Computers, Materials & Continua
DOI:10.32604/cmc.2021.014606
Article

Tech Science Press

Low Area PRESENT Cryptography in FPGA Using TRNG-PRNG Key Generation

T. Kowsalya[1], R. Ganesh Babu[2], B. D. Parameshachari[3], Anand Nayyar[4] and Raja Majid Mehmood[5,*]

[1]Department of Electronic and Communication Engineering, Muthayammal Engineering College, Rasipuram, Namakkal, Tamilnadu, India
[2]Department of Electronics and Communication Engineering, SRM TRP Engineering College, Tiruchirappalli, Tamilnadu, India
[3]Department of Telecommunication Engineering, GSSS Institute of Engineering and Technology for Women, Mysuru, India
[4]Faculty of Information Technology, Graduate School, Duy Tan University, Da Nang, 550000, Viet Nam
[5]Department of Information and Communication Technology, School of Electrical and Computer Engineering, Xiamen University Malaysia, Sepang, 43900, Malaysia
[*]Corresponding Author: Raja Majid Mehmood. Email: rmeex07@ieee.org, rajamajid@xmu.edu.my
Received: 29 September 2020; Accepted: 31 October 2020

Abstract: Lightweight Cryptography (LWC) is widely used to provide integrity, secrecy and authentication for the sensitive applications. However, the LWC is vulnerable to various constraints such as high-power consumption, time consumption, and hardware utilization and susceptible to the malicious attackers. In order to overcome this, a lightweight block cipher namely PRESENT architecture is proposed to provide the security against malicious attacks. The True Random Number Generator-Pseudo Random Number Generator (TRNG-PRNG) based key generation is proposed to generate the unpredictable keys, being highly difficult to predict by the hackers. Moreover, the hardware utilization of PRESENT architecture is optimized using the Dual port Read Only Memory (DROM). The proposed PRESENT-TRNG-PRNG architecture supports the 64-bit input with 80-bit of key value. The performance of the PRESENT-TRNG-PRNG architecture is evaluated by means of number of slice registers, flip flops, number of slices Look Up Table (LUT), number of logical elements, slices, bonded input/output block (IOB), frequency, power and delay. The input retrieval performances analyzed in this PRESENT-TRNG-PRNG architecture are Peak Signal to Noise Ratio (PSNR), Structural Similarity Index (SSIM) and Mean-Square Error (MSE). The PRESENT-TRNG-PRNG architecture is compared with three different existing PRESENT architectures such as PRESENT On-The-Fly (PERSENT-OTF), PRESENT Self-Test Structure (PRESENT-STS) and PRESENT-Round Keys (PRESENT-RK). The operating frequency of the

PRESENT-TRNG-PRNG is 612.208 MHz for Virtex 5, which is high as compared to the PRESENT-RK.

Keywords: Dual port read only memory; hardware utilization; lightweight cryptography; malicious attackers; present block cipher; pseudo random number generator; true random number generator

1 Introduction

Lightweight Cryptography (LWC) plays a vital role to obtain the higher security with low energy and low area in different sensitive applications such as implantable and wearable medical devices, radio-frequency identification tags, Wireless Nano sensors, and smart cards and secure embedded systems [1–3]. The symmetric cryptography is divided into two types such as block and stream ciphers. The block cipher processes the one input block at a time and produces the output block for each input block, but the stream cipher frequently processes the input elements and generates the one output element at a time [4,5]. FPGA is considered as a growing design platform to implement the cryptographic algorithms because of its in-house security and reconfigurability [6].

Generally, the Advanced Encryption Standard (AES) is widespread block cipher and fundamental for many security systems [7]. But AES used in the high-performance processors is not suitable in resource-constrained platforms because of its inadequate area and energy/power [8]. Therefore, the better tradeoff between the security, power, area and speed is obtained by designing the lightweight block cipher [9]. Some of the examples of the lightweight ciphers are the STES [10], SEED [11], ANU [12], KLEIN [13], PRESENT [14], and KASUMI [15] and so on. From the different lightweight cipher, PRESENT block cipher is selected as an efficient algorithm due to its hardware efficiency and it also standardized by ISO/IEC 29192-2. However, the hardware failures are considered as the natural fault in the implementation of Very-Large-Scale Integration (VLSI). This natural fault increases the sensitivity and creates the malicious attacks over the cryptographic hardware and embedded systems [16]. The conventional LWC method uses the same type of generators and identical keys to accomplish both the encryption and decryption process that leads susceptible to the attacks [17].

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

- The PRESENT architecture improves the robustness against the malicious attackers by random key generation using TRNG-PRNG module and two stage security is used during the encryption process.
- In PRESENT architecture, the key value from the TRNG-PRNG module is generated for each clock and plaintext. Therefore, the identification of key value by the unauthenticated users (i.e., malicious attackers) is difficult during the encryption/decryption process.
- A DROM is utilized to minimize the number of logical elements used in the PRESENT architecture. The DROM used in the PRESENT architecture accomplishes the operation of Substitution box (S-box).

The overall organization of the paper is: The literature survey related to existing PRESENT architecture is described in Section 2. The problem statement found from the literature survey along with solution is described in Section 3. Section 4 describes the PRESENT architecture by using key scheduling approach and TRNG-PRNG module. The results and discussion of the PRESENT-TRNG-PRNG architecture is presented in Section 5. Finally, the conclusion is made in Section 6.

2 Literature Survey

The literature survey regarding the recent PRESENT block cipher is described along its advantages and limitations in this section.

Pandey et al. [18] implemented the PRESENT lightweight block cipher algorithm to accomplish the encryption and decryption processes. The developed PRESENT architecture processed 64-bit input value along with the 80/128 bit of key length. Additionally, the dynamic keys were provided with the OTF architecture to compute the intermediate key. Next, the generated intermediate keys were used to accomplish the encryption/decryption operation. The iterative method considered in the decryption is used to achieve the better tradeoff among time and area. The total power consumption of the PRESENT architecture was high at low frequency, when processed with high number of key bits (128 bit).

De Cnudde et al. [19] developed the evaluation of PRESENT block cipher under two different physical attacks such as Side-Channel Analysis (SCA) and fault attacks (FAs). The first order implementation was used to provide the security against the side-channel. Next, the Private Circuits II is used to provide the security against the FA. The leakage detection test was used to analyze the side channel evaluation. But, the Private Circuits II used for FA resistance in the PRESENT block cipher was expensive.

Azari et al. [20] implemented the PRESENT Cipher model that incorporated both encryption and decryption process. The encryption and decryption process were accomplished by using 80/128-bit key to obtain the security for 64-bit input value. The plain text of 64 bits processing requires 16 cycles to load the data during encryption process. Here, the PRESENT cipher obtains a higher throughput based on the effective encryption and decryption process. However, the PRESENT cipher used high number of S-boxes which increased the hardware utilization.

Rashidi [21] presented the two different low-cost and high-throughput block ciphers such as HIGHT and PRESENT to improve the security. Since, the modulo 28 was one of the complex blocks in the HIGHT algorithm. Next, the parallel prefix adders such as Sklansky, Han–Carlson, Kogge–Stone and Ladner–Fischer were used to design the modular adder. Moreover, the PRESENT cipher was supported by two key lengths such as 80-bit and 128-bit. The Karnaugh mapping was used to reduce the amount of logic gates in the S-box and critical path delay. But, the computation time was high and throughput was less when the unroll factor is high in block ciphers.

Lara-Nino et al. [22] developed the standardized lightweight cipher namely PRESENT to overcome the security issues caused at the extremely constrained environments. Moreover, the data in the registers were moved to the right that used to reduce the MUX size. The PRESENT architecture used two different alternatives to generate the RK of 80 bit and 128 bit. Since, the 80 bit and 128 bit input keys were generated using 20 bit registers and 32 bit registers respectively. The key given to the PRESENT architecture was manually generated by the key generator module and it can be easily predicted by hackers.

3 Problem Statement

The problems obtained from the existing literature survey along with the solution by the PRESENT-TRNG-PRNG architecture is as follows.

The STS based PRESENT architecture requires an additional comparator to generate the output [19]. Next, the conventional PRESENT architecture uses high amount of S-box operation to accomplish the encryption process [20]. The unroll factor considered in the loop unrolling method also affects the performance of the hardware utilization [21]. The aforementioned constraints increase the number of logical elements used in the PRESENT architecture. Since, the increment of hardware utilization leads to affect the operating frequency and power of the overall PRESENT architecture. The manual key generation accomplished in the PRESENT architecture [22] generates the same key value for each clock cycle. The generation of same clock cycle for each round can be easily predicted by the malicious attackers.

Solution:

The logical components of PRESENT architecture are minimized by using the DROM. In this PRESENT architecture, 8 DROM is used instead of 16 S-boxes of the conventional PRESENT architecture. The DROM used in the PRESENT-TRNG-PRNG accomplishes the same process which is performed by the S-box. Moreover, the security of the PRESENT-TRNG-PRNG architecture is improved by using two different approaches: (1) two stage security approach and (2) unpredictable key generation using TRNG-PRNG module. The robustness of the PRESENT architecture is improved by generating the random key for each clock cycle and each plaintext.

4 PRESENT-TRNG-PRNG Architecture

In the PRESENT-TRNG-PRNG architecture, the logical elements are optimized by using the DROM to accomplish the encryption/decryption process. The PRESENT architecture is designed to support the 64-bit input value with 80-bit key length. Here, the random key generation is carried out by using the TRNG-PRNG module. The randomness of the key from the TRNG-PRNG module is improved using the two-stage security enabled during encryption process. The block diagram of the PRESENT-TRNG-PRNG architecture is shown in Fig. 1.

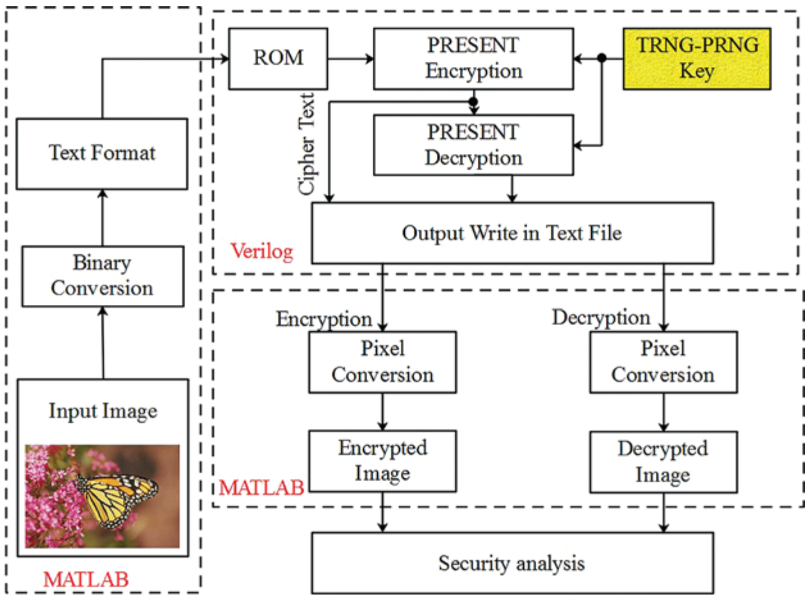


Figure 1: Block diagram of the PRESENT-TRNG-PRNG architecture

The overall working process of the PRESENT-TRNG-PRNG architecture are given as follows:

1. At first, the input image (*P*) is read from the MATLAB R2018a software and the image pixels are converted into binary format.
2. Next, the binary value of the image pixels is written in the text format using MATLAB.
3. The TRNG-PRNG module is used to generate the random key value to accomplish the encryption operation. The decryption process is generally the inverse process of the encryption operation.
4. The Verilog (Modelsim) is used to process both the encryption/decryption process. Moreover, the output of encryption and decryption is written in the text format using the Verilog (Modelsim).
5. Then the text files are used in the MATLAB to convert the encrypted and decrypted binary value into the image.

4.1 64-Bit Path Encryption

The overall architecture of the path encryption for 64-bit data is shown in Fig. 2. At first, the one pixel from the image is converted into 8 bits and total plain text of 64-bit data (*PT*) is kept in the register. The plaintext stored in the register is denoted as *Dreg*. On the other hand, the TRNG-PRNG module generates an appropriate key to accomplish the encryption operation over the 64-bit value of plaintext. The conventional PRESENT architecture manually generates the key values which are subjected to predict by the hackers. The main objective of using TRNG-PRNG module in PRESENT architecture is to obtain high security level by generating the random key value for each pixel at every clock cycle.

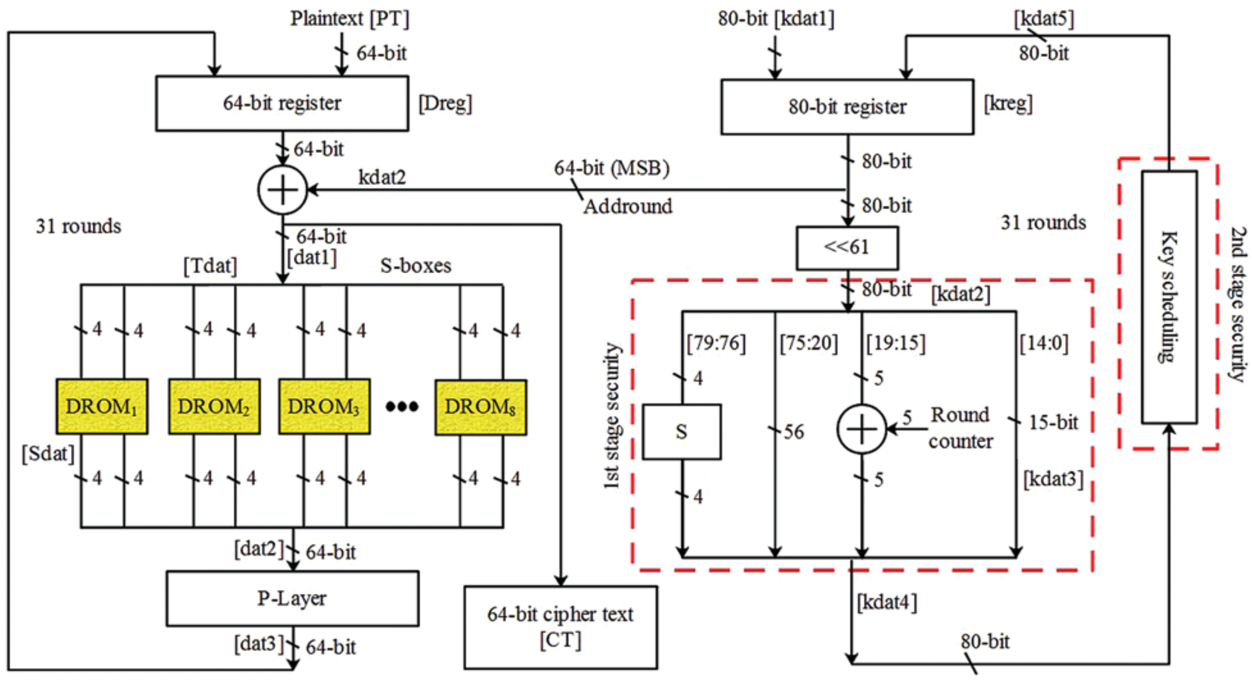


Figure 2: Architecture of 64-bit path encryption

The 80 bit of key value ($kdat1$) generated from the TRNG-PRNG module is stored in the register $kreg$. From the 80 bit of $kdat1$, the MSB of 64 bit data ($kdat2$) is selected and then it is XORed with the 64 bit of PT value as shown in Eq. (1).

$$dat1 = XOR\,(PT, kdat2) \tag{1}$$

where, the $dat1$ represents the XOR value between the plaintext and MSB of 64-bit data from the key value generated by TRNG-PRNG module.

Next, the XORed data is truncated into 16 four bit values which are shown in Eq. (2).

$$Tdat = \{dat1\,[0:\,3],\, dat1\,[4:\,7],\, \ldots,\, dat1\,[56:\,59],\, dat1\,[60:\,63]\} \tag{2}$$

From the 16 sets of 4-bit values, each 2 sets of 4-bit values are given into the DROM which processes the operation of Substitution box (S-box). For example, the $dat1\,[0:\,3]$ and $dat1\,[4:\,7]$ are given to the DROM$_1$ to process the S-box operation. Totally, there are eight DROMs are used to produce the 64-bit value based on the S-box operation shown in the Tab. 1. The conventional PRESENT architecture uses the 16 different S-box operation that leads to increase the hardware utilization and increases the delay while processing the input plain text. Hence, the PRESENT-TRNG-PRNG architecture uses only 8 DROMs to process the S-box operation which minimizes the number of logical elements used in the encryption process. The reduction in logical elements minimizes the hardware utilization and increases the speed of the encryption process.

Table 1: Operation of S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

The value obtained from the DROM is represented as $Sdat$ which is obtained through the S-box operation. The value from the 8 DROMs are concatenated together and generated one 64-bit value i.e., $dat2$ which is shown in the Eq. (3).

$$dat2 = \{Sdat\,[0:\,3]\,\|\,Sdat\,[4:\,7],\, \ldots,\, Sdat\,[56:\,59]\,\|\,Sdat\,[60:\,63]\} \tag{3}$$

Then the concatenated 64-bit value is processed through the permutation layer (P-layer). This P-layer used to move the bit value in new bit position as shown in the Tab. 2. Moreover, the value from the P-layer is represented as $dat3$ and this updated $dat3$ is considered instead of plaintext for next 31 rounds.

Table 2: Operation of P-layer

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

On the other hand, the 80-bit key value of processed under the 61 left shift operation as shown in the Eq. (4).

$$kdat2 = \{kdat1\,[18:0],\, kdat1\,[79:19]\} \tag{4}$$

Next, the $kdat2$ is divided into four sets such as $kdat2\,[79:76]$, $kdat2\,[75:20]$, $kdat2\,[19:15]$, and $kdat2\,[14:0]$, to obtain the first stage security. The 56-bit and 15-bit of the $kdat2\,[75:20]$ and $kdat2\,[19:15]$ are directly taken while updating the key value. Additionally, $kdat2\,[79:20]$ and $kdat2\,[19:15]$ are processed through the S-box and add counter to update the key value. The aforementioned steps are illustrated in the following Eq. (5).

$$\begin{aligned} &kdat3\,[14:0] = kdat2\,[14:0] \\ &kdat3\,[19:15] = XOR\,(kdat\,[19:15],\, RC) \\ &kdat3\,[75:20] = kdat2\,[75:20] \\ &kdat3\,[79:76] = Sbox\,(kdat2\,[79:76]) \end{aligned} \tag{5}$$

where, RC represents the round counter that varies from 0 to 31 for each round. The updated key values such as $kdat3\,[79:76]$, $kdat3\,[75:20]$, $kdat3\,[19:15]$ and $kdat3\,[14:0]$ are concatenated to generate a $kdat4$ which is shown in the Eq. (6).

$$kdat4 = \{kdat3\,[79:76]\,\|kdat3\,[75:20]\,\|kdat3\,[19:15]\,\|\}\,kdat3\,[14:0] \tag{6}$$

This updated $kdat4$ is given to the key scheduling process to accomplish the second stage security. Both the first and second stage security are used to improve the randomness of the key values.

4.2 Key Scheduling Process

The architecture of key scheduling used in the 64-bit path encryption is shown in Fig. 3. This key scheduling is processed for the next 31 rounds to improve the security of the plaintext against malicious attackers. The 80 bit value of $kdat4$ is truncated into four 20 bit values such as $K1$, $K2$, $K3$, and $K4$. Subsequently, these four 20-bit values are processed through the rotation operation in which the bit position is changed as shown in the Tab. 3.

The rotation operation provides four different 20-bit values such as $R1$, $R2$, $R3$, and $R4$. Next, the truncation operation is processed for each 20-bit value obtained through the rotation. This truncation operation provides five 4-bit values for each 20-bit value. Totally, 20 four-bit values i.e., $\{TR1, TR2, \ldots, TR20\}$ are acquired from the four 20-bit value. Moreover, the security is effectively improved by processing the 20 four-bit values through the DROM. Here, each DROM processes two 4-bit value, totally 10 DROMs are utilized in the key scheduling process. The DROM used to perform the operation of the S-box as shown in Tab. 1. Finally, the DROM provides 20 four-bit values such as i.e., $\{Y1, Y2, \ldots, Y20\}$. This $Y1$, $Y2, \ldots, Y20$ is given to the rotation operation that generates the 80-bit value i.e., $kdat5$.

After completing the 32 rounds, the PRESENT architecture provides the encrypted cipher text that is denoted as CT. Moreover, the decrypted value is obtained based on the inverse process of PRESENT decryption. Here the reverse architecture of PRSENT module is used during the decryption process. The process of key generation using the TRNG-PRNG module is explained in the following section.

4.3 Key Generation Using TRNG-PRNG Module

In this TRNG-PRNG module, the key value is generated for each clock cycle as well as for each plain text to improve the security. The overall architecture of 80-bit key generation using TRNG-PRNG module is illustrated in Fig. 4. Generally, the TRNG is designed by the digital circuits to produce the true randomness using the unpredictable effects. Here the TRNG is generated by using the $*random* function. The sequence generated by the TRNG is mainly based on two essential features such as uniformity and statistical independence among the actual symbol and the numbers generated in previous rounds. Moreover, the overall circuit design being used to generate 80-bit key is referred as PRNG.

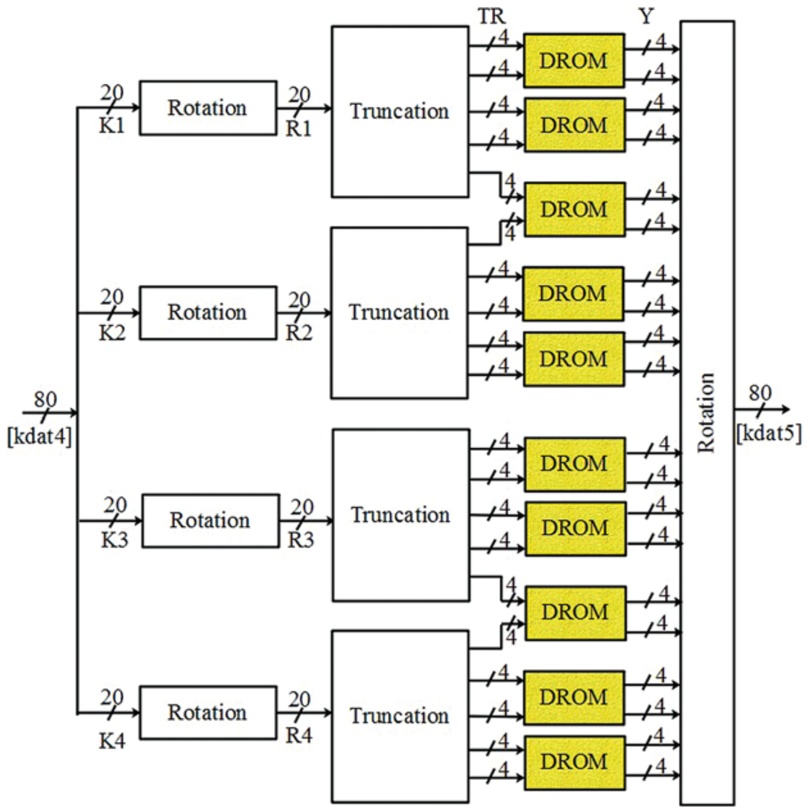


Figure 3: Architecture of key sampling process

Table 3: Operation of rotation

K1(j)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R1(j)	14	10	9	13	16	11	8	2	17	5	18	3	15	1	19	4	7	12	0	6

The steps processed in the key generation using TRNG-PRNG module are given as follows:

a. Initially, the TRNG-PRNG module generate the 80-bit true random number that is represented as $RN0$. Next, this 80-bit $RN0$ value is truncated into four 20 bits such as $T1$, $T2$, $T3$, and $T4$ which is shown in Eq. (7).

$$T1 = RN0\,[0:\ 19]$$
$$T2 = RN0\,[20:\ 39]$$
$$T3 = RN0\,[40:\ 59]$$
$$T4 = RN0\,[60:\ 79] \tag{7}$$

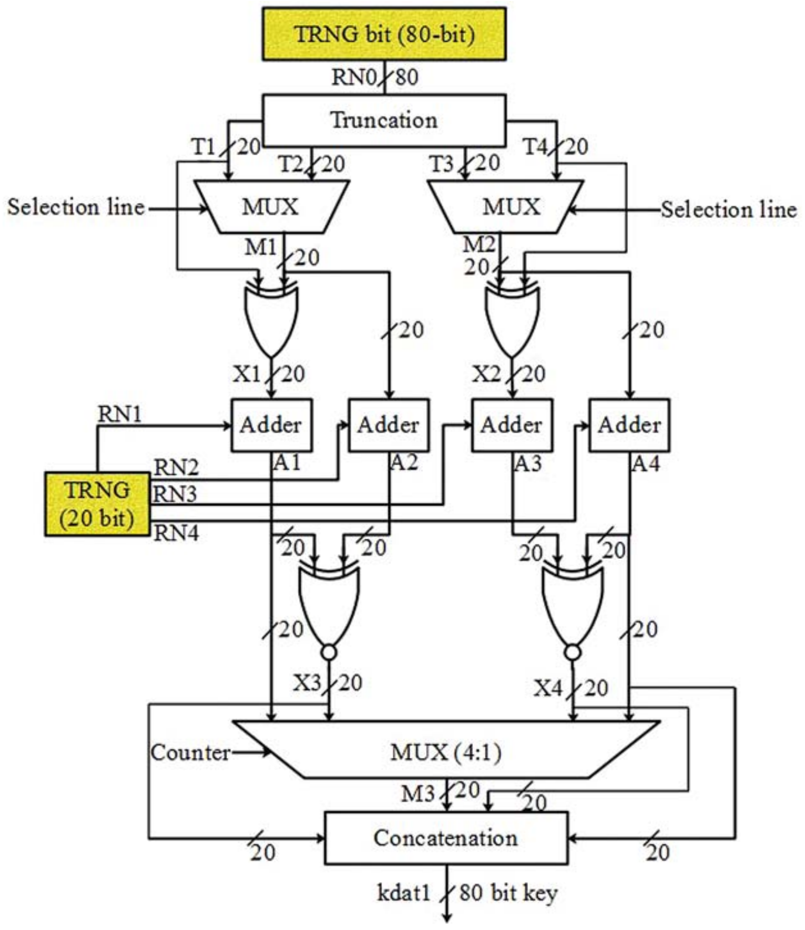


Figure 4: Architecture of TRNG-PRNG module

b. The pair of truncated values such as $T1 - T2$ and $T3 - T4$ is processed under MUX operation. The output from the MUX is operated using the selection line and it is shown in Eq. (8).

$$M1 = MUX\,(T1,\ T2)$$
$$M2 = MUX\,(T3,\ T4) \tag{8}$$

where, $M1$ and $M2$ represents the MUX operation value between the pairs of $T1 - T2$ and $T3 - T4$ respectively.

c. The values from the MUX processes $M1$ and $M2$ are XORed with the 20bit values of the $T1$ and $T4$ respectively. The XOR operation between the pairs of $M1 - T1$ and $M2 - T4$ are denoted as $X1$ and $X2$ respectively as shown in Eq. (9). For example, the XOR operation between the $M1 - T1$ pair is shown in Tab. 4.

$$X1 = XOR\ (M1,\ T1)$$
$$X2 = XOR\,(M2,\ T4) \tag{9}$$

Table 4: Sample XOR operation for $M1 - T1$

$M1$	$T1$	$X1$
0	0	0
0	1	1
1	0	1
1	1	0

d. In this key generation process, one more TRNG is used to generate four 20-bit values such as $RN1$, $RN2$, $RN3$, and $RN4$. This structure uses 4 different adders to sum the different pair values such as $X1 - RN1$, $M1 - RN2$, $X2 - RN3$ and $M2 - RN4$ as shown in Eq. (10).

$$A1 = X1 + RN1$$
$$A2 = M1 + RN2$$
$$A3 = X2 + RN3$$
$$A4 = M2 + RN4 \tag{10}$$

where, $A1$, $A2$, $A3$, and $A4$ are the values obtained through the addition process.

e. The values of $A1 - A2$ and $A3 - A4$ are processed under XNOR operation, once the addition is completed. Eq. (11) shows the process of XNOR operation and sample XNOR operation between the pair of $A1 - A2$ is shown in Tab. 5.

$$X3 = XNOR\,(A1, A2)$$
$$X4 = XNOR\,(A3, A4) \tag{11}$$

where, $X3$ and$X4$ are the XNOR values between the $A1 - A2$ and $A3 - A4$ pair respectively.

Table 5: Sample XNOR operation for $A1 - A2$

$A1$	$A2$	$X3$
0	0	1
0	1	0
1	0	0
1	1	1

f. One more MUX operation is carried out using 4 different inputs such as $A1$, $X3$, $A4$, and $X4$. The output ($M3$) from the MUX operation is mainly defined based on the counter value. For instance, the MUX gives $A1$ as output when the counter value is 0. Similarly, the MUX provides the output of $X3$, $A4$ and $X4$, when the counter value is 1, 2 and 3 respectively.

g. Finally, the concatenation operation between the $X3$, $M3$, $X4$ and $A4$ is carried out to generate the 80 bit key value i.e., $kdat1 = X3||M3||X4||A4$. This $kdat1$ of 80-bit value from the TRNG-PRNG module is given as input to the encryption process.

The optimization of hardware components using DROM leads to obtain the high operating frequency and less area utilization while designing the PRESENT architecture. Moreover, the generation of key for each round and each plaintext improves the robustness of the encrypted cipher text against attackers. Therefore, it is difficult to predict the original plain text without knowing key value generated from the TRNG-PRNG module.

5 Results and Discussion

The results and discussion of the TRNG-PRNG based PRESENT architecture is described in this section. The implementation of the PRESENT architecture along with key generation module i.e., TRNG-PRNG module is carried out using the Xilinx ISE 14.2 software. This TRNG-PRNG based PRESENT architecture is designed using the very high speed integrated circuit hardware description language and ModelSim simulator is used to perform the functional simulations. Moreover, the MATLAB R2018a software is used to convert the image file into txt file. In PRESENT architecture, the TRNG-PRNG module is used to generate the key to accomplish the encryption/decryption process. The developed PRESENT architecture supports the 80-bit key value for 64-bit input.

5.1 Performance Analysis of PRESENT-TRNG-PRNG Architecture for Different FPGA Devices

The PRESENT-TRNG-PRNG architecture is developed for 64-bit path encryption using the 80-bit key value. The 64-bit path encryption using PRESENT-TRNG-PRNG architecture is analyzed in six different Xilinx FPGA devices such as Spartan 6, Virtex 4, and Virtex 5. The sample input image considered for 64-bit path encryption is "monarch.png" highlighted in Fig. 5. Next, this sample image is converted as gray scale image and it is converted into 128×128 as shown in Fig. 6. The input image sizes are decided by the user and it is not stable.

Figure 5: Input image

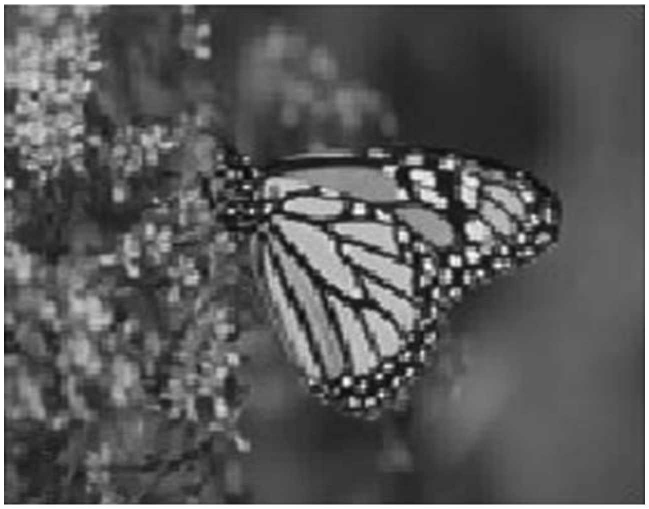
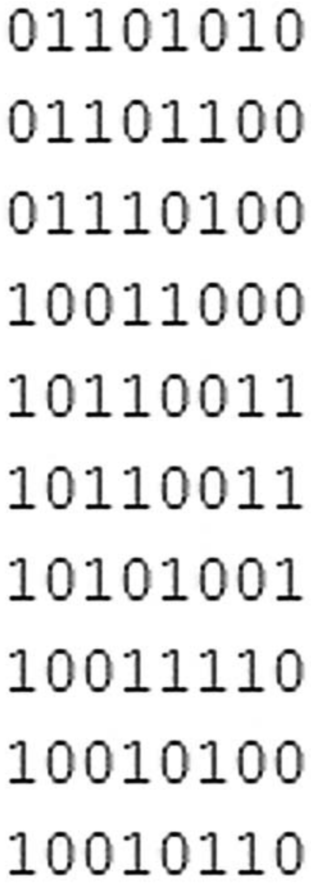


Figure 6: Gray image



```
01101010
01101100
01110100
10011000
10110011
10110011
10101001
10011110
10010100
10010110
```

Figure 7: Binary format of gray scale image

The gray scale input image shown in the Fig. 6 contains totally 16384 pixels. Additionally, the gray scale image is converted into binary format using the *dec2bin* function. The binary format of the image is shown in Fig. 7 and this binary value is stored in the memory of the FPGA processor. The histogram of the input image obtained using *imhist* function is shown in the Fig. 8. Next, the binary values of the sample image are divided into 64-bits and it is given as input to the encryption process. On the other hand, the TRNG-PRNG module generates the efficient key value of 80-bit that used to encrypt the input plain text.

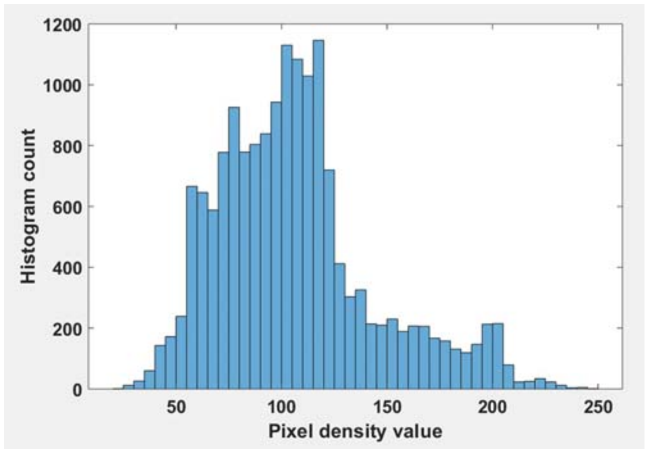


Figure 8: Histogram for input image

Fig. 9 shows the simulation waveform obtained from the ModelSim simulator. The control signals given to the PRESENT-TRNG-PRNG architecture are *clk* and *load*. The *idat* and *odat* in Fig. 9 represents the input data (PT) and cipher text (CT). Here, the input data (PT) is given to encryption and cipher text (CT) is obtained from the encryption operation. The *key* represents the 80-bit key value generated using TRNG-PRNG module.*kreg* and *dreg* are the registers used to store the input plaintext and key value from the TRNG-PRNG module respectively. Next, $dat1, dat2, dat3$ and $kdat1, kdat2$ are the intermediate variables of plaintext and key value that process 64-bit path encryption. Further, the *round* represents the number of rounds processed during the encryption process. Fig. 9 highlights that the encryption PRESENT-TRNG-PRNG architecture satisfies the test vector. For example, the output cipher text (i.e., 5579C1387B228445) marked by the red box in the Fig. 9 is equal to the cipher text given in the test vector. From the test vector analysis, it is proved that the PRESENT-TRNG-PRNG architecture works precisely during encryption. This test vector is verified for the PRESENT architecture except the 2nd stage key scheduling security.

The hardware utilization, power, delay, and frequency for the different FPGA architectures are illustrated as follows:

The hardware utilization of the PRESENT-TRNG-PRNG architecture for Spartan 6 is shown in the Tab. 6. The results shown from Tab. 6 is taken for the 64-bit path encryption using 80-bit key value. The LUT, slices and flip flops for the Spartan 6 device are 45, 35 and 48 respectively. From hardware analysis, the amount of LUT used by the Spartan 6 is less as compared to the remaining five FPGA devices. If the PRESENT-TRNG-PRNG architecture is implemented in the hardware Spartan 6, the encryption output is easily verified by using the 16-output light emitting diodes present in the Spartan 6 FPGA device. The utilization of 8 DROMs instead of 16 S-boxes in PRESENT architecture helps to minimize the hardware utilization. Moreover, the analysis of frequency, delay and power are shown in the Tab. 7. These performances are evaluated for different FPGA devices. Tab. 7 shows that the PRESENT-TRNG-PRNG architecture using Virtex 5 FPGA device provides higher frequency i.e., 612.208 MHz when compared to the remaining FPGA devices. The frequency of the PRESENT architecture with Virtex 5 device increase due to the less amount of hardware utilization.

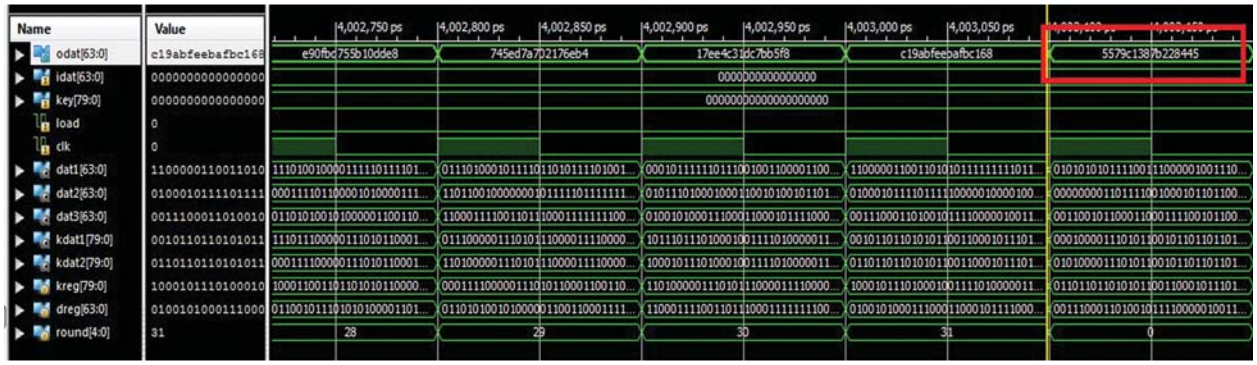


Figure 9: Simulation waveform

Table 6: Hardware utilization of PRESENT-TRNG-PRNG architecture in spartan 6 FPGA

FPGA performances	Total resources	Occupied resources	% of utilization
Number of slice registers	4800	62	1.29
Flip Flops	4800	48	1
Number of slice LUTs	2400	45	1.87
Number of logical elements	2400	78	3.25
Slices	600	38	6.33
Bonded IOB	12	6	50

Table 7: Analysis of frequency, delay and power for different FPGA devices

FPGA devices	Frequency (MHz)	Delay (ns)	Power (mW)
Spartan 3	226.341	6.126	38.22
Spartan 6	356.123	5.021	19.23
Virtex 4	423.415	4.126	210.43
Virtex 5	612.208	2.013	465.38
Artix 7	413.236	4.894	198.32
Kintex 7	530.561	3.672	423.84

The encrypted binary value of input image pixel is transferred to the MATLAB R2018a software. The encrypted image using PRESENT-TRNG-PRNG architecture and its histogram count are shown in the Figs. 10 and 12, respectively. Similarly, the decrypted image and its histogram count are shown in the Figs. 11 and 13 respectively. The amount of error occurred between the input sample image to the decrypted sample are calculated using the histogram count. Moreover, the image retrieval performance of the PRESENT-TRNG-PRNG architecture are analyzed using the MSE, PSNR and SSIM. The PRESENT-TRNG-PRNG architecture obtains significant PSNR and SSIM of 49.8762 dB and 0.8211 respectively. Hence, the PRESENT-TRNG-PRNG architecture preserves the details in the image during the encryption/decryption process.

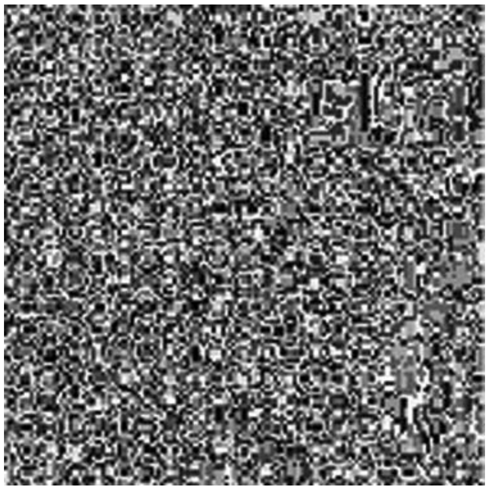


Figure 10: Encrypted image

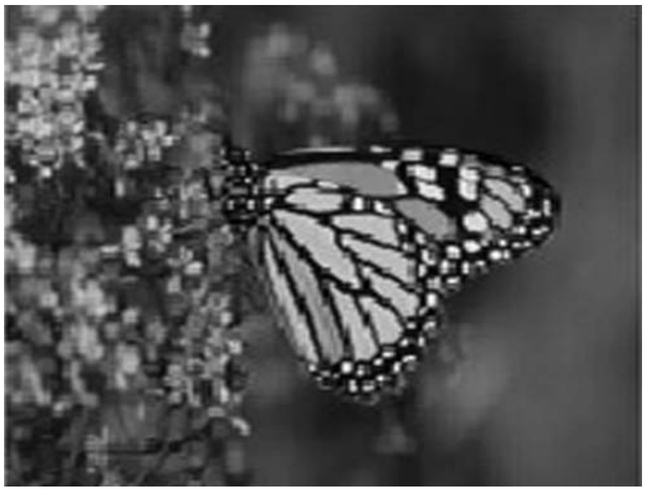


Figure 11: Decrypted image

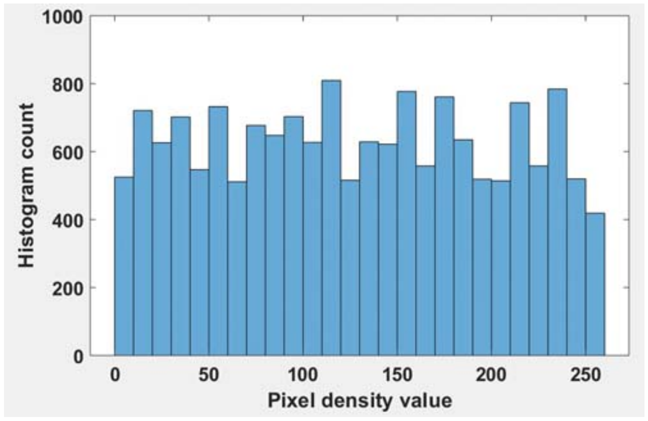


Figure 12: Histogram for encrypted image

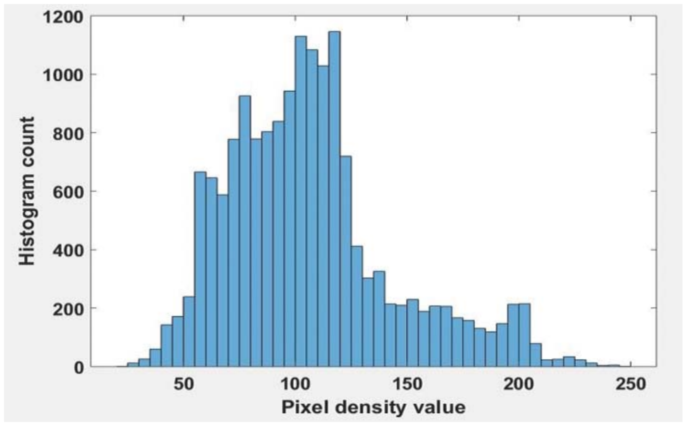


Figure 13: Histogram for decrypted image

5.2 Comparative Analysis

The effectiveness of the PRESENT-TRNG-PRNG architecture is evaluated by comparing with three existing PRESENT architecture designs. The existing methods used for the performance evaluation are PRESENT-OTF [18], PRESENT-STS [19] and PRESENT-RK [22]. The comparative analysis is accomplished by using five different FPGA devices such as Spartan 3, Spartan 6, Virtex 4, Virtex 5 and Kintex 7.

Tabs. 8 and 9 shows the comparison of the PRESENT-TRNG-PRNG architecture with the PRESENT-OTF [18], and PRESENT-RK [22] respectively. The comparison shows that the PRESENT-TRNG-PRNG architecture utilizes less amount of hardware components when compared to the PRESENT-OTF [18], and PRESENT-RK [22]. The PRESENT-STS [19] is used for high amount of S-box operation (e.g., 16 S-boxes) during encryption/decryption as well as this PRESENT-STS [19] requires additional comparator to generate the output that leads to increase the hardware utilization. But, the PRESENT-TRNG-PRNG uses only 8 DROM to accomplish the operation of the S-box. The DROM is used in both the encryption and key scheduling process that minimizes the overall hardware utilization. Moreover, the manual key generation of the PRESENT-RK [22] is vulnerable to the malicious attackers because the manually generated keys in PRESENT-RK [22] can be easily detected by the attackers. The two-stage security in the 64-bit path encryption and random key generation using TRNG-PRNG module increases the security against the malicious attackers.

Table 8: Comparison of PRESENT-TRNG-PRNG with PRESENT-OTF

Performance	Spartan 3		Virtex 5	
	PRESENT-OTF [18]	PRESENT-TRNG-PRNG	PRESENT-OTF [18]	PRESENT-TRNG-PRNG
Slices	326	114	56	39
Registers	217	98	215	70
LUT	590	201	217	102
Power (mW)	98	38.22	637	465.38

Table 9: Comparison of PRESENT-TRNG-PRNG with PRESENT-RK

Performance	Spartan 3		Spartan 6		Virtex 4		Virtex 5	
	PRESENT-RK [22]	PRESENT-TRNG-PRNG	PRESENT-RK [22]	PRESENT-TRNG-PRNG	PRESENT-RK [22]	PRESENT-TRNG-PRNG	PRESENT-RK [22]	PRESENT-TRNG-PRNG
Slices	124	114	48	38	124	47	67	39
Flip flops	153	82	153	48	153	78	153	56
LUT	215	201	170	45	215	194	190	102
Frequency (MHz)	213.81	226.341	257.40	356.123	375.66	423.415	542.30	612.208
Power (mW)	42.08	38.22	21.61	19.23	245.78	210.43	562.75	465.38

6 Conclusion

In this paper, the TRNG-PRNG module based key generation is accomplished in PRESENT architecture to generate the 80-bit key value to support the 64-bit of input value. Additionally, the randomness of the key obtained from the TRNG-PRNG module is increased using the two stage security during the 64-bit path encryption. Therefore, the key value used in the PRESENT-TRNG-PRNG architecture is unpredictable by the malicious attackers which improves the security of the input value. Moreover, the hardware utilization of the PRESENT architecture is minimized using the DROM to process the operation of S-box. Hence, the PRESENT-TRNG-PRNG architecture minimizes the logical elements while maintaining the higher security. The PRESENT-TRNG-PRNG architecture provides better performance when compared to the PRESENT-OTF, PRESENT-STS and PRESENT-RK. The operating frequency of the PRESENT-TRNG-PRNG is 612.208 MHz for Virtex 5, it is high when compared to the PRESENT-RK. In future, the architecture level optimization can be implemented as well as the hardware utilization and power consumption will be reduced for the entire PRESENT architecture.

Funding Statement: This research was supported by the Xiamen University Malaysia Research Fund (XMUMRF) (Grant No: XMUMRF/2019-C3/IECE/0007).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Aghaie, M. M. Kermani and R. Azarderakhsh, "Fault diagnosis schemes for low-energy block cipher Midori benchmarked on FPGA," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 4, pp. 1528–1536, 2016.

[2] K. Järvinen, S. S. Roy and I. Verbauwhede, "Arithmetic of $$\tau$$-adic expansions for lightweight Koblitz curve cryptography," *Journal of Cryptographic Engineering*, vol. 8, no. 4, pp. 285–300, 2018.

[3] P. Kitsos, N. Sklavos, G. Provelengios and A. N. Skodras, "FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0," *Microprocessors and Microsystems*, vol. 37, no. 2, pp. 235–245, 2013.

[4] R. Chakraborty and J. K. Mandal, "An FPGA based non-feistel block cipher through recursive substitutions of bits on prime-nonprime detection of sub-stream," *Microsystem Technologies*, vol. 25, no. 5, pp. 1679–1687, 2019.

[5] H. M. El Hennawy, A. E. Omar and S. M. Kholaif, "LEA: Link encryption algorithm proposed stream cipher algorithm," *Ain Shams Engineering Journal*, vol. 6, no. 1, pp. 57–65, 2015.

[6] D. B. Roy, S. Bhasin, J. L. Danger, S. Guilley, W. He *et al.,* "The conflicted usage of RLUTs for security-critical applications on FPGA," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 162–178, 2018.

[7] A. Aysu, E. Gulcan and P. Schaumont, "SIMON says: Break area records of block ciphers on FPGAs," *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 37–40, 2014.

[8] A. Singh, N. Chawla, J. H. Ko, M. Kar and S. Mukhopadhyay, "Energy efficient and side-channel secure cryptographic hardware for IoT-edge nodes," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 421–434, 2018.

[9] V. Dahiphale, H. Raut and G. Bansod, "Design and Implementation of novel datapath designs of lightweight cipher rectangle for resource constrained environment," *Multimedia Tools and Applications*, vol. 78, no. 16, pp. 23659–23688, 2019.

[10] D. Chakraborty, C. Mancillas-López and P. Sarkar, "STES: A stream cipher based low cost scheme for securing stored data," *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2691–2707, 2014.

[11] F. Pirpilidis, L. Pyrgas and P. Kitsos, "8-bit serialised architecture of SEED block cipher for constrained devices," *IET Circuits, Devices & Systems*, vol. 14, no. 3, pp. 316–321, 2020.

[12] V. Dahiphale, G. Bansod, A. Zambare and N. Pisharoty, "Design and implementation of various datapath architectures for the ANU lightweight cipher on an FPGA," *Frontiers of Information Technology & Electronic Engineering*, vol. 21, no. 4, pp. 615–628, 2020.

[13] A. Aghaie, M. M. Kermani and R. Azarderakhsh, "Reliable and fault diagnosis architectures for hardware and software-efficient block cipher KLEIN benchmarked on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 4, pp. 901–905, 2017.

[14] R. Sadhukhan, S. Patranabis, A. Ghoshal, D. Mukhopadhyay, V. Saraswat *et al.,* "An evaluation of lightweight block ciphers for resource-constrained applications: Area, performance and security," *Journal of Hardware and Systems Security*, vol. 1, no. 3, pp. 203–218, 2017.

[15] N. Wu, Z. A. Ali, M. M. Shaikh, M. R. Yahya and M. Aamir, "Compact and high speed architectures of KASUMI block cipher," *Wireless Personal Communications*, vol. 106, no. 4, pp. 1787–1800, 2019.

[16] M. Mozaffari-Kermani, K. Tian, R. Azarderakhsh and S. Bayat-Sarmadi, "Fault-resilient lightweight cryptographic block ciphers for secure embedded systems," *IEEE Embedded Systems Letters*, vol. 6, no. 4, pp. 89–92, 2014.

[17] N. Thangamani and M. Murugappan, "A lightweight cryptography technique with random pattern generation," *Wireless Personal Communications*, vol. 104, no. 4, pp. 1409–1432, 2019.

[18] J. G. Pandey, T. Goel and A. Karmakar, "Hardware architectures for PRESENT block cipher and their FPGA implementations," *IET Circuits, Devices & Systems*, vol. 13, no. 7, pp. 958–969, 2019.

[19] T. De Cnudde and S. Nikova, "Securing the present block cipher against combined side-channel analysis and fault attacks," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 12, pp. 3291–3301, 2017.

[20] H. D. Azari and P. V. Joshi, "An efficient implementation of present cipher model with 80 bit and 128 bit key over FPGA based hardware architecture," *International Journal of Pure and Applied Mathematics*, vol. 119, no. 14, pp. 1825–1832, 2018.

[21] B. Rashidi, "Efficient and high-throughput application-specific integrated circuit implementations of hight and present block ciphers," *IET Circuits, Devices & Systems*, vol. 13, no. 6, pp. 731–740, 2019.

[22] C. A. Lara-Nino, A. Diaz-Perez and M. Morales-Sandoval, "Lightweight hardware architectures for the present cipher in FPGA," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2544–2555, 2017.