IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 30 May 2023, accepted 29 July 2023, date of publication 2 August 2023, date of current version 14 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3301128

RESEARCH ARTICLE

Two New Lightweight Cryptographic Hash Functions Based on Saturnin and Beetle for the Internet of Things

SUSILA WINDARTA[1], (Member, IEEE), SURYADI SURYADI[2], KALAMULLAH RAMLI[1], (Member, IEEE), ANDRIANI ADI LESTARI[3], (Member, IEEE), WILDAN WILDAN[4], BERNARDI PRANGGONO[5], (Senior Member, IEEE), AND RINI WISNU WARDHANI[6], (Graduate Student Member, IEEE)

[1]Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Depok, Jawa Barat 16424, Indonesia
[2]Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok, Jawa Barat 16424, Indonesia
[3]Cryptographic Hardware Engineering Study Program, Cryptography Department, Politeknik Siber dan Sandi Negara, Ciseeng, Bogor, Jawa Barat 16120, Indonesia
[4]Cybersecurity and Cryptography Technology Research and Development Center, National Cyber and Crypto Agency, Depok, Jawa Barat 16516, Indonesia
[5]School of Computing and Information Science, Anglia Ruskin University, CB1 1PT Cambridge, U.K.
[6]School of Computer Science and Engineering, Pusan National University, Busan 609735, South Korea

Corresponding author: Kalamullah Ramli (kalamullah.ramli@ui.ac.id)

This work was supported by Universitas Indonesia through the Hibah Publikasi Terindeks Internasional (PUTI) Q2 2023 Scheme under Contract NKB-820/UN2.RST/HKP.05.00/2023. The work of Susila Windarta was supported in part by Lembaga Pengelola Dana Pendidikan (LPDP), Ministry of Finance of the Republic of Indonesia under Contract 20194210114181.

ABSTRACT With the enormous growth in Internet of Things (IoT) applications, the volume of data shared among IoT devices is vastly increasing. Extensive IoT device connectivity and substantial data transmission have made information integrity susceptible to various assaults. Therefore, hash functions are required to ensure data integrity in IoT networks. IoT systems are constrained by their complexity, necessitating the consumption of minimal computational power. As a result, lightweight hash functions have been selected as the solution for the IoT data integrity issue. We present two lightweight hash functions, ALIT-Hash and TJUILIK-Hash, based on the SATURNIN block cipher and the Beetle mode of operation. In particular, we created TJUILIK-Hash by modifying the SATURNIN block cipher. The strength of the proposed hash functions is evaluated through security analysis and performance testing. ALIT-Hash and TJUILIK-Hash both show reasonably good resistance to differential and linear cryptanalysis. Hardware implementations on a cost-effective and low-power microcontroller board (ATmega2560) demonstrate an average execution time of 0.746 microseconds for the TJUILIK-Hash algorithm. Performance evaluations on a 64-bit personal computer indicate that the ALIT-Hash and TJUILIK-Hash implementations exhibit comparable speed and throughput to seven other evaluated hash functions. Simulation experiments employing Contiki-NG and the Cooja simulator confirm the good performance of these two hash functions relative to PHOTON-Beetle-Hash, PHOTON, and SPONGENT across five metrics. The hash functions pass seven cryptographic randomness tests and pass all tests in the National Institute of Standards and Technology (NIST) Statistical Test Suite (STS). Therefore, the implementation of both proposed hash functions should be considered, as they are both cost-effective and provide an adequate level of security, which is essential for IoT devices with limited resources.

INDEX TERMS Lightweight cryptographic hash function, sponge construction, SATURNIN block cipher, Beetle mode of operation, ALIT-Hash, TJUILIK-Hash, the Internet of Things.

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

I. INTRODUCTION

The rate at which devices around us are achieving internet connectivity is accelerating. According to a current Statista

study, the number of Internet of Things (IoT) devices in operation will more than double between 2020 and 2030, increasing from 9.7 billion to over 29 billion [1]. China will have the most IoT devices in 2030, including over 5 billion consumer devices. This expansion is establishing the IoT as a promising emerging market that could be a pillar of the growing digital economy. After reaching USD 4.5 trillion in 2021, the IoT market is expected to grow to over USD 5 trillion by 2025 [2].

The massive connectivity of IoT devices and the exchange of large amounts of data present numerous security challenges. Researchers [3], [4] have identified seven security challenges in IoT systems: authentication, authorization, integrity, confidentiality, nonrepudiation, availability, and privacy. Figure 1 summarizes the attacks, issues, and requirements relevant to each layer of the IoT architecture. The reference IoT architecture used in this study has three layers: the application layer, the network layer, and the perception layer. In Fig. 1, security requirements that demand lightweight hash functions are highlighted in bold. The multitude of security requirements demanding lightweight hash functions underscores the importance of employing hash functions in IoT systems.

In IoT systems, cryptographic hash functions are among the most frequently employed cryptographic primitives [5], [6], [7], [8]. Cryptographic hash functions have been implemented in the contexts of cybersecurity and information security applications, such as data integrity [7], [8], entity authentication [7], [8], digital signatures [5], [6], [9], cryptographic protocols [10], [11], pseudorandom number generation [7], cryptographic key derivation [7], [9], [10], cryptographic key generation [9], password security, and blockchain applications [12], [13], [14].

Accordingly, governments, the private sector, and academic institutions have all endeavored to build and analyze cryptographic hash algorithms. A hash function designer must consider both performance and security issues [15], [16], [17]. As a lightweight cryptography (LWC) development initiative, the National Institute of Standards and Technology (NIST) launched a project in 2017 to standardize LWC. LWC is a vital part of the cryptographic field that seeks to provide energy- and memory-efficient cryptographic algorithms for devices with limited resources. An LWC report published by NIST [18] recognizes this initiative. In August 2018, NIST announced a ''request for algorithms'' to participate in the LWC standardization process [19]. After more than two years of competition, on March 29, 2021, NIST announced the ten finalists competing in the final round. Six candidates, namely, ASCON [20], Elephant [21], ISAP [22], PHOTON-**Beetle** [23], Sparkle (SCHWAEMM and ESCH) [24], and Xoodyak [25], used sponge-based construction. Two candidates employed block ciphers, GIFT-COFB [26] and TinyJAMBU [27]. Romulus [28] was the sole candidate based on a tweakable block cipher. Grain-128AEAD was the only candidate in the final round based on a stream cipher algorithm [29]. On February 20, 2023, NIST declared ASCON the LWC competition winner.

Given the considerations mentioned earlier, the design trend of LWC is built on permutations as cryptographic components. The majority of the variants used involve sponge construction or expansions. Sponge-constructed permutations are preferred since they are relatively easy to transform into different cryptographic primitives. Examples include hash functions [30], authenticated encryption with associated data (AEAD) [31], message authentication code (MAC) [32], and pseudorandom bit generators (PRBGs) [33]. Notably, the software and hardware performance of the previously mentioned alternatives varies due to their distinct structures. Feistel networks and substitution-permutation networks (SPNs) are employed for the round function, and the number of rounds utilized during construction significantly impacts an algorithm's performance.

The contributions of this study are summarized as follows.

1) We develop the ALIT and TJUILIK lightweight hash function algorithms, which are based on the SATURNIN block cipher algorithm and the sponge-based **Beetle** mode of operation.
2) We modify the super S-box of SATURNIN and analyze the security aspects of the modified super S-box. The modified super S-box offers an adequate level of security in terms of differential and linear cryptanalysis.
3) We elucidate a cost-effective hardware implementation for implementing lightweight hash functions on resource-constrained devices with low power consumption.
4) We present average speed and throughput performance evaluations for hash function algorithms that employ the same mode of operation.
5) We simulate and evaluate the two proposed hash functions in comparison with seven other hash functions in Contiki-NG [34] and the Cooja simulator, measuring performance based on five metrics: throughput (bits/s), energy consumption (mJ), power consumption (mW), ROM (bytes), and RAM (bytes).
6) We perform a randomness property assessment of all proposed hash algorithms using seven cryptographic randomness tests and fifteen tests from the NIST Statistical Test Suite (STS).

The remaining sections of this paper are organized as follows. Section II presents a review of the relevant literature. Section III addresses the specifications and security aspects of the SATURNIN block cipher. Section IV introduces the **Beetle** mode of operation. The design rationale for our proposals is presented in Section V. In Section VI, we propose two lightweight cryptographic hash functions (LWCHFs). Section VII presents a security analysis of the proposed LWCHFs. In Section VIII, three performance analyses and two randomness analyses of the proposed hash functions are presented. The paper is concluded in the final section.

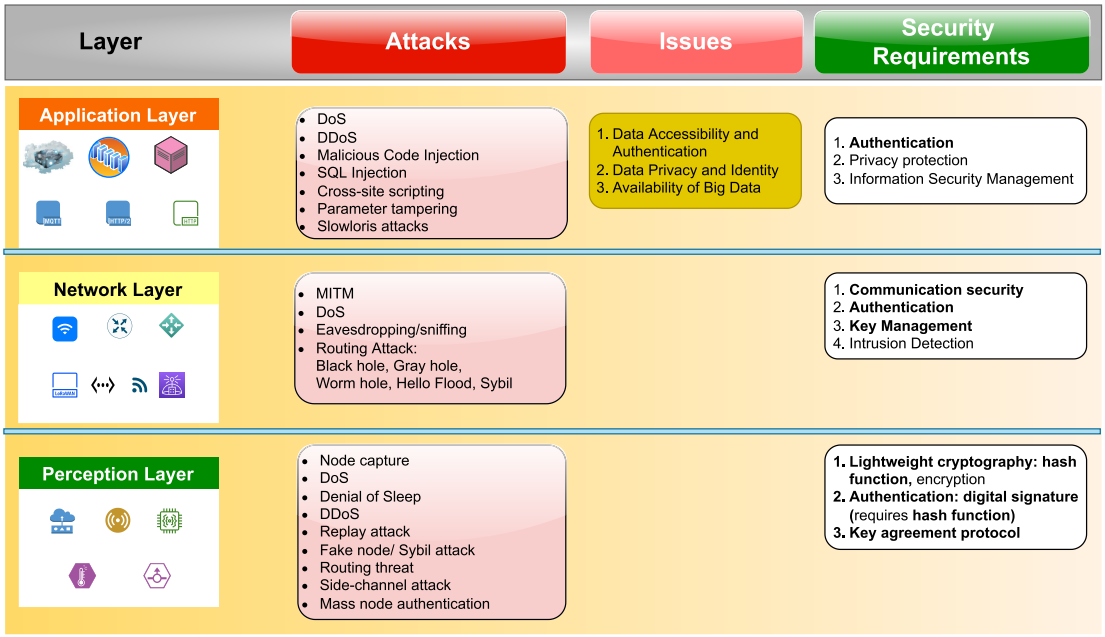

FIGURE 1. Three-layer IoT architecture: attacks, issues and security requirements.

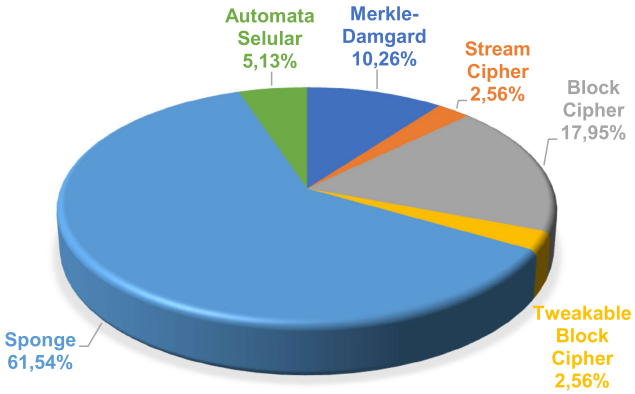

FIGURE 2. LWCHFs found in the literature.

II. RELATED WORKS

To our knowledge, the development of lightweight hash functions began with the proposal of dM-PRESENT by Bogdanov et al. [35]. Since then, researchers have developed various lightweight hash functions or families of such functions, resulting in a total of thirty-eight hash functions or families, each with unique characteristics and properties. Figure 2 summarizes the LWCHFs found in the literature.

This section focuses on seven algorithms. The first three, SPONGENT [36], [37], PHOTON [38], and LESAMNTA-LW [39], are recognized as following the ISO/IEC 29192-5:2016 standard [40]. The other four algorithms, namely, PHOTON-**Beetle**-Hash [23], ASCON-Hash [20], XOODYAK-Hash [41], and ESCH256 [24], were finalists in the NIST LWC competition. This competition aimed to identify the most suitable LWC algorithms for widespread deployment on IoT devices. ASCON-Hash emerged as the victor in the NIST LWC competition, solidifying its position as a leading lightweight hash function.

The SPONGENT family of hash functions was designed by Bogdanov et al. and presented at CHES 2011 [36]. This family is designed to provide hash values of 88, 128, 160, 224, and 256 bits in length to resist preimage attacks. The authors claim that the algorithm resists attacks targeting the hash function.

Guo et al. proposed PHOTON in their research paper [38]. This algorithm utilizes a sponge-like construction approach and the Advanced Encryption Standard (AES) and is a compact hash function requiring only 1120 gate equivalents (GE) to provide 64-bit security. Its speed is claimed to be competitive compared to similar algorithms.

The designers of LESAMNTA-LW claim that it is a secure and lightweight hash function with a 256-bit hash length [39]. The primary objective of its design is to achieve a small-scale hardware and software implementation. The algorithm employs Merkle–Damgård (MD) construction and an AES-based block builder. A 4-branch generalized Feistel network (GFN) and AES components (SubBytes and MixColumn) are used in the hash function. The MixColumn operation employs maximum distance separable (MDS) AES matrix multiplication specified via GF (2^8).

PHOTON-**Beetle**-Hash [23] utilizes the PHOTON_{256} permutations [38] as building blocks and the **Beetle** sponge mode [42]. This hash function accepts any input message $M \in \{0,1\}^*$ and returns a 256-bit hash value $\mathcal{H}(M) \in \{0,1\}^{256}$.

XOODYAK [41] is a cryptographic primitive intended for hash functions, PRBGs, authentication, encryption, or authenticated encryption (AE). XOODYAK uses the 384-bit XOODOO permutations [43], [44]. XOODOO is a family of

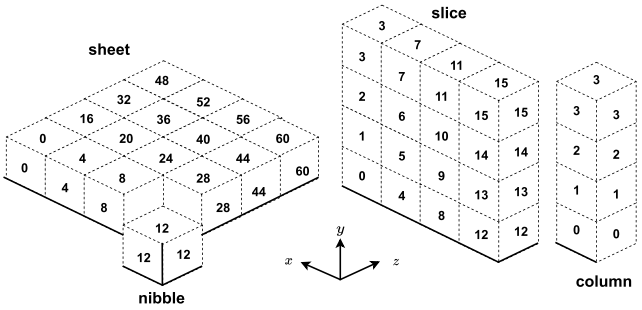

FIGURE 3. Different parts of SATURNIN's state.

permutations inspired by KECCAK-*p* [30], [45]. XOODYAK-Hash has a hash value length of 256 bits.

ESCHHash [24] has two variants: ESCH256 and ESCH384. These hash functions accept input of an arbitrary bit length and return hash values of 256 bits and 384 bits, respectively. The main proposal for these hash functions is based on the SPARKLE [46] permutation group, with rate r and capacity c.

ASCON-Hash [20] is a member of the ASCON family of cryptographic algorithms proposed for the NIST LWC competition. Indeed, ASCON was the winner of the NIST LWC competition. Previously, ASCON was the winner of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [47], which was hosted by NIST for the standardization of an AE algorithm. The ASCON hash function is a variant of the ASCON algorithm designed for hash function purposes. It uses a permutation-based design similar to that of AES, employing a 64-bit nonce and a 128-bit key to provide security. The algorithm operates on 32-bit words and consists of several operations, including bit manipulation, substitution, and permutation. To obtain the final hash value, the algorithm truncates the output of the last permutation. The designers of ASCON-Hash aimed to create a hash function that could be implemented efficiently in hardware and software. To this end, they ensured that the algorithm would have a low gate count, making it feasible to implement on devices with limited resources, such as low-power microcontrollers and IoT devices. Additionally, the hash function has a fixed output size of 256 bits.

III. SATURNIN BLOCK CIPHER

This section describes the design of the SATURNIN block cipher algorithm, a building block of the proposed hash functions.

SATURNIN [48] operates on 256-bit message blocks and 256-bit keys to generate 256-bit ciphertext blocks. The algorithm utilizes a 256-bit internal state known as a cube. The three subsets of the cube (see Fig. 3) are labeled according to the same nomenclature used for the Keccak-f state components in SHA-3 [49].

SATURNIN is an even-round SPN cipher. Following the definition of Canteaut et al., a super-round is a combination of two consecutive rounds. SATURNIN employs a 256-bit internal state ($\mathbf{X}$) and a 256-bit key state ($\mathbf{K}$), each structured as a nibble ($4 \times 4 \times 4$) cube. The following round constants are generated using two 16-bit words, RC_0 and RC_1. The encryption process of the SATURNIN block cipher is described in Algorithm 1.

ALGORITHM 1: SATURNIN State Encryption

Input: State $\mathbf{X} \in (F_2^{16})$, Key $\mathbf{K} \in (F_2^{16})$, $R \in \mathbb{N}$, $D \in \{0, 1, \ldots 8\}$
Output: State $\mathbf{X} \in (F_2^{16})$

$\mathbf{X} \leftarrow \mathbf{X} \oplus \mathbf{K}$;
$RC_0 \leftarrow$ 0xfe00$|(R \ll 4)|D$; $RC_1 \leftarrow RC_0$;
for $r = 0$ **to** $R - 1$ **do**
  $\mathbf{X} \leftarrow \mathbf{S}(\mathbf{X})$;
  $\mathbf{X} \leftarrow \mathbf{M}(\mathbf{X})$;
  $\mathbf{X} \leftarrow \mathbf{S}(\mathbf{X})$;
  if $r \mod 2 \equiv 1$ **then**
    $\mathbf{X} \leftarrow \mathbf{SR}_{\text{slice}}(\mathbf{X})$;
    $\mathbf{X} \leftarrow \mathbf{M}(\mathbf{X})$;
    $\mathbf{X} \leftarrow \mathbf{SR}^{-1}_{\text{slice}}(\mathbf{X})$;
    $\mathbf{X}_0 \leftarrow \mathbf{X}_0 \oplus RC_0$;
    $\mathbf{X}_8 \leftarrow \mathbf{X}_8 \oplus RC_1$;
    $\mathbf{X} \leftarrow \mathbf{X} \oplus \text{rot}(\mathbf{K})$;
  else
    $\mathbf{X} \leftarrow \mathbf{SR}_{\text{sheet}}(\mathbf{X})$;
    $\mathbf{X} \leftarrow \mathbf{M}(\mathbf{X})$;
    $\mathbf{X} \leftarrow \mathbf{SR}^{-1}_{\text{sheet}}(\mathbf{X})$;
    $\mathbf{X}_0 \leftarrow \mathbf{X}_0 \oplus RC_0$;
    $\mathbf{X}_8 \leftarrow \mathbf{X}_8 \oplus RC_1$;
    $\mathbf{X} \leftarrow \mathbf{X} \oplus \mathbf{K}$;
  end
  for $j = 0$ **to** 15 **do**
    $RC_0 \leftarrow \text{clockLFSR}_0(RC_0)$;
    $RC_1 \leftarrow \text{clockLFSR}_1(RC_1)$;
  end
end

Parameters. SATURNIN has two parameters. The first parameter is R, which is the number of super-rounds, i.e., $\frac{\text{number of rounds}}{2}$. The proposed hash functions use $R = 16$. The second parameter is D, a 4-bit value called the domain separator.

Initialization. The input message is set to $\mathbf{X}$, and the main key is set to $\mathbf{K}$. RC_0 and RC_1 are filled with bit strings of the following form:

$$\underbrace{1\ldots1}_{7\text{ ones}}\underbrace{R_4\ldots R_0}_{R}\underbrace{D_3\ldots D_0}_{D}$$

such that the least significant bit is on the right. The first four bits denote the domain separator D, whereas the second five specify R. The initial operation of Round 0 is to XOR the internal state with the value $\mathbf{K}$.

Round function. Beginning with Round 0, the following internal state transformations are consecutively applied:

1) An S-box layer that implements S-box σ_0 for even-numbered nibbles and S-box σ_1 for odd-numbered

TABLE 1. S-boxes σ_0 and σ_1 of SATURNIN.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\sigma_0(x)$	0	6	e	1	f	4	7	d	9	8	c	5	2	a	3	b
$\sigma_1(x)$	0	9	d	2	f	1	b	7	6	4	5	3	8	c	a	e

nibbles. Table 1 is the lookup table used to determine the contents of these two S-boxes.

2) A nibble permutation SR_r that is dependent on the integer r.
3) A linear layer MC that applies sixteen duplicates of M over $(F_2^4)^4$ simultaneously to every column of the internal state. The definition of M is as follows:

$$M : \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \mapsto \begin{pmatrix} \alpha^2(a) \oplus \alpha^2(b) \oplus \alpha(c) \oplus d \\ a \oplus \alpha(b) \oplus b \oplus \alpha^2(c) \oplus c \oplus \alpha^2(d) \oplus d \\ a \oplus b \oplus \alpha^2(c) \oplus \alpha^2(d) \oplus \alpha(d) \\ \alpha^2(a) \oplus a \oplus \alpha^2(b) \oplus \alpha(b) \oplus b \oplus c \oplus \alpha(d) \oplus d \end{pmatrix}$$

where a is the smallest nibble index and α modifies $(x_0, \ldots, x_3)$ according to the following formula:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

4) SR_r^{-1}, the nibble permutation inverse.
5) A subkey addition performed just after every super-round. The subkey is the XOR of the round constant and the main key or a rotational form of the main key.

Round constant. RC_0 and RC_1 are modified by executing the following procedure 16 times: when the most significant bit of RC_i, $0 \leq i \leq 15$, is 0, RC_i is replaced by $RC_i \ll 1$; otherwise, it is replaced by $(RC_i \ll 1)\hat{}\mathtt{poly}_i$, with $\mathtt{poly}_0 = 0 \times 1002\mathtt{d}$ and $\mathtt{poly}_1 = 0 \times 10053$. RC_0 and RC_1 are then XORed with the internal state. Bit i in RC_0 is added to Bit 0 of the nibble at position $4i$. Similarly, Bit i in RC_1 is XORed with Bit 0 of the nibble with the index $(4i + 2)$.

Round key. The main key **K** is XORed with the internal state if the round index R is even; otherwise, the rotated version of the main key is XORed. The key nibble at location $(i + 20) \mod 64$ is accepted by the i-th nibble.

IV. BEETLE, A SPONGE-BASED MODE OF OPERATION

Beetle is a sponge-based mode of operation proposed by Chakraborti et al. [42], [50]. The researchers initially proposed Beetle for the AEAD algorithm. In the LWC competition, the authors of Beetle, working with the designer of the PHOTON hash function, proposed the PHOTON-Beetle family [23]. Thus, in addition to the AEAD algorithm, PHOTON-Beetle-Hash has also been proposed.

According to [42], in conventional sponge-based modes, the ciphertext is generated by XORing the plain text with the rate portion of the permutation output. The system then feeds this same value into itself as the input rate for the next iteration of permutation. However, in the Beetle construction process, a combined feedback approach is used such that the ciphertext block and the rate portion of the subsequent feedback differ from each other; this contrasts with a conventional sponge, in which these values are identical. This change enhances the mode's security without requiring extra storage space. Figure 4 shows the Beetle construction process.

V. DESIGN RATIONALE

This section explains why we chose to implement specific design elements. The rationale includes the choices that must be made when using the SATURNIN block cipher and the Beetle mode of operation.

A. CHOICE OF SATURNIN

When developing the proposed hash functions, we chose to use a 256-bit permutation from the available literature, and for this purpose, we selected the SATURNIN block cipher. Among the factors that led to the selection of SATURNIN is its design, which is based on a 20-year-long examination of AES. Because of this design approach, SATURNIN inherits the security features of AES while allowing efficient implementation and thorough security analyses considering both classical and quantum adversaries.

A comprehensive security analysis of SATURNIN is described in the work of Canteaut et al. [48]. The first part of the analysis concentrates on classical attacks: differential cryptanalysis, linear cryptanalysis, algebraic degrees, bicliques, impossible differential attacks, and subspace trails. The second part focuses on a Demirci–Selçuk meet-in-the-middle attack in 7.5-round SATURNIN. The final examination concerns the resistance of SATURNIN to quantum attacks.

Several researchers have investigated attacks on the SATURNIN block cipher or SATURNIN-Hash. According to Bao et al. [51], the likelihood of usable differential paths might be as low as 2^{-n}. This leads to more targeted rounds than all quantum collision attacks and conventional multi-collision distinguishers. These authors applied their attack model to AES, Rijndael, and SATURNIN to demonstrate its effectiveness. Distinguishing attacks were made on all rounds of Rijndael-128-160, Rijndael-128-224, AES-192, and AES-256. Other results include those for 10-round SATURNIN-256, 12-round Rijndael-160-256, and 11-round AES-128. Using a mixed-integer linear programming-based technique, Dong et al. [52], [53] used the related-key differentials of the underlying block cipher to automate the procedure of looking for configurations that could be used for rebound attacks. Their model directs searches in the quantum context toward features that reduce the cost and complexity of subsequent rebound attacks. They applied their strategy to SATURNIN-Hash, SKINNY, and Whirlpool and obtained good results. Dong et al. [54] employed a triangulating rebound attack on multiple AES-like hash algorithms to identify classical or

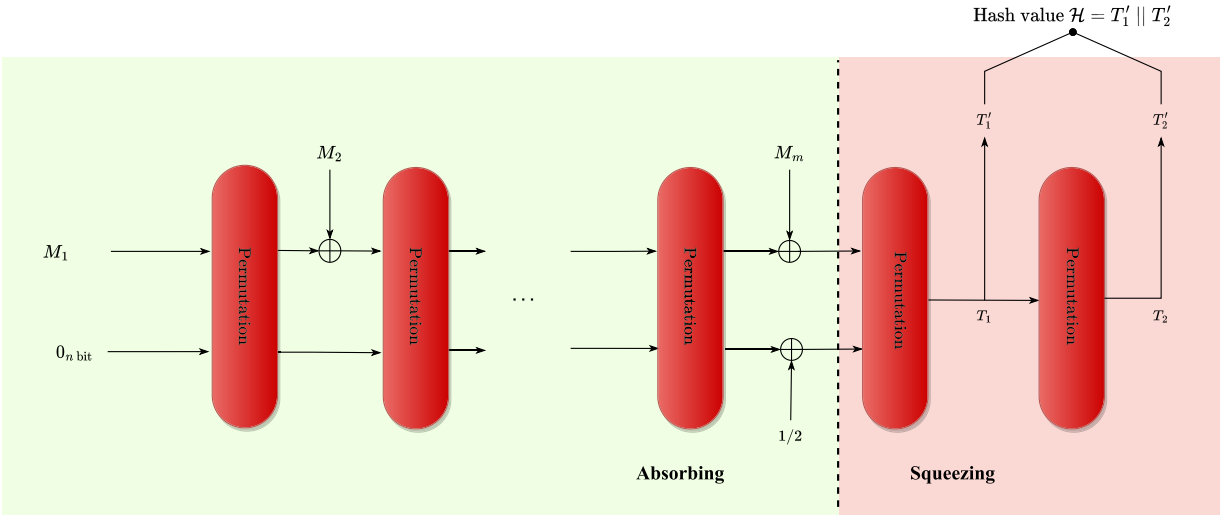

FIGURE 4. Beetle construction.

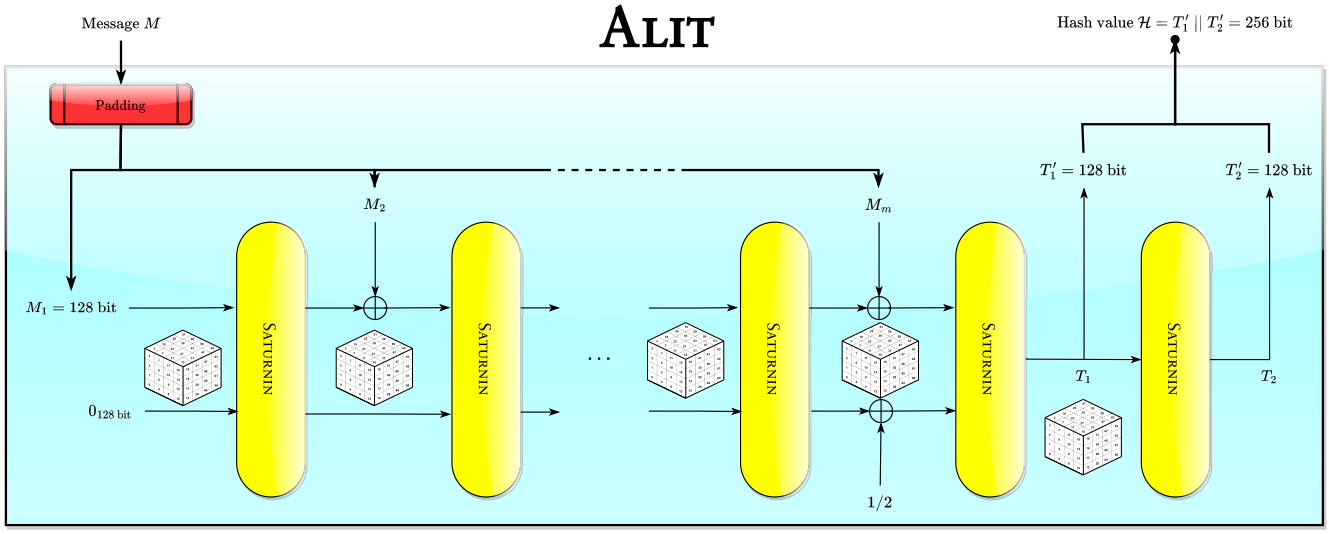

FIGURE 5. ALIT hash function employing m message blocks. $|M_1| = 128$ and $|M_i| = 32$ for $i = 2, \ldots, |M_m| \leq 32$. The hash value $\mathcal{H}$ is computed as $T_1 \,||\, T_2'$, where $T' = \text{Trunc}(T_i, 128)$.

quantum collisions. The attacked round number of targets such as AES-128, SKINNY, SATURNIN-Hash, and Grostl-512 increased by one to five. Because these attacks have a higher cost than the security target we aim to achieve, these distinguishers do not affect the security of ALIT-Hash or TJUILIK-Hash. In addition, SATURNIN can be utilized in the Beetle mode of operation without modifying the existing parameters.

B. CHOICE OF BEETLE

Since its introduction in 2007 by Bertoni et al. [55], sponge construction has become a common method for constructing lightweight cryptographic primitives. Beetle is one of the proposed modes of operation for sponge construction. The main novelty of the Beetle sponge mode [42] is its use of the permutation output and ciphertext block as feedback to produce the subsequent permutation input. The Beetle sponge mode prevents attackers from processing multiple blocks of data simultaneously, making it more difficult for them to find weaknesses in the cryptographic system and launch attacks. When a cryptographic system processes multiple data blocks simultaneously, attackers can often discover patterns or relationships between blocks. This situation can facilitate various attacks, such as collision attacks, in which the attacker tries to find two inputs that produce the same output. Therefore, the Beetle mode raises

the level of security, reduces the size of the state, and finally leads to low state implementation. This security upgrade ensures that even with a state size of only 256 bits, we can meet the desired security requirements.

VI. PROPOSED LWCHFs

A. ALIT LWCHF

The first contribution of this research is highlighted in this section. To design a lightweight hash function, which we refer to as the ALIT LWCHF, we propose combining the SATURNIN block cipher with the **Beetle** mode of operation. As described in [48], we apply SATURNIN with 16 super-rounds to make the compression function immune to related-key attacks. As a result, the developed hash function algorithm is expected to have strong cryptographic immunity in both linear and differential cryptanalysis.

The ALIT hash function converts a message $M \in \{0,1\}^*$ of arbitrary length into a fixed-length hash value $\mathcal{H} \in \{0,1\}^{256}$. The first 128 bits of the input message are designated for the first processing block, and the remainder of the input message is partitioned into 32-bit blocks. We set the main key to a 256-bit 0, written as 0^{256}. If the length of the input is not a multiple of 32 bits, it is padded with $10*$. In this procedure, the output of each permutation is XORed with the next 32-bit message block, which is concatenated with zeros, to determine the input for the subsequent permutation call. This initial state is assigned to the permutation's initial call. During the domain separation process for the last message block, a small constant is XORed into the capacity portion, depending on whether the last message block is complete or partial. A 256-bit hash value $\mathcal{H}$ is produced by combining the 128-bit values T_1 and T_2. Figure 5 illustrates the ALIT hash function, and the detailed procedure used to obtain the hash value is shown in Algorithms 2, 3, and 4.

Table 2 presents a set of test vectors for ALIT. The algorithm's input and output data are presented in hexadecimal notation. The first row of Table 2 shows the result obtained with an empty input message.

B. TJUILIK LWCHF

To obtain our second proposed LWCHF, we modify the super S-box of the SATURNIN block cipher. In this modification, the σ_0 and σ_1 S-boxes are replaced with 4-bit S-boxes from the literature [56], [57], [58], [59], [60]. The smallest differential uniformity value obtained for the resulting super S-box is 80, while the smallest linearity value is 3,072, as described in [48]. Our objective is to minimize the differential values of uniformity and linearity to provide differential and linear security superior to that of the initial concept. After investigation, we have found that the best S-box combination consists of the Serpent S-box S_3 [61] and the S-box G_3 proposed by Leander and Poschmann [56]. Both S-boxes are optimal. Serpent's block cipher algorithm was a finalist in the 2001 AES competition. Table 3 presents the S_3 and G_3 S-boxes.

ALGORITHM 2: **ALIT-Hash**[32](M)

Input: Message $M \in \{0,1\}^*$
Output: Hash value $\mathcal{H} \in \{0,1\}^{256}$

if $M = \lambda$ **then**
  $\mathbf{IV} \leftarrow 0 \,||\, 0$;
  $\mathcal{H} \leftarrow \mathbf{TAG}_{256}(\mathbf{IV} \oplus 1)$;
  return $\mathcal{H}$;
end
if $|M| \leq 128$ **then**
  $c_0 \leftarrow (|M| < 128)?1 : 2$;
  $\mathbf{IV} \leftarrow \mathbf{Pad}_{128}(M) \,||\, 0$;
  $\mathcal{H} \leftarrow \mathbf{TAG}_{256}(\mathbf{IV} \oplus c_0)$;
  return $\mathcal{H}$;
end
$M_1 \,\|\, M' \xleftarrow{(128,|M|-128)} M$;
$c_0 \leftarrow (32||M'| < 128)?1 : 2$;
$\mathbf{IV} \leftarrow M_1 \,||\, 0$;
$\mathbf{IV} \leftarrow \mathrm{HASH}_{32}(\mathbf{IV}, M', c_0)$;
$\mathcal{H} \leftarrow \mathbf{TAG}_{256}(\mathbf{IV})$;
return $\mathcal{H}$;

ALGORITHM 3: $\mathbf{HASH}_{32}(\mathbf{IV}, D, c_0)$

Input: $\mathbf{IV}, D, c_0$
Output: $\mathbf{IV}$

$D_1 \,\|\, D_2 \,\|\, \cdots \,\|\, D_d \xleftarrow{32} \mathbf{Pad}\,(D)$;
for $i = 1$ **to** d **do**
  $Y \,\|\, Z \xleftarrow{(32,224)} \text{SATURNIN}\,(\mathbf{IV})$;
  $W \leftarrow Y \oplus D_i$;
  $\mathbf{IV} \leftarrow W \,\|\, Z$;
end
$\mathbf{IV} \leftarrow \mathbf{IV} \oplus c_0$;
return $\mathbf{IV}$;

ALGORITHM 4: $\mathbf{TAG}_{256}(T_0)$

Input: T_0
Output: $\mathcal{H}$

for $i = 1$ **to** 2 **do**
  $T_i \leftarrow SATURNIN(T_{i-1})$;
end
$\mathcal{H} \leftarrow \mathbf{Trunc}(T_1, 128) \,\|\, \mathbf{Trunc}(T_2, 128)$;
return $\mathcal{H}$;

Table 4 compares the uniformity of the original and modified super S-box differentials. With the modification, the differential uniformity value decreases from 80 to 74, as shown in Table 4. Due to this reduction, the improved super S-box-based method is anticipated to be more secure than the original proposal. According to the AES design, there are always at least 25 functioning super S-boxes in play during any sequence of four successive super-rounds. For TJUILIK-Hash, which uses 16 super-rounds, the

TABLE 2. Test vectors for ALIT.

Message	Hash value
	e5a794caf76338810cbfc3973143da32 a1e082c9bff6e190adf96569eeab1ae5
00	afbefab5797009b34c2be5571dfbdc58 e80f4778eaed4979b2cb6beb0ffaa5ab
0001	f4759f2379b87e4734214641f43ba206 32ee0c064942a79fe5cef4d0c12951e5
000102	64143c247044404b970a1bf40240e937 da5589db431a0fdc71272becf8ddfc99
00010203	6ad0bd66e8ed643eb424ee59c0111e69 3cbe541292d49048792590e1867548dc
0001020304	8d6155a60a9e7f6a1bd55fefba4afaff 9ac43d0a2de52cfda643a1fe4e579981
000102030405	3e9a564190103cbaa71339ee10e7c250 f81a9350320567f4b21693b5313fa909
00010203040506	077e91cd9b4deccaca44ba8509764779 f29514e5269db3a7ef95a22a9779c442
0001020304050607	a3b0a7b433173779f347576f4c5fe1bc 6fcd08c9319f5b9c25cb1207743d65a9

TABLE 3. S_3 and G_3.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_3(x)$	0	f	b	8	c	9	6	3	d	1	2	4	a	7	5	e
$G_3(x)$	0	1	2	d	4	7	f	6	8	c	5	3	a	e	b	9

best 16-round differential characteristic probability is now $2^{-9.79\times 100} = 2^{-979}$.

Notably, the linearity of the modified super S-box is called into question by this change. When computing the linearity value, we obtain a maximum value of 3,200. This differs from the maximum linearity value of the original super S-box of SATURNIN, equal to 3,072, by 128 points. The maximum squared correlation of the linear relations equals $3,200^2 \times 2^{-32} \approx 2^{-8.713}$. Using the same reasoning used for the differential uniformity above, we find that the maximum squared correlation values for the 4-super-round and 8-super-round linear paths are no greater than $2^{-217.825}$ and $2^{-435.65}$, respectively, corresponding to reductions of $2^{220.7}$ and $2^{441.5}$ relative to 4 and 8 super-rounds of the original SATURNIN. We argue that this decrease does not affect the security against linear cryptanalysis for two reasons. First, the probability reduction is small, and second, the number of rounds used in the hash function is increased to 16 super-rounds (32 rounds). These two factors combine to ensure that the security against linear cryptanalysis remains robust, even with a moderate decrease in probability.

We call the above modification SATURNIN-Mod. We assert that all security analyses that apply to the SATURNIN block cipher also apply to SATURNIN-Mod. We use SATURNIN-Mod as a permutation to build TJUILIK-Hash. Figure 6 presents TJUILIK-Hash. Test vectors for TJUILIK-Hash are listed in Table 5. The input messages are the same as the test vectors used for ALIT-Hash. Tables 2 and 5 show the major differences in the generated hash values.

VII. SECURITY ANALYSIS

This section presents a security analysis of the ALIT and TJUILIK hash functions. The analysis covers collision resistance, preimage security, and the security of the block cipher algorithm as a permutation function. Table 6 summarizes the security claims regarding the ALIT and TJUILIK lightweight hash functions.

A. COLLISION SECURITY

Collision security is related to the computational difficulty of finding two distinct input values that produce the same hash value. A collision-resistant hash function is distinguished by its collision security. NIST requires that an LWCHF have 112-bit collision security.

To conduct a collision attack against ALIT-Hash[32] (also TJUILIK-Hash[32]), the attacker must perform p permutation calls. It is assumed that all states are accessible via permutation from the initial state. A bit sequence of 0^{256} is the initial state in this instance. An attacker can organize queries to make all query inputs or outputs accessible. Furthermore, if there is a collision in the output capacity components of two permutation calls, then the rate section of the message can be adjusted to cause state collisions, which can then be used to generate collisions in the hashes. Let $\text{Prob}_{\text{coll}}$ be the probability of this occurrence. The value of (1) determines the upper limit on $\text{Prob}_{\text{coll}}$:

$$\text{Prob}_{\text{coll}} = \frac{p^2}{2^{256-r-1}}. \tag{1}$$

The extra bit originates from the addition of a constant to the capacity part. For $r = 32$, the query complexity for collision attacks is 111.5 bits.

B. PREIMAGE SECURITY

Preimage security is associated with the ability of a hash function to withstand preimage attacks, namely, searches for an input message value given a hash value. In ALIT-Hash[r] and TJUILIK-Hash[r], the hash value size is defined as 256 bits, and the squeezing level is 128 bits. In the case of ALIT, the attacker must identify Z such that $\text{SATURNIN}(T_1 \parallel Z_1) = T_2 \parallel *$ or $\text{SATURNIN}^{-1}(T_2 \parallel Z) = T_1$ to discover the preimage hash value of $T_1 \parallel T_2$. In the case of TJUILIK, the underlying permutation is SATURNIN-Mod. The probability of this occurrence, called Prob_{pre}, is restricted to within the upper limit given in (2):

$$\text{Prob}_{\text{pre}} = \frac{p}{2^{128}}. \tag{2}$$

VIII. PERFORMANCE AND RANDOMNESS TESTS

This section describes performance and randomness tests of the two proposed lightweight hash functions. The performance evaluations consist of a hardware performance

TABLE 4. Super S-box differential uniformity.

Property	SATURNIN Super S-box	Proposed super S-box
Differential uniformity	80	74
Highest probability of a nontrivial differential	$80 \times 2^{-16} \approx 2^{-9.68}$	$74 \times 2^{-16} \approx 2^{-9.79}$
Best 4-super-round differential characteristic probability	$2^{-9.68\times 25} = 2^{-241.90}$	$2^{-9.79\times 25} = 2^{-244.76}$
Best 8-super-round differential characteristic probability	$2^{-9.68\times 50} = 2^{-483.90}$	$2^{-9.79\times 50} = 2^{-489.50}$

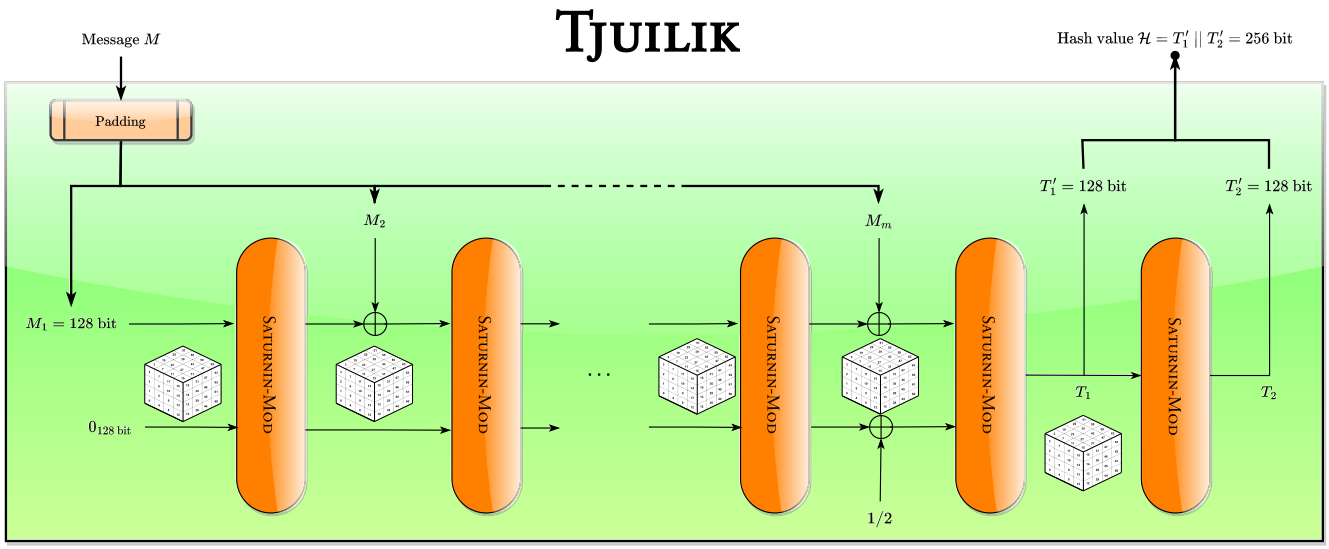

FIGURE 6. TJUILIK hash function employing m message blocks. $|M_1| = 128$ and $|M_i| = 32$ for $i = 2, \ldots, |M_m| \leq 32$. The hash value $\mathcal{H}$ is computed as $T_1 \,||\, T_2'$, where $T' = \text{Trunc}(T_i, 128)$.

evaluation, a software performance evaluation, and simulations on Contiki-NG [34] and the Cooja simulator. The randomness testing includes both cryptographic and statistical randomness tests. Fast NIST STS [62], a more efficient version of the NIST STS [63], is used for the statistical randomness tests.

A. HARDWARE PERFORMANCE

We implemented one of the proposed hash functions in a low-cost environment to achieve a reasonable average hash rate for hardware-specific IoT applications using the ATmega328P microcontroller. We performed a single hash operation with a 256-bit hash length and a 128-bit squeeze rate. This operation is a standard measurement for typical IoT sensor data in sensor nodes based on RAM-constrained devices. This investigation focused solely on TJUILIK-Hash. We assert that the hardware implementation performance of ALIT-Hash is comparable to that of TJUILIK-Hash due to their similar structures.

Experiments were carried out by optimizing the hash function for performance or memory consumption and then translating it into the required headers and function calls. We then generated test vectors with varying input sizes and compared the hash values obtained with the optimized hash function to the known hash values. These implementations prioritized efficient 16-byte, 128-byte, and 1024-byte embedded architectures, specifically AVR.

To compare its performance on AVR with that of other hash algorithms from [64] in an identical manner, we implemented TJUILIK-Hash in an ATmega2560 processor environment with a maximum CPU speed of 16 MHz. The ATmega2560 serves as the base for the Arduino Mega 2560 microcontroller board, which includes 54 digital I/O pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, and an ICSP header. Fifteen of the digital I/O pins can function as PWM outputs. Table 7 lists all results, and Fig. 7 depicts the average speed analysis for TJUILIK-Hash in this 16-bit ATmega implementation. Accordingly, we can compare the

TABLE 5. Test vectors for TJUILIK.

Message	Hash value
	f0c85a7af9a317f3731f8c43602a8a94 26f0d4a371bcbdb0689a5c038acdebc4
00	1dc379f5327adf6871b2f8833e1c07fe 991a042d6343579fabdc334587d01b27
0001	16b7e6852904c736c0568902d96425cf 9f4ce70163a964da9184a102c08c561a
000102	374c2b1c60e63a70bf2bc5e73e59219c 250e7d1947fc288f47fe400690d38ffe
00010203	ba5a1228c173ab8e64f18bf5ebe397f5 275f13dae5ff1c346ab7f64fa26d166b
0001020304	6532b9d3cba6dc74fa30a88e83ac2a24 64509357bbe1afcf8b96a9151e88fad5
000102030405	1356aa03f891abbf53746d0ffb65b06e d17139b5e50644d24655de3b85c414c3
00010203040506	578b0686e5e3e942f6dd2992a017af35 8adefc6ad56ce91eb997866cc3c73a8a
0001020304050607	a12c17c76cccb1d8c571d6eb76da5782 a75bb23b4213b3ecb705de67fcdba476

TABLE 6. Security of ALIT-Hash and TJUILIK-Hash.

Hash function	Security	Time complexity security
ALIT	Collision	2^{112} (query complexity: $2^{111.5}$)
	Preimage	2^{128}
TJUILIK	Collision	2^{112} (query complexity : $2^{111.5}$)
	Preimage	2^{128}

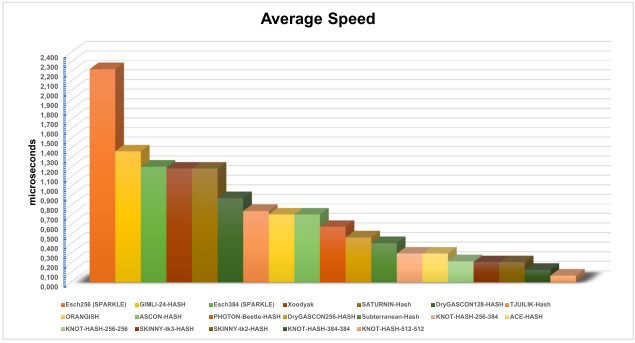

FIGURE 7. Hash algorithm performance comparison in a 16-bit AVR ATmega2560 implementation. We implement TJUILIK-Hash in 16-byte, 128-byte, and 1024-byte versions and then compare its average performance with that of other hash algorithms.

speeds of different hash algorithms and conduct a test vector analysis to compare the hash values obtained from the hardware implementation with the expected results.

Among the evaluated algorithms, TJUILIK-Hash stands out with an average execution time of 0.746 microseconds, indicating its superior efficiency and speed in generating hash values compared to other algorithms. TJUILIK-Hash outperforms most algorithms in the 128-byte and 16-byte input categories, with significantly shorter average execution times. For 1024-byte inputs, although it takes slightly longer than some other algorithms, the proposed algorithm nevertheless achieves a notably short average execution time.

Upon comparing the average execution times of all algorithms, it is evident that TJUILIK-Hash consistently demonstrates good performance across different input sizes. It outperforms several well-known hash algorithms, such as ESCH256 (Sparkle), GIMLI-24-Hash, ESCH384 (Sparkle), XOODYAK-Hash, SATURNIN-Hash, and DryGASCON128-Hash. These findings emphasize the efficiency and effectiveness of TJUILIK-Hash as a hashing algorithm, particularly on an AVR platform. The superior performance of TJUILIK-Hash in terms of average execution time makes it a good choice for applications in which speed and efficiency are crucial considerations.

B. SOFTWARE PERFORMANCE

In the software performance evaluation, we compared ALIT-Hash and TJUILIK-Hash with seven lightweight hash function algorithms: PHOTON, SPONGENT, LESAMNTA-LW, PHOTON-Beetle-Hash, ASCON-Hash, XOODYAK-Hash, and ESCH256. The first three of these hash functions follow the ISO/IEC 29192-5:2016 standard, and the remaining four were finalists in the NIST LWC competition. All lightweight hash functions have a hash value length of 256 bits. All algorithms were integrated into a C implementation and tested on a system with a 3.30 GHz AMD Ryzen 9 5900HX processor with Radeon Graphics and 32 GB of RAM running Windows 11 Pro. Our aim was to evaluate each algorithm's performance in terms of speed and efficiency through the integration process. We chose to implement all algorithms under identical conditions and specifications and not to refer to other researchers' implementation results to ensure fair comparisons.

We performed a speed test by calculating the hash value of input with a size equivalent to the longest LoRaWAN payload, that is, 242 bytes [65]. We focused on the time taken to process the data message and used the results to evaluate the impact of the hash function design on overall throughput. We present the speed test results to provide insight into the performance of hash function algorithms and their impact on overall throughput. These results can guide future improvements and optimizations of such algorithms to enhance their performance. Table 8 presents the speed and throughput performance results.

Among the algorithms in Table 8, ASCON-Hash exhibited the fastest performance, with a recorded processing time of 0.00031 ms. The achieved throughput of 819,462.228 bits/ms is significant. The Sparkle algorithm, also known as ESCH256, demonstrated a fast processing time of 0.00047 ms and a high throughput of 543,524.416 bits/ms.

The ALIT-Hash algorithm also showed noteworthy performance features, with a processing time of 0.00063 ms and a throughput of 409,600.000 bits/ms. The XOODYAK-Hash algorithm achieved a processing time of 0.00157 ms, resulting in a throughput of 163,265.306 bits/ms.

TABLE 7. Performance comparison of TJUILIK-Hash with other hash algorithms on AVR.

Algorithm	Hash bits	1024 bytes	128 bytes	16 bytes	Average (microseconds)
ESCH256 (Sparkle)	256	1.900	1.650	3.150	2.230
GIMLI-24-Hash	256	1.290	1.060	1.760	1.370
ESCH384 (Sparkle)	384	1.200	0.960	1.480	1.210
XOODYAK-Hash	256	0.920	0.830	1.830	1.190
SATURNIN-Hash	256	0.940	0.770	1.860	1.190
DryGASCON128-Hash	256	0.530	0.510	1.620	0.880
TJUILIK-Hash	256	1.525	0.238	0.475	0.746
ORANGISH	256	0.660	0.550	0.920	0.710
ASCON-Hash	256	0.660	0.550	0.910	0.710
PHOTON-Beetle-Hash	256	0.170	0.180	1.380	0.580
DryGASCON256-Hash	512	0.390	0.340	0.670	0.470
Subterranean-Hash	256	0.270	0.260	0.700	0.410
KNOT-Hash-256-384	256	0.280	0.230	0.380	0.300
ACE-Hash	256	0.280	0.230	0.380	0.300
KNOT-Hash-256-256	256	0.140	0.140	0.380	0.220
SKINNY-tk3-Hash	256	0.220	0.180	0.310	0.210
SKINNY-tk2-Hash	256	0.130	0.130	0.360	0.210
KNOT-Hash-384-384	384	0.080	0.080	0.220	0.130
KNOT-Hash-512-512	512	0.060	0.050	0.110	0.070

The cryptographic algorithm LESAMNTA-LW showed a processing time of 0.01100 ms and a correspondingly reduced throughput of 23,272.727 bits/ms compared to the other algorithms. For TJUILIK-Hash, this study found a slight increase in processing time to 0.01571 ms, leading to a throughput of 16,296.391 bits/ms.

The PHOTON-Beetle-Hash algorithm displayed reduced operational efficiency, as evidenced by its longer processing time of 0.10147 ms and resulting throughput of 2,522.839 bits/ms. The PHOTON system achieved a processing time of 0.26100 ms, yielding a throughput of 980.843 bits/ms. The SPONGENT algorithm had the lowest efficiency, with a processing time of 1.51300 ms and a throughput of 169.200 bits/ms.

TABLE 8. Comparison of software performance.

Algorithm	Speed (ms)	Throughput (bits/ms)
ASCON-Hash	0.00031	819,462.228
ESCH256 (Sparkle)	0.00047	543,524.416
ALIT-Hash	0.00063	409,600.000
XOODYAK-Hash	0.00157	163,265.306
LESAMNTA-LW	0.01100	23,272.727
TJUILIK-Hash	0.01571	16,296.391
PHOTON-Beetle-Hash	0.10147	2,522.839
PHOTON	0.26100	980.843
SPONGENT	1.51300	169.200

C. PERFORMANCE IN RESOURCE-CONSTRAINED ENVIRONMENTS: CONTIKI-NG AND COOJA SIMULATIONS

We simulated ALIT-Hash, TJUILIK-Hash, and seven other algorithms using the Contiki-NG operating system [34] and the Cooja simulator. Our simulation setup consisted of the Ubuntu 20.04.6 LTS operating system with 16 GB of RAM running on Windows Subsystem for Linux 2 on Windows 11 Pro. The simulation involved a wireless network platform in which Z1 motes from Zolertia acted as both senders and receivers. The Z1 mote has a second-generation MSP430F2617 low-power microcontroller featuring a 16-bit RISC CPU clocked at 16 MHz, a calibrated clock generator, 8 kB of RAM, and 92 kB of flash memory. It includes the popular CC2420 transceiver, which is compliant with the IEEE 802.15.4 standard and operates at 2.4 GHz with a data rate of 250 kbps [66]. The sending and receiving motes were responsible for calculating each hash function. We evaluated the algorithms based on five metrics: throughput (bits/s),

TABLE 9. Performance of ALIT-Hash and TJUILIK-Hash in the Contiki-NG and Cooja simulators compared to other hash functions.

Algorithm	Throughput (bits/s)	Energy consumption (mJ)	Power consumption (mW)	ROM (bytes)	RAM (bytes)
ESCH256 (Sparkle)	11,037.64	1.34	57.97	2,779	0
LESAMNTA-LW	7,921.25	1.87	57.90	2,446	0
ASCON-Hash	6,523.02	2.27	57.94	2,704	0
XOODYAK-Hash	3,658.35	4.05	57.92	2,077	0
ALIT-HASH	3,536.51	4.19	57.92	12,012	64
TJUILIK-Hash	957.17	15.49	57.90	3,110	64
PHOTON-Beetle-Hash	345.98	42.84	57.90	1.379	26
PHOTON	311.39	47.60	57.90	1,798	266
SPONGENT	50.25	294.97	57.90	1,125	544

energy consumption (mJ), power consumption (mW), ROM (bytes), and RAM (bytes). Table 9 presents the complete results of the simulations.

The algorithms enumerated in Table 9 possess unique characteristics. The ESCH256 (Sparkle) algorithm is recognized for its exceptional data processing speed, with its maximum throughput of 11,037.64 bits/s rendering it the most efficient algorithm. The LESAMNTA-LW hash function achieves a noteworthy throughput of 7,921.25 bits/s while simultaneously maintaining low levels of energy and power consumption of 1.87 mJ and 57.90 mW, respectively. The ASCON-Hash and XOODYAK-Hash algorithms produce intermediate throughput rates of 6,523.02 bits/s and 3,658.35 bits/s, respectively. These values are accompanied by energy and power consumption levels that are relatively balanced. PHOTON-Beetle-Hash, PHOTON, and SPONGENT demonstrate lower throughputs, implying slower data processing rates. Nevertheless, they offset this limitation by exhibiting diverse energy and power consumption rates, with PHOTON-Beetle-Hash consuming 42.84 mJ, PHOTON consuming 47.60 mJ, and SPONGENT demanding 294.97 mJ. The algorithms under consideration also exhibit variations in their respective ROM and RAM requirements. PHOTON-Beetle-Hash occupies 1,379 bytes of ROM and 26 bytes of RAM, PHOTON occupies 1,798 bytes of ROM and 266 bytes of RAM, and SPONGENT occupies 1,125 bytes of ROM and 544 bytes of RAM.

ALIT-Hash has a throughput of 3,536.51 bits/s, demonstrating its ability to process data efficiently. Although it utilizes more energy than other algorithms, its power consumption remains constant at 57.92 mW. ALIT-Hash's relatively modest RAM usage of 64 bytes demonstrates its efficient utilization of memory resources. In addition, its 12,012-byte ROM requirement achieves a balance between storage demand and performance.

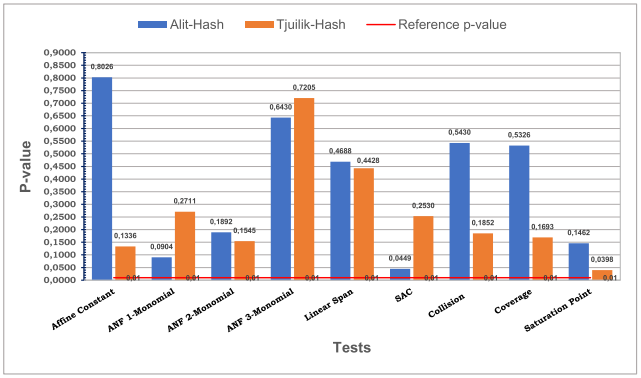

FIGURE 8. Results of cryptographic randomness tests.

Despite its lower throughput of 957.17 bits/s, TJUILIK-Hash also possesses distinctive advantages. Like ALIT-Hash, TJUILIK-Hash uses only 64 bytes of RAM, demonstrating its memory efficiency. The memory efficiency of TJUILIK-Hash makes it suitable for resource-constrained environments.

D. CRYPTOGRAPHIC RANDOMNESS TESTS

A cryptographic randomness test is a statistical test that examines a function's cryptographic characteristics [67]. We conducted seven tests for cryptographic randomness testing: affine constant, algebraic normal form d-monomial, linear span, strict avalanche criterion, collision, coverage, and saturation point tests. Filiol [68] in 2002 introduced a novel statistical testing methodology for assessing the quality of symmetric ciphers and hash functions. An assessment of the statistical aspects of a cryptographic algorithm's output, such as the value distribution and the presence of correlations, forms the basis of this methodology. First, the algebraic normal forms (ANFs) of random Boolean functions are

completely characterized employing the Möbius transform. Then, output bits from the cryptosystem under test are described using sets of Boolean functions and compared with those from purely random Boolean functions. These tests are called the affine constant and ANF d-monomial tests. Doğanaksoy et al. [67] proposed the next four tests to address corresponding cryptographic properties, as follows:

- For optimal diffusion, each output bit should change with a probability of 50 percent whenever an input bit changes. This requirement is known as the strict avalanche criterion (SAC). The SAC test aims to determine whether an algorithm satisfies the SAC property.
- A Boolean function should be far from the set of all affine functions. This attribute is measured in terms of nonlinearity and relates to confusion. The3 linear span test examines an algorithm by assessing the linear dependence of outputs generated from a set of firmly linearly dependent inputs.
- Finding two different inputs that produce the same output should be challenging; this property is known as collision resistance. The collision test consists of determining the number of collisions in a random subset of outputs corresponding to the input set.
- Block ciphers with a fixed plaintext and hash function are one-way functions that must act as random mappings. The coverage test checks the size of the output set corresponding to a subset of the input set.

The last test is the saturation point test, which was proposed by Sulak [69]. The saturation point test measures the randomness of the output of a hash function algorithm based on the saturation point, which is the index value of the integer at which all possible numbers have appeared in the sequence. This index is found by evaluating the appearance of all possible integer numbers in the sequence.

ALIT-Hash and TJUILIK-Hash were subjected to several different security evaluations, and Fig. 8 presents a comparison of the outcomes of these evaluations. A p value of 0.01 is considered to indicate a meaningful significance level when analyzing the results. The tests performed include the affine constant, ANF 1-monomial, ANF 2-monomial, ANF 3-monomial, linear span, SAC, collision, coverage, and saturation point tests. According to the findings, in most tests, TJUILIK-Hash has a lower p value than ALIT-Hash. This suggests that TJUILIK-Hash may have a higher level of security than ALIT-Hash does. However, it is important to interpret these findings within the context of the particular tests that were performed and the circumstances under which they were performed.

E. NIST RANDOMNESS TESTS

A hash function is expected to output random sequences, making it difficult to identify the algorithm based on an inspection of its output. The algorithm output must appear identical to that of random permutations and mappings.

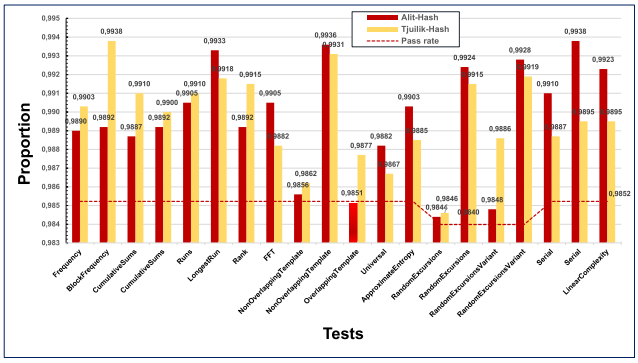

FIGURE 9. The proportion of sequences passing the NIST STS evaluation.

Therefore, it is crucial to evaluate the unpredictability of a hash function's output through statistical randomness tests. The NIST STS, as described in SP 800 22 rev 1a, is utilized for this purpose in this work.

The NIST STS consists of 15 statistical tests. This open-source randomness test application seeks to uncover nonrandom features in output sequences. The tests are divided into two categories: parameterized and nonparameterized. The eight nonparameterized tests are designed to reveal patterns, biases, or anomalies in random data that may indicate that they are not truly random. These tests are the binary matrix rank, frequency, runs, spectral DFT, longest runs of ones, cumulative sums, random excursion, and random excursion variant tests. The remaining seven tests are classified as parameterized tests. These tests are the linear complexity, approximate entropy, block frequency, overlapping templates, Maurer's universal, serial, and nonoverlapping tests. Each of these tests necessitates the introduction of parameter values.

Two methods are used to interpret the test results: assessing the percentage of bit sequences that pass each test and examining the resultant p value distribution for uniformity using the chi-square (χ^2) test and the Kolmogorov–Smirnov (K-S) test. The probability of a random sequence passing a test is equal to $1 - \alpha$, the complement of the significance level. For a large number of random sequences, the proportion of sequences that pass a test will vary but will be approximately $(1 - \alpha)$.

In these experiments, the randomness of a cryptographic method was determined based on the proportion of test samples passing each test, as expressed in (3):

$$p_\alpha = (1 - \alpha) - 2,6\sqrt{\frac{\alpha\,(1 - \alpha)}{s}}, \tag{3}$$

where the significance level α is 0.01 and the sample size s is 3,900 sequences of hash values. Utilizing the χ^2 goodness-of-fit test and the K-S test, we determined whether the p values resulting from each test were uniformly distributed on the interval (0, 1). For the χ^2 goodness-of-fit test, the p value interval (0, 1) was divided into ten subintervals to determine whether the p value frequency in each subinterval was approximately equal to the predicted frequency for a

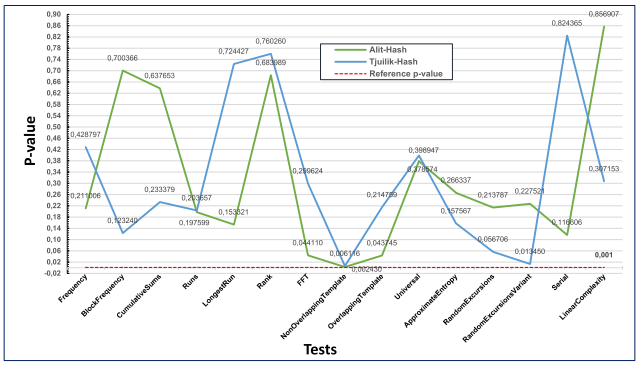

FIGURE 10. P value uniformity test of the NIST STS.

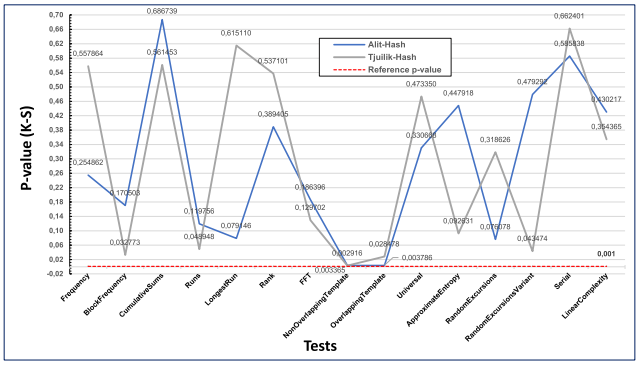

FIGURE 11. Kolmogorov–Smirnov (K-S) p value uniformity test of the NIST STS.

uniform distribution, $s/10$. The K-S test [70] is utilized to examine continuous distributions of ungrouped data based on the following test statistic:

$$D = \max_{1 \le i \le s} \left(F\left(Y_i\right) - \frac{i-1}{s}, \frac{i}{s} - F\left(Y_i\right) \right), \tag{4}$$

where F is the continuous theoretical cumulative distribution being tested and s is the sample size. The distributional form hypothesis is rejected if D exceeds the table-derived critical value.

In this study, we generated 3,900 hash value sequences of 1 million bits each for each hash function, equivalent to 487.758 MB of data. We provide the results of proportional tests conducted for multiple subtests, such as the nonoverlapping template matching, serial, cumulative sums, random excursion, and random excursion variant tests, using minimum and maximum values. However, for the p value uniformity test, we consider only the minimum value. Figures 11, 10, and 9 present the results of the Fast NIST STS evaluation. The overlapping template test showed ALIT-Hash to be a failure, with a proportion value of 0.9851, below the reference proportion value of 0.98522. In comparison, TJUILIK-Hash passed all fifteen tests conducted. Therefore, it follows that the output of TJUILIK-Hash cannot be distinguished from that of random permutations or mappings. In contrast, ALIT-Hash generates nonrandom output when the number of occurrences of predefined target strings in a given pattern does not match the expected number.

IX. CONCLUSION

The current study proposes two new lightweight hash functions based on the SATURNIN block cipher algorithm and the Beetle mode of operation. In addition, we propose a modification of the super S-box of the SATURNIN block cipher as the basis of the second proposed hash function. Security analysis results show that the two hash functions both have a reasonably good level of security with respect to differential and linear cryptanalysis. Additionally, the differential security level of TJUILIK-Hash is better than that of ALIT-Hash due to the S-box changes. The hardware utilization on a microcontroller board that is both cost-effective and low in power, and implementation on the ATmega2560 resulted in an average processing time of 0.746 microseconds for the TJUILIK-Hash algorithm. Furthermore, the results of performance tests on a personal computer demonstrate that the ALIT-Hash and TJUILIK-Hash implementations are comparable in speed and throughput to the implementations of seven other existing hash functions. Simulations conducted using Contiki-NG and the Cooja simulator indicate that the two proposed hash functions exhibit favorable performance relative to PHOTON-Beetle-Hash, PHOTON, and SPONGENT, as measured in terms of five metrics. ALIT-Hash exhibits superior performance in throughput (3,536.51 bits/s) and energy efficiency compared to TJUILIK-Hash, whereas TJUILIK-Hash demonstrates superior power and ROM consumption. The results of cryptographic randomness tests show that the two proposed hash functions pass seven tests with a p value of at least 0.01. These results indicate that both hash functions have the cryptographic properties expected of a good hash function. When applied as a PRNG, TJUILIK-Hash passed all tests in the NIST STS, whereas ALIT-Hash failed one of the tests, namely, the overlapping template test. This finding indicates that our modification increases the randomness of TJUILIK-Hash. Therefore, the two proposed hash functions should be considered for implementation because they are low-cost algorithms that provide a high level of security, which is crucial for IoT devices with limited resources. In the future, it will be necessary to conduct additional research on hardware compatibility based on other metrics.

REFERENCES

[1] Statista. *IoT Connected Devices Worldwide 2019–2030*. Accessed: Dec. 23, 2022. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] *GSMA | Safety, Privacy and Security Across the Mobile Ecosystem | Public Policy*. Accessed: Dec. 19, 2022. [Online]. Available: https://www.gsma.com/publicpolicy/resources/safety-privacy-and-security-across-the-mobile-ecosystem

[3] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[4] H. HaddadPajouh, A. Dehghantanha, R. M. Parizi, M. Aledhari, and H. Karimipour, "A survey on Internet of Things security: Requirements, challenges, and solutions," *Internet Things*, vol. 14, Jun. 2021, Art. no. 100129.

[5] L. Zhou, C. Su, and K.-H. Yeh, "A lightweight cryptographic protocol with certificateless signature for the Internet of Things," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 3, pp. 1–10, May 2019.

[6] S. Banerjee, V. Odelu, A. K. Das, S. Chattopadhyay, J. J. P. C. Rodrigues, and Y. Park, "Physically secure lightweight anonymous user authentication protocol for Internet of Things using physically unclonable functions," *IEEE Access*, vol. 7, pp. 85627–85644, 2019.

[7] S. Shin and T. Kwon, "A privacy-preserving authentication, authorization, and key agreement scheme for wireless sensor networks in 5G-integrated Internet of Things," *IEEE Access*, vol. 8, pp. 67555–67571, 2020.

[8] R. Kalaria, A. S. M. Kayes, W. Rahayu, and E. Pardede, "A secure mutual authentication approach to fog computing environment," *Comput. Secur.*, vol. 111, Dec. 2021, Art. no. 102483. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003072

[9] K.-L. Tsai, F.-Y. Leu, L.-L. Hung, and C.-Y. Ko, "Secure session key generation method for LoRaWAN servers," *IEEE Access*, vol. 8, pp. 54631–54640, 2020.

[10] N. Hayati, S. Windarta, M. Suryanegara, B. Pranggono, and K. Ramli, "A novel session key update scheme for LoRaWAN," *IEEE Access*, vol. 10, pp. 89696–89713, 2022.

[11] N. Hayati, K. Ramli, S. Windarta, and M. Suryanegara, "A novel secure root key updating scheme for LoRaWANs based on CTR_AES DRBG 128," *IEEE Access*, vol. 10, pp. 18807–18819, 2022.

[12] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[13] L. Wang, X. Shen, J. Li, J. Shao, and Y. Yang, "Cryptographic primitives in blockchains," *J. Netw. Comput. Appl.*, vol. 127, pp. 43–58, Feb. 2019.

[14] F. H. Pohrmen and G. Saha, "LightBC: A lightweight hash-based blockchain for the secured Internet of Things," in *Proc. Int. Conf. Innov. Comput. Commun.*, D. Gupta, A. Khanna, S. Bhattacharyya, A. E. Hassanien, S. Anand, and A. Jaiswal, Eds. Singapore: Springer, 2021, pp. 811–819.

[15] D. R. Stinson, "Some observations on the theory of cryptographic hash functions," *Des., Codes Cryptogr.*, vol. 38, no. 2, pp. 259–277, Feb. 2006. [Online]. Available: http://www.springerlink.com/index/F621241047Q60866.pdf

[16] A. Biryukov and L. Perrin, "State of the art in lightweight symmetric cryptography," *Cryptol. ePrint Arch.*, vol. 2017, pp. 1–55, Jan. 2017. [Online]. Available: https://eprint.iacr.org/2017/511.pdf

[17] S. Windarta, S. Suryadi, K. Ramli, B. Pranggono, and T. S. Gunawan, "Lightweight cryptographic hash functions: Design trends, comparative study, and future directions," *IEEE Access*, vol. 10, pp. 82272–82294, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9846993/

[18] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "NISTIR 8114 report on lightweight cryptography," Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep. 8114, 2017.

[19] *Announcing Request for Nominations for Lightweight Cryptographic Algorithms*, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2018, pp. 43656–43657.

[20] C. Dobraunig, F. Mendel, M. Eichlseder, and M. Schläffer. (2021). *Ascon v1.2 Submission to NIST*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf

[21] T. Beyne, Y. L. Chen, C. Dobraunig, and B. Mennink. (2021). *Elephant v2*. [Online]. Available: https://www.esat.kuleuven.be/cosic/elephant/

[22] C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, B. Mennink, R. Primas, and T. Unterluggauer. (2021). *ISAP v 2.0*. [Online]. Available: https://isap.iaik.tugraz.at/

[23] Z. Bao, A. Chakraborti, N. Datta, J. Guo, M. Nandi, T. Peyrin, and K. Yasuda. (2021). *PHOTON-Beetle Authenticated Encryption and Hash Family*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/photon-beetle-spec-final.pdf

[24] C. Beierle, A. Biryukov, L. C. dos Santos, J. Großschädl, A. Moradi, L. Perrin, A. R. Shahmirzadi, A. Udovenko, V. Velichkov, and Q. Wang. (2021). *Schwaemm and ESCH: Lightweight Authenticated Encryption and Hashing using the Sparkle Permutation Family*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/sparkle-spec-final.pdf

[25] J. Daemen, P. M. C. Massolino, A. Mehrdad, and Y. Rotella, "The subterranean 2.0 cipher suite," *IACR Trans. Symmetric Cryptol.*, vol. 2020, pp. 262–294, Jun. 2020.

[26] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo. (2021). *GIFT-COFB v1.1*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/gift-cofb-spec-final.pdf

[27] H. Wu and T. Huang. (2021). *TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2)*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf

[28] C. Guo, T. Iwata, M. Khairallah, K. Minematsu, and T. Peyrin. (2021). *Romulus v1.3*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/romulus-spec-final.pdf

[29] M. Hell, T. Johansson, A. Maximov, W. Meier, and H. Yoshida. (2021). *Grain-128AEADv2—A Lightweight AEAD Stream Cipher*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/grain-128aead-spec-final.pdf

[30] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. (2011). *The Keccak Reference*. Accessed: Jun. 15, 2020. [Online]. Available: https://keccak.team/files/Keccak-reference-3.0.pdf

[31] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications," in *Selected Areas in Cryptography*, A. Miri and S. Vaudenay, Eds. Berlin, Germany: Springer, 2012, pp. 320–337.

[32] G. Bertoni, J. Daemen, M. Peeters, G. van Assche, and V. R. Keer, "Permutation-based encryption, authentication and authenticated encryption," in *Directions Authenticated Ciphers (DIAC)*. Stockholm, Sweden: ECRYPT, 2012.

[33] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Sponge-based pseudo-random number generators," in *Proc. 12th Int. Workshop Cryptograph. Hardw. Embedded Syst.*, vol. 6225, 2010, pp. 1–15.

[34] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The contiking open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, p. 6, Jun. 2022. [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Home

[35] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, and Y. Seurin, "Hash functions and RFID tags: Mind the gap," in *Cryptographic Hardware and Embedded Systems—CHES 2008*, E. Oswald and P. Rohatgi, Eds. Berlin, Germany: Springer, 2008, pp. 283–299.

[36] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: A lightweight hash function," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, in Lecture Notes in Computer Science, vol. 6917. Cham, Switzerland: Springer, 2011, pp. 312–325, doi: 10.1007/978-3-642-23951-9_21.

[37] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: The design space of lightweight cryptographic hashing," *IEEE Trans. Comput.*, vol. 62, no. 10, pp. 2041–2053, Oct. 2013.

[38] J. Guo, T. Peyrin, and A. Poschmann, "The *PHOTON* family of lightweight hash functions," in *Advances in Cryptology—CRYPTO 2011* (Lecture Notes in Computer Science), vol. 6841. Cham, Switzerland: Springer, 2011.

[39] S. Hirose, K. Ideguchi, H. Kuwakado, T. Owada, B. Preneel, and H. Yoshida, "A lightweight 256-bit hash function for hardware and low-end devices: Lesamnta-LW BT," in *Information Security and Cryptology—ICISC 2010*, K.-H. Rhee and D. Nyang, Eds. Berlin, Germany: Springer, 2011, pp. 151–168.

[40] *Information Technology—Security Techniques—Lightweight Cryptography—Part 5: Hash-Functions*, Standard ISO/IEC 29192-5:2016, 2016. [Online]. Available: https://www.iso.org/standard/56425.html

[41] J. Daemen, S. Hoffert, S. Mella, M. Peeters, G. V. Assche, and R. V. Keer. (2021). *Xoodyak, A Lightweight Cryptographic Scheme*. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/xoodyak-spec-final.pdf

[42] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda, "Beetle family of lightweight and secure authenticated encryption ciphers," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, pp. 218–241, May 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/881

[43] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer, "The design of Xoodoo and Xoofff," *IACR Trans. Symmetric Cryptol.*, vol. 2018, no. 4, pp. 1–38, Dec. 2018. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/7359

[44] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodoo cookbook," *Cryptol. ePrint Arch.*, vol. 2018, pp. 1–32, Mar. 2018. [Online]. Available: https://eprint.iacr.org/2018/767.pdf

[45] *FIPS Pub 202 SHA-3 Standard: Permutation-Based Hash and Extendable Output Functions*, NIST, Gaithersburg, MD, USA, 2015.

[46] C. Beierle, A. Biryukov, L. C. dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, and Q. Wang, "Lightweight AEAD and hashing using the sparkle permutation family," *IACR Trans. Symmetric Cryptol.*, vol. 2020, pp. 208–261, Jun. 2020. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8627

[47] D. J. Bernstein. (2019). *Caesar: Competition for Authenticated Encryption: Security, Applicability, and Robustness*. [Online]. Available: https://competitions.cr.yp.to/caesar.html

[48] A. Canteaut, S. Duval, G. Leurent, M. Naya-Plasencia, L. Perrin, T. Pornin, and A. Schrottenloher, "SATURNIN: A suite of lightweight symmetric algorithms for post-quantum security," *IACR Trans. Symmetric Cryptol.*, vol. 2020, pp. 160–207, Jun. 2020. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8621

[49] M. J. Dworkin. (2015). *Sha-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[50] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda, "Beetle family of lightweight and secure authenticated encryption ciphers," *Cryptol. ePrint Arch.*, vol. 2018, pp. 1–25, Jan. 2018. [Online]. Available: https://eprint.iacr.org/2018/805

[51] Z. Bao, J. Guo, S. Li, and P. Pham, "Quantum multi-collision distinguishers," *Cryptol. ePrint Arch.*, vol. 2021, pp. 1–42, Jun. 2021. [Online]. Available: https://eprint.iacr.org/2021/703

[52] X. Dong, Z. Zhang, S. Sun, C. Wei, X. Wang, and L. Hu, "Automatic classical and quantum rebound attacks on AES-like hashing by exploiting related-key differentials," in *Advances in Cryptology—ASIACRYPT 2021*, M. Tibouchi and H. Wang, Eds. Cham, Switzerland: Springer, 2021, pp. 241–271.

[53] X. Dong, Z. Zhang, S. Sun, C. Wei, X. Wang, and L. Hu, "Automatic classical and quantum rebound attacks on AES-like hashing by exploiting related-key differentials," *Cryptol. ePrint Arch.*, vol. 2021, pp. 1–45, Sep. 2021. [Online]. Available: https://eprint.iacr.org/2021/1119

[54] X. Dong, J. Guo, S. Li, and P. Pham, "Triangulating rebound attack on AES-like hashing," *Cryptol. ePrint Arch.*, vol. 2022, pp. 1–54, Jun. 2022. [Online]. Available: https://eprint.iacr.org/2022/731

[55] G. Bertoni, J. Daemen, M. Peeters, and G. van Assche. (2007). *Sponge Functions*. [Online]. Available: https://keccak.team/files/SpongeFunctions.pdf

[56] G. Leander and A. Poschmann, "On the classification of 4 bit S-boxes," in *Proc. Int. Workshop Arithmetic Finite Fields*, in Lecture Notes in Computer Science, vol. 4547. Berlin, Germany: Springer, 2007, pp. 159–176.

[57] M. J. O. Saarinen, "Cryptographic analysis of all 4 × 4—Bit S-boxes," *Cryptol. ePrint Arch.*, vol. 2011, pp. 1–10, Jul. 2011. [Online]. Available: https://eprint.iacr.org/2011/218

[58] M. J. O. Saarinen, "Cryptographic analysis of all 4 × 4-bit S-boxes," in *Selected Areas in Cryptography* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7118. Berlin, Germany: Springer, 2012, pp. 118–133, doi: 10.1007/978-3-642-28496-0_7.

[59] W. Zhang, Z. Bao, V. Rijmen, and M. Liu, "A new classification of 4-bit optimal S-boxes and its application to PRESENT, RECTANGLE and SPONGENT," in *Fast Software Encryption* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9054. Berlin, Germany: Springer, 2015, pp. 494–515, doi: 10.1007/978-3-662-48116-5_24.

[60] L. Cheng, W. Zhang, and Z. Xiang, "A new cryptographic analysis of 4-bit S-boxes," in *Information Security and Cryptology* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9589. Cham, Switzerland: Springer, 2016, pp. 144–164, doi: 10.1007/978-3-319-38898-4_9.

[61] E. Biham, R. Anderson, and L. Knudsen, "Serpent: A new block cipher proposal," in *Fast Software Encryption*, S. Vaudenay, Ed. Heidelberg, Germany: Springer, 1998, pp. 222–238.

[62] M. Sýs, Z. Říha, and V. Matyáš, "Algorithm 970: Optimizing the NIST statistical test suite and the Berlekamp–Massey algorithm," *ACM Trans. Math. Softw.*, vol. 43, no. 3, pp. 1–12, Dec. 2016, doi: 10.1145/2988228.

[63] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, N. Heckert, J. Dray, S. Vo, and L. Bassham. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final

[64] R. Weatherley. (2021). *Lightweight Cryptography Primitives: Performance on AVR*. Accessed: May 15, 2023. [Online]. Available: https://rweather.github.io/lightweight-crypto/performance_avr.html

[65] L. Alliance. *RP2-1.0.3 LoRaWAN Regional Parameters*. Accessed: Jan. 7, 2023. [Online]. Available: https://resources.lora-alliance.org/technical-specifications/rp2-1-0-3-lorawan-regional-parameters

[66] Zolertia. (Apr. 2013). *The Z1 Mote · Zolertia/Resources Wiki · Github*. Accessed: May 21, 2023. [Online]. Available: https://github.com/Zolertia/Resources/wiki/The-Z1-mote

[67] A. Doganaksoy, B. Ege, O. Koçak, and F. Sulak, "Cryptographic 1102 randomness testing of block ciphers and hash functions," *Cryptol. ePrint Arch.*, vol. 2010, pp. 1–12, Nov. 2010. [Online]. Available: https://eprint.iacr.org/2010/564

[68] E. Filiol, "A new statistical testing for symmetric ciphers and hash functions," in *Information and Communications Security*, R. Deng, F. Bao, J. Zhou, and S. Qing, Eds. Berlin, Germany: Springer, 2002, pp. 342–353.

[69] F. Sulak, "A new statistical randomness test: Saturation point test," *Int. J. Inf. Secur. Sci.*, vol. 2, pp. 81–85, Sep. 2013. [Online]. Available: https://ijiss.org/ijiss/index.php/ijiss/article/view/52

[70] National Institute of Standards and Technology (NIST). (2003). *NIST/SEMATECH Engineering Statistics Handbook*. Accessed: Feb. 15, 2023. [Online]. Available: http://www.itl.nist.gov/div898/handbook/

SUSILA WINDARTA (Member, IEEE) received the degree in cryptography from the National Crypto Academy, Bogor, Indonesia, the bachelor's degree in information systems from Gunadarma University, Indonesia, and the master's degree in mathematics from the Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok, Indonesia, where he is currently pursuing the Ph.D. degree in the Department of Electrical Engineering, Faculty of Engineering. Since 2013, he has been a Lecturer with the Department of Cyber-Security Engineering, National Cyber and Crypto Polytechnic, Indonesia. His current research interests include cryptography and information security-related topics, mainly cryptographic hash functions and security protocols.

SURYADI SURYADI received the B.Sc. degree in mathematics from the Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Indonesia, in 1990, the M.Sc. degree in informatics engineering from the Bandung Institute of Technology, Indonesia, in 1998, and the D.Phil. degree from the Department of Electrical and Computer Engineering, Universitas Indonesia, in 2013. He is currently an Associate Professor (a Lecturer) with the Department of Mathematics and the Department of Electrical Engineering, Universitas Indonesia. He is the author or coauthor of more than 40 papers published in prominent international journals and conference proceedings, has written two books, and has contributed a chapter to a book. His current research interests include information security, cryptography, and computational mathematics. He is a member of IndoMS.

KALAMULLAH RAMLI (Member, IEEE) received the master's degree in telecommunication engineering from the University of Wollongong, Wollongong, NSW, Australia, in 1997, and the Ph.D. degree in computer networks from Universitaet Duisburg-Essen (UDE), NRW, Germany, in 2003. He has been a Lecturer with Universitas Indonesia (UI), since 1994, and a Professor of computer engineering, since 2009. He teaches advanced communication networks, embedded systems, object-oriented programming, and engineering and entrepreneurship. He is the prolific author, with more than 125 journals/conference papers and eight books/book chapters published. His current research interests include embedded systems, information and data security, computers and communication, and biomedical engineering.

ANDRIANI ADI LESTARI (Member, IEEE) received the degree in cryptography from the National Crypto Academy, Bogor, Indonesia, and the bachelor's degree in statistics from Universitas Terbuka. She is currently pursuing the master's degree in statistics with the Department of Statistics, Faculty of Mathematics and Natural Sciences, IPB University, Jakarta, Indonesia. Since 2019, she has been a Lecturer with the Department of Cryptography, Politeknik Siber dan Sandi Negara. Her current research interests include cryptography and information security, primarily block ciphers, cryptographic hash functions, and protocols.

WILDAN WILDAN received the bachelor's degree in cryptographic techniques from the National Crypto Institute, Bogor, Indonesia, and the master's degree in mathematics from the Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok. He has been a Researcher with the Cryptographic and Technology Cybersecurity Research and Development Center, Indonesian National Cyber and Crypto Agency, since 2017. His current research interests include cryptographic techniques, with a specific focus on block cipher cryptanalysis.

BERNARDI PRANGGONO (Senior Member, IEEE) received the B.Eng. degree in electronics and telecommunication engineering from Waseda University, Japan, the master's degree in digital communications from Monash University, Australia, and the Ph.D. degree in electronics and electrical engineering from the University of Leeds, U.K. He is currently an Associate Professor of cyber security and computer networks with the School of Computing and Information Science, Anglia Ruskin University, Cambridge, U.K. He has previously held academic and research positions with Sheffield Hallam University, Glasgow Caledonian University, Queen's University Belfast, and the University of Leeds. He has held industrial positions at Oracle, PricewaterhouseCoopers, Accenture, and Telstra. His current research interests include cybersecurity, the Internet of Things, cloud computing, and green ICT. He is a fellow of the Higher Education Academy (HEA). He is an Associate Editor of *Frontiers of Computer Science* and *Frontiers in Communications and Networks*.

RINI WISNU WARDHANI (Graduate Student Member, IEEE) received the M.Eng. degree in electrical engineering from the University of Indonesia, in 2011. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Pusan National University, South Korea. She was with the National Cyber and Crypto Agency of Indonesia, from 2003 to 2021. Her current research interests include hardware security, information security, cryptography, and quantum computing.