International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

AES Using Linear Feedback Shift Register (LFSR) for Enhanced Cryptographic Performance

K Kavitha[1], Patan Mohammad Khan[2], Gandla Likith Kumar[3], Vadlamudi Gowtham[4], Aradyula Bala Murali Krishna[5]

[1]Guide of the project, Assistant professor, Department of Electronics and Communication Engineering, Krishna University College of Engineering and Technology, Machilipatnam-521001, India

[2],[3],[4],[5]Department of Electronics and Communication Engineering, Krishna University College of Engineering and Technology, Machilipatnam-521001, India

Abstract

AES Using LFSR (Linear Feedback Shift Register) is an approach that integrates the **AES encryption algorithm** with the use of **LFSR-based techniques** to enhance the performance, security, and hardware efficiency of the cryptographic system. AES (Advanced Encryption Standard) is one of the most widely used symmetric-key encryption algorithms, offering high security and fast encryption. However, the need for efficient hardware implementations has led to the exploration of various methods to speed up the encryption process while maintaining security.

The integration of **LFSR** into the AES structure primarily targets the generation of **pseudo-random key streams** for **key expansion** or **stream cipher generation**, improving the **randomness** and **speed** of the key schedule, which is a vital part of the AES algorithm. By utilizing **LFSR-based key generation techniques**, the AES cipher can achieve faster processing speeds, lower hardware complexity, and reduced power consumption in **embedded systems**, **FPGA**, and **ASIC** implementations.

This method focuses on optimizing the AES algorithm's key expansion process by replacing traditional methods with **LFSR-based key generation**, enhancing the overall throughput while maintaining the encryption strength. The use of LFSR also provides a more flexible and scalable approach, allowing for higher-speed cryptographic operations in hardware implementations.

In this paper, we present the design and implementation of an AES encryption system using LFSR-based key generation and compare it with conventional AES implementations in terms of **performance**, **area efficiency**, and **power consumption**. Experimental results demonstrate that this LFSR-enhanced AES approach significantly improves the throughput, making it suitable for high-performance and resource-constrained environments, particularly in **IoT devices**, **mobile platforms**, and **real-time cryptographic applications**.

Keywords: AES, LFSR, Cryptography, FPGA, Key Expansion, Hardware Security, Verilog.

Chapter 1: Introduction

The selective application of technological and related procedural safeguards is an important responsibility of every Federal organization in providing adequate security to its electronic data systems. This publication specifies a cryptographic algorithm, the Advanced Encryption Standard (AES) which may be used by Federal organizations to protect sensitive data. Protection of data during transmission or while in storage may be necessary to maintain the confidentiality and integrity of the information represented by the data.

The algorithms uniquely define the mathematical steps required to transform data into a cryptographic cipher and also to transform the cipher back to the original form. The Advanced Encryption Standard is being made available for use by Federal agencies within the context of a total security program consisting of physical security procedures, good information management practices, and computer system/network access controls.

Data encryption (cryptography) is utilized in various applications and environments. The specific utilization of encryption and the implementation of the AES will be based on many factors particular to the computer system and its associated components. In general, cryptography is used to protect data while it is being communicated between two points or while it is stored in a medium vulnerable to physical theft. Communication security provides protection to data by enciphering it at the transmitting point and deciphering it at the receiving point. File security provides protection to data by enciphering it when it is recorded on a storage medium and deciphering it when it is read back from the storage medium. In this the key must be available at the transmitter and receiver simultaneously during communication.

Cryptography is probably the most important aspect of communication security becoming increasingly important as a basic building block for computer security. Cryptographic systems are characterized along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext: All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext is mapped into another element, and transposition, in which element in the plaintext are rearranged. The fundamental requirement is that no information be lost. Most systems referred to as product systems, involve multiple stages substitutions and transpositions.

2. The number of keys used: If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.

3. The way in which the plaintext is processed: A block cipher processes the input one block of elements at a time, producing an output block for each input block. A stream cipher processes the input elements continuously, producing output one element at a time, as it goes along. Before beginning, we define some terms. A symmetric encryption scheme has five ingredients:

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption Algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertext. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essential the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

The process of securing data from any means of unapproved access and data corruption through its entire life is said to be data security [1]. With the continuous improvement in the field of technology, the data getting unsecured. Every now and then hackers are trying to hack one's data. Therefore the security of data is the most concerned thing in people's minds. The security of data can be achieved by either software means or hardware means. Nowadays hardware approach to protect the data is getting more attention. This is because by means of hardware protecting the

data is more reliable, flexible and less complex. The hardware approach also gives minimal delay and provides more efficiency to data security. To protect the data from any unrecognized access there are two types of security algorithms. The first one is Symmetric security algorithms which cover Data Encryption Standard (DES) and Advanced Encryption Standard (AES). The second one is Asymmetric security algorithms which cover Rivest-Shamir-Adleman (RSA) & Elliptic Curve Cryptosystem (ECC) [2]. FPGA devices are practiced for a hardware approach to secure one's data. This is because FPGA devices are more frequent give flexibility and it has good speed and throughput as compared to software approach. FPGA devices have so much variation so that it is very difficult for hackers to breach security.

Internet of Things (IoT) is the next revolution of the internet which brings profound impact on our everyday lives. IoT is the extension of the Internet to connect just about everything on the planet. This includes real and physical objects ranging from household accessories to industrial engineering. As such these "things" that are connected to the Internet will be able to take actions or make decisions based on the information they gather from the Internet with or without human interaction. In addition, they also update the Internet with real-time information with the help of various sensors. IoT works with resource-constraint components such as sensor nodes, RFID tags etc. These components have low computation capability, limited memory capacity and energy resources, and susceptibility to physical capture. Also, they communicate through the wireless communication channel which is not secured and transmit real-time information through the treacherous wireless medium. In certain applications, confidentiality, authentication, data freshness, and data integrity might be extremely important. Therefore, encryption of data is becoming a major concern. But due to resource-constraint nature of the components, short-term security can meet the demand. By the term resource-constraint means low battery power, less computational speed etc. Encrypting data using standardized cryptographic algorithms may consume more energy which drastically reduces the lifetime of the components. Two main approaches are followed to design and implement security primitives which are fitted with extremely constraint devices. Firstly, designing new lightweight cryptosystem. For instance, are some recently proposed lightweight cryptographic algorithms. Secondly, modifying the existing standard cryptosystem in a lightweight fashion. Possible examples of the second approach are modification of the Advanced Encryption Standard Algorithm (AES), SHA-256 etc. With respect to the security aspect and implementation complexity, AES is considered as one of the strongest and efficient algorithms. Despite that like other symmetric encryption algorithms, the secret key distribution is still considered as a critical issue. Again to encrypt or decrypt a single block (128-bit) of data, an essential amount of computational processing has to be done which consumes enormous battery power. As components of IoT have resource-constraint characteristics, consuming immense power may cause expiration of such components. Analyzing related work, we come to know that Substitution Layer is the most energy consuming portion of AES in the round based design.

Chapter 2: Literature Survey

The literature survey focuses its attention towards AES, particularly to utilize under low power consumption, high security, better performance and improved efficiency. The implementation feasibility in VLSI environment is also studied and analyzed in depth.

2.1 Fault Analysis in AES-CBC Algorithm Using Hamming Code for Space Applications

National institute of standard and technology (2001) presented computer security. Two FIPS publications already prove the modes of operation for two particular block cipher algorithms. Four of these modes are equivalent to the ECB, CBC, CFB, and OFB modes with the Triple DES algorithm (TDEA) as the underlying block cipher. For any given key, the block cipher algorithm of the mode consists of two function that are inverses of each other.

Francois-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat presented (2004) discussed about the Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware. It addressed various approaches for efficient FPGA implementations of the Advanced Encryption Standard algorithm. In

implementation of block ciphers, several strategies can produce effective designs. Inherent constraints of FPGAs were taken into account in order to define an efficient methodology. Inside these architectures, the authors proposed algorithmic optimizations for the substitution box, and also efficient combinations between the diffusion layer and the key addition.

Farhadian.A and Aref.M.R (2009) presented efficient method for simplifying and approximating the s-boxes based on power functions. In this paper cipher algorithms, power functions over finite fields and special inversion functions have an important role in the S-box design structure. A new systematic efficient method is introduced to crypt analyze such S-boxes. This method is very simple and does not need any heuristic attempt and can be considered as a quick criterion to find some simple approximations. Using this new method, approximations can be obtained for advanced encryption standard (AES) like S-boxes, such as AES, Camellia, and Shark and so on. Finally as an application of this method, a simple linear approximation for AES S-box is presented.

Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh (2011) presented a Compact Rijndael Hardware Architecture with S-Box Optimization. Encryption and decryption data paths are combined and all arithmetic components are reused. An extremely small size of 5.4 K gates is obtained for a 128-bit key Rijndael circuit using a 0.11-µm CMOS standard. In order to minimize the hard-ware size, the order of the arithmetic functions were changed, and the encryption-decryption data paths are efficiently combined with respect to cell library. Logic optimization techniques such as factoring were applied to the arithmetic components, and gate counts were reduced.

Ashkan Masoomi and Roozbeh Hamzehiyan (2012) presented a new approach for detecting and correcting errors in satellite communications based on Hamming Error Correcting Code. A novel model to detect and correct Single Event Upsets in on-board implementations of the AES algorithm was based on hamming error correcting code. Single Even Upset (SEU) faults occur in the on-board during encryption due to radiation. Some of the AES modes like ECB, CBC, OFB, CFB and CTR performances have been analyzed. From that, CTR mode has been recommended as the optimum choice for satellite applications.

Ramesh Babu, George Abraham and Kiransinh Borasia (2012) presented a Review on Securing Distributed Systems Using Symmetric Key Cryptography. It was used to evaluate the importance of Symmetric Key Cryptography for Security in Distributed Systems. Two symmetric key cryptographic algorithms DES and AES were commonly used. These two algorithms were evaluated on the parameters such as key size, block size, number of iterations. From the literatures reviewed of various implementations and analysis of both the algorithms, it can be concluded that AES algorithm has over-shadowed the DES algorithm in many areas.

Karri, R., Wu, K., Mishra, P., and Kim, Y. (2002) Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers. They presented algorithm level, round level, and operation level CED (Concurrent Error Detection) architectures for symmetric block ciphers. The algorithm was independent and can be applied to almost any symmetric block cipher. The proposed scheme introduced moderate area overhead and interconnects complexity to achieve permanent as well as transient fault tolerance.

2.2 A Secure Implementation of Nonlinear AES S-Box and Fault Analysis in AES-CM Algorithm Using Hamming Code for Space Applications

Ross Anderson, Eli Biham, Lars Knudsen (1999) presented a Proposal for the Advanced Encryption Standard. Its design is highly conservative, yet still allows a very efficient implementation. With a 128-bit block size and a 256-bit key, it is as fast as DES on the market leading Intel Pentium/MMX platforms yet we believe it to be more secure than three-key triple-DES. The linear transformations were just bit permutations, which were applied as rotations of the 32-bitwords in the bit slice implementation.

A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar (2000) presented an FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists. Reprogrammable devices such as Field Programmable Gate

Arrays (FPGAs) are highly attractive options for hardware implementations of encryption algorithms as they provide cryptographic algorithm agility, physical security, and potentially much higher performance than software solutions.

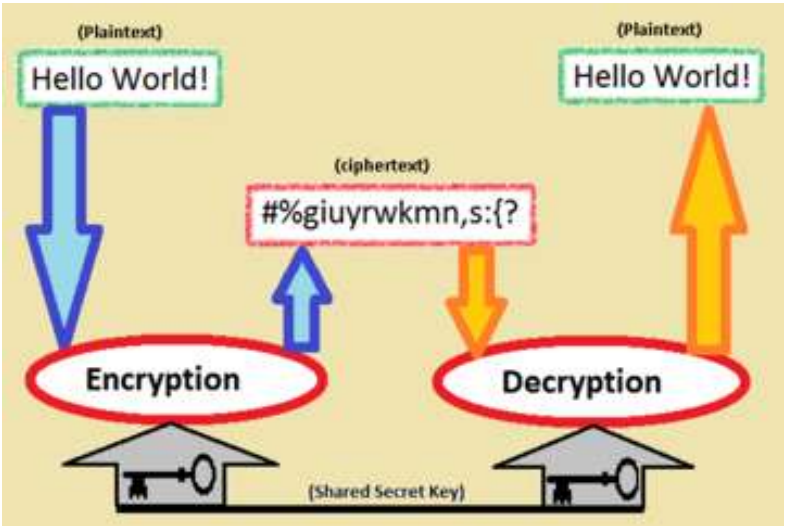
Pawel Chodowiec, Kris Gaj, Peter Bellows and Brian Schott (2001) presented Experimental Testing of the Gigabit IPSec-Compliant Implementations of Rijndael and Triple DES Using SLAAC-1V FPGA Accelerator Board. Full implementations of the new Advanced Encryption Standard, Rijndael, and the older American federal standard, Triple DES, were developed and experimentally tested using the SLAAC-1V FPGA accelerator board, based on Xilinx Virtex 1000 devices.

Xinmiao Zhang and Keshab K. Parhi (2004) presented high speed VLSI architectures for the AES algorithm. A novel high-speed architecture for the hardware implementation of the Advanced Encryption Standard (AES) algorithm. Composite field arithmetic is employed to reduce the area requirements, and different implementations for the inversion in subfield GF (2^4) are compared.

Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., and Piuri, V (2002) presented Concurrent fault detection for a hardware implementation of the Advanced Encryption Standard (AES). It is very important not only to protect the encryption/decryption process from random faults. It will also protect the encryption/decryption circuitry from an attacker who may maliciously inject faults in order to find the encryption secret key.

Chapter 3: AES (Advanced Encryption Standard)

The selective application of technological and related procedural safeguards is an important responsibility of every Federal organization in providing adequate security to its electronic data systems. This publication specifies a cryptographic algorithm, the Advanced Encryption Standard (AES) which may be used by Federal organizations to protect sensitive data. Protection of data during transmission or while in storage may be necessary to maintain the confidentiality and integrity of the information represented by the data.



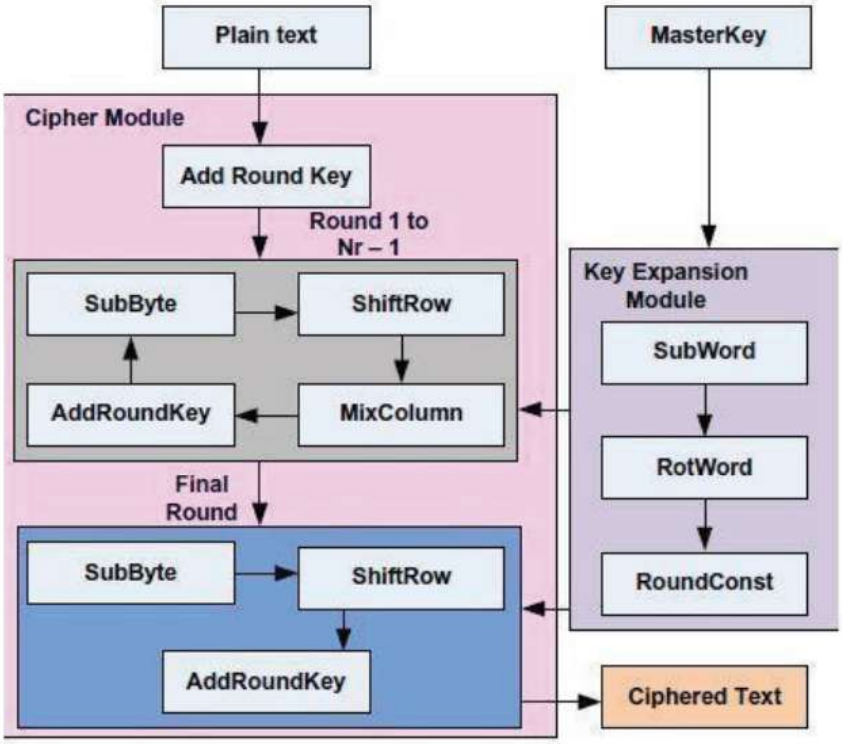
[FIGURE: Advanced Encryption Standard (AES) diagram]

Existing Design

The AES Algorithm

Advanced encryption standard is the most widely used cryptographic algorithm that is compatible with hardware and software at the same time. The data input given is encrypted by AES using a key provided by the user or the developer. AES is a block cipher that transform the 128-bit input data by permutations and combinations using a key of size 128-bit, 192-bit or 256-bit secret key. The National Institute of standards and Technology have announced in Jan

1997 to initiate the development of new AES cipher. In October 2001 they have announced that the new Rijindal algorithm have won the competition between different algorithms developed and proposed that the new Advanced Encryption Standard will be Rijindal algorithm. From there on-wards AES is being widely used as the base of cryptography in terms of hardware and software. It have replaced the existing DES and Triple DES to provide better security benefits and secure architecture. AES is being widely used in the commercial market for different kind of data transactions. It paved the way to make the cryptographic algorithm widespread in the market. There are several hardware architecture using the benefits of AES to perform better encryption with the help of other hash algorithms like SHA, MD5 etc.



[FIGURE: AES encryption diagram]

Steps In AES Algorithm

- **Key Expansions:** The input key is expanded for the number of rounds plus one for the addround key step in each stage. The keys are obtained by from the cipher key by the Rijndael's key schedule.
- **Initial Round:** AddRoundKey: Each byte of the state is combined with the round key block using bitwise XOR.

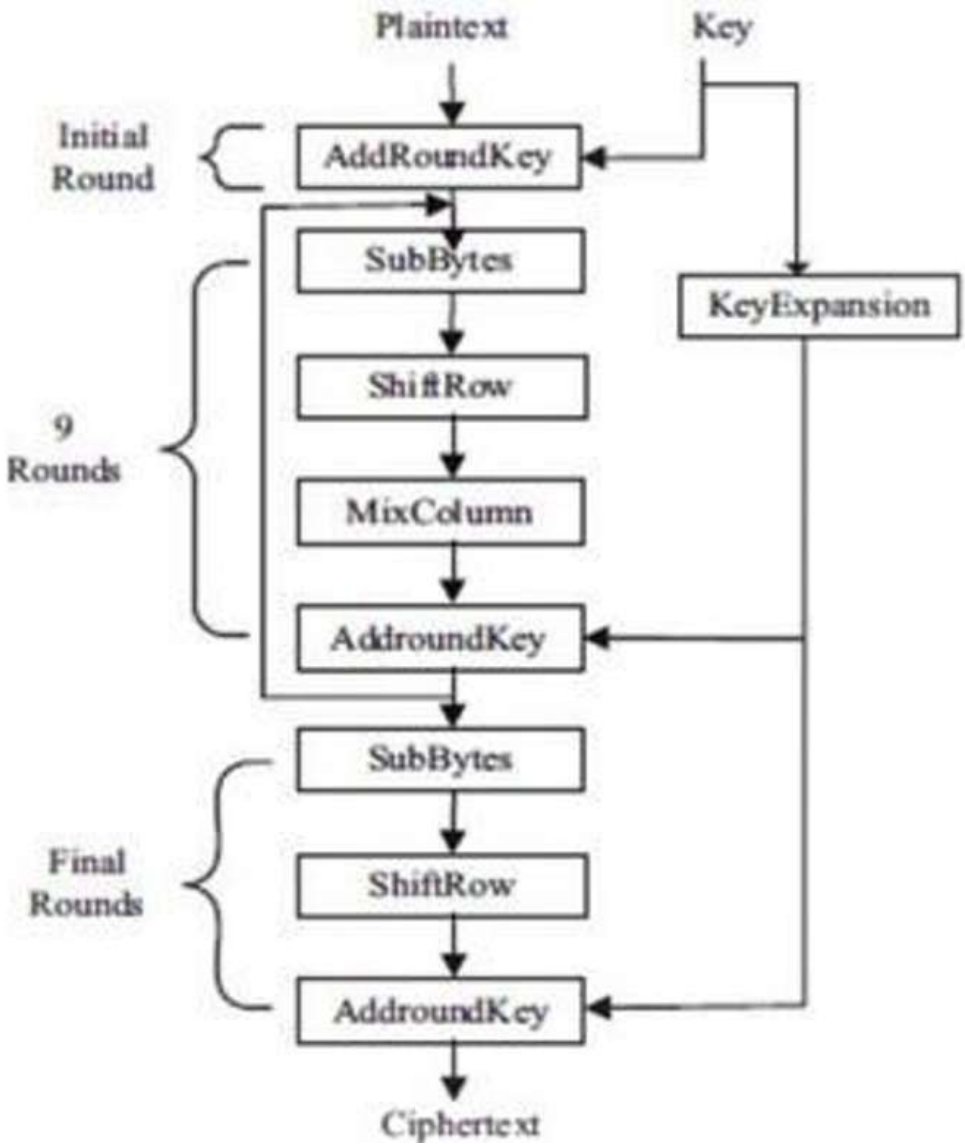
Rounds:

- a) SubBytes
- b) ShiftRows
- c) MixColumns
- d) AddRoundKey

Last Round:

- a) SubBytes
- b) ShiftRows

- c) AddRoundKey



[FIGURE: Working of AES diagram]

1. SUB-BYTES: This non linear byte substitution operation will operate on each byte independently. The substitution table or the sbox is invertible and is constructed by the combination of two transformations:
i) Multiplicative inverse is taken in the Rijndael's finite field.
ii) Then doing the affine transformations.

2. SHIFT ROW STEP: In this step each columns are shifting row wise. The initial row is not getting shifted, it will be same as previous. The 2nd row is left shifted to left one time, the 3rd row two times and the 4th row three times.

3. MIX COLUMNS: By treating each column as a four-term polynomial the mix column operation makes the transformation on the state column by column. The columns are considered as polynomials over GF 28 and multiplied modulo x4 +1 with a fixed polynomial a(x), given by

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

4. KEY EXPANSION: The initial key we provide as the input to AES will not be sufficient for the proceeding 10 rounds. The key expander algorithm follows a Key expander (or generator) operation basically follows five steps to generate a unique key for each round in AES. Every key produced will be same width as that of input key. The input

to key expander circuit will be the key from the LFSR here, thereby making the keys also unpredictable by the attacker.

5. ADDROUNDKEY: The main function of the Add Round Key is to associate the keys generated by the key expander step to XOR with the output got from the mixcolumn step. The initial 128 bit key is expanded by the key expansion method to increase the key size for multiple rounds. The round key length will be matching with the block size length, that is 16 bytes. The output of addround key is got by XORing of Key expansion output and the Mix columns output. The output given above is encrypted output of output1. The output of the Add Round Key step is given as input to the next round to process. The feedback creates a loop and runs for 10 rounds of this stage.

Proposed Design

Advanced Encryption Algorithm (AES)

The Advanced Encryption Algorithm (AES), a symmetric block cipher algorithm that can processes blocks of 128-b, using cipher keys with lengths of 128,192 and 256-bits. The input and output for the AES algorithm each consist of sequence of 128-b (digit with values of 0 or 1). These sequences will sometimes refer to as blocks and the number of bits they contain will be referred to as their length. Internally, the AES algorithm's operation is are performed on a two-dimensional array of bytes called the state as shown in the Figure.1. The state consists of four rows of bytes, each containing Nb bytes, where Nb is the block length. Here Nb = 4, which reflects the number of 32-b words (number of columns) in the state.

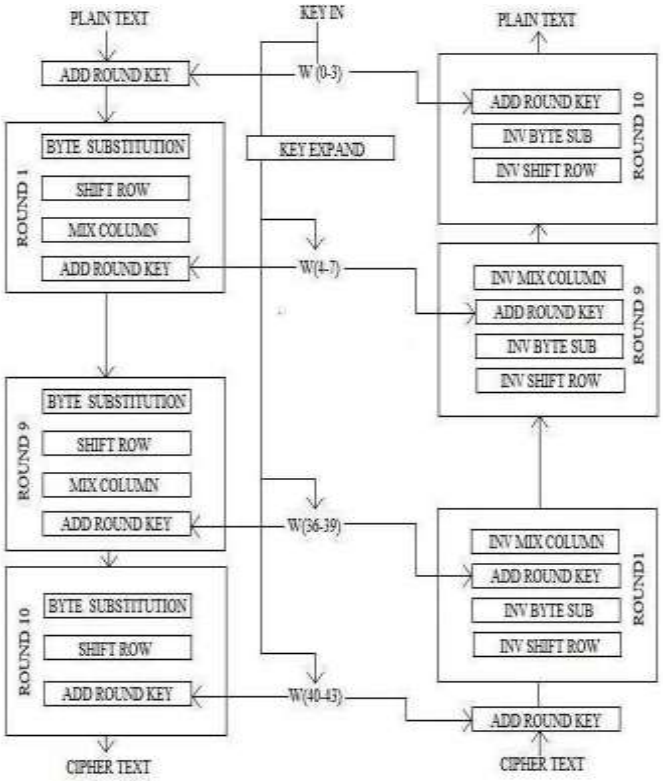
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

[FIGURE: State Array]

For AES algorithm, the length of the cipher key, K, is 128,192 or 256 bits. The key length is represented by Nk = 4, 6, or 8, which reflects the number of 32-b words (number of columns) in the cipher key. The number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by Nr, where Nr = 10 when Nk = 4, Nr = 12 when Nk = 6 and Nr = 14 when Nk = 8. Here we consider Nr = 10 and Nk = 4 for design of the architecture.

	Key Length *(Nk words)*	Block Size *(Nb words)*	Number of Rounds *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

[FIGURE: Key-Block-Round combination]



[FIGURE: AES Encryption and Decryption]

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1      // round 1 to 9
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)                   // last round
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

[FIGURE: Pseudo Code for AES Cipher Process]

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1      // round 9 to 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)                    // last round
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

[FIGURE: Pseudo Code for AES Decipher Process]

1. Byte Substitution

Encryption and Decryption Transformation

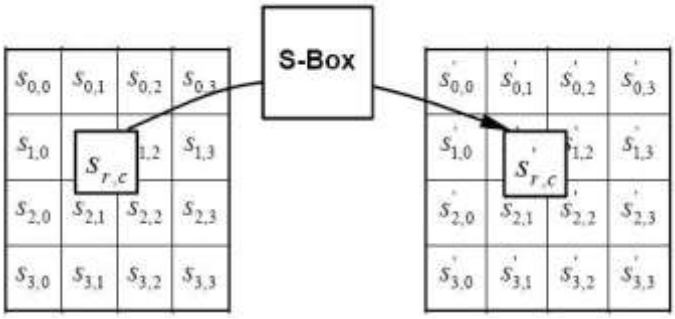
The substitute byte transformation, called the byte sub, is a simple table lookup. AES defines a 16X16 matrix of byte values, called an S-Box that contains a permutation of all possible 256 8-b values. Each individual byte of the state is mapped into a new byte in the following way: The leftmost 4-b of the byte value is used as a row value and the rightmost 4-b are used as a column value. These row and column values serve as indexes into the S-Box to select a unique 8-b output value.

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

[FIGURE: AES S-Box Table]

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

[FIGURE: AES Inverse S-Box Table]



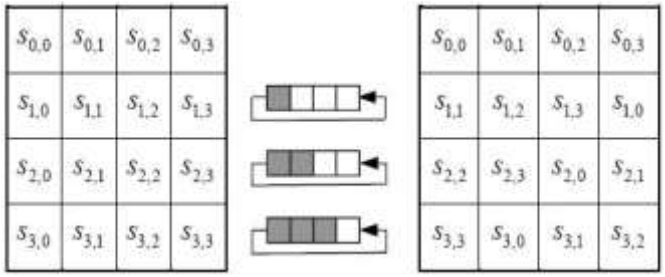
[FIGURE: AES Byte Substitution Operation]

[FIGURE: AES Inverse S-Box Table]

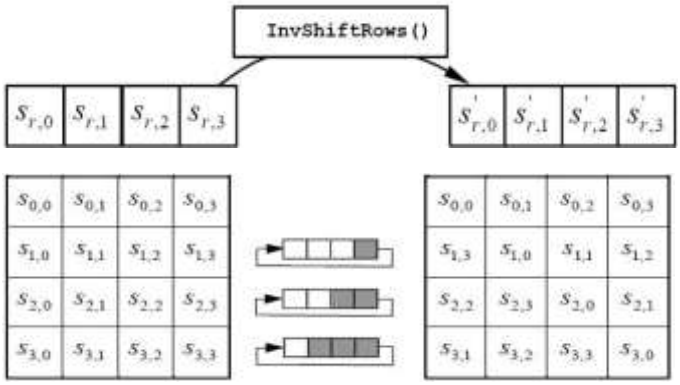
Shift Row

Encryption and Decryption Transformation

In this step, a normal left circular shift operation is done where in the first row is not altered and the next three rows are moved by 1, 2, 3 bytes respectively.



[FIGURE: AES Shift Row operation]



[FIGURE: AES Inverse Shift Row operation]

Mix Column

Encryption and Decryption Transformation

The mix column transformation operates on the state column-by-column, treating each column as a four term polynomial. The column are considered as polynomial over GF (2^8) and multiplied with modulo x^4+1 with a fixed polynomial a(x), given by

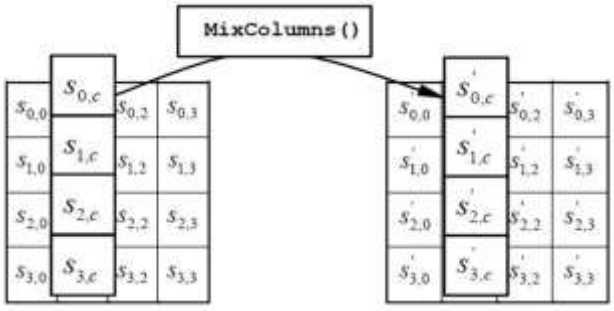
$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

Let s`(x) = a(x) x s(x):

As a result of this multiplication, the four bytes in a column are replaced.

While for the decryption process the Inv mix column is inverse of the mix column transformation, where in it is multiplied with fixed polynomial $a^{-1}(x)$, given by

$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$



[FIGURE: AES Mix Column Operation]

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

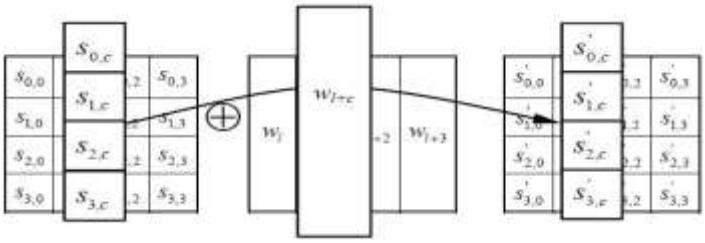
$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

[FIGURE: AES Mix Column Operation]

Add Round Key

Encryption and Decryption Transformation

In the Add round key transformation, the 128-b of the state is bitwise XORed with128-b of the round key. The Add round key operation in decryption transformation is same, as XOR operation is its own inverse.



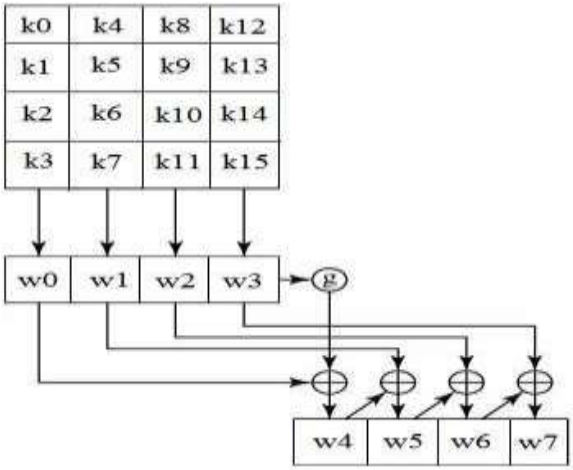
[FIGURE: AES Add Round Key Transformation]

Key Expansion

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The following pseudocode describes the expansion:

text

```
KeyExpansion (byte key[16], word w[44])
{
   word temp
   for (i = 0; i < 4; i++)
      w[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);
   for (i = 4; i < 44; i++)
   {
      temp = w[i-1];
      if (i mod 4 = 0)
         temp = SubWord (RotWord (temp)) XOR Rcon[i/4];
      w[i] = w[i-4] XOR temp
   } }
```

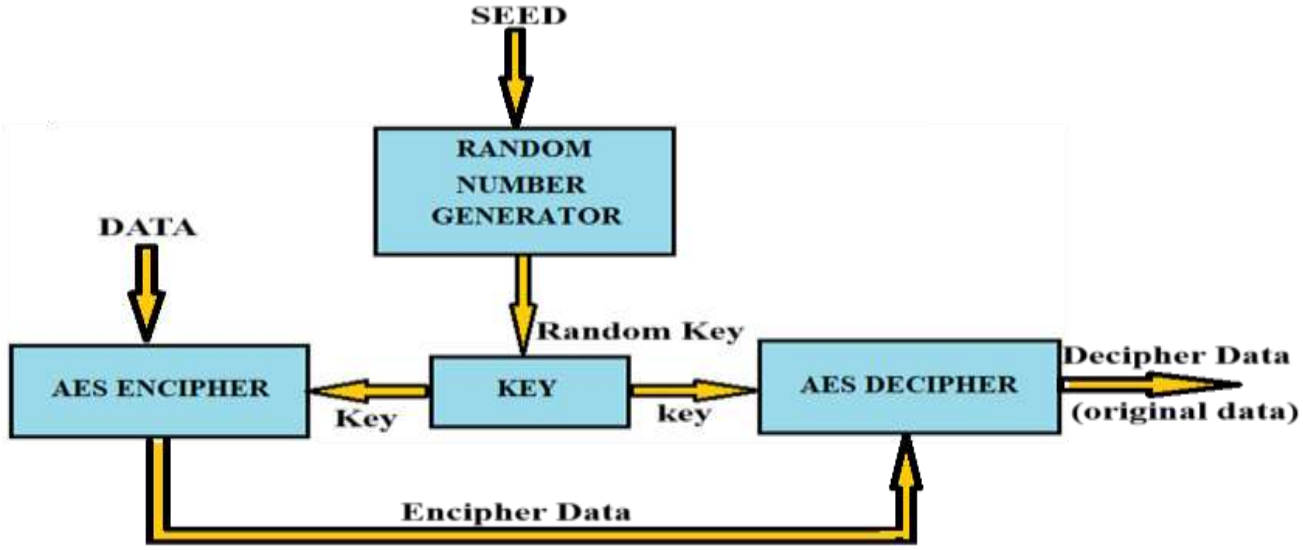


[FIGURE: AES Key Expansion]

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

[FIGURE: Round Constant Rcon[j] Table]

In AES Algorithm is working with one key, to improving the security of the design here we are adding one module for supplying key randomly, for applying random key we are taking random number generator module in this project.



[FIGURE: Block diagram proposed design]

Chapter 4: Proposed LFSR-Based AES Methodology

We present the methodology for implementing **AES encryption using Linear Feedback Shift Registers (LFSR)**. The methodology focuses on integrating LFSRs into the AES encryption process to enhance key generation and improve hardware efficiency. The approach involves modifying the **key expansion** process of AES by utilizing an **LFSR-based** key generation system. The process is designed to maintain the security and efficiency of AES while optimizing performance in terms of speed, area, and power consumption, especially for embedded systems and hardware-based implementations such as **FPGA** and **ASIC**.

The methodology can be broken down into the following steps:

4.1 AES Overview and Key Expansion

AES (Advanced Encryption Standard) is a symmetric encryption algorithm that operates on blocks of data (128 bits) using a **secret key** of 128, 192, or 256 bits. The encryption process involves several rounds of transformations, including substitution, permutation, and key addition. The key expansion phase generates round keys from the

original key to be used in each round of the encryption process. Traditionally, key expansion involves the use of **S-boxes** and **Rcon** values, which may increase the complexity of hardware implementations.

In our methodology, we propose the use of an **LFSR-based key generation** mechanism to replace traditional key expansion methods. The LFSR is used to generate pseudo-random sequences, which are then used to expand the original key, providing a faster and more efficient key generation process.

4.2 LFSR-Based Key Generation

The main goal of this step is to utilize an **LFSR** to generate key material for AES encryption. An LFSR is a shift register with feedback that produces a pseudo-random sequence of bits. The LFSR is characterized by its **feedback polynomial** and **initial state**, which determine the sequence of bits it generates.

The key steps involved in LFSR-based key generation for AES are:

- **LFSR Design:**
    - Select an appropriate **feedback polynomial** for the LFSR. Common feedback polynomials used in cryptography are **primitive polynomials**, which ensure that the LFSR produces a maximum-length sequence. For AES key expansion, the LFSR needs to be configured to generate sufficient randomness in the key stream.
    - Choose the **initial state** of the LFSR. The initial state can be derived from the original AES key or another secure source.
- **Key Stream Generation:**
    - The LFSR is clocked to produce a **key stream** that will be used for key expansion. This key stream is combined with the original AES key (or intermediate round keys) to generate the round keys required for the AES rounds.
- **Feedback Mechanism:**
    - The output of the LFSR is XORed with the input to generate the new state. This feedback ensures that each bit in the sequence depends on previous bits, providing the necessary randomness for cryptographic security.
- **Round Key Generation:**
    - The output key stream from the LFSR is combined with the original AES key to produce the round keys. The round keys are used in each round of AES for the AddRoundKey step.

4.3 Integration of LFSR into AES

In this step, we integrate the **LFSR-based key generation** with the AES encryption algorithm. This integration involves replacing the **traditional key expansion** process with the LFSR-based method while ensuring that all other AES operations (SubBytes, ShiftRows, MixColumns, and AddRoundKey) remain intact.

The key steps for integration include:

- **Initialization:**
    - The original AES key is loaded into the LFSR. The **initial state** of the LFSR is set based on the AES key or a derived value. The LFSR will then generate key streams to expand the key for each round of AES encryption.
- **Modified Key Expansion:**

- The LFSR generates a sequence of bits, which is used to derive the round keys. These round keys are then used in the **AddRoundKey** step of AES, replacing the traditional key expansion method.

- **AES Rounds:**
  - The standard AES rounds (SubBytes, ShiftRows, MixColumns, AddRoundKey) are applied using the round keys generated from the LFSR. The process continues for the specified number of rounds (10 rounds for 128-bit AES).

Chapter 5: Implementation Methodology

5.1 Hardware Implementation Platform

The proposed AES encryption system with LFSR-based key generation was implemented using **Verilog HDL**. The design was synthesized and simulated using **Xilinx Vivado**. The target device was **Xilinx Virtex-7 (xc7vx485tffg1157-1)** FPGA.

5.2 Top-Level Module Ports

Port Name	Width	Direction	Description
clk_a	1 bit	Input	Master clock signal
clk_l	1 bit	Input	Master clock signal
rst	1 bit	Input	Active low reset
data_in	128 bits	Input	Input data to be encrypted
aes_encout	128 bits	Output	Encrypted output data
aes_decout	128 bits	Output	decrypted output data

5.3 LFSR Module Design

A 128-bit Linear Feedback Shift Register (LFSR) was designed to autonomously generate pseudo-random keys for the AES encryption process. Unlike conventional AES implementations that require a user-provided key, this design uses an LFSR with a built-in initial seed to generate all round keys internally. The LFSR utilizes a primitive feedback polynomial to ensure maximum-length sequence generation. The feedback polynomial used in this implementation is:

$$P(x) = x^{128} + x^7 + x^2 + x + 1$$

The LFSR operates as follows:

- A fixed initial seed (hardcoded in the design) loads automatically upon reset

- On each clock cycle, the register shifts right by one bit
- The feedback bit is computed as XOR of selected tap bits
- The output is a 128-bit pseudo-random key
- No external user key is required - the LFSR generates all keys independently

5.5 Integration of LFSR with AES

The LFSR module is integrated with the AES encryption process as follows:

1. Upon reset, the LFSR initializes with a **hardcoded seed value**
2. For each encryption round, the LFSR **autonomously generates** a 128-bit round key
3. The generated round key is directly used in the AddRoundKey transformation
4. The LFSR shifts to the next state for the subsequent round
5. This process repeats for all 10 rounds of AES encryption

Key advantage: No user key input is required. The LFSR independently generates all cryptographic keys, making the system fully self-contained and suitable for embedded applications where key management is a challenge

5.6 Control FSM

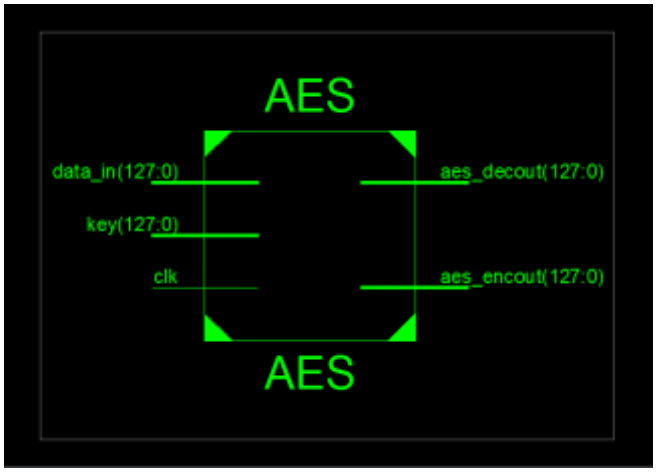
State	Operation
INIT_LFSR	Initialize LFSR with hardcoded seed
ROUND_1 to ROUND_9	Generate LFSR key → SubBytes → ShiftRows → MixColumns → AddRoundKey
FINAL_ROUND	Generate final LFSR key → SubBytes → ShiftRows → AddRoundKey
DONE	Assert encrypt_done and output ciphertext

Chapter 6: Results

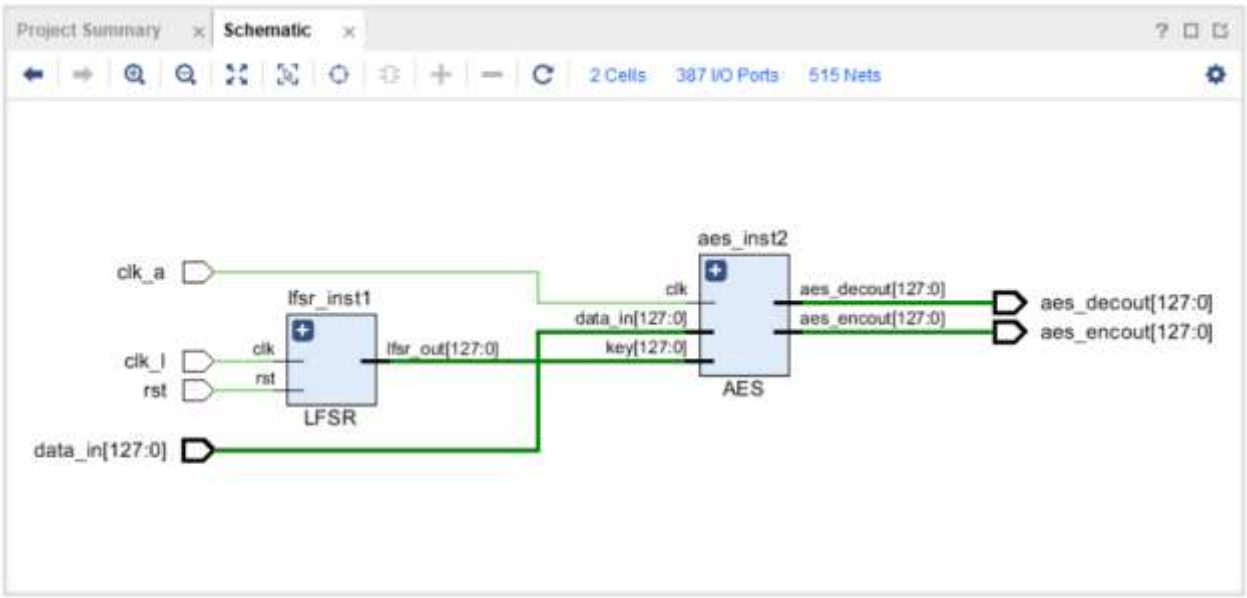
RTL Schematic

The RTL schematic is abbreviated as the register transfer level it denotes the blue print of the architecture and is used to verify the designed architecture to the ideal architecture that we are in need of development. The HDL language is used to convert the description or summary of the architecture to the working summary by use of the coding language

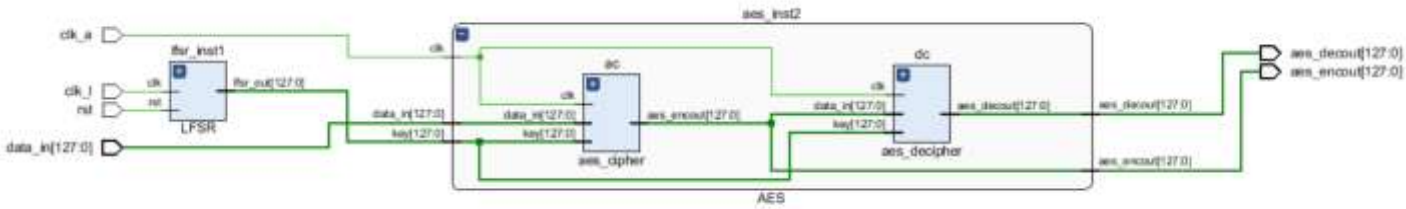
i.e verilog, vhdl. The RTL schematic even specifies the internal connection blocks for better analyzing. The figure represented below shows the RTL schematic diagram of the designed architecture.



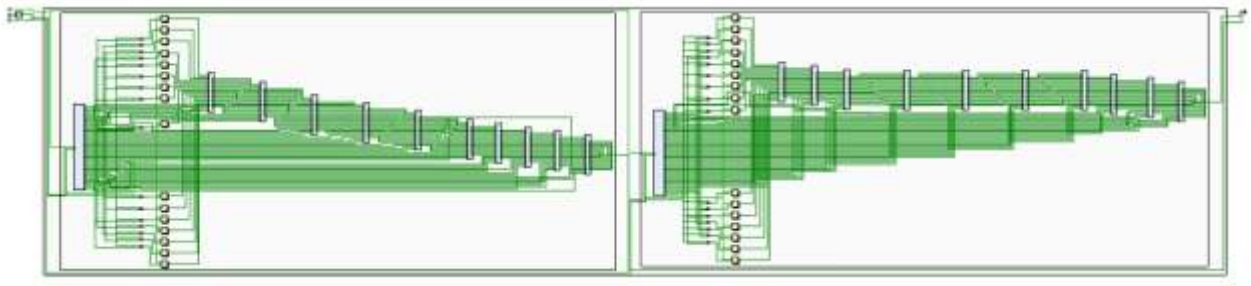
[FIGURE: RTL Schematic of AES]



[FIGURE: RTL Schematic1 of AES with LFSR]



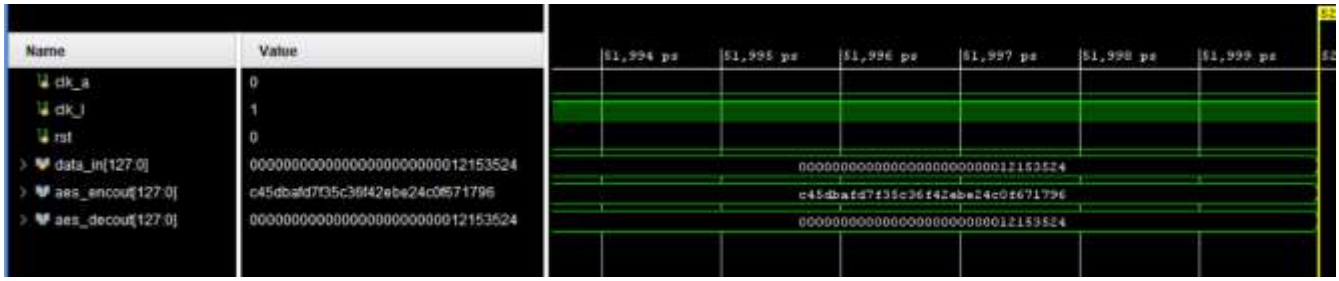
[FIGURE: RTL Schematic2 of AES with LFSR]



[FIGURE: RTL Schematic2 of AES with LFSR - continued]

Simulation

The simulation is the process which is termed as the final verification in respect to its working where as the schematic is the verification of the connections and blocks. The simulation window is launched as shifting from implantation to the simulation on the home screen of the tool, and the simulation window confines the output in the form of the wave forms. Here it has the flexibility of providing the different radix number systems.

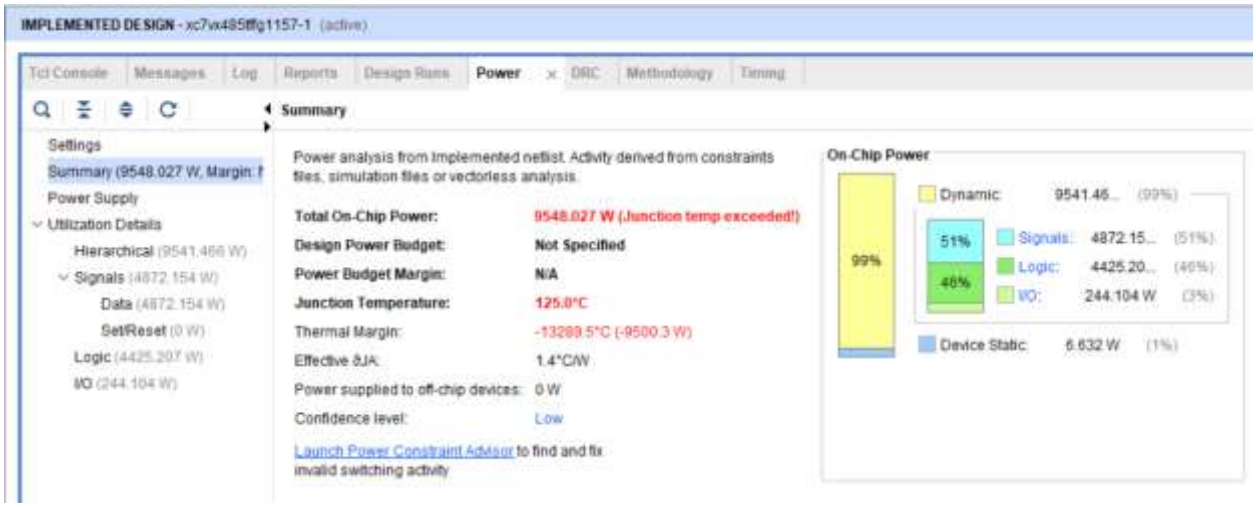


[FIGURE: Simulated Waveforms of AES with LFSR]

Parameters

Consider in VLSI the parameters treated are area, delay and power, based on these parameters one can judge the one architecture to other. Here the consideration of delay is considered the parameter is obtained by using the tool XILINX 14.7 and the HDL language is verilog language.

[TABLE: Hardware utilization]



[TABLE: Power report of design]

Chapter 7: Advantages and Applications

Advantages of AES Using LFSR-Based Key Generation:

1. **Improved Performance:** Faster Key Expansion and higher throughput using LFSR-based generation.
2. **Lower Hardware Complexity:** Area efficiency and simpler design for FPGA and ASIC implementations.
3. **Reduced Power Consumption:** Fewer gates and logic operations compared to traditional key expansion.
4. **Scalability and Flexibility:** Easy adaptation to different key sizes (128, 192, 256 bits).
5. **Hardware Optimization:** Optimized for low-power and high-speed operations in resource-constrained environments.
6. **Enhanced Security:** Maintains AES security while improving performance.
7. **Real-Time Cryptographic Applications:** Suitable for secure communications and data encryption in fast-paced environments.

Applications:

1. **Embedded Systems:** Ideal for microcontrollers and low-power devices.
2. **Internet of Things (IoT):** Secure communication for resource-constrained IoT devices.
3. **Mobile Platforms:** Reduced battery drain for smartphones and mobile devices.
4. **Wireless Communications:** Secure data transmission in Wi-Fi, Bluetooth, and Zigbee.
5. **FPGA and ASIC Implementations:** Hardware acceleration for cryptographic operations.
6. **Secure Data Storage:** Encrypted flash drives, SSDs, and cloud storage.
7. **Real-Time Cryptography:** Video streaming, VoIP, and financial transactions.
8. **Digital Payment Systems:** Secure payment and banking applications.
9. **Network Security:** VPNs, firewalls, and secure networks.

Chapter 8: Conclusion and Future Work

Conclusion

In this project, we have given 128 bits input and 128 bits security key and observed how it is delivered at the output with security. The integration of **Linear Feedback Shift Registers (LFSRs)** with **AES encryption** represents an innovative approach to enhancing the performance and efficiency of cryptographic systems. By replacing the traditional key expansion method with an **LFSR-based key generation**, this method significantly improves the **speed** of key generation, reduces **hardware complexity**, and lowers **power consumption**---making it an ideal solution for resource-constrained environments, such as **embedded systems**, **IoT devices**, and **mobile platforms**.

In hardware implementations, particularly for **FPGA** and **ASIC** designs, the use of LFSRs optimizes both **area** and **power** efficiency without compromising the security of the AES algorithm. The LFSR-based key generation also enables higher **throughput** and **real-time encryption** capabilities, ensuring robust protection for sensitive data while meeting the demands of fast-paced applications.

Furthermore, this integration maintains the **cryptographic strength** of AES while reducing latency, making it suitable for **secure communications**, **wireless protocols**, and **digital payments**.

In conclusion, **AES with LFSR-based key generation** offers a promising approach to achieve **high-performance**, **low-power**, and **secure encryption** for a wide range of applications, making it a viable option for modern cryptographic solutions in hardware and embedded systems.

Future Scope

- **Enhanced Security Techniques:** Further research can explore combining LFSRs with nonlinear feedback and chaotic systems to improve security.
- **Integration with Other Algorithms:** Hybrid systems with RSA or ECC for more secure applications.
- **Quantum-Resistant Cryptography:** Development of quantum-resistant LFSR-based encryption methods.
- **Improved Hardware Optimization:** More efficient FPGA/ASIC designs for high-speed networks and mobile devices.
- **Machine Learning Integration:** Adaptive, self-learning cryptographic systems that dynamically adjust encryption parameters.
- **Low-Power IoT Networks:** Extended application to sensor networks where security and battery life are critical.
- **Standardization and Adoption:** Potential incorporation into international cryptographic standards.

References

[1] Madakam, Somayya, R. Ramaswamy, and Siddharth Tripathi. "Internet of Things (IoT): A literature review." Journal of Computer and Communications 3, no. 05 (2015): p.164.

[2] Wang, Yong, Garhan Attebury, and Byrav Ramamurthy. "A survey of security issues in wireless sensor networks." IEEE Communications Surveys Tutorial (2006).

[3] Veeramallu, B., S. Sahitya, and Ch Lavanya Susanna. "Confidentiality in Wireless sensor Networks." International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, January 2013.

[4] Eisenbarth, Thomas, and Sandeep Kumar. "A survey of lightweight cryptography implementations." IEEE Design & Test of Computers 24.6 (2007).

[5] Banik, Subhadeep, Andrey Bogdanov, and Francesco Regazzoni. "Exploring energy efficiency of lightweight block ciphers." International Conference on Selected Areas in Cryptography. Springer, Cham, 2015.

[6] Bogdanov, Andrey, et al. "PRESENT: An ultra-lightweight block cipher." CHES. Vol. 4727. 2007.

[7] Daemen, Joan and Rijmen, Vincent. "The design of Rijndael: AES-the advanced encryption standard.", Springer Science & Business Media, 2013.

[8] Al Hasib, Abdullah, and Abul Ahsan Md Mahmudul Haque. "A comparative study of the performance and security issues of AES and RSA cryptography." Third International Conference on Convergence and Hybrid Information Technology, 2008. Vol.2.

[9] Feldhofer, Martin, Johannes Wolkerstorfer, and Vincent Rijmen. "AES implementation on a grain of sand." IEE Proceedings-Information Security 152, no. 1 (2005): p.13-20.

[10] Moradi, Amir, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. "Pushing the limits: a very compact and a threshold implementation of AES." In Eurocrypt, vol. 6632, pp. 69-88. 2011.

[11] Kerckhof, Stphanie, Franois Durvaux, Cdric Hocquet, David Bol, and Franois-Xavier Standaert. "Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint." Cryptographic Hardware and Embedded Systems CHES 2012 (2012): p.390-407.

[12] Batina, Lejla, et al. "Dietary recommendations for lightweight block ciphers: power, energy and area analysis of recently developed architectures." International Workshop on Radio Frequency Identification: Security and Privacy Issues. Springer, Berlin, Heidelberg, 2013.

[13] Kong, Jia Hao, Li-Minn Ang, and Kah Phooi Seng. "A comprehensive survey of modern symmetric cryptographic solutions for resource constrained environments." Journal of Network and Computer Applications 49 (2015): p.15-50.

[14] Wenceslao Jr, Felicisimo V., et al. "Modified AES Algorithm Using Multiple S-Boxes." The Second International Conference on Electrical, Electronics, Computer Engineering and their Applications (EECEA2015). 2015.

[15] Kawle, Pravin, et al. "Modified Advanced Encryption Standard." International Journal of Soft Computing and Engineering (IJSCE) 4 (2014).