Modeling, Hardware Architecture, and Performance Analyses of an AEAD-based Lightweight Cipher

Kartik Jhawar
Birla Institute of Technology and Science, Pilani
Jugal Gandhi
Academy of Scientific and Innovative Research
Diksha Shekhawat
Academy of Scientific and Innovative Research
Aniket Upadhyay
Birla Institute of Technology and Science, Pilani
Avadh Harkishanka
Birla Institute of Technology and Science, Pilani
Nitin Chaturvedi
Birla Institute of Technology and Science, Pilani
M. Santosh
Central Electronics Engineering Research Institute
Jai Gopal Pandey (jai@ceeri.res.in)
Central Electronics Engineering Research Institute

Research Article

Keywords: AEAD , Image encryption , Lightweight cryptography , TinyJAMBU algorithm , Hardware implementation , FPGAs

Posted Date: July 21st, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3177687/v1

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Journal of Real-Time Image Processing on February 15th, 2024. See the published version at https://doi.org/10.1007/s11554-024-01416-w.

Modeling, Hardware Architecture, and Performance Analyses of an AEAD-based Lightweight Cipher

Kartik Jhawar[1,4] · Jugal Gandhi[1,2,3] · Diksha Shekhawat[2,3] · Aniket Upadhyay[4] · Avadh Harkishanka[4] · Nitin Chaturvedi[4] · M. Santosh[2,3] · Jai Gopal Pandey[2,3]

Received: date / Accepted: date

Abstract Ensuring data security and integrity is crucial for achieving the highest level of protection and performance in modern cyber-physical systems (CPS). Authenticated encryption with associated data (AEAD) is an efficient and secure way to encrypt data that ensures confidentiality and authenticity. In this study, we focus on image encryption using the TinyJAMBU cipher within the AEAD scheme. In this paper, image encryption using the TinyJAMBU cipher with software and hardware modeling has been proposed, and image encryption evaluation over standard matrices has been performed. The hardware architecture for TinyJAMBU has been implemented on the Xilinx Virtex-7 FPGA device. The implementation results are compared with the realization of other contemporary ciphers that bring TinyJAMBU-128's implementation better in terms of look-up tables (LUTs), slice utilization, and power consumption. In the experimentation phase, the results of TinyJAMBU-128/192/256 for image encryption have been compared with existing image encryption techniques. It has been observed that, compared to other implementations, the proposed image encryption application using TinyJAMBU provides better results for PSNR, MSE, RMSE, and UACI.

Keywords AEAD · Image encryption · Lightweight cryptography · TinyJAMBU algorithm · Hardware implementation · FPGAs.

[2] CSIR - Central Electronics Engineering Research Institute (CEERI), Pilani, Rajasthan, India- 333031
[3] Academy of Scientific and Innovative Research (AcSIR), Ghaziabad, Uttar Pradesh, India - 201002
[4] Birla Institute of Technology and Science, Pilani, India-333031
E-mail: jai@ceeri.res.in

[1] Equal contribution

1 Introduction

In the era of cyber-physical systems (CPS), with cloud computing, edge computing, and the Internet of Things (IoT), smart devices have become an integral part of our lives. They are used to provide essential functions related to the transmission, storage, and access of various types of data. IoT devices can be found in a wide range of applications, from smart homes and wearable devices to industrial machinery and healthcare devices. They can potentially improve efficiency, reduce costs, and improve safety and security in many applications [19]. The rapid growth and widespread adoption of IoT devices in various industries and regions have significantly affected the global economy [27]. In these technology enablers, data security is essential for which lightweight cryptography is deployed. Lightweight ciphers are used, especially when the form factors of the devices are smaller and more portable [26].

The significance of lightweight image encryption has grown due to the widespread use of image-based appli-

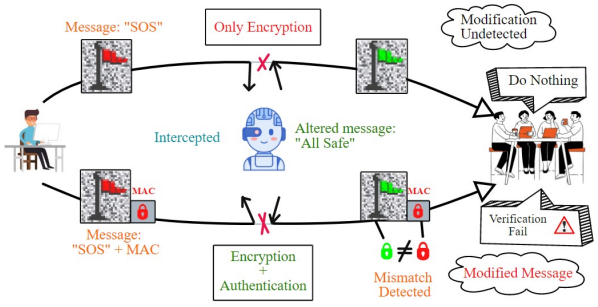

Fig. 1: Image authentication: The vital need in image encryption.

cations in resource-constrained environments, including mobile devices and IoT applications [16]. These devices typically have limited processing power, memory, and energy resources, making it difficult to perform computationally intensive encryption and decryption operations. Consequently, the adoption of lightweight image encryption techniques becomes critical for ensuring the security of image data in IoT devices (refer Fig. 1). Images often contain sensitive information, such as personal or confidential data, which requires protection against unauthorized access or disclosure. Lightweight image encryption offers a viable solution for securing this information while minimizing the resource requirements of the encryption/decryption process.

The proposed work provides high-level modeling, hardware modeling, and associated analyses for AEAD-based TinyJAMBU lightweight cipher for image encryption. Image encryption is performed on 128/192/256-bit of the TinyJAMBU cipher. In addition, the results are compared in related lightweight image encryption works. When considering the favorable results of image encryption for TinyJAMBU, its hardware architecture is proposed that works with 64-bit input plaintext and associated data. The cipher is implemented on the Xilinx Virtex-7 485t field-programmable gate array (FPGA) device, resulting in better resource utilization and power consumption compared to the existing contemporary designs of lightweight ciphers.

Article contribution: The main contributions of this work in terms of image encryption and the FPGA implementation of TinyJAMBU-128 are as follows:

- To meet the pressing security requirements of images, a lightweight AEAD-based TinyJAMBU has been utilized.
- High-level modeling for TinyJAMBU to implement a lightweight image encryption scheme.
- The evaluation of the image encryption parameters has been analyzed for the three variants of TinyJAMBU and compared with the existing image encryption techniques. TinyJAMBU has shown 7.38%, 16.15%, 0.85%, 11.53%, and 6.88% improved results in terms of peak signal-to-noise ratio (PSNR), mean square error (MSE), root mean square error (RMSE), unified average change intensity (UACI), etc.
- Hardware architecture of TinyJAMBU-128 algorithm for lightweight image encryption has been proposed.
- The FPGA implementation of the cipher has been carried out and the results are compared with the implementation of other algorithms. The proposed work provides better results in terms of LUT utilization, slice utilization, and power consumption.

The rest of the paper is divided into eight sections and the purpose of each section is discussed below. Section 2 discusses the related work, followed by Section 3 discusses the details of the TinyJAMBU-128 cipher and its pseudocode. The software modeling of TinyJAMBU-based lightweight image encryption is explained in Section 4. Section 5 provides the proposed hardware design of TinyJAMBU for image encryption. Experimental setup and related results of the proposed design and its lightweight image encryption application are provided in Section 6. In Section 7, the article is concluded.

2 Related Work

Recent studies in cryptography have shown a significant focus on lightweight AEAD-based ciphers. The confidentiality and integrity of the data that are being transmitted are adequately addressed by the AEAD mode of operation. In particular, in the context of resource-constrained environments of lightweight cryptography, this mode of operation becomes beneficial. Modern lightweight ciphers such as the low-latency lightweight block cipher (LLLWBC) [37], GFRX [38], and DBST [32], which is a lightweight block cipher employing dynamic S-box, have emerged as notable advancements in the field. LLLWBC is a low-latency block cipher that employs a generalized variant of the Feistel structure, namely extended GFS. The GFRX algorithm employs an add-rotate XOR (ARX) structure in combination with distinct non-linear components to address all the branches of a generalized Feistel structure, thus enhancing the diffusion effect with fewer rounds. DBST is constructed using a generalized Feistel variant structure that combines linear and non-linear transformations. It operates on 128-bit blocks and utilizes a 64-bit key size. Table 1 presents fundamentals for the NIST finalists of lightweight ciphers. Lynx [14] is one of the recently proposed families of lightweight AEAD-based block ciphers. It comprises members of the 1-pass and rate-1 variants, providing strong integrity security against birthday-bound attacks in nonce-respecting, nonce- misuse, and related key scenarios. Naito et al. introduced SAEB [24], which uses XOR exclusively for block encryption. The SAEB cipher utilizes a state size that is equal to the block size, enabling the processing of a single data block at a time.

The TinyJAMBU cipher has gained attention in recent years for its suitability in various applications and has been implemented on hardware platforms. Wu et al. in [31] aimed to design a TinyJAMBU cipher optimized for devices where a secret key is stored. The inability to control state bits leads to better authentication security. TinyJAMBU-128 uses a 128-bit keyed permutation with a message block size of 32-bit, and when the nonce is reused, the attacker's forgery advantage reduces to

Table 1: Specifications for NIST finalist of lightweight ciphers [8].

Algorithm	Type	Variant(s)	Primary Version	State (Bits)	Key (Bits)	Mode	Rate/Block (Bits)	Tag (Bits)	Security (Bits)
ASCON [10]	Sponge	ASCON-128	ASCON-p	320	128	Duplex	64	128	128
		ASCON-128a	ASCON-p	320	128	Duplex	128	128	128
ELEPHANT [7]	Sponge	Jumbo	Spongent	176	128	Elephant	176	64	127
		Dambo	Spongent	160	128	Elephant	160	64	112
		Delirium	Keccak	200	128	Elephant	176	128	127
GIFT-COFB [4]	Block	GIFT-COFB	GIFT-128	192	128	COFB	128	128	128
GRAIN-128 AEAD [15]	Stream	GRAIN-128 AEAD	-	256	128	-	1	64	128
ISAP [9]	Sponge	ISAP-A-128	ASCON-p	320	128	ISAP	64	128	128
		ISAP-K-128	Keccak	400	128	ISAP	144	128	128
		ISAP-K-128A	Keccak	400	128	ISAP	144	128	128
		ISAP-A-128A	ASCON-p	320	128	ISAP	64	128	128
PHOTON-Beetle [5]	Sponge	PHOTON-Beetle-AEAD [128]	PHOTON-256	256	128	Beetle	128	256	121
		PHOTON-Beetle-AEAD	PHOTON-256	256	128	Beetle	32	256	128
Romulus [20]	Block	Romulus-M	SKINNY-128-384	384	128	COFB	128	128	128
		Romulus-N	SKINNY-128-384	384	128	COFB	128	128	128
		Romulus-T	SKINNY-128-384	384	128	COFB	128	128	128
SPARKLE [6]	Sponge	SCHWAEMM256-128	SPARKLE	384	128	SPARKLE	256	128	120
		SCHWAEMM128-128	SPARKLE	256	128	SPARKLE	128	128	120
		SCHWAEMM192-192	SPARKLE	384	192	SPARKLE	192	192	184
		SCHWAEMM256-256	SPARKLE	512	256	SPARKLE	256	256	248
TinyJAMBU [31]	Sponge	TinyJAMBU	TinyJAMBU	128	128	TinyJAMBU	32	64	128
Xoodyak [2]	Sponge	Xoodyak	Xoodoo	384	128	Cyclist	352	128	128

less than 2^{-15}, adding to the strong authentication security. The hardware area is significantly reduced with the constant use of fixed keys. Efficient input loading on the hardware is another important feature discussed in [31]. It uses at least 1024 rounds to encrypt a 32-bit plain text block, making it impossible to recover the key by differential and linear cryptanalysis with a probability smaller than 2^{-64}.

AEAD-based lightweight ciphers have numerous applications in the protection of data transmission, storage, and access control. Bakhs-handeh et al. [3] proposed an authenticated image encryption scheme based on memory cellular automata and chaotic maps that utilizes a permutation-diffusion architecture. The method is designed to provide a high level of security through its diffusion mechanism and offers computational efficiency and ease of implementation. Another chaotic map, together with the rapid image encryption and authentication scheme based on cellular automata, is proposed in [34], which presents a keyed hash function that generates a 128-bit hash value from the plain image and secret hash keys. The hash value serves as the encryption and decryption key, whereas secret hash keys are utilized to authenticate the decrypted image. Thus, several lightweight AEAD-based ciphers have been proposed for image encryption. To the best of our knowledge, none of the mentioned proposals have investigated the utilization of the TinyJAMBU cipher specifically for image security applications. In this paper, we explore the potential suitability of TinyJAMBU for image encryption applications. We extensively evaluated the TinyJAMBU cipher with all its variants for use in image encryption. The experimental results indicate that it is useful among existing AEAD-based ciphers, in particular, for image encryption. It performed well on the basis of various evaluation parameters and histogram analysis compared to other image encryption techniques.

Abdulgadir et al. [1] have provided lightweight side-channel resistant implementations of TinyJAMBU for hardware security applications considering the issue of protecting intellectual property (IP) from theft. This was done with the aim of ensuring that the area and power consumption remained significantly lower than current standards, such as AES-GCM. A TinyJAMBU software implementation on a Siemens S7-1200 programmable logic controller has been given by [11]. The focus here was on assessing the execution speed and memory usage of each variant of the cipher on the industrial controller. In [21], a TinyJAMBU software implementation is performed on the ARM Cortex M0 and evaluated in terms of three performance metrics: latency, throughput, and memory usage. The impact of the length of the input data and the parameters on implementation performance was also discussed. Existing research on lightweight image encryption has focused on develop-

ing new algorithms and evaluating their performance on software platforms.

The implementation of these algorithms on hardware platforms, such as FPGAs, has received limited contribution. This is a significant gap in research, as hardware implementation offers several advantages over software implementation, including improved performance, reduced power consumption, and increased security. Additionally, efficient image encryption techniques for IoT devices that have limited computational resources are needed. Therefore, there is a strong case for researching the hardware implementation of lightweight image encryption algorithms, such as TinyJAMBU and evaluating their effectiveness in securing images in IoT devices. In the proposed work, we address this gap and also implemented TinyJAMBU on a hardware platform and its use for image encryption applications.

3 TinyJAMBU-128 Cipher

The TinyJAMBU cipher algorithm is a lightweight cryptographic algorithm that operates on a 64-bit block size and a key size of 128/192/256 bits. It uses a permutation-based design with a round function consisting of substitution and linear diffusion layers. The key is expanded using a key scheduling algorithm that generates round keys for each round of cipher. TinyJAMBU also incorporates the JAMBU mode of operation for authenticated nonce-based encryption. The JAMBU mode uses a nonce as an input, providing both confidentiality and plaintext authenticity. The cipher encrypts plaintext in a series of blocks, each combined with previous block's ciphertext using a message authentication code (MAC) function. This function uses a secret key to generate a tag that is appended to the ciphertext. TinyJAMBU is designed to be secure against known attacks, including differential and linear attacks.

3.1 The Keyed Permutation

The keyed permutation in TinyJAMBU is the core operation that provides confusion and diffusion properties to the algorithm. It is a permutation that takes input, a 128-bit state and a 128-bit key and produces as output a new state. The permutation round consists of four steps, each of which applies a set of operations that include bitwise XOR, bit rotation, substitution using a nonlinear function, and a linear mixing operation. Algorithm 1 depicts the working of TinyJAMBU cipher. The basis of the algorithm lies in a nonlinear feedback shift register (NFSR) of 128-bit that is used to update the state [31] as shown in line 2 to 6.

Algorithm 1: TinyJAMBU-128 Algorithm

Input: N: 96-bit nonce, K: 128-bit key, AD: associated data of length $adlen$, M: plaintext of length $mlen$, F: 3-bit frameBits

Output: C: cipher text of length $mlen$, T: 64-bit authentication tag

1 **Function** state update(**S**, **K**, **N**)
2 **for** $i = 1$ *to* N **do**
3 feedback $\leftarrow s_0 \oplus s_{47}\ (\sim (s_{70} \& s_{85})) \oplus s_{91} \oplus K_{imodklen}$
4 **for** $j = 0$ *to* 126 **do**
5 $s_j \leftarrow s_{j+1}$
6 $s_{127} \leftarrow$ feedback

/* Key setup */
7 $\{s_0, s_1, ..., s_{127}\} \leftarrow \{0, 0, ..., 0\}$
8 stateUpdate(S, K, 1024)
/* Nonce setup */
9 $\{f_0, f_1, f_2\} \leftarrow \{1, 0, 0\}$
10 **for** $j = 0$ *to* 2 **do**
11 $s_{36..38} \leftarrow s_{36..38} \oplus f_{0..2}$
12 stateUpdate(S, K, 640)
13 $s_{96..127} \leftarrow s_{96..127} \oplus n_{32j..32j+31}$

/* Processing the full blocks of AD */
14 $\{f_0, f_1, f_2\} \leftarrow \{1, 1, 0\}$
15 **for** $j = 0$ *to* $\lfloor adlen/32 \rfloor$ **do**
16 $s_{36..38} \leftarrow s_{36..38} \oplus f_{0..2}$
17 stateUpdate(S, K, 640)
18 $s_{96..127} \leftarrow s_{96..127} \oplus ad_{32j..32j+31}$

/* Processing the partial blocks of AD */
19 **if** $adlen\ mod\ 32 \neq 0$ **then**
20 $s_{36...38} \leftarrow s_{36...38} \oplus f_{0..2}$
21 stateUpdate(S, K, 640)
22 $lenp \leftarrow adlen$ mod 32
23 $startp \leftarrow adlen\text{-}lenp$
24 $s_{96..96+lenp-1} \leftarrow s_{96..96+lenp-1} \oplus ad_{startp...adlen-1}$
25 $s_{32..33} \leftarrow s_{32..33} \oplus (lenp/8)$
26 **else**
27 go to **Next Step**

/* Processing the full blocks of plaintext */
28 $\{f_0, f_1, f_2\} \leftarrow \{1, 0, 1\}$
29 **for** $j = 0$ *to* $\lfloor mlen/32 \rfloor$ **do**
30 $s_{36..38} \leftarrow s_{36\cdots38} \oplus f_{0\cdots2}$
31 stateUpdate(S, K, 1024)
32 $s_{96..127} \leftarrow s_{96..127} \oplus m_{32j..32j+31}$
33 $c_{32j..32j+31} \leftarrow s_{64..95} \oplus m_{32j..32j+31}$

/* Processing partial blocks of plaintext */
34 **if** $mlen\ mod\ 32 > 0$ **then**
35 $s_{36..38} \leftarrow s_{36..38} \oplus f_{0..2}$
36 stateUpdate(S, K, 1024)
37 $lenp \leftarrow mlen$ mod 32
38 $startp \leftarrow mlen$ - $lenp$
39 $s_{96..96+lenp-1} \leftarrow s_{96..96+lenp-1} \oplus m_{startp..mlen-1}$
40 $c_{startp..mlen-1} \leftarrow s_{64..64+lenp-1} \oplus m_{startp..mlen-1}$
41 $s_{32..33} \leftarrow s_{32..33} \oplus (lenp/8)$
42 **else**
43 go to **Algorithm 2**

3.2 Initialization

The first step is initialization, which is shown in Algorithm 1 from lines 7 to 13. It has been divided into two steps: the key and the nonce setup. The setup of keys involves a simple reset of the state followed by updating the state using NFSR for 1024 rounds (P_{1024}). For three consecutive iterations, the frame bits in the message are set to a value of '001'. During each iteration, the state bits 36 to 38 are XORed with the corresponding frame bit following the P_{640} permutation. Additionally, the state bits from 96 to 127 and are XORed with the corresponding nonce bits during the nonce setup step. This process is performed iteratively to ensure that the output of function is unpredictable and resistant to differential cryptanalysis.

3.3 Processing of the Associated Data

The processing of the associated data step is shown in Algorithm 1 from lines 14 to 18. $AD = AD_{0...adlen-1} = d_0,..., d_{adlen-1}$ are processed block by block, where each block is 32 bits and *adlen* is the number of bits of the total associated data. Depending on the number of blocks of AD, the following steps occur: XORing state bits 36 to 38 with the corresponding frame bits set to '011', P_{640} permutation, and XORing state bits 96 to 127 with the corresponding AD bits. If the last block of associated data is not complete, the partial block processing steps are performed as specified in Algorithm 1. These steps involve XORing of frame bits with the remaining bits of the incomplete block, followed by the P_{640} permutation, and then updating the state twice using functions that depend on length of the associated data, *adlen*, and the modulus of *adlen* with 32.

3.4 Processing of the Plaintext

In Algorithm 1, the processing of the entire block of the plaintext is shown from lines 28 to 33. $M = M_{0...mlen-1} = m_0...m_{adlen-1}$ are processed block by block, where each block is of 32 bits and *mlen* is the number of bits in the total plain text message. Depending on the number of blocks in *M*, many rounds of the following steps take place starting with, XORing state bits 35 to 38 with the corresponding frame bits set to '101', P_{1024} and XORing state bits 96 to 127 with the corresponding bits *M* following to generate the corresponding 32-bit of cipher text. Similarly to the processing of the associated data partial block, the partial block, if any of the plaintext is processed through similar steps, ultimately generates the last 32 bits of the plaintext message.

3.5 Finalization

The Algorithm 2 provides the finalization step. This step generates the authentication tag required for the authenticity check. When the message encryption is complete, the values of *FrameBits* are set to '111'. This triggers the XOR operation of the state bits numbered 36 to 38 with *FrameBits*, followed by the P_{1024} permutation. The resulting state bits numbered 64 to 95 are then extracted and used as the lower 32 bits of the authentication tag. The same XOR operation is performed again, followed by the P_{640} permutation to generate the upper 32 bits of the authentication tag.

Algorithm 2: TinyJAMBU Finalization Stage

1 $\{f_0, f_1, f_2\} \leftarrow \{1, 1, 1\}$
2 $s_{36..38} \leftarrow s_{36..38} \oplus f_{0..2}$
3 state update(S, K, 1024)
4 $T_{36..38} \leftarrow s_{64..95}$
5 $s_{36..38} \leftarrow s_{36..38} \oplus f_{0..2}$
6 state update(S, K, 640)
7 $T_{32..63} \leftarrow s_{64..95}$

4 A Software Modeling of TinyJAMBU-based Lightweight Image Encryption

The high-level modeling and associated software implementation of TinyJAMBU 128/192/256-bit variants for image encryption are shown in Fig. 2. The results are analyzed using various evaluation parameters and histogram analysis of the ciphered images. A software model of the AEAD-based TinyJAMBU algorithm for lightweight image encryption involves creating a mathematical representation or simulation of the encryption process using software tools. The model considers specific steps and operations of the TinyJAMBU algorithm, including key generation, initialization, associated data processing, plaintext processing, and finalization. It also incorporates the properties and characteristics of TinyJAMBU, such as its lightweight design, high security, and efficiency. The software model can be used to analyze and evaluate the performance and security of TinyJAMBU-based image encryption scheme. It can help to assess the resistance of the encryption scheme to various attacks. The model can also be used to measure computational overhead, memory requirements, and algorithm processing time. Also, the software model can help develop practical applications of TinyJAMBU-based lightweight image encryption. It can be used to create software tools to encrypt images using TinyJAMBU on different platforms.

In this, the channel of an image has been extracted from an RGB image, and each channel is stored separately. For each channel, the decimal equivalent array

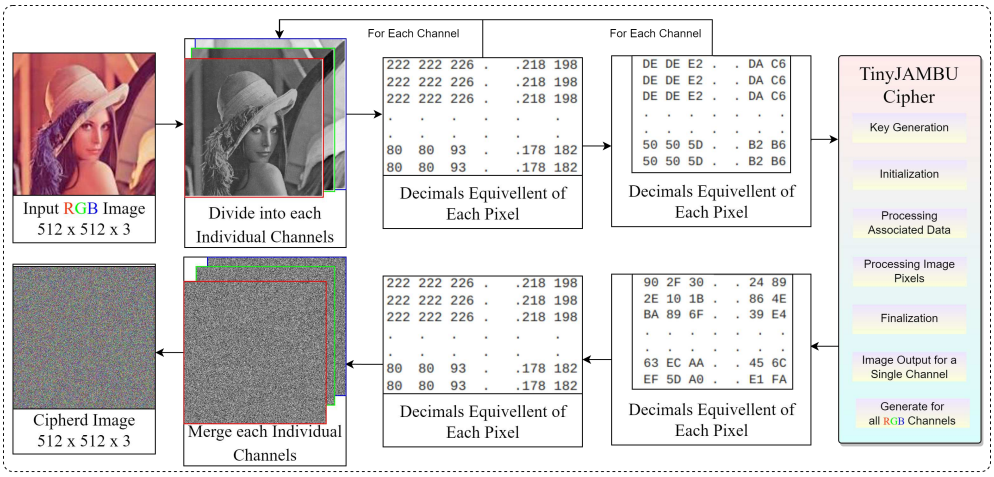

Fig. 2: Encryption of a 512×512 image using TinyJAMBU-128 cipher.

of pixels has been extracted, and its hex form has been created. Then, this array was passed through a Tiny-JAMBU algorithm using flattened matrices, and the ciphered version was extracted. The image reconstruction from this array has been carried out for all channels, and a ciphered image of the input has been generated using the TinyJAMBU-128 algorithm. After encrypting an image using the TinyJAMBU-128 algorithm or any other encryption algorithm, it is important to perform a security analysis of the encrypted image to assess its security and evaluate its resistance to potential attacks. Security analysis of an encrypted image involves evaluating the level of protection provided by the encryption algorithm and its parameters.

5 Hardware Modeling of TinyJAMBU-128

The need of hardware for lightweight image encryption arises due to the high computational complexity involved in encryption algorithms. As the size of the image increases, the computational requirements of the encryption algorithm also increase. The hardware implementation of Tiny-JAMBU-128 is faster and more efficient than the software implementation, making it suitable for real-time image encryption applications. It can also be used in resource-constrained IoT devices, where lightweight encryption algorithms are preferred. It involves designing a hardware architecture based on the algorithm's specifications and implementing it on FPGA or other hardware devices. The hardware implementation of TinyJAMBU-128 can be optimized to reduce power consumption, increase performance, and minimize the hardware resources.

Hardware-based image encryption provides faster and more efficient encryption compared to software-based approaches. Therefore, in this paper, we have proposed a hardware implementation for the TinyJAMBU-128 bit variant following the algorithm described in Section 3. Here, a TinyJAMBU-128 architecture with a controller is proposed for 64-bit plaintext and associated data input. The latency of the encryption operation is 7955 cycles due to the usage of keyed permutation. Thus, it involves 640 and 1024 rounds of permutations at various steps in the encryption. The total latency consists of 1025 cycles for key setup, 1926 cycles for nonce setup, and 1284, 2054, and 1666 cycles for processing the full blocks of the associated data step, processing the full blocks of the plaintext step, and the finalization step, respectively. However, similar steps are performed for the 96-bit and 256-bit variants. The proposed hardware architecture and controller for 64-bit input TinyJAMBU-128 are discussed in Section 5.1 and Section 5.2.

5.1 Hardware Architecture for TinyJAMBU-128

The hardware architecture for TinyJAMBU-128 mainly consists of four steps: initialization, associated data processing, plaintext processing, and finalization. A custom controller is used to provide the control signals to the various modules of the architecture. The proposed architecture of TinyJAMBU-128 is shown in Fig. 3. The detailed operation of the architecture is as follows.

5.1.1 Initialization

In this step, the key configuration is performed, which consists of resetting 'State Update' module. This is followed by 1024 rounds of state update, in which an 11-bit counter tracks the current round. Once the 'State

Update' module completes all 1024 rounds, the 'done' flag is set to '1', signaling the system to proceed to the next step. After this, nonce setup is done which comprises three distinct sub-steps: XOR of frame bits with state value, state update, and XORed output. First, the frame bits are XORed, which are distinct in each step of the process, as illustrated in Algorithm 1. The three 4×1 multiplexers (muxes) are utilized, and the frame bits are XORed with the state value and subsequently update the state from 36 to 38, as shown in Fig. 3. Then, the state is updated by 640 rounds, and the state bits are XORed with a 32-bit nonce value. A universal mux is used to select 32 bits of the plaintext message, 32 bits of associated data, and 32 bits of a nonce. Finally, the resulted XOR output is given as one of the inputs to the corresponding 2×1 mux and the other input is the unupdated state value from bits 96 to 127.

5.1.2 Processing of the Associated Data

In this step, processing of all associated data blocks has been carried out using hardware modules similar to the initialization setup of the nonce, and if *adlen* mod $32 > 0$, then different steps are followed to process the partial blocks. Also, XORing of the state 96–127 bits with associated data is done. It is contrast to XORing with the nonce as in the previous step, along with the change in frame bit value.

5.1.3 Processing of the Plaintext Message

The process consists of two divisions, one for processing full blocks of plaintext and the other for processing partial blocks of plaintext, identified by the flag of *mlen* mod $32 > 0$. For full block processing, similar steps as in the nonce setup are followed with a change in the third step. The cipher text is then generated and stored in a file in the register, which is mapped using a 4×1 mux. If the flag of *mlen* modifies $32 > 0$, then several different steps are performed. Similarly to the previous step, partial block processing is unnecessary in this design case, as the hardware is designed for 64-bit input message size and 64-bit associated data, resulting in two full blocks.

5.1.4 Finalization

The finalization process is performed directly after completing the plaintext processing. In this, the normal state bits are updated similar to the nonce setup and the associated data processing steps. This is followed by 1024 rounds of state updates to generate the first 32 bits of the tag, which are then stored in the same register file. Then, further steps are performed to generate the next 32 bits of the tag, which are also stored in the same register file. Finally, the sender-side encryption process is completed. The final output is 64-bit cipher bits and 64-bit authentication tag bits.

5.2 Design of a Controller

A controller for the proposed architecture is shown in Fig. 4. In hardware architecture, the term 'state' refers to the 128-bit value of the 'State Update' hardware module, as shown in Fig. 3. The finite state machine (FSM) state (FSM_state) which is distinct from 'state', refers to the current state of the controller.

5.2.1 Initialization

When the $state\_rst$ signal is set to '1', the 128-bit state is initialized to '0' and P_{1024} is applied to the state for 1024 rounds. The $bit11cntTill$ control signal loads the 11-bit counter to count 1024 rounds, while en_11bit enables the counting in the datapath to count until the flag $bit11\_countdone$ is set to '1', which changes FSM_state to FrameBitsXOR. For the nonce setup, the frame bits, '001' are set using f_sel and output of the XOR operation is written back into the 36-38 bits of the state using $S36to38\_sel$ and reg_write pin of the 'State Update' module. Similarly, the P_{640} operation is applied, and when $bit11\_countdone$ is set to '1', the FSM state changes to NonceXOR, indicating that the 8×1 mux in the datapath should select the 32 bits of nonce in every loop using $Snon\_ad\_m\_sel$ select line.

5.2.2 Processing the Associated Data

Once the three loops of the nonce setup step are complete, FSM_state changes FrameBitsXOR for the processing of associated data. In this step, the frame bit value is set to '011' by setting f_sel to '1'. The additional control signals are similar to FrameBitsXOR FSM_state in the previous step. This step is followed by the P_{640} and the ADXOR state, with control signals similar to the initialization step. However, in ADXOR FSM_state, XOR of 32 bits of the associated data selected using 8×1 mux of the datapath is performed with the corresponding 96-127 state bits for each loop. This step has two loops instead of three as in the initialization step, as the input associated data are 64 bits.

5.2.3 Processing of Plaintext

In the previous step, after the two loops have been completed, f_sel is set to '2' for the frame bits, which is '101' for this step. Similar control signals are used as in

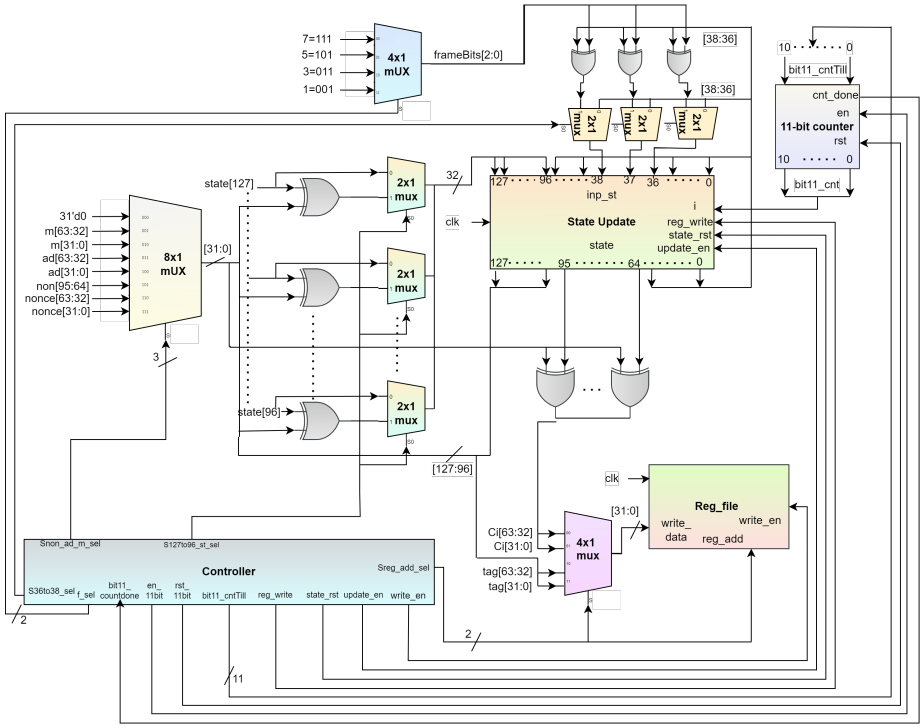

Fig. 3: The proposed hardware architecture of 64-bit TinyJAMBU-128.

FrameBitsXOR FSM_state. P_{1024} is executed by setting the 11-bit counter to '1024'. When $bit11\_countdone$ is flagged, the state changes to MsgXOR, which is the XOR of the 32 bits of plaintext message with the state bits, 96–127. For this, $Snon\_ad\_m\_sel$ is set to '6' and '7' for each loop of this step. In the last state of each loop, the 32 bits of ciphertext are stored in Reg_file of the datapath. Another control signal, $Sreg\_add\_sel$, is set to '0' and '1' for each loop, addressing the Reg_file to store the 32 bits of a ciphertext in two loops. The two loops are used due to the 64-bit input Msg of the proposed architecture.

5.2.4 Finalization

After two loops of the previous step, FSM_state changes to FrameBitsXOR of the finalization step. Frame bits are '111', and f_sel is set to '3'. The controller releases signals similar to those in the FrameBitsXOR step, followed by signals to execute P_{1024}. When $bit11\_countdone$ is flagged, FSM_state changes to store the first 32 bits of the authentication tag generated as 64–95 state bits in Reg_file of the datapath. For this, $sreg\_add\_sel$ is set to '3' to set the destination address of Reg_file. Another FrameBitsXOR step and P_{640} follow this step. Finally, the remaining 32 bits of tag are generated as 64–95 state bits and stored in Reg_file in storing tag FSM_state with the destination address set by $sreg\_add\_sel$.

6 Experimental Setup, Results, and Comparison

To validate the TinyJAMBU encryption technique, numerical simulations, and tests are performed using MATLAB, C++, and Python languages. The test images used in the experiment are obtained from the USC-SIPI image database [30]. The development is carried out on an Intel Core i7-10750H CPU@5.00GHz ×16 processor and 32GB of RAM. The FPGA implementation has been done using Xilinx's Vivado v2022.1 in Verilog on the Xilinx Virtex-7 FPGA device.

6.1 High-level Modeling Results

We have performed the matrices on the TinyJAMBU algorithm and compared the results with the existing related literature. The evaluation results performed on the three TinyJAMBU variants are collectively tabulated in Table 2. Based on the results of the PSNR, MSE, RMSE, and NPCR values for the three ciphered images, it is observed that for the 256-bit variant, it gives better results. For entropy and UACI, the 128-bit variant performs better. Taking into account the majority of evaluation parameters that favor the 256-bit variant, the TinyJAMBU-256 variant is used for further comparison with other image encryption techniques.

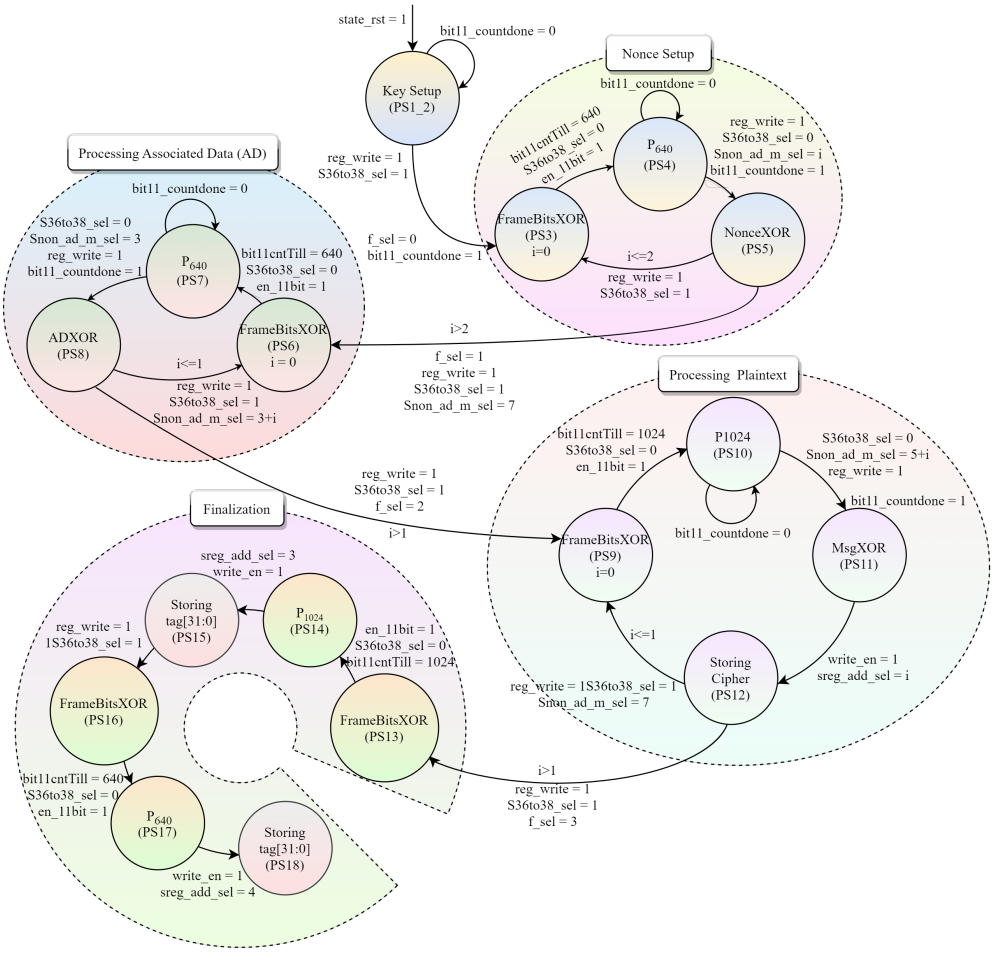

Fig. 4: Controller for the 64-bit input TinyJAMBU-128.

Table 2: A comparison of the image evaluation parameters for all variants of TinyJAMBU.

Parameters	**TinyJAMBU-128**			**TinyJAMBU-192**			**TinyJAMBU-256**		
	Lenna	Pepper	Aeroplane	Lenna	Pepper	Aeroplane	Lenna	Pepper	Aeroplane
Entropy	7.9590	7.9589	7.9585	7.9588	7.9585	7.9579	7.9590	7.9587	7.9590
PSNR	8.6001	8.0973	7.9649	8.5954	8.1024	7.9603	8.5877	8.0890	7.9542
MSE	8975.6787	10077.4295	10389.3806	8985.2949	10065.4359	10400.2373	9001.2945	10096.6990	10414.9395
RMSE	94.7400	100.3864	101.9283	94.7907	100.3266	101.9815	94.8751	100.4823	102.0536
NPCR	99.5337	99.5663	99.6143	99.5354	99.5689	99.6089	99.5381	99.5587	99.6167
UACI	29.6180	24.0427	28.4108	29.6408	24.0846	28.4979	29.7267	24.0762	28.5852

The comparison of the evaluation parameters with existing image encryption techniques using different cryptographic algorithms is given in Table 3. This showcases the proposed image encryption algorithm has better results in terms of entropy, PSNR, MSE, RMSE, NPCR, and UACI with respect to the existing architectures. Compared to [28], the proposed encryption techniques show 70.33% and 32.05% better PSNR and MSE; UACI value is improved by 11.53% compared to [18] and 5.96% compared to [36]; better PSNR and MSE values of 11.10% and 19.5226% with respect to [29].

6.1.1 Histogram Analysis

Histogram analysis is used in image encryption to evaluate the statistical distribution of pixel values in both the original and the encrypted image. It helps to ensure that the encryption process does not significantly alter the distribution of the pixel values, which could result in a distorted or unrecognizable image. Histograms of the original and encrypted images obtained by running the three variants of TinyJAMBU on the three selected images are shown in Fig. 5. A flat profile in the encrypted histogram means that the encrypted image has

Table 3: A comparison of the image evaluation parameters with respect to other designs.

Image	Evaluation Parameter	[18]	[23]	[12]	[35]	[17]	[36]	[28]	[39]	[29]	**This work**
Lenna	Entropy	7.9973	7.9989	7.9991	7.4461	7.4455	7.9664	7.9992	-	-	7.9590
	PSNR	-	45.6210	8.6403	-	-	-	28.9500	9.5400	9.5600	8.5877
	MSE	-	8319.0817	8893.0400	-	-	-	6816.1536	7229.0000	7244.0000	9001.2945
	MSE	-	91.2090	-	-	-	-	82.5600	-	-	94.8751
	NPCR	99.6100	99.5870	-	99.8300	99.6100	99.6100	-	-	-	99.5381
	UACI	33.4800	30.7010	-	30.2800	30.2800	27.8500	-	-	-	29.6180
Pepper	Entropy	7.9972	7.9989	7.9991	-	-	-	-	-	-	7.9589
	PSNR	-	45.7160	8.1030	-	-	-	28.4200	8.9900	8.9100	8.0890
	MSE	-	8692.2060	10064.2000	-	-	-	8717.9570	8205.0000	8431.0000	10096.6990
	RMSE	-	93.2320	-	-	-	-	93.3700	-	-	100.4823
	NPCR	99.6100	99.6010	-	-	-	-	-	-	-	99.5689
	UACI	-	31.0360	-	-	-	-	-	-	-	24.0427

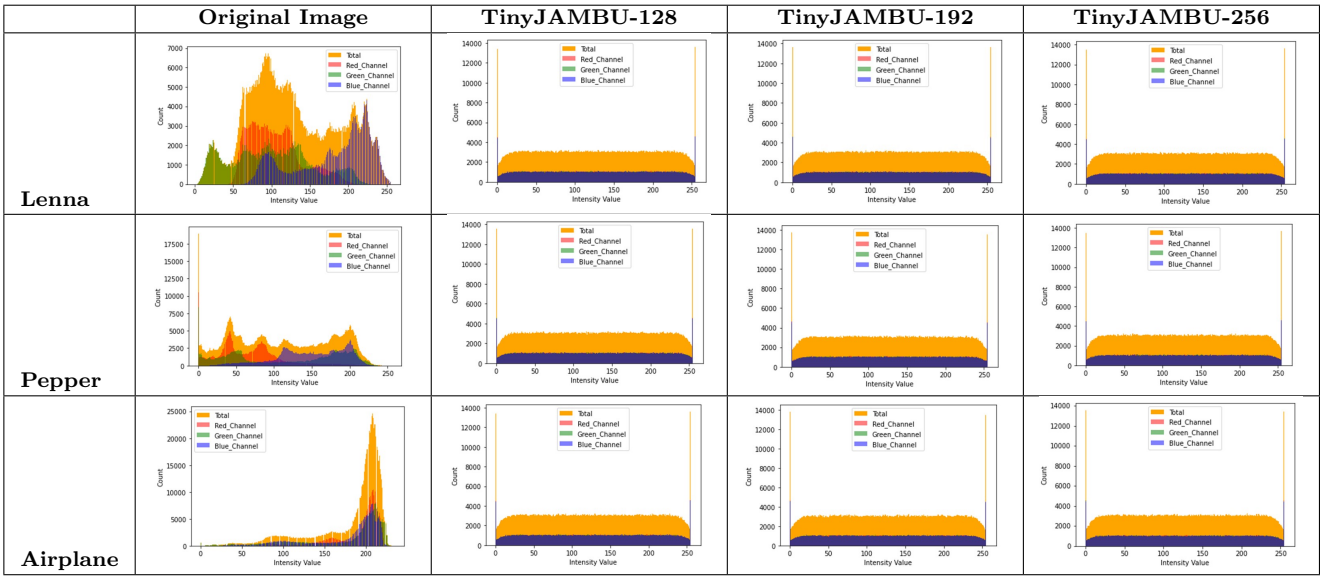

Fig. 5: Comparisons of histograms for original and encrypted images for TinyJAMBU variants.

a uniformly distributed pixel intensity. In other words, each value of pixels in the encrypted image has an equal probability of occurring. This is desirable in image encryption, as it makes it difficult for an attacker to discern meaningful information from the encrypted image. Additionally, a flat profile can indicate that encryption process has succeeded in disguising the original image and making it difficult to recover the original content.

6.2 Hardware Implementation Results

The hardware implementation results of the proposed architecture are compared with other hardware implementations performed on the Virtex-7 FPGA device. The proposed architecture utilizes 0.08%, 0.02%, and 0.14% of LUTs, flip-flops, and slices, respectively. A comparison of F_{max}, LUTs, flip-flops, slices, and dynamic power of the proposed TinyJAMBU-128 with existing architectures are given in Table 4.

Table 4: A comparison of the Virtex-7 FPGA implementations with existing techniques.

Parameters	[22]	[25]	[33]	[13]	**This Work**
F_{max} (MHz)	479.00	57.94	39.50	374.00	128.15
LUTs	760	492	455	1614	258
Flip-flops	128	-	89	-	144
Slices	345	-	128	1355	109
Power (mW)	185	143	16	-	18

Compared to the existing architectures, the utilization of LUTs and slices is reduced by 66.05% and 68.40% with respect to [22], 84.01% and 91.95% with respect to [13], 43.29% and 14.84% with respect to [33] and 47.56% with respect to [25]. The proposed TinyJAMBU-128 architecture has maximum frequency, F_{max} of 128.15 MHz, which is better than [33] and [25]. Compared to

the existing architectures implemented on the Virtex-7 FPGA device, the power usage of the proposed work is reduced by 87.41% with respect to [25] and 90.27% with respect to [22].

7 Conclusion

In this paper, high-level modeling is presented for a lightweight AEAD-based cipher, TinyJAMBU to encrypt images. An in-depth evaluation of the image encryption performance of TinyJAMBU is conducted across its three variants through a comprehensive analysis of evaluation parameters and histogram analysis. Compared to [28]he proposed encryption techniques demonstrate significant improvements of 70.33% and 32.05% higher PSNR and MSE values, respectively. Furthermore, the UACI value is enhanced by 11.53% compared to [18] and 5.96% compared to [36]. Additionally, the PSNR and MSE values show improvements of 11.10% and 19.5226%, compared to [29]. Based on the assessment of TinyJAMBU's image encryption performance, an hardware architecture for TinyJAMBU-128 with 64-bit input has been proposed. An implementation of the proposed architecture has been performed on the Virtex-7 FPGA device, illustrating better utilization of LUT, slices and power consumption than existing architectures. The LUTs and slices are reduced by 66.05% and 68.40% with respect to [22], 84.01% and 91.95% with respect to [13], 43.29% and 14.84% with respect to [33] and 47.56% with respect to [25]. The power consumption is reduced by 87.41% with respect to [25] and 90.27% with respect to [22]. The findings of this study highlight the potential of TinyJAMBU for edge-node-based secure image encryption applications.

Acknowledgment

We acknowledge the fellowship support provided by CSIR-HRDG through the CSIR-GATE-SRF, which supported Jugal Gandhi during this research.

References

1. Abdulgadir A, Lin S, Farahmand F, Kaps JP, Gaj K (2021) Side-channel resistant implementations of a novel lightweight authenticated cipher with application to hardware security. In: Proceedings of the 2021 on Great Lakes Symposium on VLSI, pp 229–234
2. Assche GV, Keer RV (2020) Xoodyak, a lightweight cryptographic scheme. IACR Transactions on Symmetric Cryptology
3. Bakhshandeh A, Eslami Z (2013) An authenticated image encryption scheme based on chaotic maps and memory cellular automata. Optics and Lasers in Engineering 51(6):665–673
4. Banik S, Chakraborti A, Inoue A, Iwata T, Minematsu K, Nandi M, Peyrin T, Sasaki Y, Sim SM, Todo Y (2020) GIFT-COFB. Cryptology ePrint Archive
5. Bao Z, Chakraborti A, Datta N, Guo J, Nandi M, Peyrin T, Yasuda K (2019) Photon-beetle authenticated encryption and hash family. NIST Lightweight Compet Round 1:115
6. Beierle C, Biryukov A, dos Santos LC, Großschädl J, Perrin L, Udovenko A, Velichkov V, Wang Q (2020) Lightweight aead and hashing using the sparkle permutation family. IACR Transactions on Symmetric Cryptology pp 208–261
7. Beyne T, Chen YL, Dobraunig C, Mennink B (2022) Multi-user security of the elephant v2 authenticated encryption mode. In: Selected Areas in Cryptography: 28th International Conference, Virtual Event, September 29–October 1, 2021, Revised Selected Papers, Springer, pp 155–178
8. Buchanan WJ, Maglaras L (2023) Review of the nist light-weight cryptography finalist. arXiv preprint arXiv:230314785
9. Dobraunig C, Eichlseder M, Mangard S, Mendel F, Mennink B, Primas R, Unterluggauer T (2020) Isap v2. 0. IACR Transactions on Symmetric Cryptology
10. Dobraunig C, Eichlseder M, Mendel F, Schläffer M (2021) Ascon v1. 2: Lightweight authenticated encryption and hashing. Journal of Cryptology 34:1–42
11. Duka AV (2022) Software implementation and benchmarking of tinyjambu on programmable logic controllers. In: The 16th International Conference Interdisciplinarity in Engineering: Inter-Eng 2022 Conference Proceedings, Springer, pp 889–899
12. ElBeltagy M, Alexan W, Elkhamry A, Moustafa M, Hussein HH (2022) Image encryption through rössler system, prng s-box and recamán's sequence. In: 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), IEEE, pp 0716–0722
13. Harb S, Ahmad MO, Swamy M (2022) A high-speed FPGA implementation of aes for large scale embedded systems and its applications. In: 2022 13th International Conference on Information and Communication Systems (ICICS), IEEE, pp 59–64
14. Hasan M, Chang D (2023) Lynx: Family of lightweight authenticated encryption schemes based on tweakable blockcipher. Cryptology ePrint

Archive
15. Hell M, Johansson T, Maximov A, Meier W, Sönnerup J, Yoshida H (2019) Grain-128aeadv2-a lightweight aead stream cipher. The NIST Lightweight Cryptography Standardization Process
16. Hemmati A, Rahmani AM (2022) The internet of autonomous things applications: A taxonomy, technologies, and future directions. Internet of Things 20:100635
17. Hussain S, Jamal SS, Shah T, Hussain I (2020) A power associative loop structure for the construction of non-linear components of block cipher. IEEE Access 8:123492–123506, DOI 10.1109/ACCESS.2020.3005087
18. Kareem SM, Symmetric Algorithm D (2023) Des and aes algorithms for image encryption. Al-Qadisiyah Journal of Pure Science
19. Katagi M, Moriai S, et al. (2008) Lightweight cryptography for the internet of things. sony corporation 2008:7–10
20. Khairallah M, Khairallah M (2022) Romulus: Lighweight aead from tweakable block ciphers. Hardware Oriented Authenticated Encryption Based on Tweakable Block Ciphers pp 115–134
21. Kitahara T, Hira R, Hara-Azumi Y, Miyahara D, Li Y, Sakiyama K (2022) Optimized software implementations of ascon, grain-128aead, and tinyjambu on arm cortex-m0. In: 2022 Tenth International Symposium on Computing and Networking Workshops (CANDARW), pp 316–322, DOI 10.1109/CANDARW57323.2022.00030
22. Mishra Z, Mishra S, Acharya B (2021) High throughput novel architecture of sit cipher for iot application. In: Nanoelectronics, Circuits and Communication Systems: Proceeding of NCCS 2019, Springer, pp 267–276
23. Mondal B, Singh JP (2022) A lightweight image encryption scheme based on chaos and diffusion circuit. Multimedia Tools and Applications 81(24):34547–34571
24. Naito Y, Matsui M, Sugawara T, Suzuki D (2019) Saeb: a lightweight blockcipher-based aead mode of operation. Cryptology ePrint Archive
25. Poojary A, Kiran Kumar V, Nagesh H (2022) FPGA implementation novel lightweight mbrisi cipher. Journal of Ambient Intelligence and Humanized Computing pp 1–13
26. Rashidi B (2021) Flexible and high-throughput structures of camellia block cipher for security of the internet of things. IET Computers & Digital Techniques 15(3):171–184
27. Report GVR (2018) IoT Device Management Market Size & Share Report, 2030. URL `https://www.grandviewresearch.com/industry-analysis/iot-device-management-market`
28. Roy S, Shrivastava M, Pandey CV, Nayak SK, Rawat U (2021) Ievca: An efficient image encryption technique for iot applications using 2-d vonneumann cellular automata - multimedia tools and applications. URL `https://link.springer.com/article/10.1007/s11042-020-09880-`
29. Som S, Kotal A, Mitra A, Palit S, Chaudhuri BB (2014) A chaos based partial image encryption scheme. In: 2014 2nd International Conference on Business and Information Management (ICBIM), pp 58–63, DOI 10.1109/ICBIM.2014.6970933
30. Weber AG (2019) The usc-sipi image database: Version 5, original release: October 1997, signal and image processing institute, university of southern california, department of electrical engineering
31. Wu H, Huang T (2021) TinyJAMBU: A family of lightweight authenticated encryption algorithms (version 2). Submission to the NIST Lightweight Cryptography Standardization Process
32. Yan L, Li L, Guo Y (2023) Dbst: a lightweight block cipher based on dynamic s-box. Frontiers of Computer Science 17(3):173805
33. Yang G, Shi Z, Chen C, Xiong H, Li F, Hu H, Wan Z (2022) Hardware optimizations of fruit-80 stream cipher: Smaller than grain. ACM Transactions on Reconfigurable Technology and Systems
34. Yang H, Wong KW, Liao X, Zhang W, Wei P (2010) A fast image encryption and authentication scheme based on chaotic maps. Communications in Nonlinear Science and Numerical Simulation 15(11):3507–3517
35. Yousaf A, Razaq A, Baig H (2022) A lightweight image encryption algorithm based on patterns in rubik's revenge cube. Multimedia Tools and Applications 81(20):28987–28998
36. Yousaf MA, Alolaiyan H, Ahmad M, Dilbar M, Razaq A (2020) Comparison of pre and post-action of a finite abelian group over certain nonlinear schemes. IEEE Access 8:39781–39792, DOI 10.1109/ACCESS.2020.2975880
37. Zhang L, Wu R, Zhang Y, Zheng Y, Wu W (2023) Lllwbc: A new low-latency light-weight block cipher. In: International Conference on Information Security and Cryptology, Springer, pp 23–42
38. Zhang X, Tang S, Li T, Li X, Wang C (2023) Gfrx: A new lightweight block cipher for resource-constrained iot nodes. Electronics 12(2):405
39. Zhu C (2012) A novel image encryption scheme based on improved hyperchaotic sequences. Optics communications 285(1):29–37