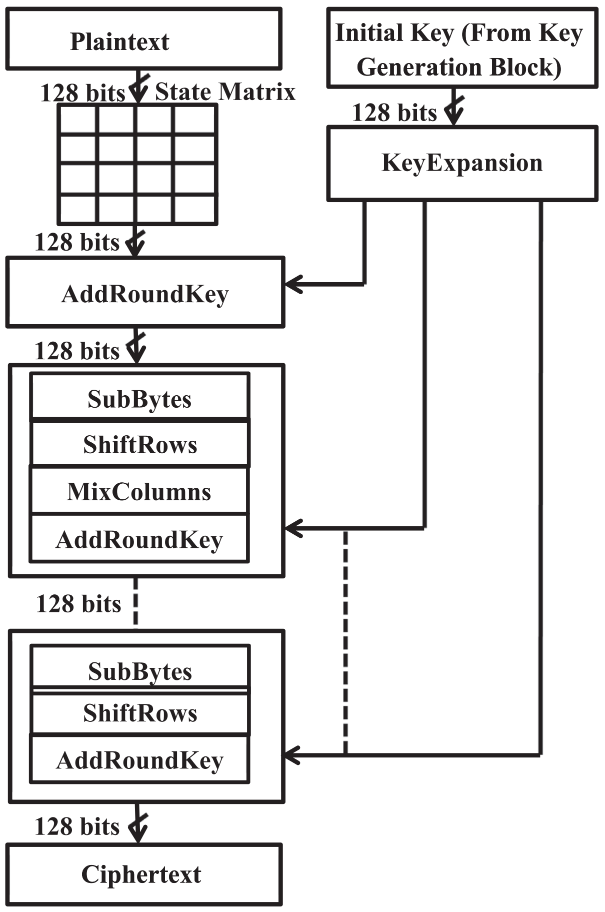

Fig. 1. Block Diagram of AES Algorithm.

against attacks. The SubBytes transformation substitutes each byte from the state matrix with the value stored in S-box. The S-box is formed of a lookup table of size 256 bytes. The S-box values are calculated by taking multiplicative inverse in finite field GF(2^8) where input element with all bits zero is mapped to itself and applying affine transformation over GF(2). The multiplicative inverse in finite field GF(2^8) is given by Eq.(1). The affine transformation over GF(2) is expressed in Eq.(2).

$$S(y) = Affine\,transform(y^{-1}) \tag{1}$$

$$Affine\,transform = \begin{bmatrix} 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1\\ 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1 \end{bmatrix} x \begin{bmatrix} i_7\\ i_6\\ i_5\\ i_4\\ i_3\\ i_2\\ i_1\\ i_0 \end{bmatrix} + \begin{bmatrix} 0\\1\\1\\0\\0\\0\\1\\1 \end{bmatrix} \tag{2}$$

3.2. ShiftRows transformation

The ShiftRows transformation shifts rows 1, 2 and 3 of the State matrix cyclically towards left by 1, 2 and 3 positions respectively. The offset value is dependent on the row number. Thus the first row remains unchanged. Cyclic rotation of rows imparts diffusion property in AES algorithm. The ShiftRows transformation is represented in Fig. 2.

3.3. MixColumns transformation

The MixColumns transformation performs operations on each column of the state matrix one at a time. It is a linear diffusion process. Each column of the state matrix is considered as a four-term polynomial over GF(2^8). The column is then multiplied by modulo ($y^4 + 1$) with a fixed polynomial a(y) given by Eq. (3),

$$a(y) = \{03\}y^3 + \{01\}y^2 + \{01\}y^1 + \{02\} \tag{3}$$

Eq.(3) can also be represented as matrix multiplication as Eq. (4):

$$p\prime(y) = a(y) \times p(y) \tag{4}$$

and in matrix form as Eq. (5):

$$\begin{bmatrix} P'_{0,c}\\ P'_{1,c}\\ P'_{2,c}\\ P'_{3,c} \end{bmatrix} = \begin{bmatrix} 02&03&01&01\\ 01&02&03&01\\ 01&01&02&03\\ 03&01&01&02 \end{bmatrix} \begin{bmatrix} P_{0,c}\\ P_{1,c}\\ P_{2,c}\\ P_{3,c} \end{bmatrix} \tag{5}$$

3.4. AddRoundKey transformation

The AddRoundKey transformation is the final transformation in each round. In this transformation the round key obtained is XORed with the state by bitwise operation. The Nb words from key schedule of each Round Key are XORed with the columns of the State as shown in Eq. (6),

$$\left[P'_{0,c}, P'_{1,c}, P'_{2,c}, P'_{3,c}\right] = [P_{0,c}, P_{1,c}, P_{2,c}, P_{3,c}] \oplus [W_{round*Nb+c}] \tag{6}$$

where $[W_{round}]$ are the words from key schedule, *round* is a value in the range $0 \leqslant round \leqslant N_r$ and N_r is the round number.

3.5. Key expansion module

The key expansion module generates 128-bit keys required for each round of AES algorithm based on initial 128-bit key. The key expansion module consists of SubBytes, ShiftRows and RoundConst functions. SubBytes and ShiftRows functions are explained in sub-sections 3.1 and 3.2. The RoundCons function performs a bitwise XOR operation using a round constant array. Round constant array contains values given by [x^{i-1}, {00}, {00}, {00}] with x^{i-1} being powers of x (x denoted as {02}) in the field GF(2^8). Thus each round key is generated column wise using Eq.(7).

$$N(r,c) = \begin{cases} N(r-1,c) + sbox[Rword(N(r-1,c+3))] \\ \quad +Rcon(r-1) & c = 1 \\ N(r,c-1) + N(r-1,c) & 2 \leqslant c \leqslant 4 \end{cases} \tag{7}$$

In Eq.(7), N(r, c) denotes cth 32-bit column of rth round key, where $1 \le c \le 4$ and $r > 1$. The initial key ($r = 1$) is XORed with input 128-bit plaintext before the first round.

4. Design approach for AES algorithm with enhanced security

As compared to traditional AES algorithm, the proposed work suggests technique to modify the S-box values using PN Sequence Generator to improve the quality of encryption. The initial key required for encryption/decryption is also generated using the PN Sequence Generator instead of using a pre-defined key. In the proposed work, 8-bit PN Sequence Generator is used for generating

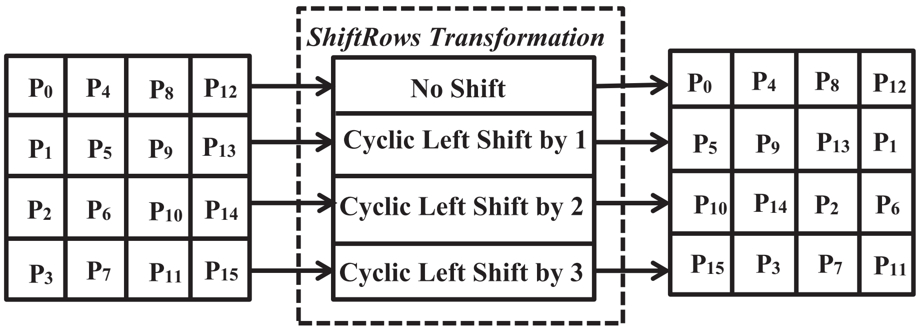

Fig. 2. ShiftRows transformation.

the S-box values and initial key. The key and plaintext sensitivity tests are performed as per Strict Avalanche Criterion. The avalanche values for traditional AES algorithm are compared with avalanche values for AES algorithm with modified S-box values. For this comparison, pre-defined initial keys as well as initial keys generated using 8-bit PN Sequence Generator are considered. Further the proposed design is synthesized using different FPGA devices and comparison with existing FPGA implementations for speed and area optimization is done.

4.1. S-box values generation using 8-bit PN Sequence Generator

A PN Sequence Generator is used to generate sequence of pseudorandom binary numbers. A PN Sequence Generator is designed using Linear Feedback Shift Register (LFSR) described by the Generator Polynomial. LFSR is shift register whose input bit is a linear function of previous state and is generated by XORing selected bits from all the bits of the shift register. The number of states generated by the LFSR is determined by the feedback taps of the Generator Polynomial. The S-box of AES algorithm consists of 256 distinct 8-bit values. For generation of these S-box values, 8-bit PN Sequence Generator is used with maximal length feedback taps to generate $2^8 - 1 = 255$ random values across $(01)_h$ to $(FF)_h$. The value $(00)_h$ is randomly fed into the S-box. To generate, 8-bit maximum length sequence, the Generator polynomial can be set to a value from the following feedback taps viz. [8 6 5 2], [8 6 5 3], [8 6 5 4], [8 7 6 1], [8 6 4 3 2 1], [8 4 3 2], [8 7 6 5 2 1], [8 6 5 1], [8 5 3 1]. In this paper, the tap [8 6 5 4] is selected as a proof of concept and the generator polynomial formed is represented in Fig. 3.

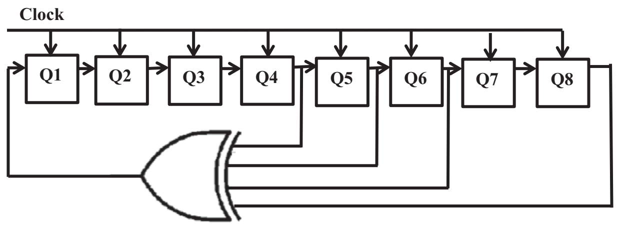

Fig. 3. 8-Bit PN Sequence Generator.

As seen in Fig. 3. bits 8, 6, 5, 4 are XORed and feedback from MSB side on each clock cycle resulting in cyclic shifting of previous value. Thus, a random sequence with a very large repetition period is obtained by combining elements from taps of the shift register and giving a feedback to the input of the generator. The randomness in the output values generated from PN Sequence Generator depends not only on the feedback taps but also on the non-zero initial 8-bit seed value given to the generator. The change in the seed value shifts the starting value and changes the sequence of the generated values. This results in generating sequence known only to the designers. These values can then be used to form the S-box. The invertible S-box is responsible to strengthen the AES algorithm against various attacks. The lack of knowledge of the taps and seed value selected to the attackers will make the AES algorithm invulnerable to attacks.

The modified S-box in hexadecimal number system for encryption using feedback taps [8 6 5 4] and initial seed value as $(1d)_h$ is shown in Table 1.

The individual bytes from the state matrix are replaced with the corresponding value stored in modified S-box in SubBytes Transformation. For this, the higher nibble and lower nibble of the

Table 1
S-box for Encryption.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	1d	0e	07	03	81	c0	60	30	98	4c	26	93	49	24	92	c9
1	64	b2	d9	ec	76	3b	9d	4e	27	13	09	04	82	41	a0	50
2	a8	d4	6a	b5	da	00	6d	b6	5b	ad	d6	6b	35	9a	4d	a6
3	d3	69	34	1a	0d	86	c3	e1	f0	f8	7c	be	df	6f	b7	db
4	ed	f6	7b	bd	5e	af	d7	eb	75	ba	5d	2e	17	8b	45	22
5	11	08	84	c2	61	b0	d8	6c	36	1b	8d	c6	e3	f1	78	3c
6	9e	cf	e7	73	39	9c	ce	67	33	19	8c	46	a3	d1	68	b4
7	5a	2d	96	4b	25	12	89	44	a2	51	28	94	4a	a5	52	a9
8	54	2a	95	ca	e5	72	b9	dc	ee	77	bb	dd	6e	37	9b	cd
9	e6	f3	79	bc	de	ef	f7	fb	fd	7e	bf	5f	2f	97	cb	65
a	32	99	cc	66	b3	59	ac	56	2b	15	8a	c5	62	31	18	0c
b	06	83	c1	e0	70	b8	5c	ae	57	ab	55	aa	d5	ea	f5	fa
c	7d	3e	9f	4f	a7	53	29	14	0a	85	42	21	90	c8	e4	f2
d	f9	fc	fe	ff	7f	3f	1f	0f	87	43	a1	d0	e8	f4	7a	3d
e	1e	8f	c7	63	b1	58	2c	16	0b	05	02	01	80	40	20	10
f	88	c4	e2	71	38	1c	8e	47	23	91	48	a4	d2	e9	74	3a

Table 2
S-box for Decryption.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	25	eb	ea	03	1b	e9	b0	02	51	1a	c8	e8	af	34	01	d7
1	ef	50	75	19	c7	a9	e7	4c	ae	69	33	59	f5	00	e0	d6
2	ee	cb	4f	f8	0d	74	0a	18	7a	c6	81	a8	e6	71	4b	9c
3	07	ad	a0	68	32	2c	58	8d	f4	64	ff	15	5f	df	c1	d5
4	ed	1d	ca	d9	77	4e	6b	f7	fa	0c	7c	73	09	2e	17	c3
5	1f	79	7e	c5	80	ba	a7	b8	e5	a5	70	28	b6	4a	44	9b
6	06	54	ac	e3	10	9f	a3	67	6e	31	22	2b	57	26	8c	3d
7	b4	f3	85	63	fe	48	14	89	5e	92	de	42	3a	c0	99	d4
8	ec	04	1c	b1	52	c9	35	d8	f0	76	aa	4d	6a	5a	f6	e1
9	cc	f9	0e	0b	7b	82	72	9d	08	a1	2d	8e	65	16	60	c2
a	1e	da	78	6c	fb	7d	2f	c4	20	7f	bb	b9	a6	29	b7	45
b	55	e4	11	a4	6f	23	27	3e	b5	86	49	8a	93	43	3b	9a
c	05	b2	53	36	f1	ab	5b	e2	cd	0f	83	9e	a2	8f	66	61
d	db	6d	fc	30	21	bc	2a	46	56	12	24	3f	87	8b	94	3c
e	b3	37	f2	5c	ce	84	90	62	dc	fd	bd	47	13	40	88	95
f	38	5d	cf	91	dd	be	41	96	39	d0	bf	97	d1	98	d2	d3

individual entry from state matrix is taken as row and column number respectively of the S-box. For example, the entry $(76)_h$ in the state matrix will be replaced by $(89)_h$ i.e the value from 7th row and 6th column.

The modified Inverse S-box generated for decryption is shown in Table 2. As in encryption, for decryption also each byte in the state matrix is substituted by the corresponding value of modified Inverse S-box in InvSubBytes Transformation. For example, the entry $(89)_h$ in the state matrix will be replaced by $(76)_h$ i.e the value from 8th row and 9th column.

4.2. Key generation using 8-bit PN Sequence Generator

In the proposed work, the key generation block uses 8-bit PN Sequence Generator to generate the initial key required for encryption/decryption process. The 8-bit PN Sequence Generator generates 255 values of 8-bit each which can be concatenated to form the 128-bit initial key. The key generation block internally selects the bytes for concatenation and form the 128-bit key.

The output states (each of 8-bit) of the PN Sequence Generator are stored in the Look up Table (LUT) and are given as an input to 256:1 multiplexer as shown in Fig. 4. An 8-bit counter generates the value of 8-bit select lines of the multiplexer. The counter is designed to count 16 states so as to select 16 input bytes and form the 128-bit (16×8) value at the output. Thus the key will consists of 16 distinct bytes and a large number of distinct keys can be obtained by changing the initial value of the counter for select lines. These keys can then be dynamically applied to different blocks of 128-bit plaintext from the entire message to be encrypted. For example, setting feedback taps [8 6 5 4] and initial seed value as $(1d)_h$ for the PN Sequence Generator and setting the initial value of the counter for select lines as $(00)_h$, $(1d0e070381c06030984c2693492492c9)_h$ is the initial key generated for encryption/decryption.

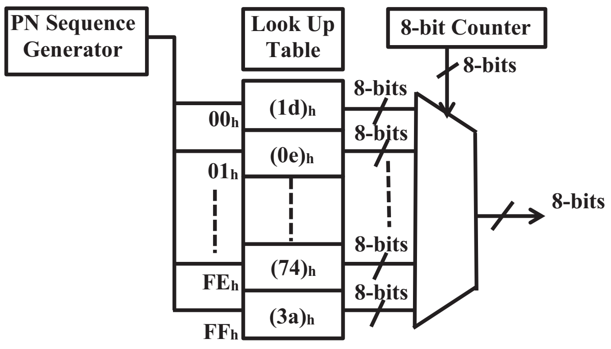

Fig. 4. Key Generation Block.

The generation of key using the key generation block eliminates the need of using external predefined keys. Moreover, the generated key is dependent on the feedback taps and initial seed value of the PN Sequence Generator and change in these parameters will change the value of initial key. In case an adversary tries to decipher the data using brute force attack, then due to use of a different key for each message, the attacker will be unable to decrypt any useful information from the message. Further, the brute force attack will identify the initial key, but without the knowledge of the S-box described in Section 4.1, the attacker will not be able to decipher the input text. Thus these two modifications ensure a very high encryption quality for the cipher.

5. Implementation

For hardware implementation of the proposed non pipelined AES Algorithm, Xilinx Spartan6 – XC6vLX150 FPGA device is used. The Xilinx ISE14.1 tool is used for Synthesis, simulation and generating the programmable file.

5.1. Performance evaluation parameters

The metrics for evaluating the performance of the proposed AES Algorithm with modified S-box and improved key generation technique are as mentioned below.

- Avalanche Effect: The strength of the cryptosystem is tested on the basis of Strict Avalanche Criterion (SAC). SAC is said to be satisfied whenever complementing a single input bit results in change of each of the output bits with a 50% probability. The plaintext and key each of 128-bit are the inputs to the AES algorithm. Thus, with a single bit complemented in plaintext or key, the cipher text should change with a probability of 50% known as Avalanche Effect.
- Throughput and Area: Hardware implementation of the algorithm is evaluated on these two parameters. Throughput signifies the number of bits processed per unit time and is specified in Gbps or Mbps. The throughput is calculated using Eq.(8).

$$Throughput = \frac{No.of\ bits\ processed * Fmax}{Latency} \tag{8}$$

In Eq. (8), No. of bits processed in 128, Fmax is maximum frequency reported by the tool and Latency is the number of clock cycles after which output is generated. For Xilinx Spartan FPGAs, area utilization is specified with respect to number of slices. Every slice of Spartan6 FPGA contains four look-up tables (LUTs) and eight storage elements.

5.2. Experimentation results

The proposed AES algorithm with modified S-box and improved key generation technique is evaluated on various FPGA devices for the performance parameters described in Section 5.1 using Xilinx ISE 14.1. The plaintext and key sensitivity on the Strict Avalanche Criterion is examined for 2048 variations using traditional AES algorithm and AES algorithm with modifications described in this work. From this, the average percentage avalanche value is calculated.

Fig. 5 shows the comparison of Percentage Avalanche Effect for traditional and modified AES algorithm for the initial key $(000102030405060708090a0b0c0d0e0f)_h$ and plaintext $(00112233445566778899aabbccddeeff)_h$. As per the SAC, a single bit change in the input should result in 50% change in output bits. Thus, for calculating the avalanche effect all 128-bits from the initial key are complemented one by one and the corresponding ciphertext with traditional AES and the modified AES are obtained. The ciphertext generated for change in each input bit is compared with the ciphertext obtained with initial key for traditional AES. The count of number of bits changed in the two values is divided by 128 and the value of percentage avalanche effect is calculated for the corresponding input. These steps are also followed for generating the value of percentage avalanche effect for remaining inputs of traditional AES and for modified AES. The plot for these values is shown in Fig. 5. The overall percentage avalanche effect is calculated by taking the count of the number of values increased for modified AES. For the above mentioned set of key and plaintext, an increased percentage avalanche effect of 58% is achieved with modified AES algorithm. Similar experimentation is carried out on different sets of plaintext and initial keys generated using the technique discussed in Section 4.2. For 2048 variations in key and plaintext the percentage avalanche effect achieved varies in the range from 53% to 61% for modified AES algorithm.

For performance evaluation, the proposed algorithm is synthesized on various FPGA devices using Xilinx ISE14.1. The result achieved for throughput and area is shown in Table 3.

The proposed non-pipelined design is synthesized using different FPGA devices and its result are compared with existing non-pipelined implementations as shown in Table 4. Latency of 10 is considered while calculating the throughput of the proposed design as it generates the output after every 10 clock cycles. The proposed design achieves the highest throughput of 6.34Gbps on XC7VX690T device with maximum clock frequency of 495.32 MHz. The number of slices used is only 0.47% of the total available slices for the selected device. Compared to design (Hussain and Jamal, 2012) which generates output after every 11 clock cycles, throughput of the proposed design is improved by 1.2 times at the expense of increased area usage by 1.67 times. Similarly an improvement in throughput by 1.8 times is achieved at the cost of 8.4 times more slice usage as compared to design in (Qiang et al., 2015). Similar is the case with (Wang and Ha, 2013). The proposed design improves the throughput by 1.8 and 1.33 times for the devices XC6VLX240T and XC4VLX60 respectively as compared to (Qiang et al., 2015). The implementation of proposed design on hardware using Xilinx Spartan6 XC6SLX150 device gives a throughput of 3.03 Gbps with maximum frequency of 237.45 MHz.

The comparison of synthesis result of the proposed non-pipelined design with existing pipelined implementations is shown in Table 5. Compared to the 6-stage pipelined design (Wang and Ha, 2013), the proposed non-pipelined design improves the throughput by 1.45 times and utilizes 0.69 times less area. The throughput obtained by (Henezen and Fichtner, 2010) is more compared to proposed non pipelined design at the expense of pipelined design and four parallel Encryption datapath. However for future consideration, introducing pipelining in the proposed design will further improve the throughput of the proposed design. Reddy and Praneeth (2011) have proposed design which uses complex combinational logic for SubBytes transformation and is 3-stage

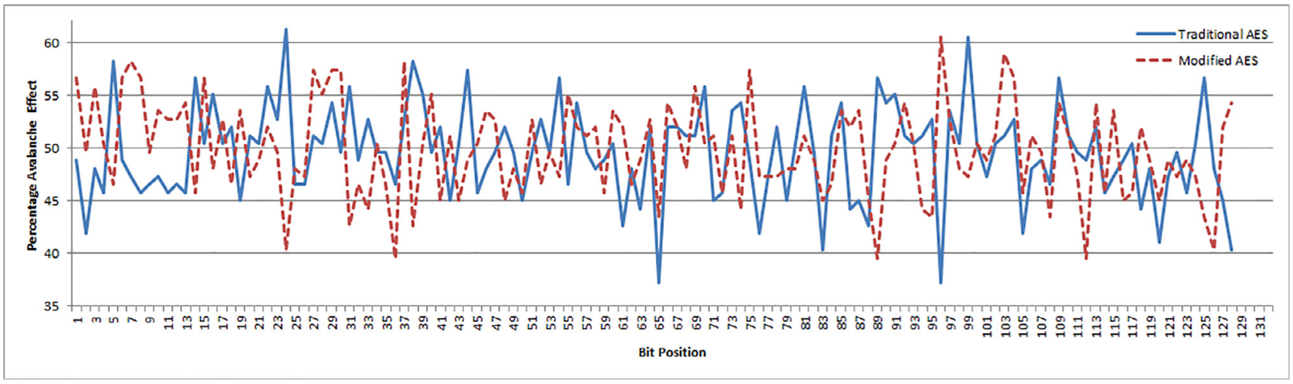

Fig. 5. Comparison of Percentage Avalanche Effect for traditional and modified AES algorithm.

Table 3
Synthesis result of Non-pipelined Modified AES algorithm on various FPGA devices.

Design	Device	Bitwidth (bits)	No. of Slices	F_{max} (MHz)	Throughput (Gbps)	Mbps/Slice
Proposed	XC7VX690T	128	697	372.98	4.34	6.22
Proposed	XC6VLX240T	128	4095	463.42	5.93	1.44
Proposed	XC5VSX240T	128	3420	199.18	25.50	7.45
Proposed	XC5VLX110T	128	3788	232.30	29.73	7.84
Proposed	XC4VLX60	128	20,818	214.48	2.74	0.13
Proposed	XC6SLX150	128	5566	237.45	3.03	0.54

Table 4
Comparison of proposed design with existing non-pipelined designs.

Design	Device	No. of Pipeline stages	Bitwidth (bits)	No. of Slices	F_{max} (MHz)	Throughput (Gbps)	Mbps/Slice
Hussain and Jamal (2012))[*]	Virtex7	Non-Pipelined	128	2444	456.00	5.30	2.17
Qiang et al. (2015)	XC7VX690T	Non-pipelined	128	486	322.58	3.44	7.08
Proposed	XC7VX690T	Non-pipelined	128	4089	495.32	6.34	1.55
Wang and Ha (2013))	XC6VLX240T	Non-pipelined	128	15,612	14.69	1.88	0.12
Qiang et al. (2015)	XC6VLX240T	Non-pipelined	128	335	323.73	3.45	10.29
Proposed	XC6VLX240T	Non-pipelined	128	4095	463.42	5.93	1.44
Qiang et al. (2015)	XC4VLX60	Non-pipelined	128	1975	192.68	2.06	1.04
Proposed	XC4VLX60	Non-pipelined	128	20,818	214.48	2.74	0.13
Proposed	XC6SLX150	Non-pipelined	128	5566	237.45	3.03	0.54

[*] The specific device used in (Hussain and Jamal, 2012) is not known.

Table 5
Comparison of proposed design with existing pipelined designs.

Design	Device	No. of Pipeline stages	Bitwidth (bits)	No. of Slices	F_{max} (MHz)	Throughput (Gbps)	Mbps/Slice
Wang and Ha (2013)	XC6VLX240T	6	128	5927	319.29	40.87	6.90
Proposed	XC6VLX240T	Non-pipelined	128	4095	463.42	59.31	1.44
Henezen and Fichtner, (2010)	XC5VSX240T	2	128	1499	233.00	119.30	8.06
Proposed	XC5VSX240T	Non-pipelined	128	3420	199.18	25.50	7.45
Reddy and Praneeth, (2011)	XC5VLX110T	3	128	8896	202.26	25.89	2.91
Proposed	XC5VLX110T	Non-pipelined	128	3788	232.30	29.73	7.84

[*]Latency is not considered while calculating throughput, based on (Wang and Ha, 2013), (Henzen and Fichtner, 2010) and (Reddy and Praneeth 2011).

pipelined. In the proposed non-pipelined design LUT based Sub-Bytes transformation is used which results in improving the throughput 1.15 times and utilizes 0.42 times less area as compared to (Reddy and Praneeth 2011).

6. Conclusion

Encryption using hardware platforms is widely being used in order to secure data and enhance throughput. In this paper, techniques to enhance the encryption quality of AES algorithm and its implementation on FPGA are proposed. First, the S-box values in the modified AES algorithm are generated using PN Sequence Generator. Second, the initial key required for encryption/decryption is also based on the output of PN Sequence Generator. The result of encryption for modified AES algorithm is tested on the Strict Avalanche Criterion for 2048 variations and the average percentage avalanche effect of 60% is achieved for modified AES algorithm as compared to tradition AES algorithm. Thus the modifications suggested, results in improved quality of encryption. FPGAs are used for efficient hardware implementation of the modified AES algorithm. The results for throughput and area are compared with existing non-pipelined and pipelined designs and are found to achieve better performance.

The proposed design achieves a throughput of 59.3Gbps when synthesized on XC6VLX240T device with a maximum frequency of 463.42 MHz and 30.39 Gbps when implemented on XC6SLX150 with a maximum frequency of 237.45 MHz.

References

Ahmad, N., Hasan, R., Jubadi, W.M., 2010. Design of AES S-box using combinational logic optimization. IEEE Symp. Ind. Electron. Appl., 696–699

Alexandru, C., Fratila, R., 2011. Cryptographic Applications using FPGA Technology. J. Mobile Embedded Distrib. Syst. 3, 10–16.

Ali, L., Aris, I., Hossain, F.S., Roy, N., 2011. Design of an ultra high speed AES processor for next generation IT security. Comput. Electr. Eng. 37, 1160–1170. https://doi.org/10.1016/j.compeleceng.2011.06.003.

Bogdanov, A., Khovratovich, D., Rechberger, C., 2011. Biclique cryptanalysis of the Full AES. Adv. Cryptol. ASIACRYPT 2011, 344–371. https://doi.org/10.1007/978-3-642-25385-0.

Chih Peng, F., Hwang, J.K., 2008. FPGA implementations of high throughput sequential and fully pipelined AES algorithm. Int. J. Electr. Eng. 15, 447–455. https://doi.org/10.1109/ISPACS.2007.4445896.

Chih-Pin, S., Tsung-Fu, L., Chih-Tsun, H., Cheng-Wen, W., 2003. A high-throughput low-cost AES processor. IEEE Commun. Mag., 86–91

Chodowiec, P., Gaj, K., 2003. Very Compact FPGA implementation of the AES algorithm. CHES 2003. LNCS 2779, 319–333. https://doi.org/10.1007/978-3-540-45238-6_26.

Ebrahim, M., Khan, S., Khalid, U. Bin, 2014. Symmetric algorithm survey: a comparative analysis. Int. J. Comput. Appl. 61, 12–19.

Elbirt, A.J., Yip, W., Chetwynd, B., Paar, C., 2001. An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. IEEE Trans. Very Large Scale Integr. Syst. 9, 545–557. https://doi.org/10.1109/92.931230.

Farooq, U., Aslam, M.F., 2017. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. J. King Saud Univ. Comput. Inf. Sci. 29, 295–302. https://doi.org/10.1016/j.jksuci.2016.01.004.

Fips-197, 2001. Advanced encryption standard (AES). Natl. Inst. Stand. Technol. 8–12. https://doi.org/10.1016/S1353-4858(10)70006-4.

Gaj, K., Chodowiec, P., 2009. FPGA and ASIC implementations of AES. Cryptogr. Eng. 235–294. https://doi.org/10.1007/978-0-387-71817-0.

Gangadari, B.R., Rafi Ahamed, S., 2016. Design of cryptographically secure AES like S-Box using second-order reversible cellular automata for wireless body area network applications. Healthcare Technol. Lett. 3, 177–183. https://doi.org/10.1049/htl.2016.0033.

Granado-Criado, J.M., Vega-Rodríguez, M.A., Sánchez-Pérez, J.M., Gómez-Pulido, J.A., 2010. A new methodology to implement the AES algorithm using partial and dynamic reconfiguration. Integr. VLSI J. 43, 72–80. https://doi.org/10.1016/j.vlsi.2009.05.003.

Hammad, I., El-Sankary, K., El-Masry, E., 2010. High-speed AES encryptor with efficient merging techniques. IEEE Embedded Syst. Lett. 2, 67–71. https://doi.org/10.1109/LES.2010.2052401.

Henzen, L., Fichtner, W., 2010. FPGA parallel-pipelined AES-GCM core for 100G ethernet applications. Proc. ESSCIRC, 202–205.

Hongge, L., Jinpeng, D., Yongjun, P., 2012. Cell array reconfigurable architecture for high-efficiency AES system. Microelectron. Reliab. 52, 2829–2836. https://doi.org/10.1016/j.microrel.2012.04.020.

Hussain, U., Jamal, H., 2012. An efficient high throughput FPGA implementation of AES for multi-gigabit protocols. Proc. 10th Int Conf. Front. Inf. Technol. FIT 2012, 215–218. https://doi.org/10.1109/FIT.2012.45.

Iyer, N., Anandmohan, P.V., Poornaiah, D.V., Kulkarni, V.D., 2011. Efficient hardware architectures for AES on FPGA. Commun. Comput. Inf. Sci., 249–257 https://doi.org/10.1007/978-3-642-25734-6_37.

Karthigai Kumar, P., Baskaran, K., 2010. An ASIC implementation of low power and high throughput blowfish crypto algorithm. Microelectron. J. 41, 347–355. https://doi.org/10.1016/j.mejo.2010.04.004.

Liu, L., Luke, D., 2003. Implementation of AES as a CMOS core. CCECE 2003. Can. Conf. Electr. Comput. Eng. 1, 53–56.

Liu, R., Parhi, K.K., 2008. Fast composite field S-box architectures for advanced encryption standard. ACM Gt. Lakes Symp. VLSI, 65–70.

Mohammed, A., Mousa, F., Radwan, T., Odeh, M., 2011. Survey paper: cryptography is the science of information security. Int. J. Comput. Sci. Secur. 5, 298–309.

Qiang, L., Zhenyu, X., Yuan, Y., 2015. High throughput and secure advanced encryption standard on field programmable gate array with fine pipelining and enhanced key expansion. IET Comput. Digit. Tech. 9, 175–184. https://doi.org/10.1049/iet-cdt.2014.0101.

Qu, S., Shou, G., Hu, Y., Guo, Z., Qian, Z., 2009. High throughput, pipelined implementation of AES on FPGA. Int. Symp. Inf. Eng. Electron. Commer., 542–545
Rais, M.H., Qasim, S.M., 2009a. Efficient hardware realization of advanced encryption standard algorithm using Virtex-5 FPGA. Int. J. Comput. Sci. Netw. Secur. 9, 59–63.
Rais, M.H., Qasim, S.M., 2009b. A Novel FPGA implementation of AES-128 using reduced residue of prime numbers based S-Box. Int. J. Comput. Sci. Netw. Secur. 9, 305–309.
Rashidi, B., Rashidi, B., 2013. Implementation of an optimized and pipelined combinational logic Rijndael S-Box on FPGA. I.J. Comput. Netw. Inf. Secur. 1, 41–48. https://doi.org/10.5815/ijcnis.2013.01.05.
Reddy, R.S.S.K., Praneeth, P., 2011. VLSI implementation of AES crypto processor for high throughput. Int. J. Adv. Eng. Sci. Technol. 6, 22–26.
Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D., 2003. Efficient Implementation of Rijndael encryption in reconfigurable hardware: improvements and design tradeoffs. Cryptogr. Hardware Embedded Syst. 2003, 334–350. https://doi.org/10.1007/978-3-540-45238-6_27.
Wang, Y., Ha, Y., 2013. FPGA-based 40.9-gbits/s masked AES with area optimization for storage area network. IEEE Trans. Circuits Syst. II Express Briefs 60, 36–40. https://doi.org/10.1109/TCSII.2012.2234891.
Zhang, X., Parhi, K.K., 2004. High-speed VLSI architectures for the AES algorithm. IEEE Trans. Very Large Scale Integr. Syst. 12, 957–967. https://doi.org/10.1109/TVLSI.2004.832943.
Zhang, Y., Wang, X., 2010. Pipelined implementation of AES encryption based on FPGA. IEEE Int. Conf. Inf. Theory Inf. Secur, 170–173.