

Article

Implementation of Speed-Efficient Key-Scheduling Process of AES for Secure Storage and Transmission of Data

Thanikodi Manoj Kumar ¹, Kavitha Rani Balmuri ², Adam Marchewka ^{3,*} ,
Parameshchari Bidare Divakarachari ⁴  and Srinivas Konda ⁵

¹ Department of Electronics and Communication Engineering, Karpagam Institute of Technology, Coimbatore 641105, Tamil Nadu, India; manojkumar.pla@karpagamtech.ac.in

² Department of Information Technology, CMR Technical Campus, Hyderabad 501401, Telangana, India; kavitharani.cse@cmrtc.ac.in

³ Faculty of Telecommunications, Computer Science and Electrical Engineering, Bydgoszcz University of Science and Technology, 85-796 Bydgoszcz, Poland

⁴ Department of Telecommunication Engineering, GSSS Institute of Engineering and Technology for Women, Mysuru 570016, India; paramesh@gsss.edu.in

⁵ Department of Computer Science Engineering, CMR Technical Campus, Kandlakoya, Hyderabad 501401, India; Srinivas.cse@cmrtc.ac.in

* Correspondence: adimar@pbs.edu.pl

Abstract: Nowadays, a large number of digital data are transmitted worldwide using wireless communications. Therefore, data security is a significant task in communication to prevent cyber-crimes and avoid information loss. The Advanced Encryption Standard (AES) is a highly efficient secure mechanism that outperforms other symmetric key cryptographic algorithms using message secrecy. However, AES is efficient in terms of software and hardware implementation, and numerous modifications are done in the conventional AES architecture to improve the performance. This research article proposes a significant modification to the AES architecture's key expansion section to increase the speed of producing subkeys. The fork-join model of key expansion (FJMKE) architecture is developed to improve the speed of the subkey generation process, whereas the hardware resources of AES are minimized by avoiding the frequent computation of secret keys. The AES-FJMKE architecture generates all of the required subkeys in less than half the time required by the conventional architecture. The proposed AES-FJMKE architecture is designed and simulated using the Xilinx ISE 5.1 software. The Field Programmable Gate Arrays (FPGAs) behaviour of the AES-FJMKE architecture is analysed by means of performance count for hardware resources, delay, and operating frequency. The existing AES architectures such as typical AES, AES-PNSG, AES-AT, AES-BE, ISAES, AES-RS, and AES-MPPRM are used to evaluate the efficiency of AES-FJMKE. The AES-FJMKE implemented using Spartan 6 FPGA used fewer slices (i.e., 76) than the AES-RS.

Keywords: advanced encryption standard; data security; field programmable gate arrays; fork-join model of key expansion; hardware resources; propagation delay



Citation: Kumar, T.M.; Balmuri, K.R.; Marchewka, A.; Bidare Divakarachari, P.; Konda, S. Implementation of Speed-Efficient Key-Scheduling Process of AES for Secure Storage and Transmission of Data. *Sensors* **2021**, *21*, 8347. <https://doi.org/10.3390/s21248347>

Academic Editor: Massimo Ficco

Received: 22 October 2021

Accepted: 10 December 2021

Published: 14 December 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Nowadays, the growth of lightweight, robust, and effective encryption algorithms are required to provide network security for information technology applications. The developed encryption algorithms are essential for maximizing the throughput and data size of IoT, and it is used in mobile transmissions, video streaming, real-time communications, and so on [1–6]. The methods of encryption/decryption are classified into two types such as symmetric and asymmetric methods. In that, symmetric cryptography uses only one key for encryption and decryption, whereas asymmetric cryptography uses two distinct keys to accomplish the encryption and decryption [7–9]. Symmetric cryptography is extensively used among all cryptographic methods due to its low energy necessities and simplicity.

Hence, the symmetric block cipher, namely AES, was developed in 2001 by the National Institute of Standards and Technology (NIST); however, this AES is an alteration for the typical data encryption standard [10–13].

Some of block cipher algorithms used in the communication applications are Khudra [14], KASUMI [15], LRBC [16], PRESENT [17], SLIM [18], HIGHT [19], Simon [20], KLEIN [21], Midori [22], and CLEFIA [23]. However, the lightweight cipher affects the performance when the system has a huge number of devices during the communication [24]. Additionally, the lightweight block ciphers offer only a lower level of security than the conventional algorithms [25]. Authentication, confidentiality, and integrity are considered as the significant objectives of the cryptographic protocols. The AES is served as a significant cryptographic algorithm, whereas it satisfies the essential security goals of availability, confidentiality, and integrity during the communication on the insecure transmission medium [26,27]. Since the cryptographic process with an extreme computation complexity of ciphers avoids the key from the attempt of brute force [28]. The AES is implemented in different hardware platforms such as graphics processing units, embedded processors, ASIC, and FPGA because of its extensive utilization [29]. The configuration of hardware units using FPGA's reconfigurable logic resources is used for achieving high pipelining and parallelism. The balancing over the pipeline is obtained by adding and relocating the registers. The usage of multiple ports in the memory units is used to increase the speed of the read/write operations [30].

FPGA chips can operate simultaneously and it has a comparatively flexible architecture. Hence, the test cycles and design cost of the FPGA chips are lower [31–34]. Since two different LUT-based methods, namely substitution box (S-Box) and T-box, are used for an effective design of AES over the FPGA, LUT-based encryption and decryption are not only memory intensive but also asymmetric because of its transformation sequence and AES functions of encryption and decryption. Hence, the process of encryption and decryption are designed individually, and it occupies a significant amount of BRAM over the LUT-based AES [35]. However, the clock speed, library, and throughout are difficult to achieve by AES design due to its complexity, its user scheduling process, and the dynamic nature of its design. Since the S-box of the AES's sub-byte process consumes more power than the other modules of the circuit [36]. The multiplicative inversion used in the sub-byte transformation requires higher resources and finite field arithmetic [37].

Some of the conventional AES architectures are described as below:

Benhadjyoussef et al. [38] presented the fault-resistant method for securing the AES against attacks. Parity checking was used to develop the error detection for the time redundancy of subbytes function and linear operations. Specifically, the error detection code depends on the cyclic redundancy check that was used to identify the parity of the Shift Rows, Mix Columns, and Add Round Key functions. On the other hand, faults were inserted in the SubBytes transformation to identify the temporal redundancy. However, the information redundancy method caused high overhead, which affects the system performance. Sheikhpour, Ko, and Mahani, [39] developed the 32-bit AES encryption/decryption for IoT and resource-constrained applications. Here, the low-cost fault-resilient structure was developed for the data path. Subsequently, an on-the-fly key expansion unit was also designed for the key generation of encryption/decryption processes. Here, the area was minimized using resource-sharing among encryption and decryption operations. Sikka et al. [40] presented the design of the AES for automotive applications. In this work, the 128-bit key of AES was designed using the High-Level Synthesis (HLS) tool. Specifically, HLS was based on the bit widths while designing the AES over the FPGA. However, the re-computation of the signal width increased the overall latency of the AES algorithm.

Zodpe and Sapkal [41] presented the PN Sequence Generator (PNSG) for creating the S-box and initial keys for Encryption/Decryption. Here, the Linear Feedback Shift Register (LFSR) was used for designing the PNSG, whereas the LFSR was represented using the generator polynomial. The designed PNSG was used to offer different random number sequences by using the initial seed and feedback tap. The robustness of the

AES cryptography was enhanced by using the feature of PNSG. However, the design of AES using non-pipelined stages required high hardware resources. Shahbazi and Ko [42] presented the 128-bit of AES in counter mode for high traffic applications. Inner and outer pipelining methods were used to achieve high throughput, and an affine transformation (AT) method was designed to minimize the area. The developed affine transformation was the hybrid method of affine and inverse isomorphic transformation. In AES, the operations of Sub-Bytes and Shift Rows were swapped, and then Add Round Key was combined with the Shift Rows. Moreover, the Mix Column operation was divided into two distinct phases for achieving the latency. However, the accomplishment of mix column in one clock caused high latency. Madhavapandian and MaruthuPandi [43] developed a 128-bit AES cryptographic method for securing the Transmission Control Protocol/Internet Protocol (TCP/IP). An effective mix column Boolean Expression (BE) using gate replacement and resource sharing structure was used to modify the mix column operation. Accordingly, the optimized architecture of AES was used to minimize power consumption. However, it was required that the time complexity be minimized because it increased the delay during the communication using TCP/IP protocol.

Arul et al. [44] developed the Iterative Structure of the AES (ISAES) for lessening the hardware resources. The architecture of renovated S-box was used in the AES to minimize the area. Here, the usage of LUT in the composite field arithmetic was accomplished in the multiplication operation. Moreover, the Vedic multiplier was employed in the Mix Column transformation, which was used to decrease the hardware resources of AES. However, the operating frequency of the AES was less because of the high delay. Wegener et al. [45] developed AES S-box by using the function of the Rotational Symmetry (RS). In this work, AES was designed by using the internal MUXes and slice registers, and this AES does not require any Block RAM (BRAM). The Boolean masking with a less amount of two shares over an AES's decomposition was applied to generate the higher robustness against the attacks. Here, the masked AES design was used to optimize the LUT implementation. However, the replication of linear operators and their independent operation was increased the overall area of AES. Kumar [46] developed the architecture of MPPRM for designing the AES's SubBytes/InvSubBytes transformation. These transformations were utilized for designing the subpipelining architecture. The hardware resources such as AND and XOR gates were reduced using the MPPRM in SubBytes and InvSubBytes transformations. Here, a 128-bit key was generated by the key expansion structure, and this key was given to the subpipelined structure. Due to the utilization of the delay module in the output of AND gate, a high-speed encryption/decryption was achieved in AES. Here, the AES's area was increased because of the recurrent key generation in the encryption process.

The problems of the conventional AES are stated as follows: The frequent computation of input, output, and intermediate signal width leads to an increase in the delay of the AES [40]. The hardware resources also increased because of AES design using non-pipelined stages [41]. Due to the high delay, the operating frequency is decreased in the AES architecture [44]. The area of the overall AES is increased because of the replication of linear function and independent operation [45]. Hence, the AES is developed with an effective FJMKE architecture to avoid the aforesaid issues. The FJMKE architecture is used to create the multiple subkeys simultaneously, which helps to decrease the delay in the AES. The generation of multiple subkeys using the FJMKE leads to reducing the combinational logics as it avoids frequent calculation subkeys.

The conventional AES architecture generates the subkeys according to the previous step subkeys, whereas the proposed AES-FJMKE architecture generates the subkeys only based on the main key that helps to reduce the overall propagation delay.

The research contributions are as follows:

- In this research, the FJMKE architecture is used to generate the subkeys in a parallel way, whereas the generation of the subkeys does not depend on the subkeys from the previous step this leads to minimize the propagation delay. This multiple subkey generation decreases the delay while encrypting the plain text.

- The combinational logic of the overall AES is minimized by avoiding the frequent computation of secret keys using FJMKE architecture, which lessens AES's resources.
- There are six different FPGA devices, namely Virtex 4, Virtex 5, Virtex 6, Spartan 3, Spartan 6, and Kintex 7, that are used for analysing the AES-FJMKE architecture.

The organization of this research article is as follows. Section 2 explains the typical AES architecture. Section 3 describes the AES-FJMKE architecture in detail. The performance evaluation of the AES-FJMKE architecture is discussed in Section 4. Further, Section 5 discusses the research's conclusion and future endeavours.

2. Related Works

This section provides information about the conventional AES algorithm along with its encryption and decryption processes.

2.1. Advanced Encryption Standard

In the AES cryptographic algorithm [47–51], only one secret key is employed for encrypting and decrypting plain texts. AES can be classified as AES-128, AES-192, or AES-256, depending on the key sizes used in encryption and decryption operations, whereas the number denotes the number of bits that exist in the secret key in the AES versions. All versions of the AES process input plain text in a step-by-step fashion. The number of rounds for AES varies between versions and is dependent on the key size, as shown in Table 1.

Table 1. Types of AES algorithm.

AES Types	Key Sizes	Rounds (Nr)	No. of Key (Nr + 1)
AES-128	128	10	11
AES-192	192	12	13
AES-256	256	14	15

Separate subkeys are used for each round of AES operation. Indeed, these subkeys are generated from the primary original key via a process called key scheduling. In all three AES variants, the processing of input data is 128 bits. The term “number of rounds” refers to the number of times a single data block is encrypted and decrypted using different subkeys (one at a time) obtained during the key expansion process. All operations are identical throughout the operation's rounds. Before beginning the encryption operation, a pre-round transformation is performed using the primary original secret key, and the other subkeys are utilized in each round.

2.2. Encryption and Decryption of AES

AES uses a secret key to encrypt a 128-bit of plain text to generate the ciphertext. The AES performs all four operations in all rounds except the last round. The final round of encryption and the initial round of decryption will not use Mix Columns. A plain text string of 128 bits in length is ordered as a 4×4 state matrix, with each element represented by a byte.

- Substitution;
- Shift Rows;
- Mix columns;
- Add round key.

2.2.1. Substitution

A byte value is substituted for other bytes in this process. The AES algorithm contains only one non-linear process: substitution. The core processes of substitution are matrix multiplication and affine transformation. By replacing the Rijndael S-box byte value directly, the decryption process employs inverse S-box substitution.

2.2.2. Shift Rows

The second, third, and fourth rows of the state matrix are left shifted 1, 2, and 3 times, respectively. The first row of the state matrix remains unchanged. In addition, the right shift operation is carried out on the rows during the decryption process.

2.2.3. Mix Columns

In this phase, the alteration is performed in the column. The simple function is matrix multiplication. Equations (1) and (2) shows the function of Mix column and Inv Mix column.

$$\begin{bmatrix} S'1 \\ S'2 \\ S'3 \\ S'4 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S1 \\ S2 \\ S3 \\ S4 \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} S'1 \\ S'2 \\ S'3 \\ S'4 \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S1 \\ S2 \\ S3 \\ S4 \end{bmatrix} \quad (2)$$

where $S'1, S'2, S'3, S'4$ are the output obtained after mix column operation and $S1, S2, S3, S4$ are the input given to the mix column process.

2.2.4. Add Round Key

The secret key performs its actual function during the Add Round Key step. All the preceding operations are easily reversible. Before initiating the Add Round Key process, the secret and all subkeys generated during the key expansion process are organized as a 4×4 state matrix. The Add Round Key step's core process is the modulo EXOR addition between the key's state matrix and the output of the mix column operation. The ciphertext is formed by the output of the added round key of the final round of encryption. These four operations are repeated according to the number of rounds of operation specified for the various AES versions. A separate subkey must be used for each round of operation, and the subkey information must be kept confidential. The main secret key is identified easily when the unauthorized person knows about the subkey's information. This research focuses primarily on the critical scheduling process.

2.3. Existing Key Expansion Architecture

AES-128 is considered in this research, and the detailed key scheduling process for conventional AES-128 is explained in detail. The critical scheduling process for AES-128 is depicted in Figure 1. All operations in the AES key expansion process are performed at the word level. Thus, the 128-bit primary secret key is divided into four 32-bit words. As shown in Table 1, the encryption operation for AES-128 requires ten subkeys. The most intriguing aspect of this key expansion architecture is the interdependence of the subkeys, which prevents subkey predictability.

Due to the fact that mathematical operations are performed on words, a subkey can be created by combining four consecutive words. For instance, in Figure 1, $W4, W5, W6,$ and $W7$ are combined to create the first subkey. AES 128 generates 44 words as a result of the key scheduling process. The first four words are derived from the main one and are used in the round prior to transformation. The remaining 40 words are divided into 10 distinct subkeys. $T4$ denotes a temporary word. $T8$ denotes a permanent word. The key expansion architecture is nonlinear due to the generation process of temporary words.

The operations for creating temporary words include S-Box substitution, word rotation, and the EXOR operation with a constant value. The following equation explains how temporary words are created, and Table 2 lists the R constant values ($Rcon$) for each

round. *Temporary words* are formed from the previous subkey's final word, as shown in Equation (3).

$$\text{Temporary word} = \text{Subword}(\text{Rotword}(W_{i-1})) \oplus \text{Rcon} \quad (3)$$

where *Rotword* is used to perform onebyte circular shift on the word (W_i).

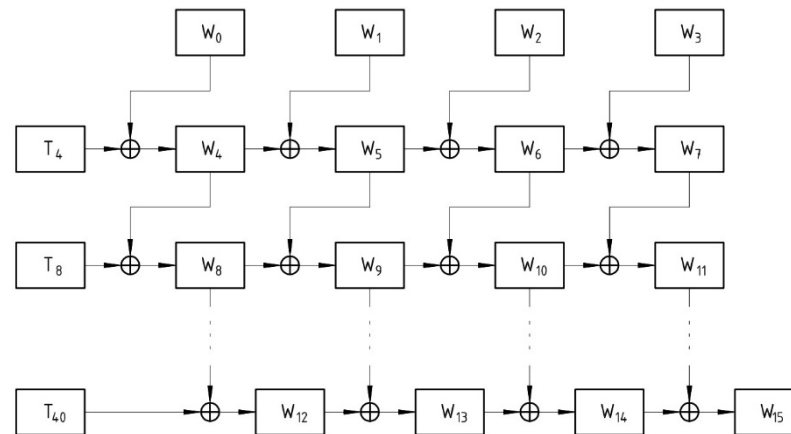


Figure 1. Typical key expansion architecture.

Table 2. R constant values for different rounds in AES–128.

Round	Rcon	Round	Rcon
1	(01000000)16	6	(20000000)16
2	(02000000)16	7	(40000000)16
3	(04000000)16	8	(80000000)16
4	(08000000)16	9	(1B000000)16
5	(10000000)16	10	(36000000)16

3. AES-FJMKE Architecture

In the proposed AES-FJMKE architecture, an effective key expansion design is developed using the fork-join model. The developed FJKME is used to generate multiple keys at a time. These multiple keys are used to encrypt different sets of plain text that help to minimize the delay. Accordingly, the multiple key generations using FJKME lead to minimizing the combinational blocks of overall AES architecture. Moreover, the other operations, namely Substitution, Shift Rows, Mix Columns, and Add Round Key are similar to the conventional AES architecture. The main objective of this AES-FJMKE architecture is to achieve less propagation delay while achieving security. The overall architecture of AES using FJKME is illustrated in Figure 2.

FJMKE Architecture

The nominal AES employs a sequential key-scheduling process that generates all subkeys sequentially. The concurrent generation of subkeys is not possible due to the dependency of the temporary word for each subkey over the final word of the preceding subkey. However, interdependence between subkeys is critical in maintaining the secrecy of subkeys. In a traditional architecture, the final word is generated after all previous words have been generated, which is a time-consuming process. With the modifications made to the key scheduling process, the AES-FJMKE architecture aims to reduce time delays. Sequential processes can be made simultaneous by incorporating additional circuitry into the conventional architecture. All subkeys are concurrently generated in this AES architecture, and the current subkey does not require waiting for the previous subkey generation. The time required to generate the subkeys is minimized by using the FJMKE. The structure of FJMKE within the block remains sequential; therefore, the subkeys are

sequentially generated in AES. The time consumption for generating the subkeys in AES is high when the overall key expansion is performed at one time. Therefore, the entire architecture of AES is split into two parallel blocks in this fork-join model to reduce the time required to generate subkeys. For AES-128, the first block generates the first five subkeys, while the second block generates the remaining five subkeys. This architecture differs slightly from conventional architecture in that the sixth subkey is dependent on the main key rather than the fifth subkey. In conventional AES architecture, the dependency between the subkeys is high, while generating the subkeys for successive rounds increases the delay. However, the designed FJMKE architecture only depends on the main key during the subkey generation, which lessens the propagation delay for the AES. When compared to the conventional architecture, this modification reduces the total execution time for generating ten subkeys by half. After completing the subkey generation, the subkeys from these two blocks are concatenated together as ten subkeys. From these ten subkeys (i.e., ten different outputs), each subkey is taken for each round to accomplish the encryption/decryption processes. Figure 3 shows the designed key expansion architecture of AES.

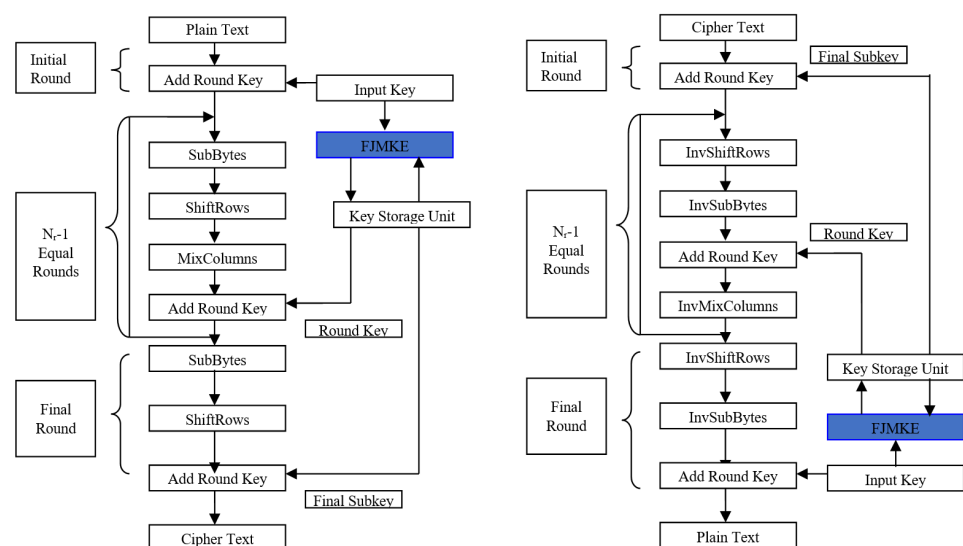


Figure 2. Architecture of AES-FJMKE.

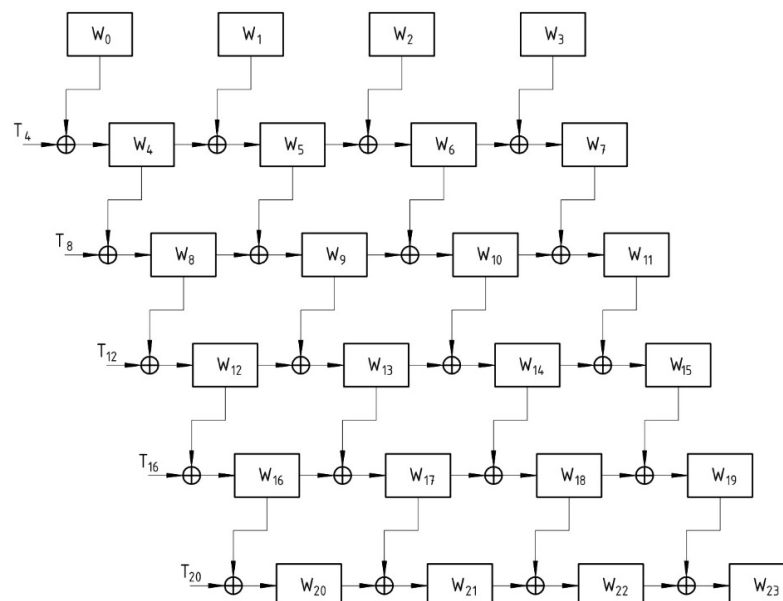


Figure 3. FJMKE for AES-128.

For example, the generation of W_4 , W_5 , W_6 , and W_7 in the FJMKE is illustrated in Figure 4. In conventional AES architecture, the generation of W_6 depends on the W_5 computed from the previous step. Therefore, the generation of current subkeys has to wait until the completion of previous step subkey generation. Hence, the subkey generation of the conventional AES requires more clock cycles. On the other hand, the proposed FJMKE architecture generates all subkeys in a parallel way without waiting for any subkey from the previous step, which minimizes the propagation delay. Specifically, the generation of W_6 does not require waiting until the completion of W_5 generation. On the contrary, the designed FJMKE architecture generates the subkeys by using main secret keys, which lessens the propagation delay. For example, the conventional AES requires four clock cycles for generating the subkeys W_4 , W_5 , W_6 , and W_7 , while the FJMKE requires only one clock cycle for generating the subkeys W_4 , W_5 , W_6 , and W_7 as it performs concurrent subkey generation. The logical elements used in the AES-FJMKE architecture is slightly higher than the conventional AES which is within an acceptable level. However, this slight increment in logics does not create an impact on the overall AES-FJMKE architecture because the designed FJMKE based key expansion is used only one time during the encryption/decryption processes. Therefore, the interdependence between subkeys is avoided and the propagation delay for producing subkeys is significantly minimized with the help of a circuit built on the basis of the following Equations (4)–(7).

$$W_4 = T_4 \oplus W_0 \quad (4)$$

$$W_5 = T_4 \oplus W_0 \oplus W_1 \quad (5)$$

$$W_6 = T_4 \oplus W_0 \oplus W_1 \oplus W_2 \quad (6)$$

$$W_7 = T_4 \oplus W_0 \oplus W_1 \oplus W_2 \oplus W_3 \quad (7)$$

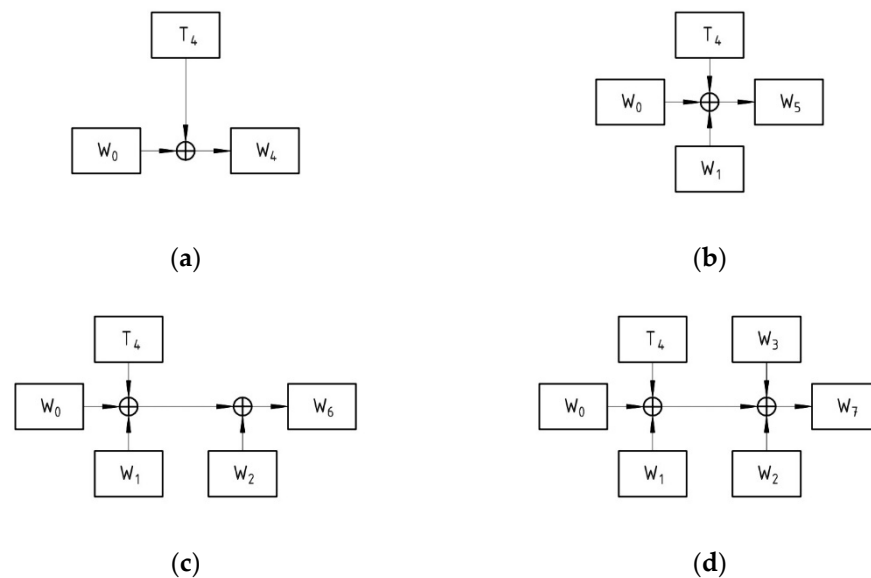


Figure 4. Subkeys generation, (a) W_4 , (b) W_5 , (c) W_6 , and (d) W_7 .

Similarly, multiple keys are generated from the designed FJMKE architecture, and it is used to accomplish the encryption over multiple plain texts. For example, the generation of W_{15} using the FJMKE architecture is expressed in the following Equation (8).

$$\begin{aligned} W_{15} = & T_{12} \oplus T_8 \oplus T_4 \oplus W_0 \oplus T_8 \oplus T_4 \oplus W_0 \oplus T_4 \oplus W_0 \\ & \oplus W_1 \oplus T_8 \oplus T_4 \oplus W_0 \oplus T_4 \oplus W_0 \oplus W_1 \oplus T_4 \\ & \oplus W_0 \oplus W_1 \oplus W_2 \oplus T_8 \oplus T_4 \oplus W_0 \oplus T_4 \oplus W_0 \oplus W_1 \\ & \oplus T_4 \oplus W_0 \oplus W_1 \oplus W_2 \oplus T_4 \oplus W_0 \oplus W_1 \oplus W_2 \oplus W_3 \end{aligned} \quad (8)$$

The concept underlying the preceding expression is to perform all necessary mathematical operations at each and every step. Rather than using the result from the previous step, the result is obtained by recalculating. While this process requires more space and energy, the total time required to produce the end result is reduced. In a conventional architecture, each step or process of subkey generation must be delayed until the previous subkey generation process is complete. However, in this AES-FJMKE architecture, there is no requirement to wait until the previous word of the subkey is complete before beginning the generation of the current word of the subkey. This significantly reduces the propagation delay associated with generating the required number of subkeys. The model described above can be extended to include all ten subkey generation processes.

The architecture of the modified subkey generation process using the FJMKE is illustrated in Figure 4. All four words in the first subkey, W_4 , W_5 , W_6 , and W_7 , can be generated concurrently. There is no reason to delay the process of generating words W_5 until the block that generates W_4 is executed. Additionally, processes for generating W_6 and W_7 can be started concurrently with the process for generating words W_4 . This property qualifies this architecture for applications that require rapid execution. The AES-FJMKE architecture retains all of the conventional algorithm's diffusion and confusion operations.

4. Results and Discussion

The results of the proposed AES-FJMKE are described in this section. The synthesis and simulation of the AES-FJMKE architecture is done using the Xilinx ISE 5.1 software. Here, the Hardware Description Language (HDL) is utilized for designing the AES architecture. For this AES architecture, the FJMKE architecture is developed for lessening the delay while decreasing the hardware resources. Moreover, the developed AES-FJMKE architecture is employed for processing the 128-bit of plaintext.

4.1. Performance Evaluation for AES-FJMKE

The design and evaluation of AES-FJMKE is made by using six distinct FPGA devices, namely Virtex 4, Virtex 5, Virtex 6, Spartan 3, Spartan 6, and Kintex 7. The performance of the AES-FJMKE is evaluated by means of performance count for hardware resources, delay, and operating frequency. The evaluation of the results of the AES for different FPGA devices is shown in the tables below.

The hardware utilization analysis of the AES-FJMKE developed in the Virtex, Spartan, and Kintex devices are shown in Tables 3–5, respectively. Moreover, the delay and operating frequency evaluation for the AES-FJMKE is shown in Table 6, where the delay and operating frequency are the time consumption-related-parameters. The results shown in the analysis are taken for the AES with 128-bit cryptography. From Tables 3–5, it is known that the designed AES-FJMKE consumes 1–67% of resources during the implementation. On the other hand, the operating frequency for the AES-FJMKE designed in the Virtex 5 is 751.247 MHz, which is higher than the other FPGA devices. The higher operating frequency is achieved by avoiding frequent computation of keys during the encryption/decryption processes. For verification purposes, the generation of multiple keys using a single input is illustrated in the simulation waveform of Figure 5. Here, the simulation waveform of key generation is taken for the Virtex 4 FPGA device. In Figure 5, the input is represented as W , and multiple subkeys are represented as k_1 – k_{10} . From the analysis, we conclude that the FJMKE offers distinct subkeys for each input value. There is no similarity between the generated subkey values of FJMKE. Moreover, the overall simulation waveform for the AES-FJMKE architecture is shown in Figure 6. There, the plain text, secret key and cipher text are represented as *plain*, *key*, and *cipher*, respectively. The generated cipher text for the plain text with different secret keys is shown in Table 7. Table 7 shows that the AES-FJMKE architecture offers different cipher text for the same plain text according to the secret key. Hence, it is proved that the AES-FJMKE architecture offers higher robustness against the unauthorized users. Accordingly, the difficulty of accessing the data during the communication is difficult by the unauthorized users.

Table 3. Analysis of used resources for AES-FJMKE designed in Virtex devices.

Virtex FPGA Devices	FPGA Performances	Used Resources	Available Resources	Total Usage (%)
Virtex 4 FPGA	Number of slice registers	8452	10,944	77.22
	Flip Flops	8452	10,944	77.22
	Number of slice LUTs	7415	10,944	67.75
	Number of logical elements	7415	10,944	67.75
	Slices	3847	5472	70.3
Virtex 5 FPGA	Bonded IOB	135	240	56.25
	Number of slice registers	18,237	28,800	63.32
	Flip Flops	18,237	28,800	63.32
	Number of slice LUTs	14,011	28,800	48.64
	Number of logical elements	14,011	28,800	48.64
Virtex 6 FPGA	Slices	4850	7200	67.36
	Bonded IOB	102	480	21.25
	Number of slice registers	2700	93,120	2.89
	Flip Flops	2700	93,120	2.89
	Number of slice LUTs	8269	46,560	17.75
Virtex 6 FPGA	Number of logical elements	8254	46,560	17.72
	Slices	966	11,640	8.29
	Bonded IOB	89	240	37.08

Table 4. Analysis of used resources for AES-FJMKE designed in Spartan devices.

Spartan FPGA Devices	FPGA Performances	Used Resources	Available Resources	Total Usage (%)
Spartan 3 FPGA	Number of slice registers	523	3840	13.61
	Flip Flops	541	3840	14.08
	Number of slice LUTs	1859	3840	48.41
	Number of logical elements	1859	3840	48.41
	Slices	972	1920	50.62
Spartan 6 FPGA	Bonded IOB	58	141	41.13
	Number of slice registers	78	18,224	1
	Flip Flops	81	18,224	1
	Number of slice LUTs	189	9112	2.07
	Number of logical elements	197	9112	2.16
Spartan 6 FPGA	Slices	76	2278	3.33
	Bonded IOB	123	18,224	1

Table 5. Analysis of used resources for AES-FJMKE designed in Kintex 7 devices.

FPGA Performances	Used Resources	Available Resources	Total Usage (%)
Number of slice registers	7087	82,000	8.64
Flip Flops	7074	82,000	8.62
Number of slice LUTs	8104	41,000	19.76
Number of logical elements	8104	41,000	19.76
Slices	451	10,250	4.4
Bonded IOB	204	300	68

Table 6. Examination of delay and operating frequency for AES- FJMKE.

FPGA Devices	Delay (ns)	Operating Frequency (MHz)
Virtex 4	14.568	521.730
Virtex 5	2.402	751.247
Virtex 6	3.133	449.309
Spartan 3	3.229	101.491
Spartan 6	1.916	210.433
Kintex 7	2.540	97.308

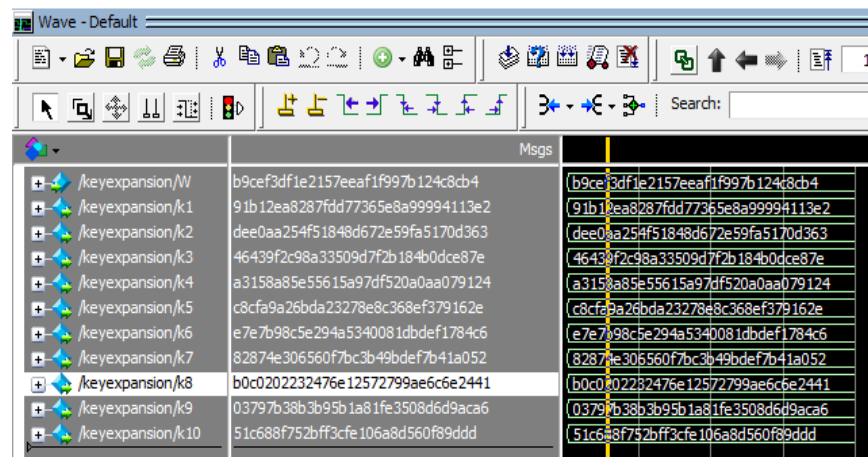


Figure 5. Simulation waveform of generated keys of AES-FJMKE designed in Virtex 4.

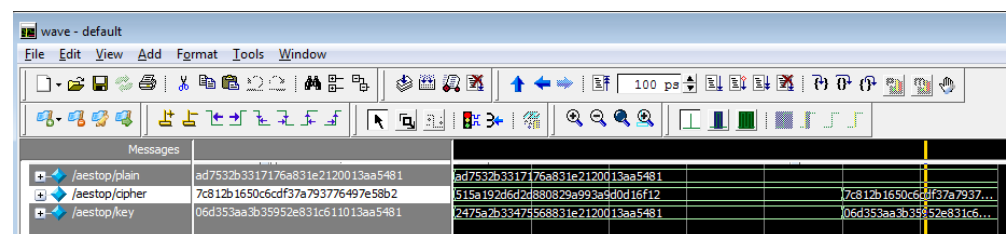


Figure 6. Overall simulation waveform of AES-FJMKE architecture.

Table 7. Results of cipher text for AES-FJMKE architecture.

Plain Text	Secret Key	Cipher Text
AD7532B3317176A831E2120013AA5481	2475A2B33475568831E2120013AA5481	515A192D6D2D880829A993A9D0D16F12
AD7532B3317176A831E2120013AA5481	6D353AA3B35952E831C611013AA5481	7C812B1650C6CDF37A793776497E58B2

4.2. Comparative Evaluation

A comparison of the AES-FJMKE architecture is presented in this section. In this research, six different existing methods, namely AES-PNSG [41], AES-AT [42], typical AES [43], AES-BE [43], ISAES [44], AES-RS [45], and AES-MPPRM [46], are used to evaluate the AES-FJMKE architecture. Here, the evaluation is performed using six FPGA devices such as Virtex 4, Virtex 5, Virtex 6, Spartan 3, Spartan 6, and Kintex 7. The evaluation among the AES-FJMKE architecture and existing methods are provided as follows.

Tables 8–13 shows the evaluation of the AES-FJMKE with AES-PNSG [41], AES-AT [42], typical AES [43], AES-BE [43], ISAES [44], AES-RS [45], and AES-MPPRM [46], respectively. Tables 8–13 compare the data of Virtex 4, Virtex 5, Virtex 6, Spartan 3, Spartan 6, and Kintex 7, respectively. Finally, the graphical illustration for the slice LUTs is shown in Figure 7. From the analysis, it is known that the AES-FJMKE architecture provides better performance in terms of hardware utilization, delay, and operating frequency. The slice registers of AES-FJMKE designed in the Virtex 6 is slightly high than the typical AES [43] and AES-BE [43]. From Table 10, it is noted that there are 2700 slice registers used in the AES-FJMKE architecture, which is higher than the number of slice registers of typical AES [23], i.e., 2688. However, the delay achieved by the AES-FJMKE architecture is 3.133 ns, which is less when compared to the typical AES [23], i.e., 3.205 ns. Hence, it is proved that the AES-FJMKE architecture achieves less delay than the typical AES architecture due to its concurrent generation of subkeys using FJMKE architecture. However, the increment in slice register is only at a negligible level that does not create any huge impact in terms of overall performances of AES-FJMKE. The reasons for the poor performance of existing architectures are specified as follows: the AES-PNSG [41] requires higher hardware

resources, because of the non-pipelined stages-based AES implementation. Next, the AES-AT [42], AES-BE [43], and ISAES [44] have resulted in a high delay while encrypting and decrypting the plain texts. Further, the design of AES-MPPRM [46] requires a frequent computation of secret keys to accomplish encryption/decryption processes, which increases the hardware resources of the overall AES. However, a simultaneous generation of multiple keys using the AES-FJMKE architecture leads to minimize the delay as well as helps to reduce the combinational logics during the key generation. Therefore, FJMKE architecture is better than the existing AES architecture, because the generation of subkeys in the FJMKE does not consume much time as it is only depends on the main secret key. On the other hand, the subkey generation of conventional AES architectures mainly depends on the subkey from the previous step. So, the current subkey generation of conventional AES architectures has to wait until the completion of previous step subkey generation, which increases the propagation delay. Moreover, the key expansion using FJMKE is performed only one time during the encryption/decryption processes which further decreases the propagation delay. Therefore, the developed AES-FJMKE architecture achieved less delay and less hardware resources while maintaining the security of the plaintext.

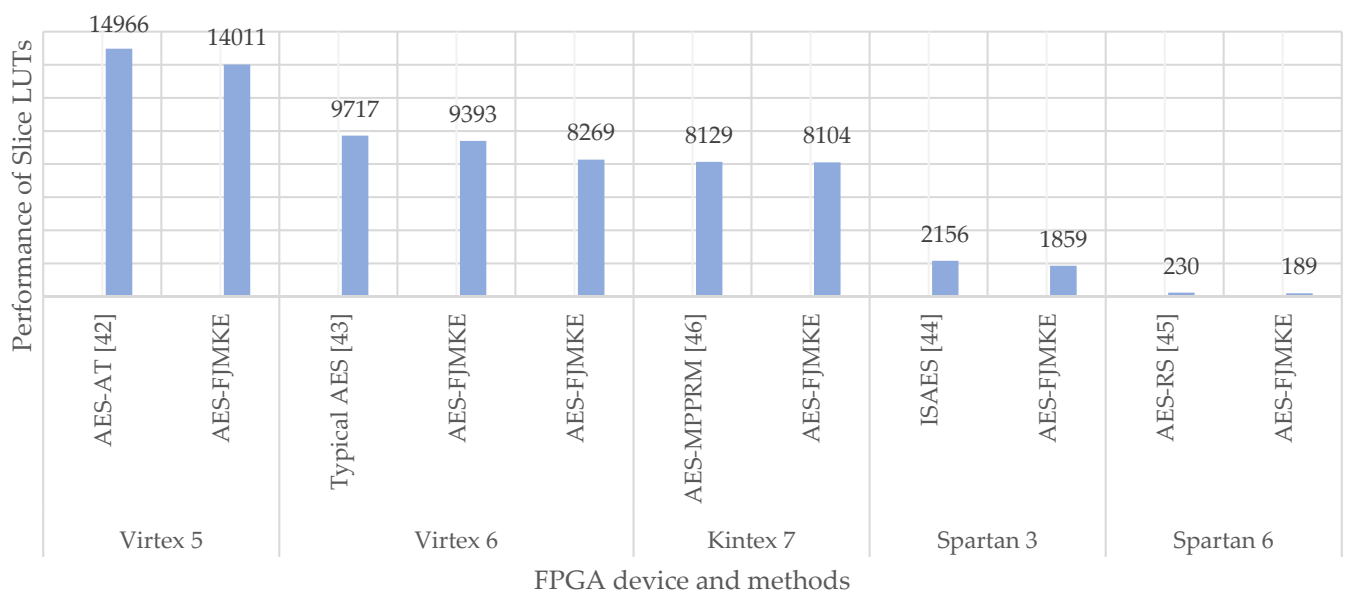


Figure 7. Graphical illustration of Slice LUTs.

Table 8. Evaluation of AES-FJMKE and AES-PNSG for Virtex 4.

Performances	AES-PNSG [41]	AES-FJMKE
Operating frequency (MHz)	214.48	521.730
Slices	20,818	2592

Table 9. Evaluation of AES-FJMKE and AES-AT for Virtex 5.

FPGA Performances	AES-AT [42]	AES-FJMKE
Operating frequency (MHz)	622.4	751.247
Slice LUTs	14,966	14,011
Slice registers	19,123	18,237
Slices	5974	4850

Table 10. Comparison of AES-FJMKE and AES-BE for Virtex 6.

Performances	Typical AES [43]	AES-BE [43]	AES-FJMKE
Operating frequency (MHz)	312.061	315.806	449.309
Delay (ns)	3.205	3.167	3.133
Slice LUTs	9717	9393	8269
Slice registers	2688	2688	2700

Table 11. Comparison of AES-FJMKE and ISAES for Spartan 3.

FPGA Performances	ISAES [44]	AES-FJMKE
Operating frequency (MHz)	67.75	101.491
Slices	1132	972
Slice LUTs	2156	1859
Flip Flops	680	541
IoB	389	58

Table 12. Comparison of AES-FJMKE and AES-RS for Spartan 6.

FPGA Performances	AES-RS [45]	AES-FJMKE
Operating frequency (MHz)	120	210.433
Slices	108	76
Slice LUTs	230	189
Flip Flops	92	81

Table 13. Comparison of AES-FJMKE and AES-MPPRM for Kintex 7.

Performances	AES-MPPRM [46]	AES-FJMKE
Operating frequency (MHz)	81.328	97.308
Delay (ns)	2.982	2.540
Slice registers	7120	7087
Flip Flops	7119	7074
Slice LUTs	8129	8104
Logical elements	8129	8104
Slices	467	451
Bonded IOB	211	204

4.3. Case Study

The AES-FJMKE architecture is developed to achieve the secure broadcasting of the human leg X-ray image. The pixel value of the input image is obtained using the MATLAB 6.5 programming software whereas these pixels are used as input data for the AES-FJMKE architecture. Since the input pixels are in the range of 0 to 255, the input pixels are $\{6c, 9d, 99, \dots, 4e\}$. Figure 8 shows the input X-ray image and its histogram representation. Next, the *dec2bin* command is used to convert the input pixel values into binary values which are encrypted using the AES-FJMKE architecture. From the AES-FJMKE, the output is acquired in a hexadecimal form that has an 8-bit size. Next, this ciphertext is securely broadcasted worldwide. Hence, the recovery of input data by an unauthorized person is difficult because of the effective encryption processed by the AES-FJMKE. The encrypted pixels are $\{fc, fc, fd, \dots, 2\}$ and its corresponding encrypted and histogram image is shown in Figure 9. The ciphertext values are transformed into original pixel values in the receiving end by using the same AES-FJMKE architecture. Next, the decrypted pixels are given to MATLAB to develop the input image. From the encrypted and histogram images, it is concluded that the AES-FJMKE architecture offers a high level of security.

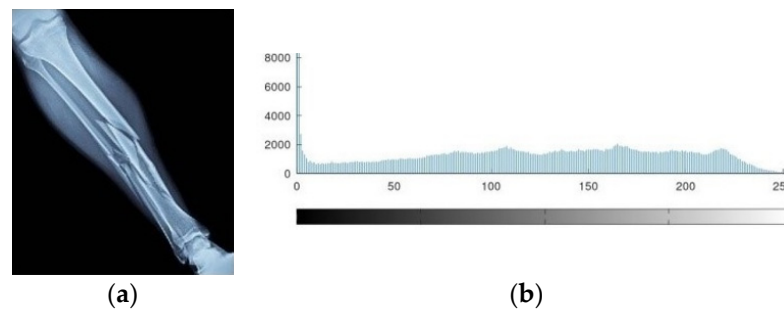


Figure 8. Input data, (a) Leg X-ray image, (b) Histogram.

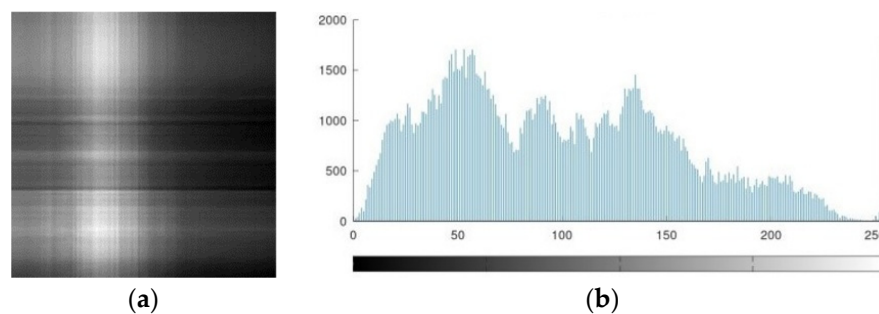


Figure 9. Output data, (a) encrypted image, (b) histogram.

In FJMKE architecture, the generation of the subkeys mainly depends on the main key; therefore, the subkeys are simultaneously generated while accomplishing the encryption/decryption processes. Accordingly, the propagation delay is minimized for the overall architecture. However, the logical elements of the AES-FJMKE architecture is slightly higher than the typical AES architecture, whereas this slight increment in the resources does not create any huge impact in overall performances, because the designed key expansion using FJMKE is accomplished only one time during the encryption/decryption processes. Moreover, the designed FJMKE provides a unique subkeys for each input value; hence, there is similarity among the generated subkeys. Accordingly, the AES-FJMKE architecture provides high robustness against the unauthorized users.

5. Conclusions

The purpose of this research is to propose a fast and efficient key-scheduling process for the AES algorithm. This AES-FJMKE architecture generates the required number of subkeys at a faster rate with minimal chip area sacrifice. This AES-FJMKE architecture ensures that the security of the messages processed does not affect the original plaintext. Therefore, the simultaneous multiple subkey generation using the AES-FJMKE architecture helps to decrease the delay. On the other hand, this multiple subkey generation is employed for decreasing the combinational logics by avoiding the frequent computation of subkeys. Hence, the AES-FJMKE architecture minimizes the delay and hardware utilization of the AES. Accordingly, the lesser delay in the computation process improves the operating frequency of the AES-FJMKE. The AES-FJMKE provides better performance than the AES-PNSG, AES-AT, typical AES, AES-BE, ISAES, AES-RS, and AES-MPPRM. The AES-FJMKE implemented using Spartan 6 FPGA used fewer slices (i.e., 76) than the AES-RS. However, without affecting the area's consumption, the propagation delay of the key scheduling process can be further reduced. This can be accomplished by incorporating optimization techniques into other transformations, such as pipelined generation of subkeys and optimization of the temporary word generation process using the S-box implementation. The developed AES can be effectively encrypted and decrypt an entire nation's sensitive passport information.

Author Contributions: The paper investigation, resources, data curation, writing—original draft preparation, writing—review and editing, and visualization were done by T.M.K. and K.R.B. The paper conceptualization, software, were conducted by S.K. The validation, and formal analysis, methodology, supervision, project administration, and funding acquisition of the version to be published were conducted by A.M. and P.B.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analysed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Visconti, P.; Capoccia, S.; Venere, E.; Velázquez, R.; Fazio, R.D. 10 Clock-Periods Pipelined Implementation of AES-128 Encryption-Decryption Algorithm up to 28 Gbit/s Real Throughput by Xilinx Zynq UltraScale+ MPSoC ZCU102 Platform. *Electronics* **2020**, *9*, 1665. [\[CrossRef\]](#)
2. Li, Y.; Kasuya, M.; Sakiyama, K. Comprehensive Evaluation on an ID-Based Side-Channel Authentication with FPGA-Based AES. *Appl. Sci.* **2018**, *8*, 1898. [\[CrossRef\]](#)
3. Lara-Nino, C.A.; Diaz-Perez, A.; Morales-Sandoval, M. Lightweight elliptic curve cryptography accelerator for internet of things applications. *Ad Hoc Netw.* **2020**, *103*, 102159. [\[CrossRef\]](#)
4. Benhadjoussef, N.; Karmani, M.; Machhout, M. Power-based Side-Channel Analysis against AES Implementations: Evaluation and Comparison. *Int. J. Comput. Sci. Netw. Secur.* **2020**, *21*, 265.
5. Xing, B.; Wang, D.; Yang, Y.; Wei, Z.; Wu, J.; He, C. Accelerating DES and AES Algorithms for a Heterogeneous Many-core Processor. *Int. J. Parallel Program.* **2021**, *49*, 463–486. [\[CrossRef\]](#)
6. Yang, C.H.; Chien, Y.S. FPGA Implementation and Design of a Hybrid Chaos-AES Color Image Encryption Algorithm. *Symmetry* **2020**, *12*, 189. [\[CrossRef\]](#)
7. Saranya, K.; Vijeyakumar, K.N. A Low Area FPGA Implementation of Reversible Gate Encryption with Heterogeneous Key Generation. *Circuits Syst. Signal Process.* **2021**, *40*, 3836–3865. [\[CrossRef\]](#)
8. Subramanian, K.; Venkatachalam, M.; Saroja, M. Adaptive Counter Clock Gated S-Box Transformation Based AES Algorithm of Low Power Consumption and Dissipation in VLSI System Design. *J. Phys. Conf. Ser.* **2021**, *1979*, 012066. [\[CrossRef\]](#)
9. Garipcan, A.; Erdem, E. Design, FPGA implementation and statistical analysis of a high-speed and low-area TRNG based on an AES S-box post-processing technique. *ISA Trans.* **2021**, *117*, 160–170. [\[CrossRef\]](#)
10. Shanthi Rekha, S.; Saravanan, P. Low-cost AES-128 implementation for edge devices in IoT applications. *J. Circuits Syst. Comput.* **2019**, *28*, 1950062. [\[CrossRef\]](#)
11. Chong, K.S.; Ng, J.S.; Chen, J.; Lwin, N.K.Z.; Kyaw, N.A.; Ho, W.G.; Chang, J.; Gwee, B.H. Dual-Hiding Side-Channel-Attack Resistant FPGA-Based Asynchronous-Logic AES: Design, Countermeasures and Evaluation. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2021**, *11*, 343–356. [\[CrossRef\]](#)
12. Mishra, Z.; Acharya, B. High throughput and low area architectures of secure IoT algorithm for medical image encryption. *J. Inf. Secur. Appl.* **2020**, *53*, 102533. [\[CrossRef\]](#)
13. Hafsa, A.; Sghaier, A.; Malek, J.; Machhout, M. Image encryption method based on improved ECC and modified AES algorithm. *Multimed. Tools Appl.* **2021**, *80*, 19769–19801. [\[CrossRef\]](#)
14. Dai, Y.; Chen, S. Security analysis of Khudra: A lightweight block cipher for FPGAs. *Secur. Commun. Netw.* **2016**, *9*, 1173–1185. [\[CrossRef\]](#)
15. Wu, N.; Ali, Z.A.; Shaikh, M.M.; Yahya, M.R.; Aamir, M. Compact and high speed architectures of KASUMI block cipher. *Wirel. Pers. Commun.* **2019**, *106*, 1787–1800.
16. Biswas, A.; Majumdar, A.; Nath, S.; Dutta, A.; Baishnab, K.L. LRBC: A lightweight block cipher design for resource constrained IoT devices. *J. Ambient Intell. Humaniz. Comput.* **2020**, *1*–15. [\[CrossRef\]](#)
17. Pandey, J.G.; Goel, T.; Karmakar, A. Hardware architectures for PRESENT block cipher and their FPGA implementations. *IET Circuits Devices Syst.* **2019**, *13*, 958–969. [\[CrossRef\]](#)
18. Aboushousha, B.; Ramadan, R.A.; Dwivedi, A.D.; El-Sayed, A.; Dessouky, M.M. SLIM: A lightweight block cipher for internet of health things. *IEEE Access* **2020**, *8*, 203747–203757. [\[CrossRef\]](#)
19. Mohd, B.J.; Hayajneh, T.; Khalaf, Z.A.; Ahmad Yousef, K.M. Modeling and optimization of the lightweight HIGHT block cipher design with FPGA implementation. *Secur. Commun. Netw.* **2016**, *9*, 2200–2216. [\[CrossRef\]](#)
20. Abed, S.E.; Jaffal, R.; Mohd, B.J.; Alshayegi, M. FPGA modeling and optimization of a Simon lightweight block cipher. *Sensors* **2019**, *19*, 913. [\[CrossRef\]](#)

21. Aghaie, A.; Kermani, M.M.; Azarderakhsh, R. Reliable and fault diagnosis architectures for hardware and software-efficient block cipher KLEIN benchmarked on FPGA. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2017**, *37*, 901–905. [\[CrossRef\]](#)
22. Aghaie, A.; Kermani, M.M.; Azarderakhsh, R. Fault diagnosis schemes for low-energy block cipher Midori benchmarked on FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2016**, *25*, 1528–1536. [\[CrossRef\]](#)
23. Rashidi, B. Efficient and flexible hardware structures of the 128 bit CLEFIA block cipher. *IET Comput. Digit. Tech.* **2020**, *14*, 69–79. [\[CrossRef\]](#)
24. Kaedi, S.; Doostari, M.A.; Ghaznavi-Ghouschi, M.B. Low-complexity and differential power analysis (DPA)-resistant two-folded power-aware Rivest–Shamir–Adleman (RSA) security schema implementation for IoT-connected devices. *IET Comput. Digit. Tech.* **2018**, *12*, 279–288. [\[CrossRef\]](#)
25. Pirpilidis, F.; Pyrgas, L.; Kitsos, P. 8-bit serialised architecture of SEED block cipher for constrained devices. *IET Circuits Devices Syst.* **2020**, *14*, 316–321. [\[CrossRef\]](#)
26. Kumar, T.M.; Karthigaikumar, P. FPGA implementation of an optimized key expansion module of AES algorithm for secure transmission of personal ECG signals. *Des. Autom. Embed. Syst.* **2018**, *22*, 13–24. [\[CrossRef\]](#)
27. Shahbazi, K.; Ko, S.B. Area-efficient nano-AES implementation for Internet-of-Things devices. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *29*, 136–148. [\[CrossRef\]](#)
28. Dofe, J.; Pahlevanzadeh, H.; Yu, Q. A comprehensive FPGA-based assessment on fault-resistant AES against correlation power analysis attack. *J. Electron. Test.* **2016**, *32*, 611–624. [\[CrossRef\]](#)
29. Aldaya, A.C.; Sarmiento, A.J.C.; Sánchez-Solano, S. AES T-Box tampering attack. *J. Cryptogr. Eng.* **2016**, *6*, 31–48. [\[CrossRef\]](#)
30. Chellam, M.B.; Natarajan, R. AES hardware accelerator on FPGA with improved throughput and resource efficiency. *Arab. J. Sci. Eng.* **2018**, *43*, 6873–6890. [\[CrossRef\]](#)
31. Koyuncu, I.; Özcerit, A.T. The design and realization of a new high speed FPGA-based chaotic true random number generator. *Comput. Electr. Eng.* **2017**, *58*, 203–214. [\[CrossRef\]](#)
32. Pirpilidis, F.; Stefanidis, K.G.; Voyiatzis, A.G.; Kitsos, P. On the effects of ring oscillator length and hardware Trojan size on an FPGA-based implementation of AES. *Microprocess. Microsyst.* **2017**, *54*, 75–82. [\[CrossRef\]](#)
33. Farooq, U.; Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. *J. King Saud Univ.-Comput. Inf. Sci.* **2017**, *29*, 295–302. [\[CrossRef\]](#)
34. Al-Asli, M.; Elrabaa, M.E.; Abu-Amara, M. FPGA-based symmetric re-encryption scheme to secure data processing for cloud-integrated internet of things. *IEEE Internet Things J.* **2018**, *6*, 446–457. [\[CrossRef\]](#)
35. Kundi, D.S.; Aziz, A.; Ikram, N. A high performance ST-Box based unified AES encryption/decryption architecture on FPGA. *Microprocess. Microsyst.* **2016**, *41*, 37–46. [\[CrossRef\]](#)
36. Nandan, V.; Gowri Shankar Rao, R. Low-power and area-efficient design of AES S-Box using enhanced transformation method for security application. *Int. J. Commun. Syst.* **2020**, e4308. [\[CrossRef\]](#)
37. Wong, M.M.; Wong, M.D.; Zhang, C.; Hijazin, I. Circuit and system design for optimal lightweight AES encryption on FPGA. *IAENG Int. J. Comput. Sci.* **2018**, *45*, 52–62.
38. Benhadjoussef, N.; Karmani, M.; Machhout, M.; Hamdi, B. A Hybrid Countermeasure-Based Fault-Resistant AES Implementation. *J. Circuits Syst. Comput.* **2020**, *29*, 2050044. [\[CrossRef\]](#)
39. Sheikhpour, S.; Ko, S.B.; Mahani, A. A low cost fault-attack resilient AES for IoT applications. *Microelectron. Reliab.* **2021**, *123*, 114202. [\[CrossRef\]](#)
40. Sikka, P.; Asati, A.R.; Shekhar, C. High-throughput field-programable gate array implementation of the advanced encryption standard algorithm for automotive security applications. *J. Ambient Intell. Humaniz. Comput.* **2020**, *12*, 7273–7279. [\[CrossRef\]](#)
41. Zode, H.; Sapkal, A. An efficient AES implementation using FPGA with enhanced security features. *J. King Saud Univ.-Eng. Sci.* **2020**, *32*, 115–122. [\[CrossRef\]](#)
42. Shahbazi, K.; Ko, S.B. High throughput and area-efficient FPGA implementation of AES for high-traffic applications. *IET Comput. Digit. Tech.* **2020**, *14*, 344–352. [\[CrossRef\]](#)
43. Madhavapandian, S.; MaruthuPandi, P. FPGA implementation of highly scalable AES algorithm using modified mix column with gate replacement technique for security application in TCP/IP. *Microprocess. Microsyst.* **2020**, *73*, 102972. [\[CrossRef\]](#)
44. Arul Murugan, C.; Karthigaikumar, P.; Sathya Priya, S. FPGA implementation of hardware architecture with AES encryptor using sub-pipelined S-box techniques for compact applications. *Automatika* **2020**, *61*, 682–693. [\[CrossRef\]](#)
45. Wegener, F.; De Meyer, L.; Moradi, A. Spin me right round rotational symmetry for fpga-specific AES: Extended version. *J. Cryptol.* **2020**, *33*, 1114–1155. [\[CrossRef\]](#)
46. Kumar, T.M.; Reddy, K.S.; Rinaldi, S.; Parameshachari, B.D.; Arunachalam, K. A Low Area High Speed FPGA Implementation of AES Architecture for Cryptography Application. *Electronics* **2021**, *10*, 2023. [\[CrossRef\]](#)
47. Algreto-Badillo, I.; Ramírez-Gutiérrez, K.A.; Morales-Rosales, L.A.; Pacheco Bautista, D.; Feregrino-Urbe, C. Hybrid Pipeline Hardware Architecture Based on Error Detection and Correction for AES. *Sensors* **2021**, *21*, 5655. [\[CrossRef\]](#)
48. Soltani, A.; Sharifian, S. An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA. *Microprocess. Microsyst.* **2015**, *39*, 480–493. [\[CrossRef\]](#)
49. Rahimunnisa, K.; Karthigaikumar, P.; Rasheed, S.; Jayakumar, J.; SureshKumar, S. FPGA implementation of AES algorithm for high throughput using folded parallel architecture. *Secur. Commun. Netw.* **2014**, *7*, 2225–2236. [\[CrossRef\]](#)

-
50. Farashahi, R.R.; Rashidi, B.; Sayedi, S.M. FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm. *Microelectron. J.* **2014**, *45*, 1014–1025. [[CrossRef](#)]
 51. Rahimunnisa, K.; Karthigaikumar, P.; Christy, N.A.; Kumar, S.S.; Jayakumar, J. PSP: Parallel sub-pipelined architecture for high throughput AES on FPGA and ASIC. *Cent. Eur. J. Comput. Sci.* **2013**, *3*, 173–186. [[CrossRef](#)]