Journal of Electronic Testing (2022) 38:181–193
https://doi.org/10.1007/s10836-022-05997-x

Low Area FPGA Implementation of AES Architecture with EPRNG for IoT Application

N. Siva Balan[1] · **B. S. Murugan**[2]

Received: 28 August 2021 / Accepted: 18 March 2022 / Published online: 6 April 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
Nowadays, the Internet of Things (IoT) is widely used in the daily lives of humans, which range from tiny wearable devices to huge industrial systems. However, designing the IoT application is difficult, because the devices in the IoT network are susceptible to security threats (e.g. malicious attacks). Therefore, an effective cryptographic process must be developed with a minimum amount of hardware resources. In this paper, an optimized Advanced Encryption Standard (AES) architecture is proposed to improve the security between the IoT devices. The following key strategies are involved in the proposed AES architecture: 1) Efficient Pseudo Random Number Generator (EPRNG) using the two-level True Random Number Generator based key generation module is used to generate a different optimal key value for each clock cycle, 2) the number of logical elements used in the AES architecture is minimized because there is no registers are required for storing the generated keys as it is automatic key generation. The performances of the EPRNG-AES architecture are analyzed in terms of the number of slice registers, flip flops, number of slice Look Up Table (LUT), number of logical elements, slices, bonded Input/ Output Block (IOB), power, delay, and operating frequency. The EPRNG-AES architecture is evaluated with five different AES architectures such as AES-PNSG, LAES, AES-HLS, AES-CTR and AES-MMC. The EPRNG-AES architecture designed in the Kintex 7 uses 153 slices, which is less when compared with the number of slices in LAES and AES-HLS.

Keywords Advanced encryption standard algorithm · Efficient pseudo random number generator · Internet of things · Security threats · Two-level true random number generator

1 Introduction

In recent decades, Internet of Things (IoT) has become a popular technology, which interconnects the user with the physical world to provide an efficient and dynamic platform for communication [3, 7, 12]. The physical objects are integrated with various types of sensors which accomplish various tasks such as sensing of environment, data processing, and data communication to different destinations through internet [18]. There are many applications in the IoT technology such as smart cities, supply chain, digital health monitors, and industrial control. IoT technologies adopt different types of devices and systems, such as smartphones, sensors (i.e., wireless sensor networks), near field communication mobiles, radio frequency identification tags, and actuators [5, 17]. The IoT device requires adequate memory to send and receive the sensor data using a smart device [10].

IoT technology comprises of several interconnected objects that are highly susceptible to malicious attacks and eavesdropping. If a malicious attack occurs in the IoT system, it may affect the functions of the whole network, which might result in a risky situation e.g., spoofing the data [1, 22]. IoT involves certain key functions such as global standard, object identification, integration, ownership, trust, regulation, security, and privacy [6, 8]. To overcome the security issues in IoT, the cryptographic algorithm needs to be implemented. The cryptographic schemes are hardware independent, generic, and provide a higher level of robustness to the

Responsible Editor: S. Bhunia

✉ N. Siva Balan
Balan.mtech@outlook.com; Balan.mtech@gmail.com

B. S. Murugan
muruganbs@gmail.com

[1] Research Scholar, Department of CSE, Kalasalingam Academy of Research and Education, Srivilliputtur, Tamil Nadu 626128, India

[2] Department of CSE, Kalasalingam Academy of Research and Education, Srivilliputtur, Tamil Nadu 626128, India

IoT devices [11]. Moreover, the cryptographic algorithm provides a well-built security mechanism and encryption/decryption operations with low power consumption [21]. The AES is one of the frequently used cryptographic algorithms, which gives better security and is cost-efficient. The AES has a simple implementation process both in software and in hardware [9, 23]. In this research work, optimal key generation and key expansion design will be used to design the AES with high security.

The important contributions of this research paper are given as follows:

- The EPRNG-AES architecture processes the 128-bits of input, key, and ciphertext values. Here, the EPRNG with two-level TRNG module generates different key values for each clock cycle, which provides the security for the response values. These secured response values improve the security between the IoT device and server.
- Accordingly, different key values and response values create higher dissimilarity between the encrypted values, which increases the robustness against unauthorized users. Hence, the unauthorized users cannot access the data transmitted among the IoT devices and server.
- The proposed EPRNG-AES architecture is analyzed in four different Field Programmable Gate Array (FPGA) devices, which are Kintex 7, Virtex 5, Virtex 6 and Spartan 6. Moreover, this EPRNG-AES architecture is analyzed under different security constraints such as Side-Channel Attacks (SCA), Denial of Service (DoS), Offline Password Guessing Attack (OPGA), Session Key Agreement (SKA), and validity of the strict key.

The overall organization of the paper is given as follows: Sect. 2 describes the related works done in the cryptographic algorithms and AES architectures. The problem statement found from the related work is stated in Sect. 3. Section 4 provides a clear explanation of the EPRNG-AES architecture. The results and discussion of the EPRNG-AES architecture are provided in Sect. 5. Finally, the conclusion is made in Sect 6.

2 Related Work

The related works about the recent techniques used in the cryptographic process and AES architecture are described in this section.

Lara et al. [14] developed a Lightweight Authentication and Key Distribution (LAKD) for accomplishing machine-to-machine communication. The LAKD mainly depended on lightweight operations such as subtraction, addition, and XOR. This LAKD had two stages that were: registration and mutual authentication. In that, the sensor node and gateway, exchanged the secret in the registration phase, and also exchanged the session key, which was generated in the mutual authentication phase. The designed LAKD was offered high security, but it is vulnerable to side-channel attacks for larger networks.

Megouache et al. [16] presented a secure multi-cloud architecture for improving the integrity and confidentiality of the data. The developed secure cloud has three different techniques, which are virtual private network (VPN), RSA, and hashing algorithm. Initially, the VPN was created between the provider and customer while RSA was used to provide the data security. Furthermore, the hashing function was used to provide the data integrity, which guaranteed the information recovery from the clouds. Here, the VPN was used to reduce the risk of data loss during the communication. However, the developed multi-cloud architecture was susceptible to DDoS attacks.

Chen et al. [4] developed the Secured Mutual Authentication Protocol (SMAP) for edge-based smart grid communication. This SMAP utilized XOR computations, one-way hash functions, and Elliptic Curve Cryptosystem (ECC), to accomplish the data security. Moreover, the Burrows-Abadi-Needham (BAN) logic was used to verify the security of the data. The smart meter registration phase, edge node registration phase, as well as login and authentication phases that were used in this SMAP consumed less amount of time and were more resistant to the attacks. However, the collision occurred while processing large volume of data in the SMAP.

Zodpe and Sapkal [24] presented the PN Sequence Generator (PNSG) to create the S-box. The PSNG was used to generate a distinct sequence of random numbers by using the initial seed value and feedback taps. The characteristics of the PNSG were used to enhance the efficiency of the cryptosystem. The AES doesn't require any external key, because it already has an initial key generation method. But, the design of AES that used nonpipelined stages, consumed high hardware resources during implementation.

Kumar et al. [13] developed the Lightweight AES (LAES) algorithm for securing voice data, and this LAES was designed in Kintex-7 and Artix-7 FPGAs. The MixColumns operation was reduced in the LAES algorithm which provided less delay. Moreover, the less MixColumns operation also minimized the logic operation. Hence, the LAES's complexity was reduced during the voice data encryption. But, the usage of multipliers was increased by decreasing the MixColumns function.

Arul Murugan et al. [2] designed the iterative structure of AES-128 encryption to improve security. In S-box, the multiplication was accomplished by using the LUT in the Composite Field Arithmetic (CFA). The hardware resources were minimized by using the Vedic multiplier in the MixColumns function. The sub-byte transformation was rapidly

performed by using the CFA. However, the design of MixColumns transformation has higher complexity than the shift-row transformation.

Sikka et al. [20] developed the high throughput FPGA design of the AES for automotive applications. The AES with a 128-bit key and block size of 10 was designed by using the Vivado High-Level Synthesis (HLS) tool. The HLS depended on the application-specific bit widths utilized for designing the FPGA. Moreover, the speed of the AES was improved using the HLS. But, the regular recalculation of the signal width resulted in excess delay in AES.

Shahbazi and Ko [19] designed the AES algorithm in counter (CTR) mode for traffic applications. The following two approaches were used to modify the AES, i) shift-Rows & sub-bytes were exchanged in the first 9 rounds, and ii) the shift rows were combined with add round key. The higher throughput was obtained by using the inner pipelining, outer pipelining and loop-unrolling approaches. However, the design of MixColumns operation over the clock cycle was delivered a high latency.

Madhavapandian and MaruthuPandi [15] developed the Modified Mix Column (MMC) with gate replacement to design a compact structure of AES for transmission control protocol/internet protocol. Here, the AES was developed with the utilization of efficient mix column Boolean expression along with resource sharing architecture. The power of the AES architecture was minimized by optimizing the overall structure. But, the time complexity was required to be minimized, because it caused high delay during the communication.

3 Problem Statement

The problems found from the related work along with the solutions given by the proposed architecture are stated in this section.

Collision occurs in the SMAP [4] when it processes a huge amount of data during the communication. The data loss occurs in the communication system when the collision occurs between the devices. The design of AES architecture using the nonpipelined stages leads to higher hardware utilization [24]. Moreover, the usage of multiplexers is increased along with the reduction in the MixColumns operation [13]. The frequent recalculation of the signal width increases the delay that minimizes the operating frequency [20].

Solution: In this EPRNG-AES architecture, the security between the IoT device and server is improved by generating different key values in each clock cycle. The bit-by-bit process of the AES architecture avoids collision during the communication process. Additionally, the optimized structure of AES is used to minimize the hardware utilization while designing the EPRNG-AES architecture. The EPRNG-AES architecture doesn't require any registration to store the intermediate key values as they are simultaneously generated and processed within the circuit itself. Hence, the use of hardware resources is minimized for the EPRNG-AES architecture.

4 EPRNG-AES Architecture

In the EPRNG-AES architecture, the security of the IoT devices are improved by using optimal key generation. The response value generated for the IoT device is secured by using the modified AES architecture. Moreover, this response value is different for each cycle whereas the challenge value is the same for all clock cycles. Further, the EPRNG-AES based key generation provides a different key for each clock cycle. Hence, the ciphertexts are highly different from each other, which improves the security of the communication established through the IoT devices. On the other hand, the hardware utilization of the EPRNG-AES architecture is minimized by optimizing the AES architecture. Figure 1 shows the process of securing IoT communication using EPRNG-AES.

4.1 Setup Phase

The IoT device sends the Identity (ID) and requests, to the server while initializing the setup phase. Next, the challenge is randomly generated by the server, once the request is received in the server. The generated challenges are used for performing the communication with the IoT device. Next, the response that is generated by using the IoT device, is secured by using the AES algorithm.

Subsequently, the response values generated by the IoT device are stored in the server, which are represented as $R1, R2, \ldots Rn$. Further, the server generates an Alias Identity

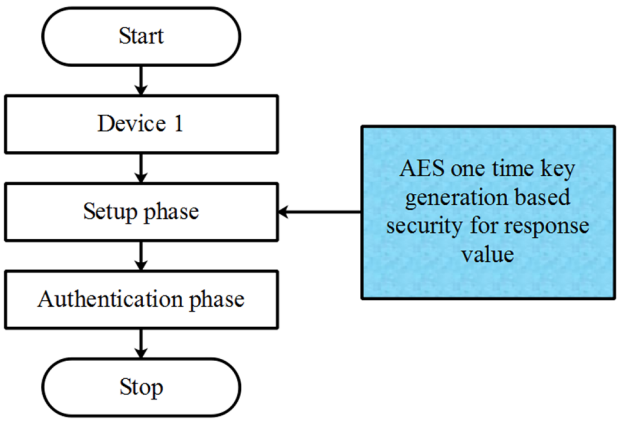

Fig. 1 Process of securing IoT communication using EPRNG-AES

(AID), Master Key (MK), Fake Identity (FI), and synchronization Key (SK), which depends on the response value. The generated MK, FI, and SK are saved in the IoT devices.

Equation (1) shows the generation of AID by using the MK and response value.

$$AID = h(R||MK) \tag{1}$$

where, the h defines the one-way hash function; R is the response value and MK is the master key of the server.

Next, the unique fake identity and pairs of synchronization keys are created by the server as expressed in Eq. (2).

$$(FD,\ SK) = \{(fid_1,\ k_1), (fid_2,\ k_2), \ldots, (fid_n,\ k_n)\} \tag{2}$$

where the fake identities are represented as $fid_1, fid_2, \ldots, fid_n$ and synchronization keys are represented as $k_1, k_2, \ldots, k_n$ and the amount of IoT devices considered during the communication is n.

4.1.1 AES Algorithm

The AES algorithm is operated with the 128-bit of plaintext and it utilizes the identical key to accomplish the encryption and decryption processes. This algorithm is processed on the data block, which contains a 4×4 byte matrix (i.e., State). The fundamental processes of the AES are conducted through the state. Figure 2 shows the operations performed by the AES encryption process [24]. In general, the AES-128 algorithm is separated into three stages as the addition of the initial round key, rounds 1–9 and the final round. The plaintext is Exclusive-ORed with the initial key at the first round. In a two dimensional 4 4×4 bytes, the transformations of the SubBytes(), ShiftRows(), MixColumns() and AddRoundKeys() are performed for each cipher round. Further, the operations of the SubBytes(), ShiftRows(), and AddRoundKeys() are accomplished in the states at the final round.

4.1.1.1 SubBytes Transformation In this phase, the nonlinear transformation is performed over the individual byte of the input state. Each byte from the state matrix is substituted with the value saved in the S-box. This SubBytes transformation creates an effective robustness against the attacks. The multiplicative inverse is taken in the finite field GF(2^8) is used to compute the values of the S-box. In finite field, the input element with bits equal to zero is mapped and the affine transformation is applied on the GF(2). Equation (3) expresses the multiplicative inverse at finite field GF(2^8) and the affine transformation on the GF(2) is shown in Eq. (4).

$$S(y) = Affine\ transformation\ (y^{-1}) \tag{3}$$

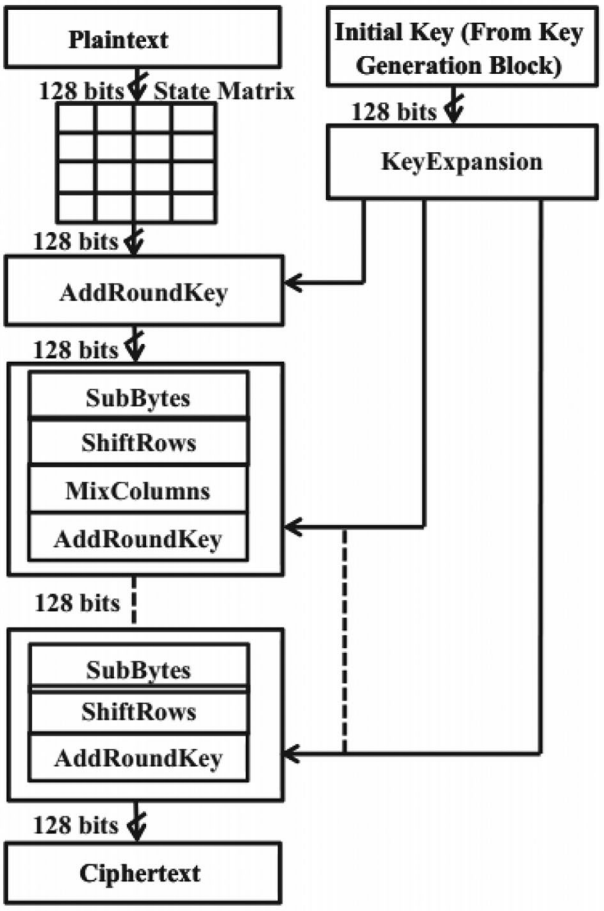

Fig. 2 Plaintext encryption using AES

$$Affine\ transformation = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{4}$$

4.1.1.2 ShiftRows Transformation The row positions 1, 2 and 3 of the state matrix are shifted periodically towards the left positions by 1, 2 and 3 respectively. Here, the number of rows defines the offset value. Accordingly, the 1st row remains unchanged in the state matrix. Hence, the diffusion property is created in the AES by using the cyclic rotation of rows. Figure 3 shows the illustration of the ShiftRows transformation [24].

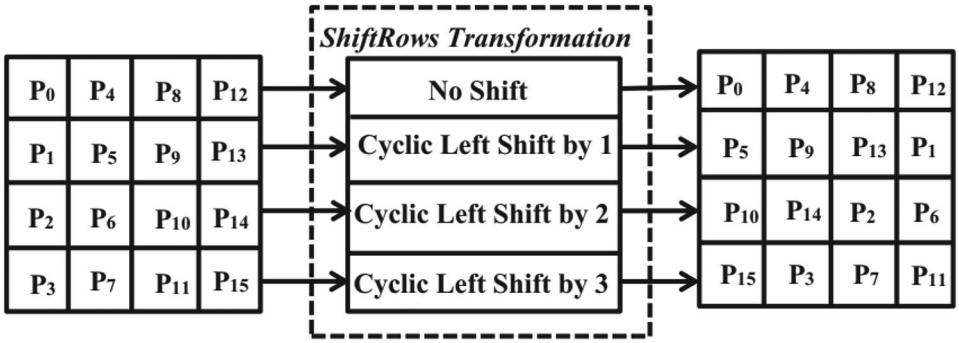

Fig. 3 ShiftRows transformation

4.1.1.3 MixColumns Transformation In this phase, the transformation of MixColumns is conducted in each column of the state matrix. This MixColumns operation is a linear diffusion process and it is performed in each column individually. Moreover, each column of the state matrix is assumed as a four-term polynomial on GF(2^8). Next, the modulo ($y^4 + 1$) is used to multiply the column value with a polynomial constant $a(y)$ as shown in Eq. (5).

$$a(y) = \{03\}y^3 + \{01\}y^2 + \{01\}y^1 + \{02\} \tag{5}$$

The matrix multiplication of Eq. (5) is shown in Eq. (6) and the matrix representation is shown in Eq. (7).

$$p'(y) = a(y) \times p(y) \tag{6}$$

$$\begin{bmatrix} P'_{0,c} \\ P'_{1,c} \\ P'_{2,c} \\ P'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} P_{0,c} \\ P_{1,c} \\ P_{2,c} \\ P_{3,c} \end{bmatrix} \tag{7}$$

4.1.1.4 AddRoundKey Transformation In AES algorithm, the transformation of the AddRoundKey is the the final transformation for each round. In this phase, the obtained round key is XORed with the state based on the bitwise operation. Here, the XOR operation is performed between the *Nb* words obtained from the key schedule of each Round Key and columns of the state value. This XOR operation is shown in the Eq. (8).

$$[P'_{0,c}, P'_{1,c}, P'_{2,c}, P'_{3,c}] = [P_{0,c}, P_{1,c}, P_{2,c}, P_{3,c}] \oplus [W_{round*Nb+C}] \tag{8}$$

where, the words from key schedule are represented as $[W_{round}]$, *round* indicates the value between the range $0 \le round \le N_r$ and N_r defines the number of round.

4.2 Key Generation Using EPRNG-AES Architecture

In the EPRNG architecture, the two-level TRNG is used as input to generate an optimal key value of 128-bit for improving the security against the various threats. The overall key generation process uses the two-level TRNG and combinational blocks, so it is defined as Efficient Pseudo Random Number Generator (EPRNG). This EPRNG generates a different number (i.e., key) for each clock cycle. Here, the two levels of the TRNG modules are used to generate a random number, which is given as input for generating the optimal key. This developed EPRNG key generation module shown in the Fig. 4, is used to generate a 20-bit key value. Therefore, six EPRNG key generation modules are used to

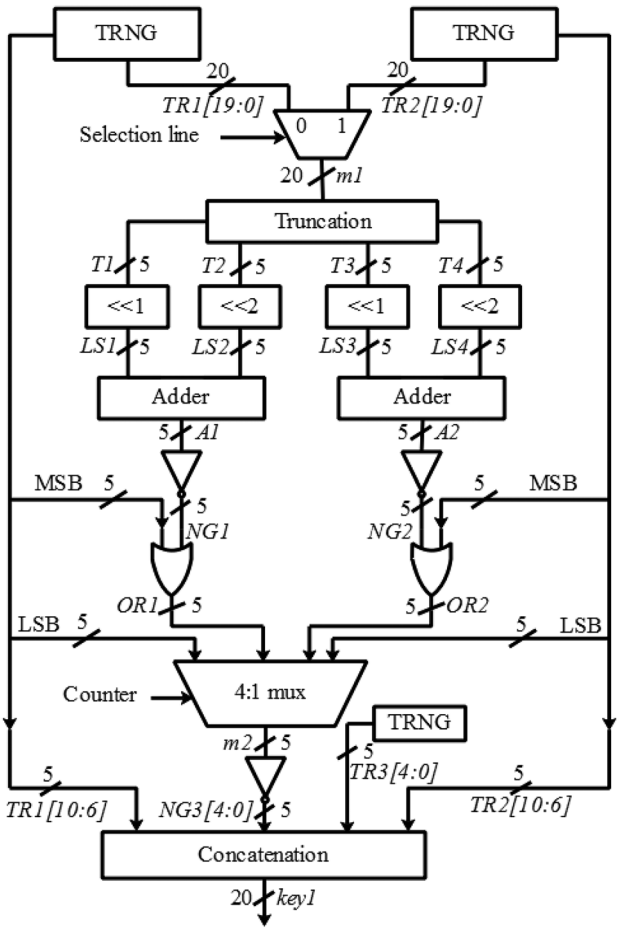

Fig. 4 The architecture of the EPRNG key generation module

Table 1 Output of 2:1 MUX

Selection line	Output
00	$TR1[19:0]$
01	$TR2[19:0]$

generate six 20-bit key values resulting in 120-bit key values. The value of zero is added in the MSB 8-bit for the remaining 8-bit values. Then, the key values are combined together {8-bit values of zero and 120-bit keys from the EPRNG key generation module}, which are used to accomplish the encryption/ decryption process.

For example, the generation of 20-bit key value i.e., $key1$ using the EPRNG key generation module is explained as follows:

1. At first, two TRNG modules are used to generate two 20-bit values, which are considered as input values for the key generation. The two 20-bit values from the TRNG are $TR1[19:0]$ and $TR2[19:0]$. Subsequently, these 20-bit values are given into the MUX, which provides the output based on the selection line as shown in Table 1.
2. The output of the MUX $m1 = TR1[19:0]$, when the selection line is 00. Otherwise, the MUX output is $m1 = TR2[19:0]$.
3. The MUX provides the 20-bit output value i.e., $m1[19:0]$ and it is processed under truncation process. This truncation divides the 20-bit input value into four 5-bits as shown in the Eq. (9).

$$T1 = m1[19:15]$$
$$T2 = m1[14:10]$$
$$T3 = m1[9:5]$$
$$T4 = m1[5:0] \quad (9)$$

where, $T1$, $T2$, $T3$ and $T4$ are the four 5-bit truncated values obtained from the MUX output.

4. Next, the four 5-bit truncated values ($T1$, $T2$, $T3$ and $T4$) are processed under left shift operation. There are two left shift operations that are performed over the $T1$, $T2$, $T3$ and $T4$ values, which are $\ll 1$ and $\ll 2$. Here, the values of $T1$ & $T3$ are processed over the $\ll 1$ left shift operation and the values of $T2$ & $T4$ are processed over the $\ll 2$ left shift operation. This left shift operation over the truncated values is shown in the Eq. (10).

$$LS1 = T1 \times 2^1$$
$$LS2 = T2 \times 2^2$$
$$LS3 = T3 \times 2^1$$
$$LS4 = T4 \times 2^2 \quad (10)$$

where, the $LS1$, $LS2$, $LS3$, and $LS4$ are the four 5-bit values obtained from the left shift operation.

5. The pairs of $LS1 - LS2$ and $LS3 - LS4$ are added to each other by using the adder as shown in Eq. (11).

$$A1 = LS1 + LS2$$
$$A2 = LS3 + LS4 \quad (11)$$

where, $A1$ and $A2$ are the 5-bit output values of the adder.

6. The output values of the adders $A1$ and $A2$ are given as input to the NOT gate. In general, the logical negation is accomplished over the input by using the NOT gate. If the input is true, the NOT gate delivers the output as false. Similarly, the true output is obtained for the false

Table 2 Output of 4:1 MUX

Counter	Output
0	$TR1[4:0]$
1	$OR1$
2	$OR2$
3	$TR2[4:0]$

Table 3 Simulation parameters

Parameter	Values
Clock period	100 ns
Duty cycle	50 ns
Initial edge	Rise
Rst,enb,enb1,enb2	1
Input data, key, the output bit	128 bit
Total simulation run time	1200 ns

Table 4 Hardware utilization of EPRNG-AES architecture for Kintex 7 FPGA device

FPGA performances	Total resources	Occupied resources	% of utilization
Number of slice registers	407,600	46	1%
Flip Flops	407,600	39	1%
Number of slice LUTs	203,800	153	1%
Number of logical elements	203,800	143	1%
Slices	50,950	59	1%
Bonded IOB	400	26	6%

Table 5 Hardware utilization of EPRNG-AES architecture for Virtex 5 FPGA device

FPGA performances	Total resources	Occupied resources	% of utilization
Number of slice registers	28,800	38	1%
Flip Flops	28,800	38	1%
Number of slice LUTs	28,800	174	1%
Number of logical elements	28,800	166	1%
Slices	7200	62	1%
Bonded IOB	480	26	5%

input. The 5-bit output values from the NOT gate are represented in Eq. (12).

$$NG1 = \overline{A1}$$

$$NG2 = \overline{A2} \tag{12}$$

where, $NG1$ and $NG2$ are two 5-bit values obtained from the NOT gate.

7. The MSB values from the TRNG and NOT gate output are given to the OR gate to generate two 5-bit values. Here, the 5-bit MSB value is taken from the two TRNG modules as $TR1[19:15]$ and $TR2[19:15]$. The pairs of $TR1[19:15] - NG1$ and $TR2[19:15] - NG2$ are processed through the OR gate to generate two 5-bit values as shown in Eq. (13).

$$OR1 = TR1[19:15] + NG1[4:0]$$

$$OR2 = TR2[19:15] + NG2[4:0] \tag{13}$$

where, the $OR1$ and $OR2$ are the two 5-bit output values taken from the OR gate.

8. Next, this $OR1$ and $OR2$ are given as inputs to the 4:1 MUX along with two more input values, which are LSB values of the two TRNG modules. The 5-bit LSB values of the TRNG module are $TR1[4:0]$ and $TR2[4:0]$. From these four 5-bit values, a single 5-bit value is taken as output ($m2$) based on the counter. The output from the 4:1 MUX is taken as specified in Table 2.
9. Next, the 5-bit output from the 4:1 MUX ($m2$) is given as input to the NOT gate and it delivers the output ($NG3$) as shown in the Eq. (14).

$$NG3 = \overline{m2} \tag{14}$$

Moreover, one more TRNG is used in this key generation module to generate a 5-bit value ($TR3$).

10. Finally, there are four inputs are given to the concatenation process for generating the 20-bit key. The four inputs given to the concatenation process are $TR1[10:6]$, $NG3$, $TR3$ and $TR2[10:6]$. Equation (15) shows the 20-bit key generation of the key generation module.

$$key1 = \{TR1[10:6], NG3[4:0], TR3[4:0], TR2[10:6]\} \tag{15}$$

Similarly, the five 20-bit key values are generated from the five more EPRNG key generation modules. The 20-bit key values obtained from the five more EPRNG key generation modules are $key2$, $key3$, $key4$, $key5$ and $key6$. Equation (16) shows the concatenation of all key values and the 8-bit values of zero in MSB.

Table 6 Hardware utilization of EPRNG-AES architecture for Virtex 6 FPGA device

FPGA performances	Total resources	Occupied resources	% of utilization
Number of slice registers	7168	38	1%
Flip Flops	7168	37	1%
Number of slice LUTs	7168	229	3%
Number of logical elements	7168	224	3%
Slices	3584	122	3%
Bonded IOB	140	26	18%

Table 7 Hardware utilization of EPRNG-AES architecture for Spartan 6 FPGA device

FPGA performances	Total resources	Occupied resources	% of utilization
Number of slice registers	11,440	56	1%
Flip Flops	11,440	49	1%
Number of slice LUTs	5720	165	2%
Number of logical elements	9112	131	1%
Slices	1430	55	3%
Bonded IOB	102	26	25%

Table 8 Analysis of delay, power and operating frequency for EPRNG-AES

FPGA devices	Power (W)	Delay (ns)	Operating frequency (MHz)
Kintex 7	0.038	4.383	305.761
Virtex 5	0.010	1.094	658.288
Virtex 6	0.022	2.981	266.902
Spartan 6	0.014	1.986	301.231

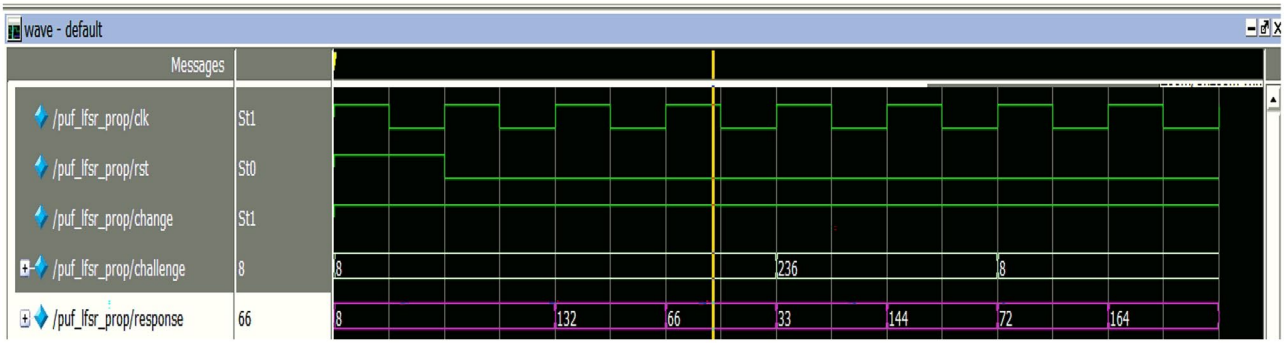

Fig. 5 Simulation waveform for response output

$$key = \{00000000,\ key1,\ key2,\ key3,\ key4,\ key5,\ key6\} \tag{16}$$

4.3 Authentication Phase

In this authentication phase, an authorization is provided to the IoT device, when both the server and device nonce are matched together. The random number request is used to verify the AID and the communication is established by transmitting the request message to the server. If the AID is matched in the authentication phase, the response value, challenge and master key are saved during the communication. Otherwise, the respective request is discarded through the IoT device. Next, the server generates the server nonce and hash key response. Hence, the authorization is provided to the IoT device by using the server, when the key hash function contains the server and device nonce values. After receiving the authentication, the data transmission is accomplished between the IoT device and server. Subsequently, the next IoT device (Device 2) also performs the same authentication process performed by Device 1. Accordingly, the setup and authentication process is performed for each IoT device.

In nutshell, the overall research has two main phases such as setup phase and authentication phase. The server receives the requests and ID of the IoT device in the setup phase. Subsequently, the challenge values are created to enable the communication the IoT device whereas the response from the IoT is secured using the EPRNG-AES architecture. The designed EPRNG provides a different key value for each clock cycle and these keys used to protect the response from the IoT device. In authentication phase, the server nonce and hash key response are generated by the server. The EPRNG-AES architecture doesn't required registers to store the generated keys which used to minimize the number of logical elements. Next, the server provides the authorization to the IoT device to enable the data broadcasting between the IoT device and server. Here, the security among the IoT device and server is improved based on the secured response values. Consequently, a different key and response values generates a dissimilarity between the encrypted values that maximizes the robustness against unauthorized users. Therefore, the designed EPRNG-AES architecture achieves higher security between the server and IoT device, while minimizing the logical elements of AES.

5 Results and Discussion

The proposed EPRNG-AES architecture has been implemented by using the Xilinx 14.4 software where the system is operated with 4 GB RAM and a 500 GB hard disk. This Xilinx software is used to analyze the hardware utilization of the EPRNG-AES architecture. Additionally, the Modelsim 10.5 software is used to analyze the EPRNG-AES architecture using the simulation waveform. The designed EPRNG-AES architecture is used to process the

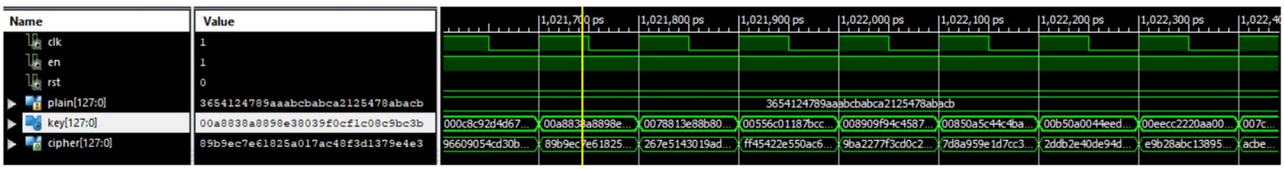

Fig. 6 Simulation waveform for AES

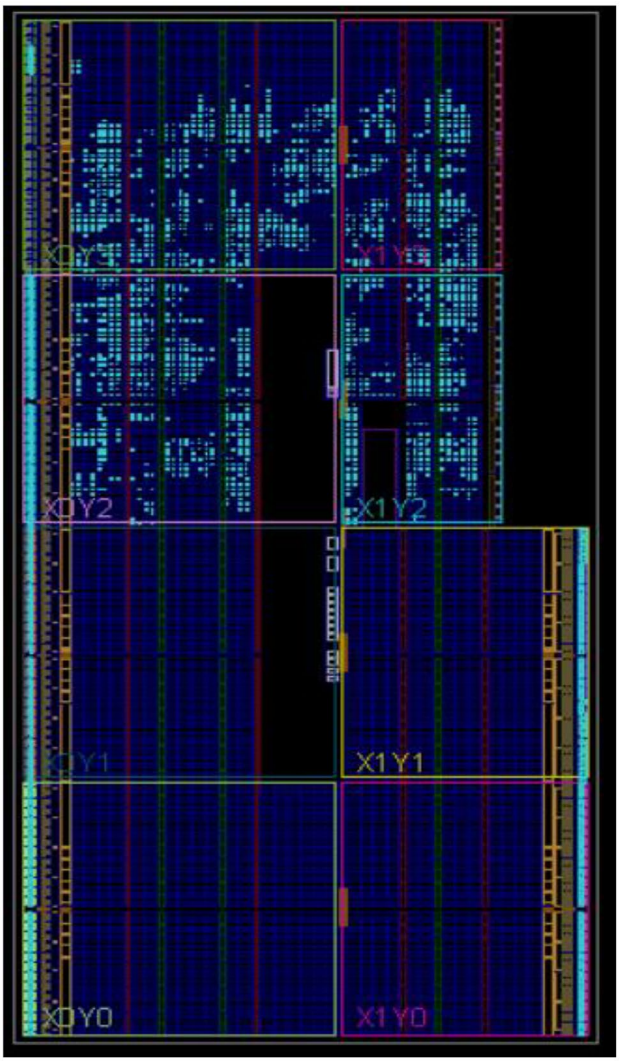

Fig. 7 Floorplanning design for EPRNG-AES architecture

128-bits of the input data, key and output data. The following Table 3 provides the simulation parameters considered for this EPRNG-AES architecture.

Table 9 Analysis of system estimation

Design	Security (%)	Complexity (%)	Cost (%)	Integrity (%)
LAKD [14]	88	8	13	88
RSA [16]	85	10	15	85
SMAP [4]	74	24	36	71
EPRNG-AES	93	5	10	92

5.1 Performance Analysis of EPRNG-AES Architecture

The performance of the EPRNG-AES architecture is analyzed by using four different FPGA devices namely Kintex 7, Virtex 5, Virtex 6 and Spartan 6. Here, the performances are analyzed in terms of the number of slice registers, flip flops, the number of slice LUTs, the number of logical elements, slices, bonded IOB, power, delay, and operating frequency. The following hardware utilization defines the number of logical elements used by AES architecture.

The analysis of hardware resource utilization for the EPRNG-AES architecture, designed in the Kintex 7, Virtex 5, Virtex 6 and Spartan 6 FPGA devices, are provided in the Tables 4, 5, 6, and 7 respectively. Moreover, the examination of the power, delay, and operating frequency for EPRNG-AES architecture, is given in Table 8. The hardware utilization is analyzed for the EPRNG-AES architecture with the 128-bits of input, key, and cipher text. The designed EPRNG-AES architecture utilizes 1–3% resources of slices, LUT, registers, flip flops and logical elements. For example, the EPRNG-AES architecture designed in the Kintex-7 FPGA device uses the 1% of slices, LUT, registers, flip flops and logical elements.

Figure 5 shows the simulation waveform for the response that is generated by using the IoT device. The control signals that are used for the response value generation are *clk* and *rst*. Moreover, the challenges and responses generated by the IoT device are challenge and response respectively.

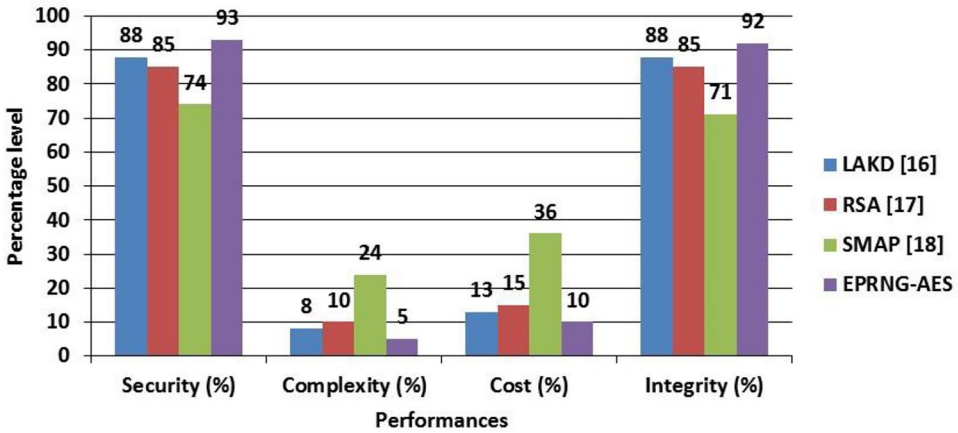

Fig. 8 Comparison graph of system estimation

Table 10 Performance comparison for Spartan 6

Methods	Slices	Operating frequency (MHz)
AES-PNSG [24]	5566	237.45
EPRNG-AES	55	301.231

From Fig. 5, it is known that the response generated in each clock cycle is different from each other, which shows that the EPRNG-AES architecture provides higher security. The simulation waveform for AES is shown in Fig. 6. In that, *clk*, *en*, and *rst* are the control signals. The plain text (i.e., response value), key and cipher text are represented as *plain* [127 : 0], *key* [127 : 0] and *cipher* [127 : 0] respectively. From Fig. 6, it is known that the difference between key-value and cipher text is high in the EPRNG-AES architecture. Hence, the EPRNG-AES architecture provides higher security against different security threats.

Figure 7 shows the floor planning design obtained for EPRNG-AES architecture. An accurate connection of EPRNG-AES architecture is verified by using this floor planning design. However, this floor planning output can be taken only when there is no error in the connections of the EPRNG-AES architecture. Therefore, the connections of the EPRNG-AES architecture are connected without any error.

5.2 Security Analysis of the EPRNG-AES Architecture

This section provides the security analysis of the EPRNG-AES architecture under different security threats. There are five different security threats such as Side-Channel Attacks (SCA), Denial of Service (DoS), Offline Password Guessing Attack (OPGA), Session Key Agreement (SKA), and validity of strict key.

a. SCA

SCA uses the data leaks from the system, which are categorized as noninvasive and passive attacks. Generally, this SCA retrieves any type of confidential data from the IoT network. This SCA doesn't concentrate on the AES algorithm instead of that it affects the physical device for obtaining the secured information. Here, the information can be retrieved by evaluating and investigating the leaked data such as power, electromagnetic analysis, timing, and so on.

b. DoS Attacks

DoS attacks generally try to stop the network or any resources from using the protector. This attack is created based on requests obtained from the registered user, messages, processor cycle for games, download requests and so on. The data traffic stream is stopped by the DoS attack and the network becomes unstable because of the DoS attack. The DoS attack sends malicious data to only one device, however, the DDoS attack sends the malicious data to multiple users.

c. OPGA

OPGA is generally a brute force attack that tries identifying the password based on the systematic checking process by using all the options. The attacker device accesses the password hash in offline and tries the combination of a key without any interference. Here, the complexity and the length of the password defines the time, which is required to crack the password. However, this EPRNG-AES architecture creates a different key value for every clock cycle. Moreover, the EPRNG uses the high confusion property of the circuit, which makes it highly difficult for the attackers to crack the key.

d. SKA

A session key is a temporary key that is utilized only once for encryption and decryption of the data at the next clock cycle. This session key behaves like a password that is being reset for every log-in time. Hence, the end-to-end security is provided by the EPRNG module without any additional security substructure.

e. Validity of strict key avalanche criteria

The input data is encrypted by using the 128-bit AES algorithm for analyzing the strict key validation. Moreover, anyone bit of the 128-bit key is altered during

Table 11 Performance comparison for Kintex 7

Methods	Slice LUTs	Flip flops	IOBs	Operating frequency (MHz)
LAES [13]	9468	-	384	-
AES-HLS [20]	577	449	265	297.3
EPRNG-AES	153	39	26	305.761

Table 12 Performance comparison for Virtex 5

Methods	Slice registers	Slice LUTs	Slices	Operating frequency (MHz)
AES-CTR [19]	19,123	14,966	5974	622.4
EPRNG-AES	38	174	62	658.288

Table 13 Performance comparison for Virtex 6

Methods	Slice registers	Slice LUTs	Delay (ns)	Power (W)
AES-MMC [15]	2688	9393	3.167	3.725
EPRNG-AES	38	229	2.981	0.022

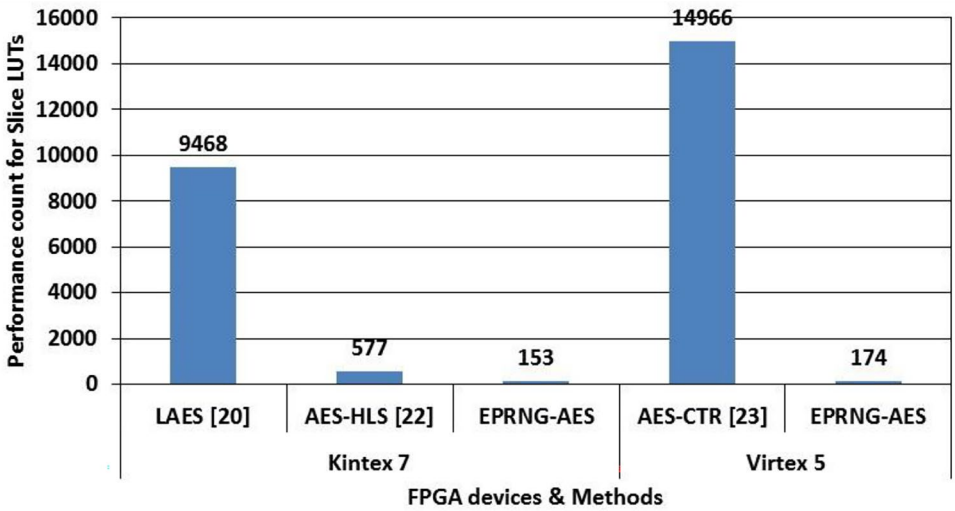

Fig. 9 Graphical illustration of slice LUTs

the decryption process. The EPRNG-AES architecture doesn't provide the decrypted value at the output, when the key is changed in the decryption process. Therefore, it shows that the EPRNG-AES architecture provides robustness against the changes in the key.

Additionally, the system estimation in terms of security, complexity, cost and integrity is evaluated for the LAKD [14], RSA [16], SMAP [4] and EPRNG-AES. This evaluation is shown in the following Table 9 and Fig. 8.

5.3 Comparative Analysis

This section shows the comparative analysis of the EPRNG-AES architecture. The proposed EPRNG-AES architecture is compared with five existing AES architectures such as AES-PNSG [24], LAES [13], AES-HLS [20], AES-CTR [19] and AES-MMC [15]. Here, the comparison is made between four different FPGA devices such as Spartan 6, Kintex 7, Virtex 5 and Virtex 6.

The comparison of the EPRNG-AES architecture for the FPGA devices of the Spartan 6, Kintex 7, Virtex 5 and Virtex 6 are provided in Tables 10, 11, 12 and 13 respectively. In that, AES-PNSG [24] is used for Spartan 6, LAES [13] & AES-HLS [20] are used for Kintex 7, AES-CTR [19] is used for the Virtex 5 and AES-MMC [15] for Virtex 6 device comparisons. Moreover, the graphical illustration of slice LUTs and operating frequency are shown in Fig. 9 and Fig. 10 respectively. From the analysis, it is concluded that the EPRNG-AES architecture provides better performance than the AES-PNSG [24], LAES [13], AES-HLS [20], AES-CTR [19] and AES-MMC [15]. For example, the EPRNG-AES architecture designed in the Virtex 5 uses 62 slices, whereas the AES-CTR [19] uses 5974 slices. The

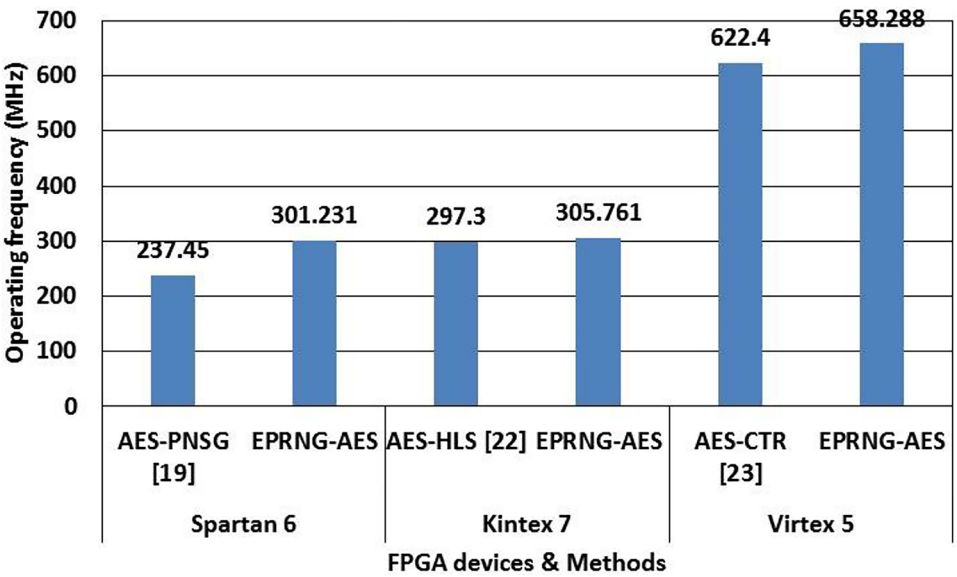

Fig. 10 Graphical illustration of operating frequency

existing AES architectures such as AES-PNSG [24], LAES [13], AES-HLS [20], AES-CTR [19] and AES-MMC [15] require a high amount of registers to store the keys obtained from the manual key generation. If there is a high amount of registers, the existing architecture requires a high amount of logical elements. However, the EPRNG-AES architecture requires only less amount of registers, because this optimized AES uses the automatic key generation using the EPRNG key generation module. This EPRNG key generation module doesn't require any register to save the key values, which minimizes the logical elements used in the EPRNG-AES architecture. Additionally, the security of the AES architecture is improved by using the EPRNG module. This EPRNG module generates a different key for each clock cycle, which improves the security against various threats such as SCA, DoS, OPGA, and SKA.

6 Conclusion

In this research paper, the EPRNG-AES architecture improves the security between the IoT device and server while minimizing the hardware resources. The EPRNG with two-level TRNG key generation module generates different key values for each clock cycle, which improves the security of the response value. Moreover, different challenge and key response values, which are obtained for each clock cycle, help to improve the dissimilarity between the ciphertexts of all clock cycles. This helps to improve the security against various security threats such as SCA, DoS, OPGA, SKA, and key attacks. On the other hand, the hardware resources are minimized by optimizing the AES architecture. Here, the register used to store the intermediate key values are not required during the encryption/decryption process. Therefore, the EPRNG-AES architecture consumes less hardware resources during the implementation. From the analysis, it is concluded that the EPRNG-AES architecture achieves better performance than the LAKD, RSA, SMAP, AES-PNSG, LAES, AES-HLS, AES-CTR and AES-MMC. The EPRNG-AES architecture designed in the Kintex 7 uses 153 slices, which is lesser than the number of slices in LAES and AES-HLS.

Funding This research received no external funding.

Data Availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Akbarzadeh A, Bayat M, Zahednejad B, Payandeh A, Aref MR (2019) A lightweight hierarchical authentication scheme for internet of things. J Ambient Intell Hum Comput 10:2607–2619. https://doi.org/10.1007/s12652-018-0937-6
2. Arul Murugan C, Karthigaikumar P, SathyaPriya S (2020) FPGA implementation of hardware architecture with AES encryptor using sub-pipelined S-box techniques for compact applications. Automatika 61:682–693. https://doi.org/10.1080/00051144.2020.1816388
3. Braeken A, Liyanage M, Jurcut AD (2019) Anonymous lightweight proxy based key agreement for IoT (ALPKA). Wireless Pers Commun 106:345–364. https://doi.org/10.1007/s11277-019-06165-9
4. Chen CM, Chen L, Huang Y, Kumar S, Wu JMT (2021) Lightweight authentication protocol in edge-based smart grid environment. EURASIP J Wirel Commun Netw 2021:68. https://doi.org/10.1186/s13638-021-01930-6
5. Chikouche N, Cayrel PL, Mboup EHM, Boidje BO (2019) A privacy-preserving code-based authentication protocol for internet of things. J Supercomput 75:8231–8261. https://doi.org/10.1007/s11227-019-03003-4
6. Das ML, Kumar P, Martin A (2020) Secure and Privacy-Preserving RFID authentication scheme for internet of things applications. Wireless Pers Commun 110:339–353. https://doi.org/10.1007/s11277-019-06731-1
7. De Smet R, Vandervelden T, Steenhaut K, Braeken A (2021) Lightweight PUF based authentication scheme for fog architecture. Wireless Netw 27:947–959. https://doi.org/10.1007/s11276-020-02491-0
8. Dhanda SS, Singh B, Jindal P (2020) Lightweight Cryptography: A Solution to Secure IoT. Wireless Pers Commun 112:1947–1980. https://doi.org/10.1007/s11277-020-07134-3
9. Farooq U, Hasan NU, Baig I, Shehzad N (2019) Efficient adaptive framework for securing the internet of things devices. EURASIP J Wirel Commun Netw 2019:210. https://doi.org/10.1186/s13638-019-1531-0
10. Jang S, Lim D, Kang J, Joe I (2016) An efficient device authentication protocol without certification authority for internet of things. Wireless Pers Commun 91:1681–1695. https://doi.org/10.1007/s11277-016-3355-0
11. Jebri S, Amor AB, Abid M, Bouallegue A (2021) Enhanced lightweight algorithm to secure data transmission in IOT systems. Wireless Pers Commun 116:2321–2344. https://doi.org/10.1007/s11277-020-07792-3
12. Khalid U, Asim M, Baker T, Hung PCK, Tariq MA, Rafferty L (2020) A decentralized lightweight blockchain-based authentication mechanism for IoT systems. Clust Comput 23:2067–2087. https://doi.org/10.1007/s10586-020-03058-6
13. Kumar K, Ramkumar KR, Kaur A (2020) A lightweight AES algorithm implementation for encrypting voice messages using field programmable gate arrays. J King Saud University-Computer Inform Sci. https://doi.org/10.1016/j.jksuci.2020.08.005
14. Lara E, Aguilar L, Sanchez MA, García JA (2020) Lightweight authentication protocol for m2m communications of resource-constrained devices in industrial internet of things. Sensors 20:501. https://doi.org/10.3390/s20020501
15. Madhavapandian S, MaruthuPandi P (2020) FPGA implementation of highly scalable AES algorithm using modified mix column with gate replacement technique for security application in TCP/IP. Microprocess Microsyst 73:102972
16. Megouache L, Zitouni A, Djoudi M (2020) Ensuring user authentication and data integrity in multi-cloud environment. HCIS 10:15. https://doi.org/10.1186/s13673-020-00224-y

17. Melki R, Noura HN, Chehab A (2020) Lightweight multi-factor mutual authentication protocol for IoT devices. Int J Inf Secur 19:679–694. https://doi.org/10.1007/s10207-019-00484-5
18. Rao V, Prema KV (2021) A review on lightweight cryptography for internet-of-things based applications. J Ambient Intell Hum Comput 12:8835–8857. https://doi.org/10.1007/s12652-020-02672-x
19. Shahbazi K, Ko SB (2020) High throughput and area-efficient FPGA implementation of AES for high-traffic applications. IET Comput Digital Tech 14:344–352. https://doi.org/10.1049/iet-cdt.2019.0179
20. Sikka P, Asati AR, Shekhar C (2021) High-throughput field-programable gate array implementation of the advanced encryption standard algorithm for automotive security applications. J Ambient Intell Hum Comput 12:7273–7279. https://doi.org/10.1007/s12652-020-02403-2
21. Singh S, Sharma PK, Moon SY, Park JH (2017) Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. J Ambient Intell Hum Comput. 1-18. https://doi.org/10.1007/s12652-017-0494-4
22. Xu L, Wu F (2019) A lightweight authentication scheme for multi-gateway wireless sensor networks under IoT conception. Arab J Sci Eng 44:3977–3993. https://doi.org/10.1007/s13369-019-03752-7
23. Zhang F, Liang ZY, Yang BL, Zhao XJ, Guo SZ, Ren K (2018) Survey of design and security evaluation of authenticated encryption algorithms in the CAESAR competition. Frontiers of Information Technology & Electronic Engineering 19:1475–1499. https://doi.org/10.1631/FITEE.1800576
24. Zodpe H, Sapkal A (2020) An efficient AES implementation using FPGA with enhanced security features. J King Saud Univ Eng Sci 32:115–122. https://doi.org/10.1016/j.jksues.2018.07.002

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

N. Siva Balan is pursuing his PhD at Department of CSE, Kalasalingam Academy of Research and Education, Srivilliputtur, Virudhunagar under the guidance of Dr. B. S. Murugan. He has received the Undergraduate Degree (BE) in Computer Science and Engineering from Anna University, and the Post Graduate degree (ME) in Information Technology from Bangalore University. He has over 16 years of teaching experience. His areas of interest include Artificial Intelligence and Machine Learning, Cloud Computing, Data Structures and Algorithms.

B. S. Murugan received the Undergraduate Degree (B.Tech.) in Information Technology from Anna University, the Post Graduate degree (M.Tech) in Information Technology from SRM University, and Ph.D. in Information Technology (Cloud Computing) from Kalasalingam University. He has more than 20 publications in National and International Conferences and International Journal proceedings. He has over 12 years of teaching experience. His areas of interest include Artificial Intelligence, Cloud Computing, Operating Systems, and DBMS. He is currently working as Associate Professor in the Department of Computer Science and Engineering at Kalasalingam Academy of Research and Education, Krishnankoil, Tamil Nadu, India.