IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.DOI

Lightweight Cryptographic Hash Functions: Design Trends, Comparative Study, and Future Directions

SUSILA WINDARTA[1] (Member, IEEE), SURYADI[2], KALAMULLAH RAMLI.[3] (Member, IEEE), BERNARDI PRANGGONO[4] (Senior Member, IEEE), TEDDY SURYA GUNAWAN[5] (Senior Member, IEEE).

[1,3]Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Depok, Jawa Barat, Indonesia
[2]Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok, Jawa Barat, Indonesia
[4]Department of Engineering and Mathematics, Sheffield Hallam University, Sheffield, United Kingdom
[5]Department of Electrical and Computer Engineering, Kulliyyah of Engineering, International Islamic University Malaysia, Kuala Lumpur, Malaysia

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

This work was supported by the Universitas Indonesia through the Hibah Publikasi Terindeks Internasional (PUTI) Q2 Scheme under Contract NKB-..../UN2.RST/HKP.06.00/2022. The work of Susila Windarta was supported in part by the Indonesia Endowment Fund for Education or Lembaga Pengelola Dana Pendidikan (LPDP).

ABSTRACT The emergence of the Internet of Things (IoT) has enabled billions of devices that collect large amounts of data to be connected. Therefore, IoT security has fundamental requirements. One critical aspect of IoT security is data integrity. Cryptographic hash functions are cryptographic primitives that provide data integrity services. However, due to the limitations of IoT devices, existing cryptographic hash functions are not suitable for all IoT environments. As a result, researchers have proposed various lightweight cryptographic hash function algorithms. In this paper, we discuss advanced lightweight cryptographic hash functions for highly constrained devices, categorize design trends, analyze cryptographic aspects and cryptanalytic attacks, and present a comparative analysis of different hardware and software implementations. In the final section of this paper, we highlight present research challenges and suggest future research topics related to the design of lightweight cryptographic hash functions.

INDEX TERMS Internet of Things, lightweight cryptographic hash function, lightweight cryptography, security

I. INTRODUCTION

THE Internet of Things (IoT) is an essential component of computer science and information technology research. An enormous amount of research on the IoT has been conducted due to the IoT applications in various fields, including automotive systems, sensor networks, healthcare, distributed control systems, cyber-physical systems, smart grids, agriculture, smart cities, smart homes, transport and logistics, and smart factories. Moreover, IoT Analytics [1] has predicted the connectivity between IoT devices to reach 30.90 billion by 2025. The increase in the number of IoT devices has led to more connections than the use of non-IoT devices. These connected devices pose the same dilemma as connectivity between people: convenience and security. Among these connected devices are devices with the same or similar resources as standard computers; however, many devices have limitations. Devices with similar resources to standard computers can use standard cryptography primitives; however, other devices require unique designs due to various limitations. Researchers in [2]–[4] defined four design limitations associated with IoT cryptography primitives, especially in hardware implementations: memory consumption, implementation size, speed or throughput, and power or energy (Fig. 3).

One of the most widely used IoT cryptographic primitives is the cryptographic hash function [5]–[8]. The cryptographic hash function is a cryptographic primitive that plays an essential role in various cyber and information security applications. The cryptographic hash function maps an arbitrary length input to a fixed-length output. The hash function outputs the hash value, message digest, digest, or fingerprint. Cryptographic hash functions have been implemented in different cryptographic mechanisms, including data integrity [7]–[10], entity authentication [7], [8], digital signatures [5], [6], [11], [12], pseudorandom number generators [7], cryptographic key derivation [7], [12], key generation [12], password security, and blockchains [13]–[17]. The use of the hash function is crucial in digital signature applications. The hash value of the message is signed using the sender's private key. The security of a digital signature is highly dependent on the security of the cryptographic hash function. If an attacker finds two messages with the same hash value and convinces the other party to sign one of the messages, the attacker can obtain a valid digital signature for the other message. Similar to password security applications, if an attacker constructs a password based on the hash value, the security of the system protected by the password may be at risk.

Therefore, government, industry, and academia have attempted to design and analyze cryptographic hash functions. When designing a hash function, the designer must consider both security and performance factors. Some previous works have described the characteristics of a good cryptographic hash function by considering these two factors [18]–[21].

In 2012, the Keccak hash function [22] was selected as the secure hash standard (SHA-3) and was published in the Federal Information Processing Standard (FIPS) 202 [23] and NIST SP 800-185 [24]. However, the hash functions designed in the SHA-3 competition are intended for devices with standard specifications. The primitives are not designed for small computing devices with limited resources, such as embedded devices, RFID devices, and sensor networks. Lightweight cryptographic algorithms for devices with limited resources have been widely discussed in the literature. Some lightweight cryptographic algorithms include the lightweight block cipher, lightweight stream cipher, lightweight public key cryptosystem, lightweight cryptographic hash function (LWCHF), and lightweight message authentication code (MAC). This article focuses on lightweight cryptographic hash functions because of the vital role these algorithms play in devices with limited resources.

There has been considerable research on the design of the LWCHF algorithm since it was first developed in 2008. In addition, many attacks on the LWCHF algorithm have been carried out. We aim to present a state-of-the-art LWCHF algorithm, including the design trends, cryptographic properties, and hardware and software implementation performance. Here, design trends refer to constructs that have been proposed in the literature, cryptographic properties refer to cryptanalytic attacks that have been carried out on each LWCHF algorithm, and the implementation performance summarizes data related to implementing hash function algorithms on hardware and software, along with the accompanying metrics. We obtain the implementation performance data from algorithm designers or implementations by other researchers.

The main contributions of this study can be summarized as follows:

- We surveyed state-of-the-art lightweight cryptographic hash functions up to early 2022. To the best of our knowledge, there have been no surveys on lightweight cryptographic hash functions developed until the final round of the NIST Lightweight Cryptography Project.
- We classify the design trends for lightweight cryptographic hash functions.
- We analyze and compare lightweight hash functions based on cryptographic properties (Table 4) and implementation aspects (Table 5).
- We analyze the challenges associated with designing and developing a lightweight cryptographic hash function.
- We identify potential gaps in future research, highlighting essential and practical considerations for developing lightweight cryptographic hash functions that require more attention.

This review should support academic and industry researchers in designing, analyzing, and implementing lightweight cryptographic hash functions. We hope our study's results can inspire researchers in future work aimed at designing and implementing LWCHFs.

The remainder of this paper is organized as follows. In Section II, we describe the methodology we used in this review. Section III highlights several surveys related to lightweight cryptographic hash functions. Section IV discusses the theoretical basis of cryptographic hash functions and their relation to lightweight cryptographic hash functions (LWCHFs). The lightweight cryptography performance metrics associated with hardware and software implementations are detailed in Section V. Section VI discusses the design trends of lightweight cryptographic hash functions. A comprehensive study of a state-of-the-art LWCHF is presented in Section VII. The results, discussion, research challenges and future directions are presented in Section VIII. Finally, we conclude our research in Section IX.

II. SURVEY METHODOLOGY

The approach we used to collect manuscripts for this survey is shown in Fig. 1. The scientific databases we searched for articles include IEEE Xplore, ACM Digital Library, Springer, ScienceDirect, and Google Scholar. The search focused on papers published between 2008 and the present day (2022). The search terms used to collect the manuscripts included several variations of "lightweight cryptographic hash function". Based on the search terms, we initially identified more than 500 papers. These papers were then filtered to fit the topic coverage based on their title, abstract, content, and conclusion.

This survey followed a semisystematic methodology [25] to narrow the literature into several stages. In Stage 1, an extensive search was used to analyze the literature on all proposed lightweight cryptographic hash functions to the best of our knowledge. In Stage 2 of our study, we conducted an in-depth examination to select literature based on the LWCHF design. The Stage 2 results were formalized as design trends and hardware and software performance com-

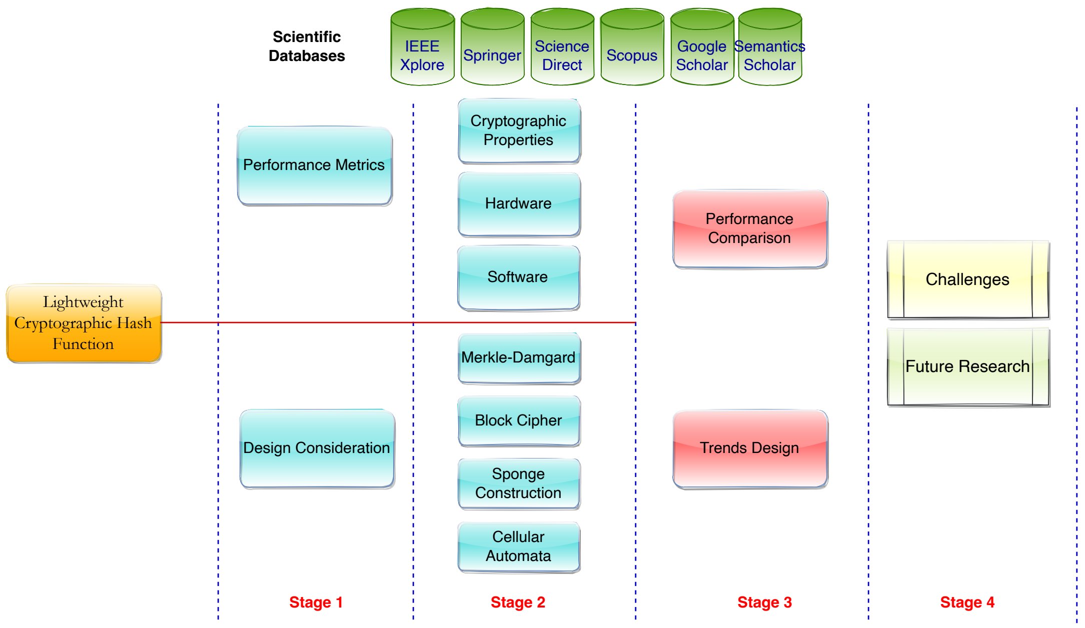

FIGURE 1. Survey Methodology.

parisons in Stage 3. In Stage 4, we concluded our evaluation of the literature and discussed some challenges of this study and potential future work.

III. RELATED WORKS

Many surveys on research progress in IoT security have been published in recent years [18], [26]–[28]. Researchers have mainly focused on IoT security solutions. Security issues are presented as components of each survey and are treated as general concepts, and security and privacy are often considered together as one concept. Unfortunately, no previous survey has detailed deep-seated IoT security issues related to lightweight cryptographic hash functions (see Table 1).

Biryukov and Perrin [18] investigated lightweight cryptographic algorithms that had been developed prior to 2017, including block ciphers, stream ciphers, and hash functions, which were designed for use in academia, government, and industry. The authors discussed in detail the design of each algorithm. However, the authors do not provide a detailed explanation of the most recent lightweight cryptographic hash function.

Ankit et al. [26] did not discuss the hash function algorithm, although this algorithm was mentioned in the introduction of their work. In addition, the author does not discuss state-of-the-art algorithms. Dhanda et al. [27] discussed 54 lightweight cryptography (LWC) algorithms, including 21 lightweight block ciphers, 19 lightweight stream ciphers, nine lightweight hash functions, and five elliptic curve cryptography (ECC) ciphers that had been developed prior to 2019. When discussing the Keccak algorithm, the author mistakenly identified the algorithm's designers as [30], while the algorithm was actually designed by [31]. The author did not specify the state-of-the-art hash function algorithm identified in the previous survey.

Thakor et al. [29] classified the critical characteristics of LWC algorithms and compared 41 LWC encryption algorithms using seven performance metrics. The seven metrics are the block/key size, memory, gate area, latency, throughput, power & energy, and hardware & software efficiency. In a recent study, Rana et al. [28] discussed state-of-the-art lightweight cryptographic protocols for IoT networks and provided a comparative analysis of popular ciphers. The authors discussed three lightweight cryptography primitives: the block cipher, stream cipher, and elliptic curve cipher.

IV. OVERVIEW OF CRYPTOGRAPHIC HASH FUNCTIONS

Cryptographic hash functions are workhorses in cryptography, and these primitives are used in almost all cryptographic applications [32]. A cryptographic hash function is defined as follows (Definition 1):

Definition 1: [19] Suppose x is the message input, and n is a positive integer. The hash function $\mathcal{H}$ is a function with at least the following properties:

1) Compression: $\mathcal{H}$ maps any input x of finite length to an output $\mathcal{H}(x)$ with length n as $\mathcal{H} : (0,1)^* \mapsto (0,1)^n$..
2) Easy computation: when the hash function $\mathcal{H}$ and input x are known, the hash value $\mathcal{H}(x)$ is easy to calculate.

Cryptographic hash functions can generally be classified into two categories [19]:

1) Modification detection codes (MDCs)
   This category is also known as message integrity codes (MICs). MDCs calculate the hash value of an input message and determine its integrity by comparing the hash values of the received messages. The MDC is an

TABLE 1. Surveys on lightweight cryptographic hash functions.

Survey	Ref.	Cryptographic primitives
State-of-the-Art in Lightweight Symmetric Cryptography	[18]	Stream ciphers, block ciphers, hash functions, and cryptographic protocol
A Survey of Lightweight Cryptographic Algorithms for IoT-Based Applications	[26]	Symmetric, asymmetric and hash functions
Lightweight Cryptography: A Solution to Secure IoT	[27]	Block ciphers, stream ciphers, hash functions, and elliptic curve cryptography
Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities	[29]	Block ciphers
Lightweight Cryptography in IoT Networks: A Survey	[28]	Block ciphers, stream ciphers, and elliptic curve ciphers
Our work	This paper	Lightweight cryptographic hash functions (LWCHFs)

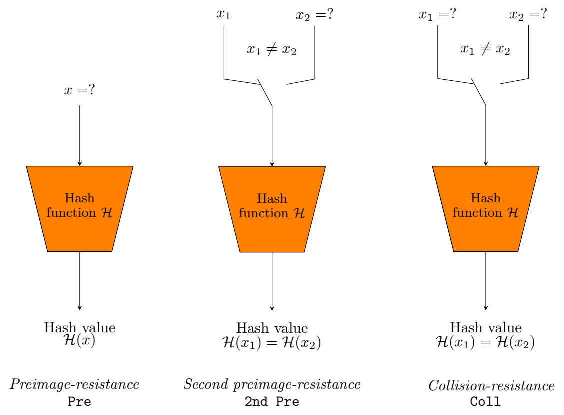
x_1
$x_2 =?$
$x_1 \neq x_2$
$x_1 =?$
$x_2 =?$
$x_1 \neq x_2$
$x =?$
Hash function $\mathcal{H}$
Hash function $\mathcal{H}$
Hash function $\mathcal{H}$
Hash value $\mathcal{H}(x)$
Hash value $\mathcal{H}(x_1) = \mathcal{H}(x_2)$
Hash value $\mathcal{H}(x_1) = \mathcal{H}(x_2)$
Preimage-resistance Pre
Second preimage-resistance 2nd Pre
Collision-resistance Coll

FIGURE 2. Three security properties of a cryptographic hash function.

unkeyed hash function with the properties specified in Definition 2. There are two subclasses of MDCs:

- One-way hash functions (OWHFs): it is computationally difficult to identify the message input according to the given hash value.
- Collision-resistant hash functions (CRHFs): it is difficult to identify any two inputs with the same hash value.

In this study, we focus on unkeyed hash functions.

2) Message authentication codes (MACs)
This category is also known as keyed hash functions. The MAC is a hash function with an additional parameter: a cryptographic key. The MAC algorithm aims to assure the integrity of the source and message without using other mechanisms. The secret key parameter allows this assurance.

Definition 2: [19] An unkeyed hash function $\mathcal{H}$ with message inputs x, x' and hash values y, y' also has the following properties:

1) Preimage resistance (one-way): given the hash value y, it is computationally difficult to determine the input x such that $\mathcal{H}(x) = y$.
2) Second-preimage resistance: given the input x, it is computationally difficult to determine another input $x \neq x$; thus, $\mathcal{H}(x') = \mathcal{H}(x)$. This property is also known as weak collision resistance.
3) Collision resistance: it is computationally difficult to find any two inputs $x' \neq x$ such that $\mathcal{H}(x') = \mathcal{H}(x)$. Another name for this property is strong collision resistance.

We denote the preimage, second preimage, and collision resistance as `Pre`, `2nd Pre` and `Coll`. Illustrations of these three properties are shown in Fig. 2.

V. LIGHTWEIGHT CRYPTOGRAPHY PERFORMANCE METRICS

Researchers in several studies have defined performance metrics for software and hardware implementations. The designer must specify which metrics are suitable for a particular application. The choice of metric is crucial because it determines the design of the lightweight cryptographic algorithm. Fig. 3 depicts the IoT device implementation metrics used in the comparison in Subsection VIII-B.

SOFTWARE IMPLEMENTATION

The software implementation metrics are defined as follows:

1) Read-only memory (ROM) or code size [33], [34]: this metric relates to the fixed amount of data required to evaluate a function independently of its input. According to [34], this metric is the size of the cryptographic primitive/algorithm/mechanism code in bytes.
2) Random access memory (RAM) consumption [33], [34]: this metric corresponds to the amount of data written to memory during each function evaluation.
3) Energy [27], [35]–[38]: this metric corresponds to the power consumption during a certain period [34] and is measured in microjoules μJ. Lower values are better for this metric. The mathematical equation for energy consumption is formulated as follows:
$$E_{\text{per bit}} = \frac{Lat \times P}{B},$$
where $E_{\text{per bit}}$ is the energy per bit, Lat is the latency, P is the power used by the hardware or software, and B is the block size.
4) Throughput: this metric measures the average amount of data processed during each clock cycle.
5) Latency: this metric corresponds to the number of clock cycles needed to calculate a plaintext/ciphertext block.

HARDWARE IMPLEMENTATION

The following metrics are used to evaluate the hardware implementation efficiency:

1) Gate equivalent (GE) [2], [4], [18], [27], [34], [35]: this metric measures the memory consumption and implementation size. The GE is defined as the area occupied by the semiconductor [34]. Lower values are better for this metric. This metric measures how much physical area is required for a circuit that implements a primitive. Gong [4] noted that the physical area allocation in an LWC implementation should be less than 2000 GE. The metric can be defined with the following equation:
$$P_{\text{area}} = \frac{L}{A_n},$$
where P_{area} is the physical area allocation, L is the application layout area and A_n is the area of the NAND2 gate.
2) Latency: this metric corresponds to the time a circuit outputs after the input is given [2], [18], [27], [34], [39]. The latency is measured in cycles/block or cycles/byte. Lower values are better for this metric. The latency can be defined as :
$$Lat = k \times t_{cycle},$$
where Lat is the latency, k is the number of clock cycles used to compute the output and t_{cycle} is the time of one cycle.
3) Throughput [2], [27], [36], [40]: this metric is measured in bits or bytes per second and corresponds to the number of plaintexts processed per unit of time. Higher values are better for this metric. The throughput can be defined as:
$$T = \frac{B \times F}{N},$$
where T is the throughput, B is the block size, F is the frequency and N is the number of cycles per block.
4) Energy consumption: this metric is the same as the corresponding software metrics.
5) Power consumption [18], [27], [28], [35]: this metric is measured in Watts (W) or μW and quantifies the amount of power required to use the circuit. Lower values are preferred for this metric. The power can be calculated as:
$$P = \frac{B \times E_{\text{per bit}}}{Lat},$$
where P is the power, B is the block size, Lat is the latency, P is the power used by the hardware or software, and $E_{\text{per bit}}$ is the energy per bit.

VI. TRENDS IN LIGHTWEIGHT CRYPTOGRAPHIC HASH FUNCTION DESIGN

This section discusses LWCHF design trends for three popular constructions: Merkle-Damgård construction, sponge construction, and block cipher-based construction. Some algorithms [41]–[43] use a particular construction, such as Merkle-Damgård or sponge, as the main construction and other constructions (e.g., block cipher-based) as building blocks to develop compression functions or permutations. In addition, we identify the round functions used in the LWCHF scheme: the substitution permutation network (SPN), Feistel network, and addition-rotation-exclusive Or (XOR) (ARX) structure. Table 2 lists the LWCHF design trends.

A. MERKLE-DAMGÅRD CONSTRUCTION

As mentioned in the introduction, research on the cryptographic hash function began with two crucial papers that underlie the development of this theory: Ralph Merkle's paper [83] and Ivan Bjerre Damgård's paper [84]. Merkle and Damgåproposed a cryptographic hash function that utilized a compression function, which is assumed to be a collision resistance function. This type of compression function can be extended to a hash function that is also collision resistant. Fig. 4 shows the Merkle-Damgård (MD) construction.

The basic idea underlying MD construction can be described as follows.

Suppose f is a compression function that is collision resistant. The function f maps $\{0,1\}^n \times \{0,1\}^k \mapsto \{0,1\}^n$, a fixed and public value initialization vector (IV) $\{0,1\}^n$ and the message $m = (m_0, m_1, \ldots, m_{L-1})$, where m_i is k bits. The hash function $\mathcal{H}$ can be constructed as:

$h_0 = IV$,

$h_{i+1} = f(h_i, m_i)$,

$\mathcal{H}(m) = h_L$.

In this case, h_i is the intermediate hash value and $\mathcal{H}(m)$ is the hash value.

MD construction is vulnerable to length extension attacks [85], [86]. To prevent these attacks, the message input length is added at the end of the message input with the required padding so that the last block is a multiple of k. This construction is known as Strengthened Merkle-Damgård.

B. LWCHFS BASED ON BLOCK CIPHERS

The use of block ciphers as building blocks in hash function design [87] is almost as old as the Data Encryption Standards (DES) algorithm [88]. Suppose that E is a block cipher with an r bit key k that maps n bit plaintext to n bit ciphertext. To the best of our knowledge, most researchers have used the Davies-Meyer (DM) construction [89] to design LWCHFs based on block ciphers. Fig. 5 depicts three well-known hash function constructions based on block ciphers: Davies-Meyer, Matyas-Meyer-Oseas, and Miyaguchi-Preenel [19], [45].

The steps of the Davies-Meyer algorithm are as follows:

Input: bit string x.

Output: n-bit hash-code.

1) Input x is divided into k-bit blocks, where k is the key length and padded, if necessary, to complete the last block. Denote the padded message with t k-bit blocks as $x_1 x_2 \ldots x_t$. n-bits IV must be predefined.

TABLE 2. Lightweight cryptographic hash function design trends.

ROUND FUNCTION	CONSTRUCTION					
	Merkle-Damgård	Block Cipher-Based	Sponge			Cellular Automata
			P-Sponge	T-Sponge	JH mode	
ROUND FUNCTION	CONSTRUCTION					
	Merkle-Damgård	Block Cipher-Based	Sponge			Cellular Automata
			P-Sponge	T-Sponge	JH mode	
Substitution Permutation Network (SPN)	ARMADILLO [44]	DM-PRESENT [45], [46]	QUARK [47]	GLUON [48]	SipHash [49]	
	Lesamnta-LW [41]	H-PRESENT-128 [45]	PHOTON [50]	SipHash [49]	SPN-Hash [51]	
	Al-Odat et al. LWCHF [52]	C-PRESENT-128 [45]	SPONGENT [53], [54]		Gimli-Hash [55], [56]	
		Lesamnta-LW [41]	SPN-Hash [51]			
		TWISH [57]	Gimli-Hash [55], [56]			
			sLiSCP-hash [43], [58] p			
			sLiSCP-hash-light [42], [59] p			
			ACE-$\mathcal{H}$-256 [60] p			
			ASCON-HASH [61]			
			KNOT-Hash [62], [63]			
			DryGascon-Hash [64]			
			ORANGISH [65]			
			PHOTON-Beetle-Hash [66]			
			ESCH [67], [68]			
			Subterranean2.0-XOF [69], [70]			
			Xoodyak Hash Mode [71]			
			HVH [72]			
Feistel Network	El Hanouti et al. LWCHF [73]	Lesamnta-LW [41]	LHash [74], [75]			
			sLiSCP-Hash [43], [58]			
			sLiSCP-Light-Hash [42], [59]			
Addition, Rotation & Exclusive Or (XOR)			Neeva-Hash [76] Bussi et al. (2016)			
			sLiSCP-Hash [43], [58]			
			sLiSCP-Light-Hash [42], [59]			
			ACE-$\mathcal{H}$-256 [60]			
Others	El Hanouti et al. LWCHF [73] Skew-Tent Map (chaos based)		LHash [74], [75] Cellular Automata			L-CAHASH [77] Cellular Automata
			Hash-One [78] NFSR			LCAHASH1.1 [79] Cellular Automata
			LNHash [80] Cellular Automata			
			LNMNT Hash [81], [82] New Mersenne Number Transform (NMNT)			

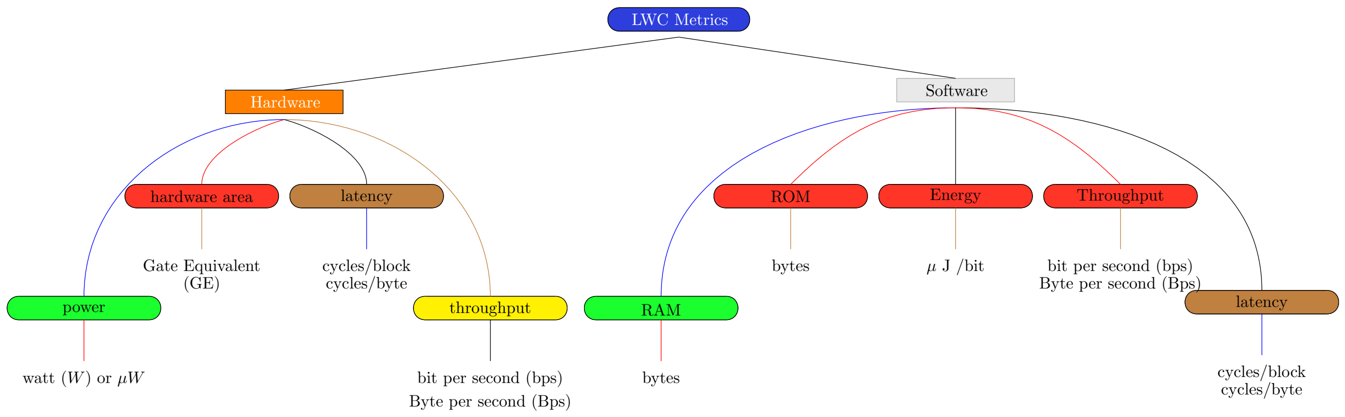

FIGURE 3. IoT device implementation metrics.

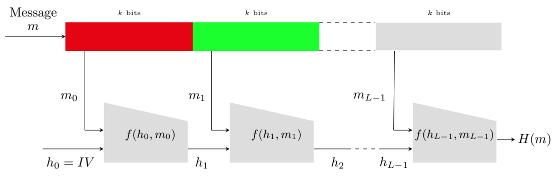

FIGURE 4. Merkle-Damgård Construction

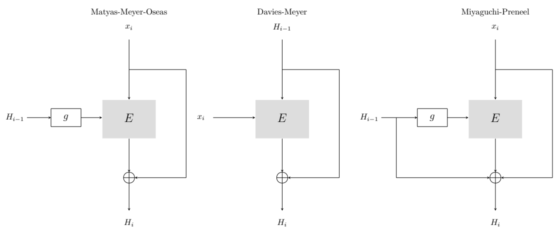

FIGURE 5. Hash functions based on block ciphers [19], [45].

2) The output is H_t:
$H_0 = IV; H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}, 1 \leq i \leq t.$

C. SPONGE CONSTRUCTION

The sponge construction method has 2 (two) stages: the absorbing and squeezing phases. Fig. 6 illustrates the sponge construction method. In this construction, the designer changes the function f by adding a new permutation or combining existing permutations.

Bertoni et al. [90] proposed the sponge construction method. This construction was further developed in 2011 [91]. Sponge construction is a method for constructing a hash function from a permutation without a publicly known key, which is referred to as P-sponge construction, or a random function, which is referred to as T-sponge construction [90]. In general, the steps in the sponge construction process can be described as follows:
Pad the message M if necessary. Then, divide the padded message into blocks of length r bits. Initialize the internal state with $b = (r + c)$ bits with bit 0, where r is the (bit) rate

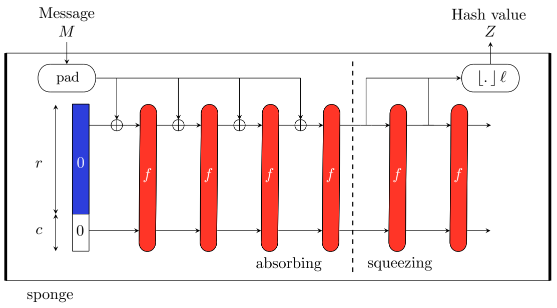

FIGURE 6. Sponge construction

and c is the capacity. Obtain the hash value by absorbing the padded message and squeezing the internal state.

The absorbing phase includes the following steps:

1) Replace the first r bits of the internal state by XORing the previous r-bit values with the r-bit padded message.
2) Replace the internal state with the output of the f function.

The above steps are repeated until the entire message block is processed. The squeezing phase includes Z/r steps, where Z is the hash value with length ℓ. The steps are as follows:

1) Store the initial r bits of the internal state.
2) Replace the internal state with the output of the f function.

The hash value Z is generated by concatenating the r-bit blocks.

The padding algorithm is relatively simple to use. For example, the Keccak, or SHA-3, algorithm [23] uses multirate padding. In the last message block, add bit 1, then bits 0 are added as necessary to ensure that the block length is a multiple of r.

Bertoni et al. [92] proved the security claim of sponge construction, which is known as the flat sponge claim. This claim proves that an attacker can "distinguish" the sponge construction output from a random oracle with a probability of $\frac{N}{2^{c/2}}$, where c is the capacity and N is the number of times

TABLE 3. The adversary efforts on sponge construction.

Type	Pre	2nd Pre	Coll
T-Sponge	$\min(n, c+r)$	$\min(n, c-\log_2(m))$	$\min(n, c)$
P-Sponge	$c-1$	$\min(n, \frac{c}{2})/2$	

the f function is called. A sponge structure with capacity c, rate r, and hash value of n bits can absorb messages of length $m < 2^{c/2}$. The resistance of sponge constructions to attacks defined as in Definition 2 is summarized in Table 3.

VII. LIGHTWEIGHT CRYPTOGRAPHIC HASH FUNCTIONS IN THE WILD

We identify 34 LWCHFs that have been used in academia and industry. As discussed in Section VI, the design focuses on the cost, performance, and security trade-offs. Fig. 7 illustrates these trade-offs. Four algorithms (11.8%) are based on the Merkle- Damgård construction: ARMADILLO, ARMADILLO2, the Al-Odat LWCHF, and the El Hanouti LWCHF. Fourteen point seven percent (5 algorithms) are block cipher-based algorithms: DM-PRESENT, H-PRESENT-128, C-PRESENT-192, LesamntaLW, and TWISH. The most significant portion (23 algorithms or 67.6%) are sponge construction-based LWCHFs: Quark, PHOTON, SPONGENT, GLUON, SPN-Hash, SipHash, LHash, Neeva-Hash, Hash-One, Gimli-Hash, sLiSCP-hash, sLiSCP-light-hash, LN-Hash, ASCON-hash, ACE-$\mathcal{H}$, KNOT-Hash, DryGascon-Hash, ORANGISH, PHOTON-Beetle-Hash, ESCH, Subterranean2.0-XOF, Xoodyak-hash, and HVH. Two algorithms, or 5.9%, are based on cellular automata: L-CAHASH and LCAHASH1.0.

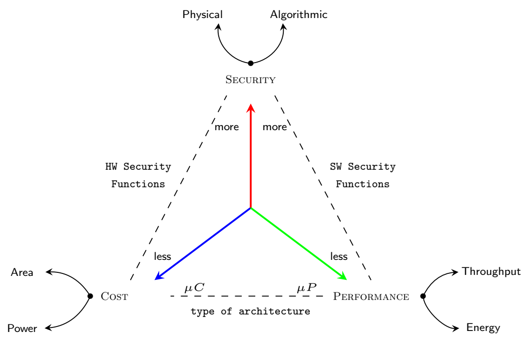

FIGURE 7. Security, performance, and cost trade-offs.

A. MERKLE-DAMGÅRD CONSTRUCTION

Badel et al. [44] proposed ARMADILLO and ARMADILLO2 as general-purpose cryptographic function designs. ARMADILLO and ARMADILLO2 can be used with fixed-input length MACs for challenge-response protocols, hashing & digital signatures, and PRNG & PRF. The proposed hash function includes five variants according to the length of the hash value: 80 bits, 128 bits, 160 bits, 192 bits, and 256 bits.

Al-Odat et al. [52] proposed a family of lightweight cryptographic hash functions based on the Merkle-Damgård construction. The algorithm has five hash value variants: 160, 224, 256, 384, and 512 bits. Unfortunately, this algorithm uses a substitution box, which is not explained in the article. Moreover, the author does not provide data on all LWCHF performance metrics; data on the power consumption, number of clock cycles, speed, and memory consumption were provided, while other performance metrics were ignored. In addition, the designer does not provide cryptanalytic results such as differential and linear cryptanalysis.

El Hanouti et al. recently proposed a lightweight hash function based on the Merkle-Damgård construction with a Feistel-like structure and a chaotic one-dimensional map known as the skew-tent map [73]. To the best of our knowledge, this proposal is the first chaotic map-based LWCHF algorithm. Other chaotic map-based hash functions [93]–[97] are not recommended for highly constrained devices. The author claims that the proposed hash function exhibits excellent performance (rapid implementation) and sufficient security properties. However, similar to the proposals of Al-Odat et al., not all performance metrics were considered in their study. Furthermore, the author does not provide supporting results concerning the cryptographic properties.

B. LWCHFS BASED ON BLOCK CIPHERS

DM-PRESENT [45] proposed the first lightweight hash function in the literature. As the name implies, it is a hash function that uses the PRESENT block cipher and the Davies-Meyer construction. There are two types of DM-PRESENT hash functions: DM-PRESENT-80 and DM-PRESENT-128. Both variants utilize 64-bit security. The designers claim that the hash functions provide a sufficient trade-off between space and throughput [45].

H-PRESENT-128 [45] is a hash function with 128-bit security. Bogdanov et al. designed H-PRESENT-128 using the Hirose construction [98]. H-PRESENT-128 is a double-block length (DBL) hash function. The compression function of H-PRESENT-128 takes two 64-bit chaining variables and one 64-bit message (H_1, H_2, M) and returns an output pair of updated chaining variables (H_1', H_2').

Bogdanov et al. designed C-PRESENT-192 [45] with the goal of developing a lightweight, collision-resistant cryptographic hash function. C-PRESENT-192 uses the same compression function as DM-PRESENT-128. The designers concluded that DM-PRESENT-128, as a building block, does not yield the expected results.

The designers claim that Lesamnta-LW [41] is a secure, lightweight hash function with a hash length of 256 bits. The main design goal is to achieve small hardware/software implementations. The designers chose the MD construction and an AES-based design for the building blocks. A 4-branch generalized Feistel network (GFN) and AES components (SubBytes and MixColumn) are utilized in the hash function. The MixColumn operation uses the AES maximum distance separable (MDS) matrix multiplication defined over

GF(2^8). TWISH [57] was designed based on the TWINE-128 [99] block cipher algorithm and uses the DM construction. TWISH is a single-block length hash function that accepts a 128-bit message input and returns a 64-bit hash value. The message input in the DM scheme acts as a key. The designer tested the security of the TWISH function by using the cryptographic randomness test proposed by [100].

C. SPONGE CONSTRUCTION

The first lightweight sponge construction-based hash function was QUARK [47]. This hash function was first proposed at CHES 2010. The version discussed in this section was updated in 2012. QUARK has been proposed as a lightweight hash function. The algorithm was inspired by the stream cipher Grain [101] and the block cipher KATAN [102]. Two nonlinear feedback shift registers (NFSRs) and a linear feedback shift register (LFSR) are used for the permutation.

PHOTON was proposed by [50] and uses both sponge and AES-like constructions. PHOTON is a compact hash function that uses 1120 gate equivalents (GE) to achieve 64-bit security. When compared with similar algorithms, the speed of this algorithm is claimed to be competitive.

SPONGENT [53], [54] is a family of hash functions designed by Bogdanov et al. and presented at CHES 2011. SPONGENT was designed as a family of hash functions with an 88-bits hash value to ensure resistance to preimages, 128 bits, 160 bits, 224 bits, and 256 bits. The authors claim that the algorithm is resistant to attacks aimed at the hash function.

GLUON [48] was developed by Berger et al. and presented at AFRICACRYPT 2012. This algorithm uses the feedback with carry shift register (FCSR) and was motivated by the stream cipher algorithms F-FCSR-v3 [103] and X-FCSR-v2 [104]. The developer proposed three instances: GLUON-128/8, GLUON-160/16, and GLUON-224/32 for 64-bit, 80-bit and 112-bit security levels, respectively.

Another algorithm is SPN-Hash [51]. This algorithm uses another type of sponge construction: the JH construction [105]. The hash function was designed by Choy et al. The main purpose of the design is to provide provable security against differential collision attacks. The S-Box used in the algorithm is the Advanced Encryption Standard (AES) [106].

The SipHash [49] algorithm has an ARX (addition, rotation & XOR) structure. This algorithm is intended for use in network traffic authentication applications and protected hash table lookups. SipHash was inspired by the BLAKE [107] and Skein [108] hash functions, which were both finalists in the SHA3 competition.

LHash [74], [75] is an LWCHF that was proposed by Wu et al. and supports three different message digest sizes: 80, 96, and 128 bits. The LWCHF provides preimage security, second preimage security between 64 and 120 bits, and collision security between 40 and 60 bits. LHash requires approximately 817 and 1028 GEs with serial implementations and 989 and 1200 GEs with 54 and 72 cycles per block in a faster implementation based on the T function. In addition, its energy consumption evaluated according to the energy per bit is extraordinary. The LHash design uses the Feistel-PG structure in the internal permutation, which take advantages of the permutation layer on the nibbles to increase the diffusion speed. The low-area implementation arises due to the hardware-friendly S-box and a linear diffusion layer. The designer evaluated LHash's resistance to known attacks and confirmed that this LWCHF provides a good security margin.

Neeva-hash [76] is a sponge construction-based LWCHF with a message digest length of 224 bits. This algorithm uses 32 rounds to generate a hash value. The only nonlinear function in the Neeva-hash LWCHF utilizes a 4×4-bit PRESENT S-Box. State b has 256 bits, the rate is 32 bits, and the capacity is 224 bits. The round function uses the ARX structure.

Mukundan et al. proposed Hash-One [78], aiming at both simplicity and security. Hash-One uses a sponge construction and two 80- and 81-bit nonlinear feedback shift registers (NFSRs) and supports message digests with sizes of 160 bits. The level of security expected by the designer is 160 bits for preimage resistance and 80 bits for collision resistance.

Gimli-Hash [56] is a derivative of the Gimli permutation function that was proposed by Bernstein et al. in 2017 [55]. The authors claim that this permutation function can be used in various platforms, such as 64-bit Intel/AMD server CPUs, 64-bit and 32-bit ARM smartphone CPUs, 32-bit ARM microcontrollers, 8-bit AVR microcontrollers, FPGAs, ASICs with side-channel protection, and ASICs without side-channel protection.

sLiSCP-hash [43] was designed by AlTawy et al. from the University of Waterloo, Canada, in 2017. Simeck-based permutations for lightweight sponge cryptographic primitives (sLiSCP) are designed for integrated duplex sponge construction and provide minimal overhead for cryptographic functions in single-hardware designs. The sLiSCP design follows the four-subblock Type-2 Generalized Feistel-like Structure (GFS). The algorithm uses the unkeyed Simeck algorithm [109], [110] with round reduction as the round function. The algorithm can be used for two applications: hashing and authenticated encryption.

In the publication [42], AlTawy et al. reviewed the sLiSCP design and developed an sLiSCP-light permutation. This permutation is the building block of sLiSCP-light-hash. The GFS design was changed to a partial substitution-permutation network (P-SPN) construction, and the resulting sLiSCP permutation hardware area was approximately 16% smaller than the previous hardware area. This change also improved the permutation function's bit diffusion and algebraic properties. This improvement reduced the number of steps and achieved better throughput in the hashing and authentication modes.

Zhang et al. presented LNHash [80], a lightweight hash function that uses linear and nonlinear cellular automata as internal permutations. The goal of this hash function is to achieve high diffusion and confusion. Six types of hash functions with different levels and capacities have been proposed.

The ACE-$\mathcal{H}$-256 [60] is a hash function developed based on the ACE permutation that has 320 input and output bits. This hash function uses the 5-block generalized version of sLiSCP-light [42]. The ACE permutation uses the SIMECK-box (SB-64) as a nonlinear layer.

ASCON-HASH [61] is a member of the ASCON family of cryptographic algorithms proposed in the NIST Lightweight Cryptography competition. Previously, ASCON was the winner of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [111], which was organized by the NIST to standardize the Authenticated Encryption (AE) algorithm.

KNOT-Hash [62], [63] belongs to the hash function family proposed in the second round of the LWC NIST competition. The hash function defines three operations used in each round: $AddRoundConstant_b$, $SubColumn_b$, and $ShiftRow_b$. These operations are performed in different states and are defined according to the width b parameter in the sponge construction, i.e., 256 bits, 384 bits, and 512 bits. The KNOT permutation is similar to the 64-bit RECTANGLE block cipher [112], [113].

DryGascon-Hash [64] is a family of hash functions designed based on the DrySponge construction and the ASCON [114] algorithm. The DrySponge construction was developed based on the duplex sponge construction [115]. The designers of DryGascon claim that the safety of Gascon permutations is similar to that of ASCON permutations [64].

The ORANGISH algorithm is a member of the ORANGE cryptographic primitive family proposed by Mridul Nandi and Bishwajit Chakraborty [65]. The permutation used in this algorithm is PHOTON_{256} [50]. The designer used this permutation mainly because it is the lightest 256-bit permutation in the literature. The hash function is similar to that of JH [105]. JH was one of the five finalists in the SHA3 competition organized by NIST [116].

PHOTON-Beetle-Hash uses the PHOTON_{256} [50] permutation as an algorithmic building block and Beetle's sponge mode [117]. The hash function accepts any message input $M \in \{0.1\}^*$ and returns a 256-bit long hash $\mathcal{H}(M) \in \{0,1\}^{256}$.

The hash function ESCH [68] has two variants: ESCH256 and ESCH384. ESCH256 and ESCH384 accept inputs with arbitrary bit lengths and return hash values of 256 bits and 384 bits, respectively. The designer chose ESCH256 as the main proposal for the hash function. This algorithm was developed based on the SPARKLE permutation [67] family, with a rate of r and a capacity of c.

Subterranean [118] is a cryptographic primitive that was originally proposed in 1992 and has been used in hash functions and stream cipher functions. A modification of the Subterranean rotation function was used in the Subterranean2.0-XOF (extendable output function) algorithm [69], [70]. This algorithm uses the Subterranean2.0 loop function with an input of arbitrary bit length and an output of 256 bits. The designers claim that Subterranean2.0-XOF has 224-bit security.

XOODYAK [71] is a cryptographic primitive intended for use in hash functions, pseudorandom bit generators (PRBGs), authentication, encryption, and authenticated encryption (AE). The permutation is the building block of XOODYAK-HASH MODE. XOODYAK uses a 384-bit permutation XOODOO [119], [120]. XOODOO is a family of permutations inspired by KECCAK-p [23], [91]. Similar to KECCAK-p, the loop function XOODOO operates on a state with 3 horizontal planes known as a plane. Each plane consists of four 32-bit lane pieces.

HVH [72] is an LWCHF designed by Huang et al. that was presented at the Security, Privacy, and Anonymity in Computation, Communication, and Storage (SpaCCS) 2020 International Workshops in Nanjing, China, 18-20 December 2020. HVH uses a sponge construction based on the lightweight block cipher VH [121]. VH is a lightweight block cipher that was proposed by Dai et al. in 2015. VH has a block size of 64 bits and a key length of 80 bits. The HVH designers defined five different output message lengths, 88-bit, 128-bit, 160-bit, 224-bit, and 256-bit, for use in different application scenarios. HVH follows the structure of the substitution permutation network (SPN). The designer claims that the HVH hash function family strikes a delicate balance between hardware and software implementations and satisfies hardware usage requirements in extreme, resource-limited environments.

LNMNT Hash is a sponge-based hash function that was proposed by Nabeel et al. at the 2021 8th International Conference on Computer and Communication Engineering (ICCCE) [82]. LNMNT Hash is based on the new Mersenne number transform (NMNT). The designer provided a security analysis in [81]. The designer analyzed the randomness, obfuscation, diffusion, hash value distribution, and differential attacks. There are four classes of LNMNT hash functions: LNMNTHash80, LNMNTHash128, LNMNTHash160, and LNMNTHash224.

D. CELLULAR AUTOMATA

L-CAHASH [77] is an LWCHF-based cellular automaton with two variants: 128-bit and 256-bit. Designers claim that linear cellular automata have good chaotic properties and match the security analyses, statistical analyses, and software performance metrics of the hash function. Security analyses include the complexity, preimage and collision resistances, and avalanche criterion. For the statistical analysis, the author used the Diehard test [122]. The software performance analysis compares L-CAHASH with GLUON, U-QUARK, D-QUARK, S-QUARK, and PHOTON.

LCAHASH1.1 [79] is an extension of L-CAHASH [77] that uses a hybrid cellular automaton with a rule set of 30, 90. Based on the cycle per byte (CPB) metric, the software performance of LCHASH1.1 is better than that of L-CAHASH.

VIII. RESULTS AND DISCUSSION

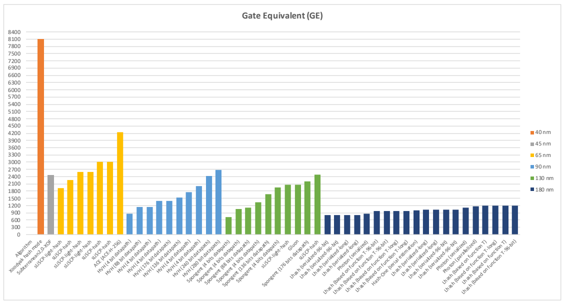

FIGURE 8. Best hardware performance in terms of the GE.

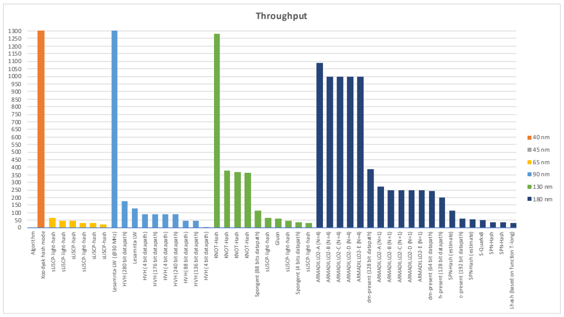

FIGURE 9. Best hardware performance in terms of the throughput.

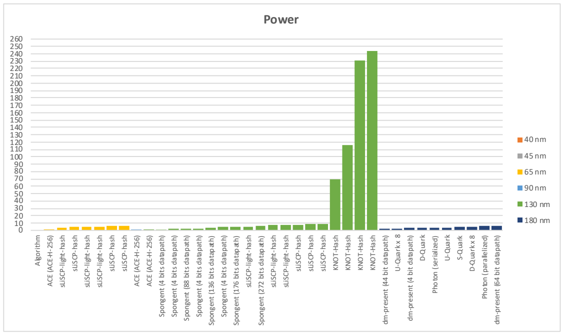

FIGURE 10. Best hardware performance in terms of the power.

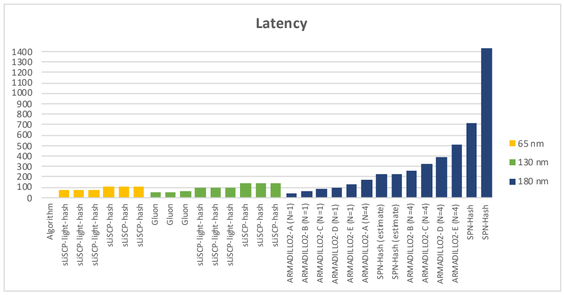

FIGURE 11. Best hardware performance in terms of the latency.

A. LWCHF CRYPTOGRAPHIC PROPERTIES

When a designer proposes an LWCHF, in addition to the implementation performance, the security properties are critical. The most commonly used term is cryptanalysis, or, in some literature, cryptanalytic attacks [33], [123]–[127]. Cryptanalytic attacks are attacks that determine the weak points of cryptographic primitives. Attacks on cryptographic hash functions are similar to attacks on other cryptographic primitives. In particular, if the LWCHF building blocks use existing cryptographic primitives, such as block ciphers or stream ciphers, automatic generic attacks on cryptographic primitives may also apply to LWCHFs.

Two important components of cryptanalysis are mathematical cryptanalysis and implementation attacks. Mathematical cryptanalysis involves attacks on the mathematical structure of cryptographic primitives. Implementation attacks exploit side-channel information, such as the execution time, RAM/ROM, power, or energy consumption, to analyze cryptographic primitives. These types of attacks are also called side-channel attacks. One example of a side-channel attack is differential fault analysis (DFA), which is commonly used with cryptographic hash functions [128]–[131]. The principle of the attack is to push errors or faults with unforeseen environmental conditions into the cryptographic implementation to reveal its internal state. We identified several mathematical cryptanalysis techniques, including differential cryptanalysis, linear cryptanalysis, integral cryptanalysis, algebraic cryptanalysis, rebound attacks, zero-sum distinguishers, slide attacks, rotational distinguishers, cube attacks, meets/misses in the middle distinguisher, invariant subspace distinguishers, boomerang attacks, yoyo games, truncated differentials, and impossible differentials.

Differential cryptanalysis and its derivatives are the most commonly considered types of cryptanalysis. Biham and Shamir [125] first proposed this attack in Crypto 1990 to attack the Data Encryption Standard (DES). This technique was also described in detail in a book published by the same researcher [123]. Differential cryptanalysis is a common technique for analyzing symmetric cryptographic primitives, particularly block ciphers and hash functions.

At EUROCRYPT 1993, Matsui [132] introduced theoretical attacks using linear cryptanalysis approaches to attack DES algorithms. Matsui performed practical attacks on the same algorithm [133]. The basic idea of this attack is to approximate the algorithm's operation with a linear expression. Integral cryptanalysis involves multiset attacks. Multiset attacks are a generic attack class that includes several attacks that appear in the literature under three different names: square attacks [134], saturation attacks [135], and integral cryptanalysis [136]. This type of attack was first discovered by Daemen, Knudsen, and Rijmen while analyzing the square block cipher [134]. A similar attack known as a saturation attack was used by Lucks [135] against the block cipher Rijndael. Biryukov and Shamir showed attacks of the same type on three arbitrary SPN rounds. Knudsen-Wagner's integral attack [136] on five rounds of MISTY [137] is in the same

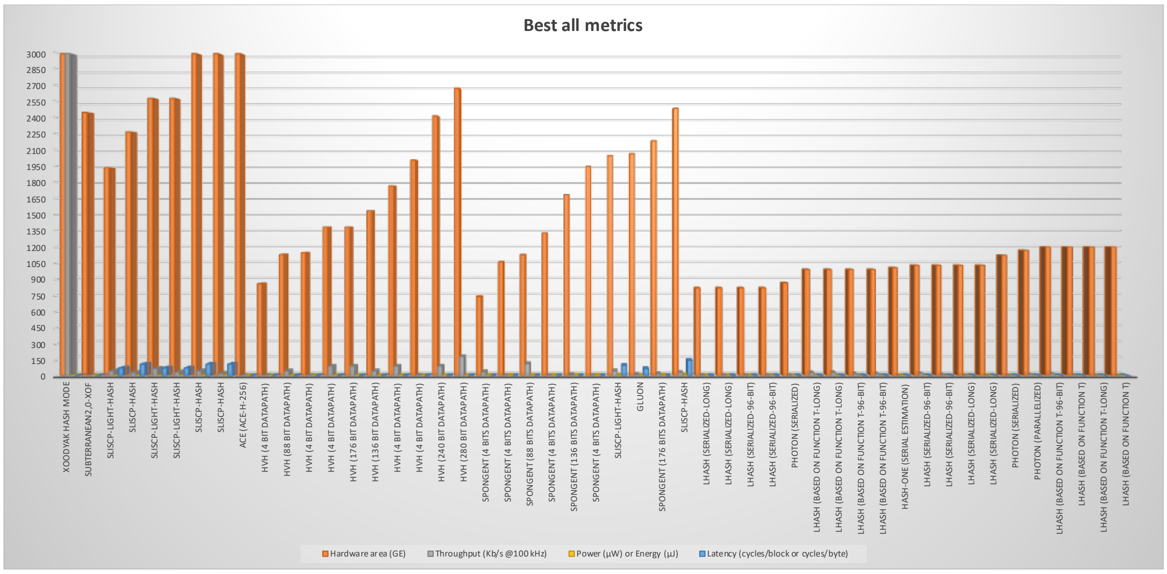

FIGURE 12. Best overall hardware performance for each type of technology.

category. Since many hash function constructs use SPNs, this attack deserves careful consideration.

The main idea of algebraic cryptanalysis is to express a cryptographic hash function with a nonlinear equation involving the message input and hash value output. The nonlinear equations are in the form of polynomial equations. One advantage of algebraic cryptanalysis is its widespread application, as a set of polynomial equations can be used to describe any cryptographic primitive.

Table 4 provides a detailed comparison of the performance of the LWCHF algorithm from the perspective of various cryptographic properties. We define the rate as the size of the message block processed during each round and denote the preimage, second preimage, and collision resistance as `Pre`, `2nd Pre`, and `Coll`. Table 4 shows that almost all the identified LWCHF algorithms were evaluated by cryptanalysis. Table 4 summarizes a third-party cryptanalysis. Although this cryptanalysis cannot be used as a benchmark, the algorithm that was affected most by the attacks can be classified as weak and need special attention when implemented. Furthermore, it is necessary to determine whether the attacks occur in full or reduced rounds.

A lightweight hash function for a particular application must consider the cryptographic properties. For example, NIST [39] requires that the hash value length of the current usage be 256 bits, and a cryptanalytic attack requires at least 2^{112} computations. Therefore, the user should not use hash functions with hash values of less than 256 bits for applications requiring high security levels. Such hash functions include ARMADILLO and ARMADILLO2 (80, 128, 160, and 192 bits), the Al-Odat et al. hash function (160), the El Hanouti hash function, DM-PRESENT, H-PRESENT, C-RESENT, TWISH, Quark (136, 176), SPN-Hash, and Hash-One. Thus, PHOTON, SPONGENT, and Lesamnta-LW were selected as lightweight hash function standards in ISO-IEC 29192-5 (2016) [138]. PHOTON and SPONGENT represent algorithms optimized for hardware, while Lesamnta-LW represents an algorithm optimized for software.

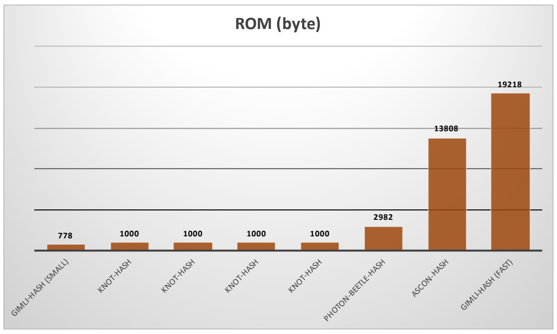

FIGURE 13. Best software performance in terms of ROM.

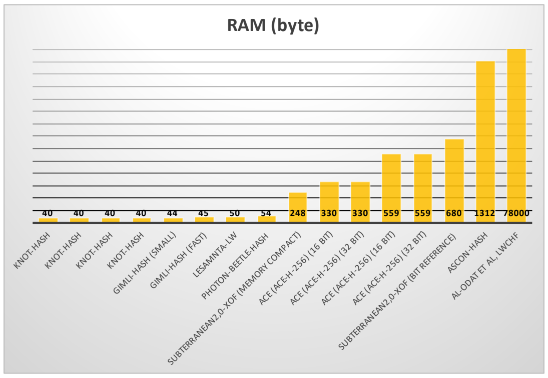

FIGURE 14. Best software performance in terms of RAM.

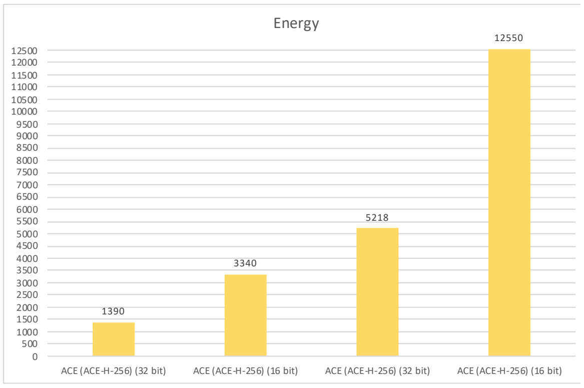

FIGURE 15. Best software performance in terms of the energy.

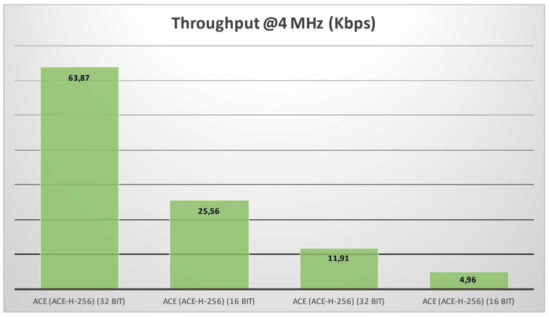

FIGURE 16. Best software performance in terms of the throughput.

B. PERFORMANCE COMPARISON

In addition to studies carried out by the designers, several studies have attempted to compare the performance of LWCHF algorithms [139]–[142]. On the one hand, these efforts have provided essential information about the performance of LWCHFs, and their shortcomings are significant to note. However, the results may not provide a complete picture of the algorithm's potential for a given metric. In addition, the implementation assumptions or goals of various LWCHFs differ, and some proposals have more varied implementations than other proposals. Thus, the results do not indicate a ranking; rather, they serve as a general recommendation.

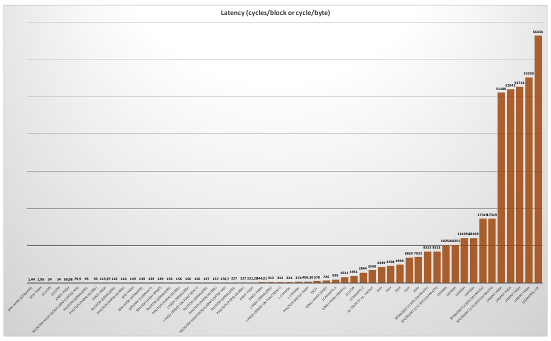

FIGURE 17. Best software performance in terms of the latency.

Due to the differences in the metrics that designers use for various hardware and software implementations, as well as differences in the devices themselves, fair comparisons are almost impossible. Table 5 presents a comparison of different hardware and software implementations. We explored 135 hardware implementations of LWCHFs with 40 nm, 45 nm, 65 nm, 90 nm, 130 nm, and 180 nm technologies. Most hardware implementations use 180 nm, 130 nm, 90 nm, 65 nm, 45 nm, and 40 nm technology. Table 5 shows that the performance metrics for many software implementations are not available. This condition occurs because there are differences in the designer's metrics.

1) Hardware

Figs. 8, 9, 10, and 11 illustrate the hardware implementation performance according to each metric. We summarize the hardware implementations for each type of technology in terms of the hardware area (GE), throughput, power, and latency. The performance of 40 nm technology is marked in orange, 45 nm technology is marked in gray, 90 nm technology is marked in light blue, 130 nm technology is marked in green, and 180 nm technology is marked in dark blue. All algorithms that exceed the lower limit of 3000 GE are not rated as efficient, including Xoodyak Hash mode, ACE-$\mathcal{H}$-256, SipHash, Lesamnta-LW, KNOT-Hash, GLUON, SPN-Hash, and C-PRESENT. The highest throughputs were generated by Xoodyak hash mode, Lesamnta-LW, KNOT-Hash, and ARMADILLO2-A. The SPONGENT, PHOTON, and LHash algorithms had the lowest throughput. The KNOT-Hash family of hash functions and the ARMADILLO family of hash functions consume the most power, resulting in low hardware efficiency. This condition is correlated with the number of GEs and the throughput of the two algorithms. The ACE-$\mathcal{H}$-256 algorithm uses the lowest power of less than 1 μW. SPONGENT-80 (with 4- and 8-bit datapaths) and SPONGENT-128 (4-bit datapath) algorithms also use relatively low power, in this case, less than 2.5 μW.

SPONGENT-80 with a 4-bit datapath requires the smallest hardware area of 738 GE. The number of GEs is 79 GE, which is less than the number of GEs in the LHash-80 implementation in serialized and long message modes.

The lowest latency was generated by Neeva-Hash, ARMADILLO2-A-80, GLUON-160, and GLUON-224, while the highest latency was generated by SPN-Hash-256, SPN-Hash-128, ARMADILLO2-E-256, and ARMADILLO2-D-160.

Fig. 12 summarizes the best hardware performance based on the technology used. Two hash function algorithms occupy the first and second positions, namely, the sLiSCP-light-hash-160 and sLiSCP-hash-160 algorithms. Both algorithms obtain good ratings for all metrics and are included in the charts showing the best metrics. Serial implementations of several algorithms, such as Photon, LHash, Hash-One, and SPONGENT, were proven to use small hardware areas between 800 GE and 1200 GE. In addition to serial implementations, designers used the bit-slice technique to reduce the

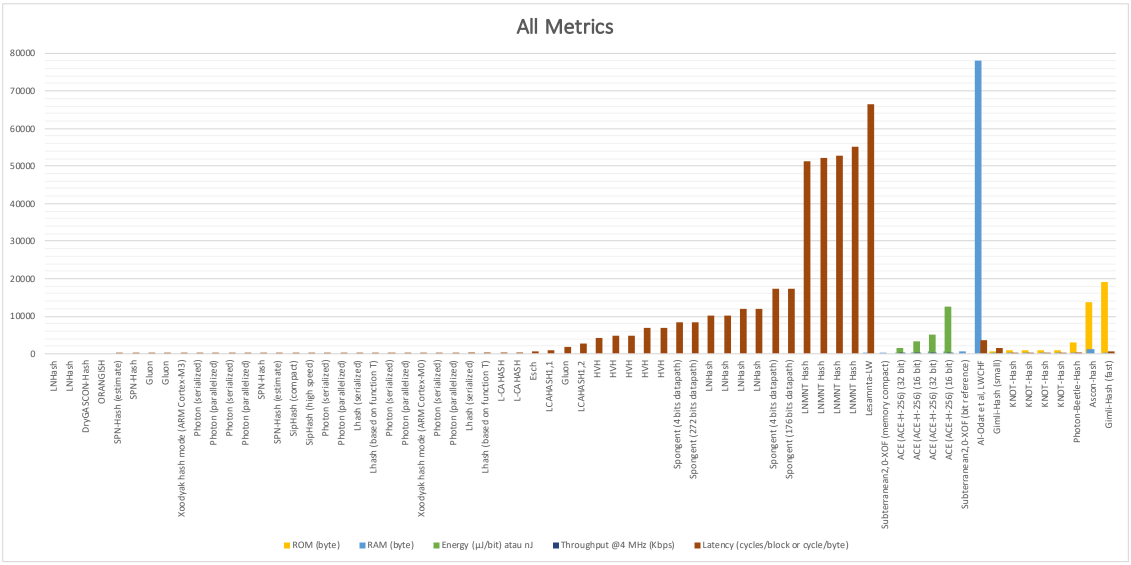

FIGURE 18. Best overall software performance.

hardware area and design complexity. Some algorithms that use this technique are ACE-$\mathcal{H}$-256, ASCON-HASH, KNOT, PHOTON-Beetle-Hash, SipHash, and sLiSCP-light-hash.

2) Software

Because the software implementations of the LWCHF algorithm are more varied than the hardware implementations, many algorithms have empty metric values. This condition shows that algorithm designers use different hardware, software, and metrics. Figs. 13, 14, 15, 16 and 17 illustrate the software implementation performance based on various metrics.

The smallest ROM metrics are achieved by Gimli-hash (small) with 778 bytes, KNOT-Hash, PHOTON-BEETLE-HASH, ASCON-HASH, and Gimli-Hash (fast). KNOT-Hash, Gimli-Hash, Lesamnta-LW, and PHOTON-Beetle-Hash have the smallest RAM size (less than 100 bytes). ACE-$\mathcal{H}$-256 has the lowest energy consumption and highest throughput. The lowest latencies were obtained by SPN-Hash, GLUON, KNOT-Hash, Xoodyak-hash mode (on ARM Cortex-M3), PHOTON, SipHash, and LHash. In contrast, Lesamnta-LW produced an enormous latency.

Fig. 18 illustrates the best software implementations of all metrics. The top three software implementation results are Gimli-Hash (small), ACE-$\mathcal{H}$-256 (32-bit), and KNOT-Hash.

C. RESEARCH CHALLENGES AND FUTURE DIRECTIONS

Designing lightweight cryptography primitives is a challenging task. The designer must balance the security, performance, and cost when implementing the algorithms in either hardware or software. We identified several issues and challenges that should be considered in future research.

LWCHF design and implementation. The cryptographic implementation investigated in this study demonstrates the overall performance of various LWCHF designs. However, the results of this study are distorted due to the dependence on tools and technology, resulting in significant deviations between studies. Therefore, it is crucial to develop another solution, such as proposing a novel hash function to compare with the existing hash function. This new paradigm may increase the quality and quantity of research on lightweight cryptographic hash functions. In particular, lightweight permutation designs with reasonable diffusion rates and resistance to differential, linear cryptanalysis, or other attacks were researched. This research opportunity was possible due to the various permutations designed for multiple cryptography primitives. These permutations can be used for various cryptography primitives, such as AEAD, hash functions, PRNG, and KDF. Some permutations include ACE [60], sLiSCP [43], sLiSCP-light [42], XOODOO [119], Sparkle [67], [68], Alzette [143], and Subterranean2.0 [69], [70].

Substitution box design. An alternative s-box with a smaller hardware implementation area and similar cryptographic properties to the proposed s-box, namely, the Simeck s-box used in permutations of the sLiSCP, sLiSCP-light, ACE-$\mathcal{H}$-256, sLiSCP-hash, and sLiSCP-light-hash algorithms, should be developed.

Optimal round function design. An optimal round function based on a permutation substitution network (SPN), Feistel network, addition, rotation, and XOR (ARX) structure, or another approach should be designed.

Security metrics standardization. The metrics for evaluating the security performance and hardware and software implementations vary widely. As mentioned in the previous discussion, because of this condition, fair comparisons of different algorithm implementations are almost impossible. Therefore, standard hardware and software security and performance metrics should be developed to analyze LWCHF security and implementations on devices with limited resources. Several attempts to develop such metrics have been made, including by NIST (USA), Cryptrec (Japan), and ECRYPT (Europe).

Novel cryptanalytic attacks. New cryptanalytic approaches for analyzing the proposed permutations or hash function algorithms, particularly differential cryptanalysis and linear cryptanalysis and attacks on secure hash function properties, including the preimage, second preimage, and collision resistance, should be researched.

IX. CONCLUSION

The lightweight cryptographic hash function has played a crucial role in the development of the IoT. This paper presents recent developments and state-of-the-art implementations of lightweight cryptographic hash functions. The hardware and software implementations of LWCHFs were examined based on nine metrics. In addition, the security, cost, and performance properties of different proposals were considered. Furthermore, a comparative analysis was presented, with the information presented in corresponding tables. A large number of studies have been conducted as the field has developed, with brand new algorithms and cryptanalytic attacks proposed in published works. We hope that the review presented in this study will contribute to the design of a robust and secure LWCHF.

TABLE 4. Cryptographic properties and known cryptanalysis of the lightweight cryptographic hash function [‡].

Algorithm	Hash value	Rate	Internal state	Construction	Pre	2nd Pre	Coll	Cryptanalysis
Merkle-Damgård Construction:								
ARMADILLO & ARMADILLO2 [44]	80 128 160 192 256	48 64 80 96 128	256 384 480 576 768	data-dependent bit transpositions [144]	2^{80} 2^{128} 2^{160} 2^{192} 2^{256}	2^{80} 2^{128} 2^{160} 2^{192} 2^{256}	2^{40} 2^{64} 2^{80} 2^{96} 2^{128}	local linearization (practical, found collision in only a few seconds on a PC) [145]; meet-in-the-middle attack [146]
Al-Odat et al. LWCHF [86]	160 224 256 384 512	512 512 512 512 512	512 512 512 512 512	JH mode	2^{160} 2^{224} 2^{256} 2^{384} 2^{512}	2^{160} 2^{224} 2^{256} 2^{384} 2^{512}	2^{80} 2^{112} 2^{128} 2^{192} 2^{256}	None
El Hanouti et al. LWCHF [73]	128	1024	1024	Feistel-like structure; skew tent map	2^{128}	2^{128}	2^{64}	none
Block Cipher-Based Construction:								
DM-PRESENT [45], [46]	64 64	80 128	64 64	Davies-Meyer	2^{64}	None	None	Multidifferential: 18-round distinguisher, 12-round collisions [147]; truncated differential [148]
H-PRESENT [45], [46]	128	128/8	128	Hirose construction [98]	2^{128}	None	None	Truncated differential [148]
C-PRESENT [45], [46]	192	64	192		2^{192}	None	None	None
Lesamnta-LW [41]	256	128	256	LW1 block cipher	2^{256}	2^{256}	2^{120}	31 of 32 rounds improved integral analysis [149]; integral cryptanalysis [150]
TWISH [57]	128	128	128	Davies-Meyer	2^{128}	2^{128}	2^{64}	None
Sponge Construction:								
QUARK [47]	136 176 256	8 16 32	136 176 256	P-Sponge	2^{128} 2^{160} 2^{224}	2^{64} 2^{80} 2^{112}	2^{64} 2^{80} 2^{112}	Improved conditional differential cryptanalysis [151]; differential cryptanalysis distinguisher for all variants [152]
PHOTON [50]	80 128 160 224 256	20/16 16 36 32 32	100 144 196 256 288	P-Sponge	2^{64} 2^{112} 2^{124} 2^{192} 2^{224}	2^{40} 2^{80} 2^{64} 2^{112} 2^{128}	2^{40} 2^{80} 2^{64} 2^{112} 2^{128}	Cube attack [153]
SPONGENT [53]	80 128 160 224 256	8 8 16 16 16	88 136 176 240 272	P-Sponge	2^{80} 2^{120} 2^{144} 2^{208} 2^{240}	2^{40} 2^{64} 2^{80} 2^{112} 2^{128}	2^{40} 2^{64} 2^{80} 2^{112} 2^{128}	Algebraic attack on 6-round Spongent-88 [154]; 23-round linear distinguisher [155]; improved zero-sum distinguisher [156]; 18-round zero-sum distinguisher [157]
SPN-Hash [49]	128 256	256 512	128 256	P-Sponge JH mode	2^{128} 2^{128}	? ?	2^{64} 2^{64}	none
GLUON [48]	128 160 224	8 16 32	136 176 256	T-Sponge	2^{128} 2^{160} 2^{224}	2^{64} 2^{80} 2^{112}	2^{64} 2^{80} 2^{112}	Collision on update function; preimage with complexity 2^{105} [158]
SipHash [49]	64	64	256	T-Sponge JH mode	2^{64}	2^{64}	None	Differential cryptanalysis [159], rotational XOR, collisions [160]
LHash [74], [75]	80 96 128 128	16 16 128 128	96 96 16 8	P-Sponge	2^{64} 2^{80} 2^{96} 2^{120}	2^{40} 2^{40} 2^{56} 2^{60}	2^{40} 2^{40} 2^{56} 2^{60}	None
Neeva-hash [76]	256	32	256	P-Sponge; ARX	2^{224}	2^{112}	2^{112}	Correcting block attack on reduced Neeva-hash (32 bits) [161]
Hash-One [78]	160	1	160	P-Sponge	2^{160}	2^{80}	2^{80}	None
Gimli-Hash [55], [56]	256	128	384	P-Sponge	2^{128}	2^{128}	2^{128}	Quantum collision: 14 rounds [162], [163]; classic collision: 12 rounds [162], [163]; quantum semifree start collision: 20 rounds [162], [163]; classic semifree start collision: 18 rounds [162], [163]; preimage: 5 rounds [164], [165]; 2nd-preimage: 3 rounds [166]; differential cryptanalysis on reduced version [167]
sLiSCP-hash [43], [58]	160 192 192	32/32 64/64 64/32	192 256 256	P-Sponge	2^{128} 2^{128} 2^{160}	2^{80} 2^{96} 2^{96}	2^{80} 2^{96} 2^{96}	Forgery and 8-step collision with 18 permutations steps [168]
sLiSCP-light-hash [43], [58]	160 192 192	32/32 64/64 64/32	192 256 256	P-Sponge	2^{128} 2^{128} 2^{160}	2^{80} 2^{96} 2^{96}	2^{80} 2^{96} 2^{96}	Differential cryptanalysis on sLiSCP-light permutation [169]
LNHash [80]	80 96 128 128 160 160	96 96 128 128 176 176	16 16 16 8 16 16	P-Sponge	2^{72} 2^{80} 2^{96} 2^{120} 2^{144} 2^{152}	2^{40} 2^{40} 2^{56} 2^{60} 2^{80} 2^{80}	2^{40} 2^{40} 2^{56} 2^{60} 2^{80} 2^{80}	None

Continued on the next page

TABLE 4. continued from previous page

Algorithm	Hash value	Rate	Internal state	Construction	Pre	2nd Pre	Coll	Cryptanalysis
ACE (ACE-$\mathcal{H}$-256) [60]	256	64	320	P-Sponge	2^{192}	2^{128}	2^{128}	Impossible differential attack on ACE permutation [170]
ASCON-HASH [61]	256	64	320	P-Sponge	2^{128}	2^{128}	2^{128}	Collision attack using 2-round differential cryptanalysis [171], [172]; active and passive side-channel key recovery attacks [173]
KNOT-hash [62], [63]	256	32	256	P-Sponge	2^{128}	2^{112}	2^{112}	Security analysis [174]
	256	128	384		2^{128}	2^{128}	2^{128}	
	384	48	384		2^{192}	2^{168}	2^{168}	
	512	64	512		2^{256}	2^{224}	2^{224}	
DryGASCON [64]	128	128	320	DrySponge	None	None	2^{64}	DryGASCON-256 [175]: 4-round subspace trails. DryGASCON-128 [175]: 3-round subspace trails, 3.5-round truncated differential, 5-round differential-linear distinguisher; practical forgery attacks on DryGASCON by exploiting internal collisions of the underlying permutation [176]
	256	128	576	P-Sponge	None	None	2^{128}	
ORANGISH [65]	128	–	128	P-Sponge	2^{128}	2^{112}	2^{112}	None
PHOTON-Beetle-Hash [66]	128	32	128	P-Sponge	2^{128}	2^{112}	2^{112}	None
ESCH [68]	256	128	384	P-Sponge	2^{128}	2^{128}	2^{128}	None
	384	128	512		2^{192}	2^{192}	2^{192}	
Subterranean 2.0 [69], [70]	256	–	257	P-Sponge	2^{224}	2^{224}	2^{224}	None
Xoodyak Hash Mode [71]	256	User	384	P-Sponge	2^{128}	2^{128}	2^{128}	None
HVH [72]	88	88	8	P-Sponge	2^{72}	2^{40}	2^{40}	None
	128	128	8		2^{120}	2^{64}	2^{64}	
	160	160	16		2^{144}	2^{80}	2^{80}	
	224	224	16		2^{208}	2^{112}	2^{112}	
	256	256	32		2^{224}	2^{128}	2^{128}	
LNMNT Hash [77]	80	–	–	P-Sponge	2^{50}	–	–	None
	128	–	–		2^{80}	–	–	
	160	–	–		2^{100}	–	–	
	224	–	–		2^{120}	–	–	
Cellular Automata:								
L-CAHASH [77]	128	128	128	Cellular Automata	2^{128}	2^{128}	2^{64}	None
	256	256	256		–	–	–	
LCAHASH1.1 [79]	128	128	128	Cellular Automata	2^{128}	2^{128}	2^{64}	None
	256	256	256		–	–	–	

‡ : order by year proposed

TABLE 5. Performance of Hardware and Software Implementations of LWCHF algorithms.

Algorithm	Hash value	Rate	Internal State	Hardware					Software				
				Technology	Hardware area (GE)	Throughput (Kb/s @100 kHz)	Power (μW) or Energy (μJ)	Latency (cycles/block)	ROM (byte)	RAM (byte)	Energy (μJ/bit)	Throughput @4 MHz (Kbps)	Latency (cycles/block)
Merkle-Damgård Construction:													
ARMADILLO and ARMADILLO2 [44]	80	48	256	180 nm	4030/2923	1090/272	77/44	44/176	–	–	–	–	–
	128	64	384		6025/4353	1000/250	118/65	64/256	–	–	–	–	–
	160	80	480		7492/5406	1000/250	158/83	80/320	–	–	–	–	–
	192	96	576		8999/6554	1000/250	183/102	96/384	–	–	–	–	–
	256	128	768		11914/8653	1000/250	251/137	128/512	–	–	–	–	–
Al-Odat et al. LWCHF [52]	160	512	512	–	–	–	–	–	–	–	–	–	–
	224	512	512		–	–	35,4 μJ	–	–	78 MB			3540
	256	512	512		–	–	–	–	–	–	–	–	–
	384	512	512		–	–	–	–	–	–	–	–	–
	512	512	512		–	–	–	–	–	–	–	–	–
El Hanouti et al. LWCHF [73]	128	1024	1024	–	–	–	–	–	–	–	–	–	–
Block Cipher-Based Construction:													
DM-PRESENT [45], [46]	64	80	64	180 nm	2213/1600	242.42/14.63	6.28/1.83	–	–	–	–	–	–
	64	128	64	180 nm	2530/1886	387.88/22.9	7.49/2.94	–	–	–	–	–	–
H-PRESENT [45], [46]	128	128/8	128	180 nm	4256/2330	200/11.45	–	–	–	–	–	–	–
C-PRESENT [45], [46]	192	64	192	180 nm	8048/4600 (estimate)	59,26/1,9	–	–	–	–	–	–	–
Lesamnta-LW [41]	256	128	256	90 nm	8240	125,550/20,000 (30 MHz)	–	–	–	50	–	–	66434 (8 bits)

Continued on the next page

TABLE 5. Continued from previous page

Algorithm	Hash value	Rate	Internal State	Hardware					Software				
				Technology	Hardware area (GE)	Throughput (Kb/s @100 kHz)	Power (μW) or Energy (μJ)	Latency (cycles/block)	ROM (byte)	RAM (byte)	Energy (μJ/bit)	Throughput @4 MHz (Kbps)	Latency (cycles/block)
TWISH [57]	128	128	128	–	–	–	–	–	–	–	–	–	–
Sponge Construction:													
QUARK [47]	136	8	136	180 nm	1379/2392	1.47/11.76	2.44/4.07	–	–	–	–	–	–
	176	16	176		1702/2819	2.27/18.18	3.10/4.76	–	–	–	–	–	–
	256	32	256		2296/4640	3.13/50.0	4.35/8.39	–	–	–	–	–	–
PHOTON [50]	80	20/16	100	180 nm	865/1168	2.82/15.15	–	–	–	–	–	–	95
	128	16	144		1122/1708	1.61/10.26	–	–	–	–	–	–	156
	160	36	196		1396/2117	2.70/20	–	–	–	–	–	–	116
	224	32	256		1736/2786	1.86/15.69	–	–	–	–	–	–	227
	256	32	288		2177/4362	3.21/20.51	–	–	–	–	–	–	157
SPONGENT [53]	80	8	88	130 nm	738/1127	35.8/111.3	1.57/2.31	–	–	–	–	–	–
	128	8	136		1060/1687	0.34/11.43	2.20/3.58	–	–	–	–	–	–
	160	16	176		1329/2190	0.40/17.78	2.85/4.47	–	–	–	–	–	–
	224	16	240		1728/2903	0.22/13.33	3.73/5.97	–	–	–	–	–	–
	256	16	272		1950/3281	0.17/11.43	4.21/6.62	–	–	–	–	–	–
GLUON [48]	128	8	136	–	2071	12.12	–	66	–	–	–	–	17319
	160	16	176		2799.3	32	–	50	–	–	–	–	8523
	224	32	256		4724	58.18	–	55	–	–	–	–	1951
SPN-Hash [49]	128	256	128	180 nm	2777/4600	36.1/55.7	–	710/230	–	–	–	–	34
	256	512	256		4625/8500	35.8/111.3	–	1430/230	–	–	–	–	34
SipHash [49]	64	64	256	–	3700/13500	–	–	–	–	–	–	–	1.96/1.44
LHash [74], [75]	80	16	96	180 nm	817/989	2.40;1.44/ 29.63;17.78	–	–	–	–	–	–	139
	96	16	96		817/989	2.40;1.44/ 29.63;17.78	–	–	–	–	–	–	139
	128	128	16		1028/1200	1.81;22.22/ 1.21;14.81	–	–	–	–	–	–	156
	128	128	8		1028/1200	0.91;11.1/ 0.40;4.94	–	–	–	–	–	–	312
Neeva-hash [76]	256	32	256	–	–	–	–	–	–	–	–	–	–
Hash-One [78]	160	1	160	180 nm	(estimate) 1006/ 2130	–	–	–	–	–	–	–	–
Gimli-Hash [55], [56]	256	128	384	180 nm	2395	–	–	–	778/19218	44/45	–	–	1611/726
sLiSCP-hash [43], [58]	160	32/32	192	65 nm/	2271/2492	29.62/29.62	4.62/7.44	108/144	–	–	–	–	–
	192	64/64	256	130 nm	3019/ 3305	44.44/22.22	5.88/8.75	108/144	–	–	–	–	–
	192	64/32	256		3019/3305	22.22/22.22	5.88/8.75	108/144	–	–	–	–	–
sLiSCP-light-hash [42], [59]	160	32/32	192	65 nm/	1938/2051	44.44/44.44	3.97/5.05	72/96	–	–	–	–	–
	192	64/64	256	130 nm	2584/2714	66.67/66.67	4.77/7.27	72/96	–	–	–	–	–
	192	64/32	256		2584/2714	33.33/33.33	4.77/7.27	72/96	–	–	–	–	–
LNHash [80]	80	96	16	–	–	–	–	–	–	–	–	–	–
	96	96	16	–	927*	–	–	–	–	–	–	–	–
	128	128	16	–	1224*	–	–	–	–	–	–	–	10251
	128	128	8	–	1224*	–	–	–	–	–	–	–	10251
	160	176	16	–	1539*	–	–	–	–	–	–	–	12109
	160	176	16	–	1539*	–	–	–	–	–	–	–	12109
ACE (ACE-$\mathcal{H}$-256) [60]	256	64	320	65 nm	4250	–	–	–	–	330/330	3340/1390	4.96/11.91	–
				90 nm	3660	–	–	–	–	559/559	12550/5218	25.56/63.87	–
				130 nm	4350	–	–	–	–	–	–	–	–
ASCON-HASH [61]	256	64	320	–	2570	14000	15	–	13808	1312	–	–	–
KNOT-Hash [62], [63]	256	32	256	130 nm	3803	376	4.17	–	1000	40	–	–	115.97†
	256	128	384		5850	1280	6.38	–	1000	40	–	–	69.08 †
	384	48	384		5608	369	6.22	–	1000	40	–	–	244.01†
	512	64	512		7420	365	8.24	–	1000	40	–	–	231.29†
DryGASCON-Hash [64]	128	128	320	Xilinx Zynq-7000 FPGA	–	–	–	65	–	–	–	–	–
	256	128	576	–	–	–	–	–	–	–	–	–	–
ORANGISH [65]	128	–	128	–	–	–	–	–	–	–	–	–	–
PHOTON-Beetle-Hash [66]	128	32	128	180	1736	–	–	–	2982	54	–	–	406.30
ESCH [68]	256	128	384	8-bit AVR ATmega128	–	578 (cycles/byte)	–	–	–	–	–	–	1978/559
	384	128	512	–	–	–	–	–	–	–	–	–	2992/830

Continued on the next page

TABLE 5. Continued from previous page

Algorithm	Hash value	Rate	Internal State	Hardware					Software				
				Technology	Hardware area (GE)	Throughput (Kb/s @100 kHz)	Power (μW) or Energy (μJ)	Latency (cycles/block)	ROM (byte)	RAM (byte)	Energy (μJ/bit)	Throughput @4 MHz (Kbps)	Latency (cycles/block)
Subterranean2.0-XOF [69], [70]	256	–	257	45 nm	2452	–	–	–	–	680/248	–	–	–
Xoodyak hash mode [71]	256	user	384	40 nm	8097	0,75 Gb/s	–	–	–	–	–	–	170.7/79.9
HVH [72]	88	88	8	?	857/1129	2.02/44.44	–	–	–	–	–	–	4339
	128	128	8		1145/1537	1.31/44.44	–	–	–	–	–	–	7023
	160	160	16		1385/1876	2.02/88.89	–	–	–	–	–	–	4708
	224	224	16		1769/2420	1.48/88.89	–	–	–	–	–	–	6869
	256	256	32		2009/2680	2.54/177.78	–	–	–	–	–	–	4939
LNMNT Hash [81], [82]	80	?	?	–	–	–	5.52	–	–	–	–	–	51180
	128	?	?	–	–	–	6.57	–	–	–	–	–	52042
	160	?	?	–	–	–	6.68	–	–	–	–	–	52742
	224	?	?	–	–	–	6.82	–	–	–	–	–	55260
Cellular Automata:													
L-CAHASH [77]	128	128	128	–	–	–	–	–	–	–	–	–	324
	256	256	256	–	–	–	–	–	–	–	–	–	374
LCAHASH1.1 [79]	128	128	128	–	–	–	–	–	–	–	–	–	995
	256	256	256	–	–	–	–	–	–	–	–	–	2844

*: estimate
†: long message

REFERENCES

[1] K. L. Lueth, "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time," 2020. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

[2] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A Survey of Lightweight-Cryptography Implementations," IEEE Design & Test of Computers, vol. 24, no. 6, pp. 522–533, 2007. [Online]. Available: http://www.computer.org/csdl/mags/dt/2007/06/mdt2007060522.html

[3] CRYPTREC, "Cryptrec cryptographic technology guideline - lightweight cryptography - (english version)," https://www.cryptrec.go.jp/report/cryptrec-gl-2003-2016en.pdf, March 2017.

[4] G. Gong, "Securing Internet-of-Things," in International Symposium on Foundations and Practice of Security. Springer, 2018, pp. 3–16.

[5] L. Zhou, C. Su, and K.-H. Yeh, "A Lightweight Cryptographic Protocol with Certificateless Signature for the Internet of Things," ACM Transactions on Embedded Computing Systems, vol. 18, no. 3, pp. 1–10, 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. 85 627–85 644, 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. 67 555–67 571, 2020.

[8] R. Kalaria, A. S. M. Kayes, W. Rahayu, and E. Pardede, "A Secure Mutual authentication approach to fog computing environment," Computers & Security, vol. 111, p. 102483, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003072

[9] A. Amiruddin, A. A. P. Ratna, and R. F. Sari, "Systematic review of internet of things security," International Journal of Communication Networks and Information Security, vol. 11, no. 2, pp. 248–255, 08 2019, copyright - Copyright Kohat University of Science and Technology (KUST) Aug 2019; Last updated - 2020-01-06. [Online]. Available: https://www.proquest.com/scholarly-journals/systematic-review-internet-things-security/docview/2333652943/se-2?accountid=17242

[10] K.-L. Tsai, F.-Y. Leu, I. You, S.-W. Chang, S.-J. Hu, and H. Park, "Low-power aes data encryption architecture for a lorawan," IEEE Access, vol. 7, pp. 146 348–146 357, 2019.

[11] K.-L. Tsai, Y.-L. Huang, F.-Y. Leu, I. You, Y.-L. Huang, and C.-H. Tsai, "Aes-128 based secure low power communication for lorawan iot environments," IEEE Access, vol. 6, pp. 45 325–45 334, 2018.

[12] 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. 54 631–54 640, 2020.

[13] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," pp. 1–9, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[14] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," IEEE Access, vol. 4, pp. 2292–2303, 2016.

[15] O. Novo, "Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1184–1195, 2018.

[16] L. Wang, X. Shen, J. Li, J. Shao, and Y. Yang, "Cryptographic primitives in blockchains," Journal of Network and Computer Applications, vol. 127, pp. 43–58, 2019.

[17] F. H. Pohrmen and G. Saha, "LightBC: A Lightweight Hash-Based Blockchain for the Secured Internet of Things," in International Conference on Innovative Computing and Communications, D. Gupta, A. Khanna, S. Bhattacharyya, A. E. Hassanien, S. Anand, and A. Jaiswal, Eds. Singapore: Springer Singapore, 2021, pp. 811–819.

[18] A. Biryukov and L. Perrin, "State of the Art in Lightweight Symmetric Cryptography," Cryptology ePrint Archive, Report 2017/511, pp. 1–55, 2017. [Online]. Available: https://eprint.iacr.org/2017/511.pdf

[19] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, 1st ed. Boca Raton, FL, USA.: CRC press, 1997.

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

[21] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," in Fast Software Encryption. FSE 2004. Lecture Notes in Computer Science, vol. 3017, 2004, pp. 371–388.

[22] G. Bertoni, J. Daemen, M. Peeters, and G. van Assche, "Keccak," in Annual International Conference on the Theory and Applications of Cryptographic Techniques: EUROCRYPT 2013, vol. 7881. Springer, 2013, pp. 313–314.

[23] NIST, "Fips pub 202 sha-3 standard : Permutation-based hash and extendable output functions," 2015.

[24] J. Kelsey, S.-j. Chang, and R. Perlner, "SHA-3 Derived Functions : cSHAKE, KMAC, TupleHash and ParallelHash," p. 185, 2016. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf

[25] H. Snyder, "Literature review as a research methodology: An overview and guidelines," Journal of Business Research, vol. 104, pp. 333–339, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0148296319304564

[26] A. Shah and M. Engineer, "A Survey of Lightweight Cryptographic Algorithms for IoT-Based Applications," in Smart Innovations in Com-

munication and Computational Sciences, S. Tiwari, M. C. Trivedi, K. K. Mishra, A. K. Misra, and K. K. Kumar, Eds. Singapore: Springer Singapore, 2019, pp. 283–293.

[27] S. S. Dhanda, B. Singh, and P. Jindal, "Lightweight Cryptography: A Solution to Secure IoT," Wireless Personal Communications, vol. 112, no. 3, pp. 1947–1980, 2020. [Online]. Available: https://doi.org/10.1007/s11277-020-07134-3

[28] M. Rana, Q. Mamun, and R. Islam, "Lightweight cryptography in IoT networks: A survey," Future Generation Computer Systems, vol. 129, pp. 77–89, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004404

[29] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," IEEE Access, vol. 9, pp. 28 177–28 193, 2021.

[30] E. B. Kavun and T. Yalçin, "A lightweight implementation of keccak hash function for radio-frequency identification applications," in RFIDSec, 2010.

[31] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," Submission to NIST (Round 2), vol. 3, no. 30, pp. 320–337, 2009.

[32] B. Schneier, "NIST Hash Workshop Live-blogging (5)," 2005. [Online]. Available: https://www.schneier.com/blog/archives/2005/11/nist_hash_works_4.html

[33] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," Journal of Cryptographic Engineering, vol. 8, pp. 141–184, 2017.

[34] ISO, "Iso/iec 29192-1:2012(en) information technology — security techniques — lightweight cryptography — part 1: General," https://www.iso.org/standard/56425.html, 2012.

[35] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert, "Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint," in CHES, 2012.

[36] M. Alizadeh, W. H. Hassan, M. Zamani, S. Karamizadeh, and E. Ghazizadeh, "Implementation and evaluation of lightweight encryption algorithms suitable for rfid," Journal of Next Generation Information Technology, vol. 4, pp. 65–77, 2013.

[37] B. Aslan, F. Y. Aslan, and M. T. Sakalli, "Energy consumption analysis of lightweight cryptographic algorithms that can be used in the security of internet of things applications," Secur. Commun. Networks, vol. 2020, pp. 8 837 671:1–8 837 671:15, 2020.

[38] A. Caforio, F. Balli, S. Banik, and F. Regazzoni, "A deeper look at the energy consumption of lightweight block ciphers," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 170–175, 2021.

[39] NIST, "Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process," 2018. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

[40] C. Pei, Y. Xiao, W. Liang, and X. Han, "Trade-off of security and performance of lightweight block ciphers in industrial wireless sensor networks," EURASIP Journal on Wireless Communications and Networking, vol. 2018, pp. 1–18, 2018.

[41] 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, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 151–168.

[42] R. AlTawy, R. Rohit, M. He, K. Mandal, G. Yang, and G. Gong, "sLiSCP-light: Towards Hardware Optimized Sponge-specific Cryptographic Permutations," ACM Trans. Embed. Comput. Syst., vol. 17, no. 4, pp. 1–26, 2018.

[43] ——, "Towards a Cryptographic Minimal Design: The sLiSCP Family of Permutations," IEEE Transactions on Computers, vol. 67, no. 9, pp. 1341–1358, 2018.

[44] S. Badel, N. Dagtekin, J. Nakahara, K. Ouafi, N. Reffé, P. Sepehrdad, P. Susil, and S. Vaudenay, "ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware," in Cryptographic Hardware and Embedded Systems, CHES 2010, vol. 6225. Springer, 2010, pp. 398–412.

[45] 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, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 283–299.

[46] A. Y. Poschmann, "Lightweight cryptography - cryptographic engineering for a pervasive world," Cryptology ePrint Archive, Paper 2009/516, 2009. [Online]. Available: https://eprint.iacr.org/2009/516

[47] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A Lightweight Hash," pp. 1–15, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00145-012-9125-6

[48] T. P. Berger, J. D'Hayer, K. Marquet, M. Minier, and G. Thomas, "The GLUON Family: A Lightweight Hash Function Family Based on FCSRs," in Progress in Cryptology - AFRICACRYPT 2012, A. Mitrokotsa and S. Vaudenay, Eds. Berlin, Heidelberg: Springer, 2012, pp. 306–323.

[49] J.-P. Aumasson and D. J. Bernstein, "SipHash: A Fast Short-Input PRF BT," in Progress in Cryptology-INDOCRYPT 2012, S. Galbraith and M. Nandi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 489–508.

[50] J. Guo, T. Peyrin, and A. Poschmann, "The PHOTON Family of Lightweight Hash Functions," in Advances in Cryptology – CRYPTO 2011. CRYPTO 2011. Lecture Notes in Computer Science, vol 6841, vol. 6841. Springer, 2011.

[51] J. Choy, H. Yap, K. Khoo, J. Guo, T. Peyrin, A. Poschmann, C. H. Tan, A. Mitrokotsa, and S. Vaudenay, "SPN-Hash: Improving the Provable Resistance against Differential Collision Attacks," in Progress in Cryptology-AFRICACRYPT 2012. AFRICACRYPT 2012. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 270–286.

[52] Z. A. Al-Odat, E. M. Al-Qtiemat, and S. U. Khan, "An Efficient Lightweight Cryptography Hash Function for Big Data and IoT Applications," in 2020 IEEE Cloud Summit, 2020, pp. 66–71.

[53] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: A Lightweight Hash Function," in Cryptographic Hardware and Embedded Systems-CHES 2011. CHES 2011. Lecture Notes in Computer Science, vol 6917, vol. 6917. Springer, 2011, pp. 312–325.

[54] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: The design space of lightweight cryptographic hashing," IEEE Transactions on Computers, vol. 62, no. 10, pp. 2041–2053, 2013.

[55] D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F.-X. Standaert, and Y. Todo, "Gimli: a cross-platform permutation," in International Conference on Cryptographic Hardware and Embedded Systems. Springer, 2017, pp. 299–320.

[56] ——, "Gimli 20190927," 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/gimli-spec-round2.pdf

[57] D. I. Afryansyah, Magfirawaty, and K. Ramli, "The Development and Analysis of TWISH: A Lightweight-Block-Cipher-TWINE-Based Hash Function," in 2018 Thirteenth International Conference on Digital Information Management (ICDIM), 2018, pp. 210–215.

[58] R. AlTawy, R. Rohit, M. He, K. Mandal, G. Yang, and G. Gong, "sliscp: Simeck-based permutations for lightweight sponge cryptographic primitives," in Selected Areas in Cryptography – SAC 2017, C. Adams and J. Camenisch, Eds. Cham: Springer International Publishing, 2018, pp. 129–150.

[59] ——, "sLiSCP-light: Towards Lighter Sponge-specific Cryptographic Permutations," 2017. [Online]. Available: http://cacr.uwaterloo.ca/techreports/2017/cacr2017-04.pdf

[60] M. Aagaard, R. AlTawy, G. Gong, K. Mandal, and R. Rohit, "ACE: An authenticated encryption and hash algorithm," Submission to NIST-LWC, 2019.

[61] C. Dobraunig, F. Mendel, M. Eichlseder, and M. Schläffer, "Ascon v1.2 Submission to NIST," p. 52, 2021. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf

[62] W. Zhang, T. Ding, B. Yang, Z. Bao, Z. Xiang, F. Ji, and X. Zhao, "KNOT: Algorithm Speci cations and Supporting Document," 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/knot-spec-round.pdf

[63] ——, "Update on Security Analysis and Implementations of KNOT," 2020. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/status-update-sep2020/KNOT_Update.pdf

[64] S. Riou, "DryGASCON Lightweight Cryptography Standardization Process round 1 submission," 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/drygascon-spec-round2.pdf

[65] B. Chakraborty and M. Nandi, "ORANGE," 2019. [Online]. Available: https://www.isical.ac.in/ lightweight/Orange/

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

[67] C. Beierle, A. Biryukov, L. Cardoso dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, and Q. Wang, "Lightweight AEAD and Hashing using the Sparkle Permutation Family," IACR Transactions on Symmetric Cryptology, vol. 2020, no. S1 SE - Articles, pp. 208–261, Jun 2020. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8627

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

[69] J. Daemen, P. M. C. Massolino, and Y. Rotella, "The Subterranean 2.0 cipher suite," 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/subterranean-spec-round2.pdf

[70] J. Daemen, P. M. C. Massolino, A. Mehrdad, and Y. Rotella, "The subterranean 2.0 cipher suite," IACR Transactions on Symmetric Cryptology, vol. 2020, no. S1, p. 262–294, Jun. 2020. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8622

[71] J. Daemen, S. Hoffert, S. Mella, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodyak, a lightweight cryptographic scheme," 2021. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/xoodyak-spec-final.pdf

[72] Y. Huang, S. Li, W. Sun, X. Dai, and W. Zhu, "Hvh: A lightweight hash function based on dual pseudo-random transformation," in Security, Privacy, and Anonymity in Computation, Communication, and Storage, G. Wang, B. Chen, W. Li, R. Di Pietro, X. Yan, and H. Han, Eds. Cham: Springer International Publishing, 2021, pp. 492–505.

[73] I. El Hanouti, H. El Fadili, S. Hraoui, and A. Jarjar, "A Lightweight Hash Function for Cryptographic and Pseudo-Cryptographic Applications," in WITS 2020, S. Bennani, Y. Lakhrissi, G. Khaissidi, A. Mansouri, and Y. Khamlichi, Eds. Singapore: Springer Singapore, 2022, pp. 495–505.

[74] W. Wu, S. Wu, L. Zhang, J. Zou, and L. Dong, "LHash: A Lightweight Hash Function (Full Version)," IACR Cryptol. ePrint Arch., vol. 2013, p. 867, 2013.

[75] ——, "LHash: A Lightweight Hash Function," in Information Security and Cryptology. Inscrypt 2013. Lecture Notes in Computer Science, vol 8567., L. D., X. S., and Y. M., Eds., vol. 8567. Cham: Springer, 2014, pp. 291–308.

[76] K. Bussi, D. Dey, M. Kumar, and B. K. Dass, "Neeva: A lightweight hash function," Cryptology ePrint Archive, Report 2016/042, 2016, https://ia.cr/2016/042.

[77] C. Hanin, B. Echandouri, F. Omary, and S. El Bernoussi, "L-CAHASH: A Novel Lightweight Hash Function Based on Cellular Automata for RFID," in Ubiquitous Networking, E. Sabir, A. García Armada, M. Ghogho, and M. Debbah, Eds. Cham: Springer International Publishing, 2017, pp. 287–298.

[78] P. Megha Mukundan, S. Manayankath, C. Srinivasan, and M. Sethumadhavan, "Hash-One: a lightweight cryptographic hash function," IET Information Security, vol. 10, no. 5, pp. 225–231, 2016.

[79] A. Sadak, B. Echandouri, F. Ezzahra, C. Hanin, and F. Omary, "Lcahash-1.1: A new design of the lcahash system for iot," International Journal of Advanced Computer Science and Applications, 2019.

[80] X. Zhang, Q. Xu, X. Li, and C. Wang, "A Lightweight Hash Function Based on Cellular Automata for Mobile Network," in 2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN), 2019, pp. 247–252.

[81] N. Nabeel, M. H. Habaebi, and M. D. R. Islam, "Security Analysis of LNMNT-LightWeight Crypto Hash Function for IoT," IEEE Access, vol. 9, pp. 165 754–165 765, 2021.

[82] N. Nabeel, M. H. Habaebi, and M. R. Islam, "Lnmnt-new mersenne number based lightweight crypto hash function for iot," 2021 8th International Conference on Computer and Communication Engineering (ICCCE), pp. 68–71, 2021.

[83] R. C. Merkle, "One Way Hash Functions and DES," in Advances in Cryptology — CRYPTO' 89 Proceedings, G. Brassard, Ed. New York, NY: Springer New York, 1990, pp. 428–446.

[84] I. B. Damgård, "A Design Principle for Hash Functions," in Advances in Cryptology-CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. New York, NY: Springer New York, 1989, pp. 416–427. [Online]. Available: http://link.springer.com/10.1007/0-387-34805-0_39

[85] T. Duong and J. Rizzo, "Flickr's api signature forgery vulnerability," 2009. [Online]. Available: https://packetstormsecurity.com/files/81729/flickr_api_signature_forgery.pdf

[86] Z. Al-Odat and S. Khan, "Constructions and attacks on hash functions," in 2019 International Conference on Computational Science and Computational Intelligence (CSCI). Los Alamitos, CA, USA: IEEE Computer Society, dec 2019, pp. 139–144. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CSCI49370.2019.00030

[87] B. O. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas Jr, C. H. Meyer, J. Oseas, S. Pilpel, and M. Schilling, "Data authentication using modification detection codes based on a public one way encryption function," Mar. 13 1990, uS Patent 4,908,861.

[88] NBS, "Data Encryption Standard," in In FIPS PUB 46, Federal Information Processing Standards Publication, 1977, pp. 42–46.

[89] B. Preneel, "Davies-meyer hash function," in Encyclopedia of Cryptography and Security, 2005.

[90] G. Bertoni, J. Daemen, M. Peeters, and G. van Assche, "Sponge Functions," pp. 1–22, 2007. [Online]. Available: https://keccak.team/files/SpongeFunctions.pdf

[91] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The keccak reference," pp. 1–14, 2011.

[92] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the indifferentiability of the sponge construction," in Advances in Cryptology – EUROCRYPT 2008, N. Smart, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 181–197.

[93] Y. Li, G. Ge, and D. Xia, "Chaotic hash function based on the dynamic s-box with variable parameters," Nonlinear Dynamics, vol. 84, pp. 2387–2402, 2016.

[94] M. Alawida, A. Samsudin, N. Alajarmeh, J. S. Teh, M. Ahmad, and W. H. Alshoura, "A Novel Hash Function Based on a Chaotic Sponge and DNA Sequence," IEEE Access, vol. 9, pp. 17 882–17 897, 2021.

[95] J. S. Teh, K. Tan, and M. Alawida, "A chaos-based keyed hash function based on fixed point representation," Cluster Computing, vol. 22, no. 2, pp. 649–660, 2019. [Online]. Available: https://doi.org/10.1007/s10586-018-2870-z

[96] N. Abdoun, S. El Assad, T. Manh Hoang, O. Deforges, R. Assaf, and M. Khalil, "Designing Two Secure Keyed Hash Functions Based on Sponge Construction and the Chaotic Neural Network," Entropy, vol. 22, no. 9, p. 1012, sep 2020. [Online]. Available: https://www.mdpi.com/1099-4300/22/9/1012

[97] J. S. Teh, M. Alawida, and J. J. Ho, "Unkeyed hash function based on chaotic sponge construction and fixed-point arithmetic," Nonlinear Dynamics, vol. 100, no. 1, pp. 713–729, 2020. [Online]. Available: https://doi.org/10.1007/s11071-020-05504-x

[98] S. Hirose, "Some Plausible Constructions of Double-Block-Length Hash Functions," in International Workshop on Fast Software Encryption FSE 2006, vol. 4047. Springer, 2006, pp. 210–225. [Online]. Available: http://www.iacr.org/cryptodb/archive/2006/FSE/3233/3233.pdf

[99] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "$$\$textnormal ${$$\$textsc ${$TWINE$}$$}$ $: A Lightweight Block Cipher for Multiple Platforms," in International Conference on Selected Areas in Cryptography. Springer, 2012, pp. 339–354.

[100] A. Doganaksoy, B. Ege, O. KoÃ§ak, and F. Sulak, "Cryptographic Randomness Testing of Block Ciphers and Hash Functions," Turkey, p. 564, 2010.

[101] M. Hell and T. Johansson, "Breaking the F-FCSR-H Stream Cipher in Real Time," in ASIACRYPT 2008, vol. 5350. Springer, 2008, pp. 557–569.

[102] C. De Cannière, O. Dunkelman, and M. Knežević, "Katan and ktantan — a family of small and efficient hardware-oriented block ciphers," in Cryptographic Hardware and Embedded Systems - CHES 2009, C. Clavier and K. Gaj, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 272–288.
[103] F. A. Thierry Berger and C. Lauradoux, "F-FCSR (Phase 3 Profile 2)," 2008. [Online]. Available: https://www.ecrypt.eu.org/stream/p3ciphers/ffcsr/ffcsr_p3.pdf
[104] F. Arnault, T. P. Berger, C. Lauradoux, and M. Minier, "X-fcsr: a new software oriented stream cipher based upon fcsrs," 2007, this paper was accepted as a short paper at Indocrypt 2007 marine.minier@insa-lyon.fr 13782 received 25 Sep 2007, last revised 26 Sep 2007. [Online]. Available: http://eprint.iacr.org/2007/380
[105] H. Wu, "The hash function JH," 2011. [Online]. Available: https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf
[106] NIST, "Announcing the advanced encryption standard (aes)," 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[107] J.-P. Aumasson, W. Meier, R. Phan, and L. Henzen, The Hash Function BLAKE. Springer Publishing Company, Incorporated, 2014.
[108] N. Ferguson, "The Skein Hash Function Family," Argument, vol. 30, no. 4, p. 79, 2010. [Online]. Available: http://www.schneier.com/skein.html
[109] G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong, "The simeck family of lightweight block ciphers," in Cryptographic Hardware and Embedded Systems – CHES 2015, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 307–329.
[110] ——, "The simeck family of lightweight block ciphers," Cryptology ePrint Archive, Paper 2015/612, 2015, https://eprint.iacr.org/2015/612. [Online]. Available: https://eprint.iacr.org/2015/612
[111] D. J. Bernstein, "Caesar: Competition for authenticated encryption: Security, applicability, and robustness," 02 2019. [Online]. Available: https://competitions.cr.yp.to/caesar.html
[112] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. M. R. Verbauwhede, "Rectangle: A bit-slice ultra-lightweight block cipher suitable for multiple platforms," IACR Cryptol. ePrint Arch., vol. 2014, p. 84, 2014.
[113] ——, "Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms," Science China Information Sciences, vol. 58, pp. 1–15, 2015.
[114] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2," Submission to the CAESAR Competition, 2016.
[115] 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, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 320–337.
[116] NIST, "Federal Register::Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family," 2007. [Online]. Available: https://www.federalregister.gov/documents/2007/11/02/E7-21581/announcing-request-for-candidate-algorithm-nominations-for-a-new-cryptographic-hash-algorithm-sha-3
[117] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda, "Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers," IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, no. 2 SE - Articles, pp. 218–241, may 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/881
[118] L. Claesen, J. Daemen, M. Genoe, and G. Peeters, "Subterranean: A 600 Mbit/sec cryptographic VLSI chip," in Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93, 1993, pp. 610–613.
[119] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer, "The design of Xoodoo and Xoofff," IACR Transactions on Symmetric Cryptology, vol. 2018, no. 4 SE - Articles, pp. 1–38, dec 2018. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/7359
[120] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodoo cookbook," Cryptology ePrint Archive, Report 2018/767, 2018. [Online]. Available: https://eprint.iacr.org/2018/767.pdf
[121] D. Xuejun, H. Yuhua, C. Lu, T. Lu, and S. Fei, "VH: A Lightweight Block Cipher Based on Dual Pseudo-random Transformation," in IEEE CLOUD 2015, 2015.
[122] G. Marsaglia, "The marsaglia random number cdrom including the diehard battery of tests of randomness," Jan 2016. [Online]. Available: https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/
[123] E. Biham and A. Shamir, Differential Cryptanalysis of the Data Encryption Standard. Berlin, Heidelberg: Springer-Verlag, 1993.
[124] ——, "Differential Cryptanalysis of the Full 16-Round DES," in CRYPTO 1992, vol. 740. Springer, 1992, pp. 487–496.
[125] ——, "Differential Cryptanalysis of DES-like Cryptosystems," in CRYPTO 1990, vol. 537. Springer, 1990, pp. 2–21.
[126] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," J. Cryptology, vol. 7, pp. 229–246, 1994.
[127] M. J. Wiener, "The Full Cost of Cryptanalytic Attacks," J. Cryptology, vol. 17, pp. 105–124, 2004.
[128] N. Bagheri, N. Ghaedi, and S. K. Sanadhya, "Differential fault analysis of sha-3," in INDOCRYPT, 2015.
[129] R. Altawy and A. M. Youssef, "Differential fault analysis of streebog," in ISPEC, 2015.
[130] M. Safkhani and M. a. Arghavani, "A survey of cube, differential fault analysis attacks and linear structures on keccak hash function (sha-3)," 2, vol. 5, no. 2, 2017. [Online]. Available: http://monadi.isc.org.ir/article-1-76-en.html
[131] P. Luo, Y. Fei, L. Zhang, and A. A. Ding, "Differential fault analysis of sha-3 under relaxed fault models," Journal of Hardware and Systems Security, vol. 1, pp. 156–172, 2017.
[132] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in EUROCRYPT 1993, vol. 765. Springer, 1993, pp. 386–397.
[133] ——, "The First Experimental Cryptanalysis of the Data Encryption Standard," in CRYPTO 1994, vol. 839. Springer, 1994, pp. 1–11.
[134] J. Daemen, L. Knudsen, and V. Rijmen, "The Block Cipher SQUARE," Lecture Notes in Computer Science, vol. 1267, pp. 149–165, 1997.
[135] S. Lucks, "Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys," 2000.
[136] L. Knudsen and D. Wagner, "Integral cryptanalysis (Extended Abstract)," Fast Software Encryption, pp. 112–127, 2002. [Online]. Available: http://link.springer.com/10.1007/3-540-45661-9_9 http://link.springer.com/chapter/10.1007/3-540-45661-9_9
[137] M. Matsui, "New Block Encryption Algorithm MISTY," in FSE 1997, vol. 1267. Springer, 1997, pp. 54–68.
[138] ISO, "Iso/iec 29192-5:2016(en) information technology — security techniques — lightweight cryptography — part 5: Hash-functions," https://www.iso.org/standard/56425.html, 2016.
[139] J.-P. Kaps, W. Diehl, M. Tempelmeier, F. Farahmand, E. Homsirikamol, and K. Gaj, "A comprehensive framework for fair and efficient benchmarking of hardware implementations of lightweight cryptography," IACR Cryptol. ePrint Arch., vol. 2019, p. 1273, 2019.
[140] M. O. A. Al-Shatari, F. A. Hussin, A. A. Aziz, G. Witjaksono, and X.-T. Tran, "FPGA-Based Lightweight Hardware Architecture of the PHOTON Hash Function for IoT Edge Devices," IEEE Access, vol. 8, pp. 207 610–207 618, 2020.
[141] NIST, "Benchmarking of lightweight cryptographic algorithms on microcontrollers," GitHub Repository. https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking, 2020.
[142] S. Renner, E. Pozzobon, and J. Mottok, "Lwc benchmark," GitHub repository. https://lab. las3.de/gitlab/lwc/compare., 2021.
[143] C. Beierle, A. Biryukov, L. Cardoso dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, and Q. Wang, "Alzette: A 64-bit arx-box," in Advances in Cryptology – CRYPTO 2020, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 419–448.
[144] A. A. Moldovyan and N. A. Moldovyan, "A cipher based on data-dependent permutations," Journal of Cryptology, vol. 15, pp. 61–72, 2001.
[145] M. Naya-Plasencia and T. Peyrin, "Practical cryptanalysis of armadillo2," in Fast Software Encryption. FSE 2012. Lecture Notes in Computer Science, vol 7549, 2012.
[146] M. A. Abdelraheem, C. Blondeau, M. Naya-Plasencia, M. Videau, and E. Zenner, "Cryptanalysis of armadillo 2," in Advances in Cryptology-ASIACRYPT 2011. ASIACRYPT 2011. Lecture Notes in Computer Science, vol 7073, 2011.
[147] T. Koyama, Y. Sasaki, and N. Kunihiro, "Multi-differential Cryptanalysis on Reduced DM-PRESENT-80: Collisions and Other Differential Properties," in ICISC, 2012.
[148] C. Blondeau, T. Peyrin, and L. Wang, "Known-key distinguisher on full present," IACR Cryptol. ePrint Arch., vol. 2015, p. 575, 2015.
[149] Y. Sasaki and K. Aoki, "Improved integral analysis on tweaked lesamnta," in ICISC, 2011.

[150] R. Shiba, K. Sakamoto, F. Liu, K. Minematsu, and T. Isobe, "Integral and impossible-differential attacks on the reduced-round lesamnta-lw-bc," IET Information Security, 2021.

[151] K. Zhang, J. Guan, and X. Fei, "Improved conditional differential cryptanalysis," Secur. Commun. Networks, vol. 8, pp. 1801–1811, 2015.

[152] J. Yang, M. Liu, D. Lin, and W. Wang, "Symbolic-like computation and conditional differential cryptanalysis of quark," in IWSEC, 2018.

[153] C. Lu, Y. Lin, S. Jen, and J. Yang, "Cryptanalysis on PHOTON hash function using cube attack," in 2012 International Conference on Information Security and Intelligent Control, 2012, pp. 278–281.

[154] M. Walter, "Algebraic methods in analyzing lightweight cryptographic symmetric primitives," 2012.

[155] M. A. Abdelraheem, "Estimating the probabilities of low-weight differential and linear approximations on present-like ciphers," in Information Security and Cryptology – ICISC 2012, T. Kwon, M.-K. Lee, and D. Kwon, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 368–382.

[156] S. Fan and M. Duan, "Improved Zero-Sum Distinguisher for SPONGENT-88," in Proceedings of the 2015 International Conference on Electromechanical Control Technology and Transportation. Atlantis Press, 2015/11, pp. 582–587. [Online]. Available: https://doi.org/10.2991/icectt-15.2015.111

[157] L. Sun, W. Wang, and M. Wang, "Milp-aided bit-based division property for primitives with non-bit-permutation linear layers," IACR Cryptol. ePrint Arch., vol. 2016, p. 811, 2016.

[158] L. Perrin and D. Khovratovich, "Collision spectrum, entropy loss, t-sponges, and cryptanalysis of gluon-64," in Fast Software Encryption, C. Cid and C. Rechberger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 82–103.

[159] C. Dobraunig, F. Mendel, and M. Schläffer, "Differential Cryptanalysis of SipHash," in Selected Areas in Cryptography – SAC 2014, A. Joux and A. Youssef, Eds. Cham: Springer International Publishing, 2014, pp. 165–182.

[160] W. Xin, Y. Liu, B. Sun, and C. Li, "Improved cryptanalysis on siphash," in Cryptology and Network Security, Y. Mu, R. H. Deng, and X. Huang, Eds. Cham: Springer International Publishing, 2019, pp. 61–79.

[161] B. Hayat Susanti, M. Rakha Rafi Bayhaqi, and M. W. Ardyani, "Correcting block attack on the 32-bit reduced neeva," in 2020 1st International Conference on Information Technology, Advanced Mechanical and Electrical Engineering (ICITAMEE), Oct 2020, pp. 85–90.

[162] A. F. Gutiérrez, G. Leurent, M. Naya-Plasencia, L. Perrin, A. Schrottenloher, and F. Sibleyras, "New results on Gimli: full-permutation distinguishers and improved collisions," Cryptology ePrint Archive, Report 2020/744, 2020.

[163] A. Flórez Gutiérrez, G. Leurent, M. Naya-Plasencia, L. Perrin, A. Schrottenloher, and F. Sibleyras, "New Results on Gimli: Full-Permutation Distinguishers and Improved Collisions," in Advances in Cryptology – ASIACRYPT 2020, S. Moriai and H. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 33–63.

[164] F. Liu, T. Isobe, and W. Meier, "Exploiting weak diffusion of gimli: Improved distinguishers and preimage attacks," Cryptology ePrint Archive, Paper 2020/561, 2020, https://eprint.iacr.org/2020/561. [Online]. Available: https://eprint.iacr.org/2020/561

[165] ——, "Exploiting weak diffusion of gimli: Improved distinguishers and preimage attacks," IACR Transactions on Symmetric Cryptology, vol. 2021, no. 1, p. 185–216, Mar. 2021. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8837

[166] ——, "Preimages and collisions for up to 5-round gimli-hash using divide-and-conquer methods," Cryptology ePrint Archive, Report 2019/1080, 2019, https://ia.cr/2019/1080.

[167] ——, "Automatic Verification of Differential Characteristics: Application to Reduced Gimli," in Advances in Cryptology – CRYPTO 2020, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 219–248.

[168] Y. Liu, Y. Sasaki, L. Song, and G. Wang, "Cryptanalysis of reduced sliscp permutation in sponge-hash and duplex-AE modes," in International Conference on Selected Areas in Cryptography. Springer, 2018, pp. 92–114.

[169] L. Kraleva, R. Posteuca, and V. Rijmen, "Cryptanalysis of the permutation based algorithm spoc," in Progress in Cryptology – INDOCRYPT 2020, K. Bhargavan, E. Oswald, and M. Prabhakaran, Eds. Cham: Springer International Publishing, 2020, pp. 273–293.

[170] J. Liu, G. Liu, and L. Qu, "A new automatic tool searching for impossible differential of nist candidate ace," Mathematics, vol. 8, no. 9, p. 1576, Sep 2020. [Online]. Available: http://dx.doi.org/10.3390/math8091576

[171] R. Zong, X. Dong, and X. Wang, "Collision Attacks on Round-Reduced Gimli-Hash/Ascon-Xof/Ascon-Hash," Cryptology ePrint Archive, Report 2019/1115, 2019.

[172] C. Tezcan, "Analysis of ascon, drygascon, and shamash permutations," International Journal of Information Security Science, vol. 9, no. 3, pp. 172–187, 2020.

[173] K. Ramezanpour, A. Abdulgadir, W. Diehl, J.-P. Kaps, and P. Ampadu, "Active and passive side-channel key recovery attacks on ascon," 2020. [Online]. Available: https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/active-passive-recovery-attacks-ascon-lwc2020.pdf

[174] W. Zhang, T. Ding, C. Zhou, and F. Ji, "Security analysis of knot-aead and knot-hash," 2020. [Online]. Available: https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/security-analysis-of-KNOT-lwc2020.pdf

[175] C. Tezcan, "Analysis of ascon, drygascon, and shamash permutations," Cryptology ePrint Archive, Paper 2020/1458, 2020, https://eprint.iacr.org/2020/1458. [Online]. Available: https://eprint.iacr.org/2020/1458

[176] H. Liang, S. Mesnager, and M. Wang, "Cryptanalysis of the aead and hash algorithm drygascon," Cryptography and Communications, vol. 14, no. 3, pp. 597–625, 2022. [Online]. Available: https://doi.org/10.1007/s12095-021-00542-7

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

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

SURYADI SURYADI (Member, IndoMS) received a B.S. degree in mathematics from the Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Indonesia, 1990, a master's degree in informatics engineering from Institute Technology Bandung, Indonesia, 1998, and a Ph.D. degree from the Department of Electrical and Computer Engineering, Universitas Indonesia, Indonesia, in 2013. They have been a Lecturer (Associate Professor) in the Department of Mathematics, Faculty of Mathematics and Natural Sciences Universitas Indonesia, and the Department of Electrical Engineering, Universitas Indonesia. His research interests include information security, cryptography, and computational mathematics. He is an author and coauthor, has published over 30 papers in leading international journals and conferences, and has written two books and contributed to one book chapter.

BERNARDI PRANGGONO (Senior Member, IEEE) Dr. Bernardi Pranggono is currently a Senior Lecturer in the Department of Engineering and Mathematics, Sheffield Hallam University. Dr. Pranggono received his B. Eng. degree in electronics and telecommunication engineering from Waseda University, Japan, an M. DigComms degree in digital communications from Monash University, Australia, and a Ph.D. degree in electronics and electrical engineering from the University of Leeds, UK. He has previously held academic and research positions at 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 an associate editor of Frontiers of Computer Science and Frontiers in Communications and Networks. Dr. Pranggono is a Fellow of the Higher Education Academy (HEA) and a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE).

TEDDY SURYA GUNAWAN (Senior Member, IEEE) Teddy Surya Gunawan received his BEng degree in Electrical Engineering with cum laude award from Institut Teknologi Bandung (ITB), Indonesia in 1998. He obtained his M.Eng degree in 2001 from the School of Computer Engineering at Nanyang Technological University, Singapore, and PhD degree in 2007 from the School of Electrical Engineering and Telecommunications, The University of New South Wales, Australia. His research interests are in speech and audio processing, biomedical signal processing and instrumentation, image and video processing, and parallel computing. He is currently an IEEE Senior Member (since 2012), was chairman of IEEE Instrumentation and Measurement Society – Malaysia Section (2013 and 2014), Professor (since 2019), Head of Department (2015-2016) at Department of Electrical and Computer Engineering, and Head of Programme Accreditation and Quality Assurance for Faculty of Engineering (2017-2018), International Islamic University Malaysia. He is Chartered Engineer (IET, UK) and Insinyur Profesional Madya (PII, Indonesia) since 2016 (upgraded to Insinyur Profesional Utama since 2021), registered ASEAN engineer since 2018, and ASEAN Chartered Profesional Engineer since 2020.

• • •