electronics

MDPI

Article

Design and Performance Analysis of a SPECK-Based Lightweight Hash Function

Abdullah Sevin * and Ünal Çavuşoğlu

Department of Computer Engineering, Sakarya University, Serdivan 54050, Sakarya, Turkey; unalc@sakarya.edu.tr
* Correspondence: asevin@sakarya.edu.tr

Abstract: In recent years, hash algorithms have been used frequently in many areas, such as digital signature, blockchain, and IoT applications. Standard cryptographic hash functions, including traditional algorithms such as SHA-1 and MD5, are generally computationally intensive. A principal approach to improving the security and efficiency of hash algorithms is the integration of lightweight algorithms, which are designed to minimize computational overhead, into their architectural framework. This article proposes a new hash algorithm based on lightweight encryption. A new design for the lightweight hash function is proposed to improve its efficiency and meet security requirements. In particular, efficiency reduces computational load, energy consumption, and processing time for resource-constrained environments such as IoT devices. Security requirements focus on ensuring properties such as collision resistance, pre-image resistance, and distribution of modified bit numbers to ensure reliable performance while preserving the robustness of the algorithm. The proposed design incorporates the SPECK lightweight encryption algorithm to improve the structure of the algorithm, ensuring robust mixing and security through confusion and diffusion, while improving processing speed. Performance and efficiency tests were conducted to evaluate the proposed algorithm, and the results were compared with commonly used hash algorithms in the literature. The test results show that the new lightweight hash algorithm has successfully passed security tests, including collision resistance, pre-image resistance, sensitivity, and distribution of hash values, while outperforming other commonly used algorithms regarding execution time.

Keywords: information security; lightweight algorithm; security tests; performance analysis; hashing

Citation: Sevin, A.; Çavuşoğlu, Ü. Design and Performance Analysis of a SPECK-Based Lightweight Hash Function. *Electronics* **2024**, *13*, 4767. https://doi.org/10.3390/electronics13234767

Academic Editor: Aryya Gangopadhyay

Received: 13 October 2024
Revised: 25 November 2024
Accepted: 26 November 2024
Published: 2 December 2024

1. Introduction

Hash algorithms are crucial in cryptography and are widely used across various fields, such as digital signatures, message authentication, and password security. A hash algorithm is a method of compressing data of varying lengths into a fixed-size representation. This fixed-length message is described as a hash value. Hash functions are one-way, and it should not be possible to obtain the data over the obtained hash value. Block cipher algorithms can be used to construct hash algorithms or hash algorithms can be developed to perform hash operations. The hash algorithms can be divided into two different parts: keyed and keyless. The output size of hash algorithms differs according to the design. Hash algorithms are used in various security and data management applications such as digital signatures, encryption and password management, data integrity control, blockchain, and database indexing.

A cryptographic hash function collision occurs when two different input messages produce the same hash output. Since hash functions map arbitrary-sized input data to fixed-sized hash values, it is theoretically possible for different inputs to produce the same output, especially when the number of potential inputs increases while the output space remains limited. This directly violates the basic property of any cryptographic hash function known as collision resistance. It guarantees that finding any two different inputs hashing to the

same value will be computationally infeasible. A collision in the hash function breaks the integrity of that hash function. Thus, an attacker can easily manipulate data without being noticed— breaching security assurances like data authenticity and integrity [1].

The security of many cryptographic systems depends on the properties of hash functions, especially collision resistance. For instance, one of the reasons that the SHA-2 family, e.g., SHA-256, is widely used in blockchain technologies is that it is believed to be collision-resistant [2]. Finding a collision will undermine confidence in digital signatures, message authentication codes, and many other cryptographic protocols. Hash algorithms aim to enhance security and prevent collisions using cryptographic operations during the hashing process. These operations include complex transformations, permutations, and mathematical functions designed to make it difficult to reverse-engineer the input and ensure that even minor changes in the input produce significantly different hash values, reducing the risk of collisions. These algorithms usually contain many processing steps and rounds, designed to be secure by making it computationally hard to either invert the hash output or predict it. Each round, therefore, is a transformation and permutation applied to input data that scramble and obscure completely the original information.

The Internet of Things (IoT) is a technology that allows physical devices to interact with each other through data sharing over the Internet. It connects a range of "smart" devices, from home appliances to industrial machinery and vehicles, for automated data collection, monitoring, and management in real time [3]. Recent developments in the IoT have seen interconnected devices leap to an unprecedented level, enabling easy communication and data flow across different applications and industries. Rapid growth in IoT-enabled devices has opened new frontiers in automation, data-driven insights, and system optimization across healthcare, manufacturing, and urban infrastructure sectors. In the context of IoT, many devices are designed to be small, low-cost, and energy-efficient to ensure they can function effectively in various environments. Due to their limited processing capacity, these devices cannot handle resource-intensive operations, such as complex cryptographic algorithms or large data-processing tasks. Also, their limited energy resources, often due to reliance on batteries or energy-harvesting systems, restrict their ability to perform continuous or high-power tasks.

Blockchain is a fast-evolving technology that relies on hash algorithms to secure and validate transactions. Blockchain networks using Proof-of-Work (PoW) consensus mechanisms are mining processes that require substantial computational power and, consequently, the consumption of significant resources and energy. With rising demand for both blockchain scalability and environmental sustainability, alternative consensus mechanisms have seen the development of Proof-of-Stake (PoS), delegated PoS (DPoS), and Proof-of-Authority (PoA) to reduce energy demands and increase efficiency. These mechanisms attempt to grasp the network security and validation of transactions while reducing the environmental impact and resources usually associated with blockchain operations [4]. The VQL (Efficient and Verifiable Cloud Query Services for Blockchain Systems) system guarantees data integrity and reduces costs using verification mechanisms when querying blockchain data in the cloud environment. It provides a decentralized trust structure by allowing users to confirm query results' accuracy quickly. VChain+ (Optimizing Verifiable Blockchain Boolean Range Queries) increases the reliability of data access with cryptographic proofs as a system optimized for verifiable logical range queries. These two systems significantly contribute to scalability of the blockchain and reliability, creating a wider usage potential. The proposed system is thought to contribute to blockchain platforms in terms of security and cost [5].

Lightweight algorithms require less processing power and energy consumption while providing a high level of security, which means they can secure data against attack while still ensuring confidentiality, integrity, and authenticity without decreasing performance. Using lightweight hash algorithms in blockchain technology offers significant transaction verification and data integrity advantages. Since these algorithms require less processing power and energy consumption, they are ideal, especially for low-power devices or IoT-

based blockchain networks. Lightweight hash algorithms increase the overall performance of the network, ensuring that transactions are verified quickly and thus improving the user experience. In addition, thanks to their lightweight structures, they reduce transaction costs and contribute to the scalability of blockchain networks. Since such algorithms require less storage, they are preferred to ensure data security, especially in resource-constrained environments. In summary, lightweight hash algorithms make blockchains more efficient, accessible, and sustainable [6,7].

Especially in the last 5 years, many lightweight encryption algorithms have been developed. Throughout the development of these algorithms, it became evident that they effectively fulfilled specific requirements. In particular, the block sizes of these algorithms are smaller than the commonly used block cipher algorithms, and the key sizes used in the block cipher algorithm are smaller. The iterative processes of lightweight algorithms are designed to optimize computation, meaning they use fewer resources and can run faster than the more complex cryptographic operations. This provides efficiency due to the constraints placed on the amount and complexity of operations during each iteration, reducing the processing time and energy usage. Thus, lightweight algorithms are suitable in applications on constrained devices where both speed and energy efficiency are required, which is the case in IoT environments due to the large message sizes and energy consumption of traditional hash algorithms that refer to well-established cryptographic hash functions like MD5, SHA-1, and SHA-2, which have been widely used across various fields, including data integrity verification, digital signatures, and password storage. Lightweight hash algorithms have been developed using lightweight algorithms with less processing load. These algorithms have lower output sizes; however, lower output sizes weaken algorithms for collisions. In the design of these algorithms, the design should consider the security and cost balance. Another critical issue to consider in the design of lightweight hash algorithms is the smaller message sizes.

There are some basic lightweight-based hash algorithms developed in the literature [8–11]. Here, some of the other studies with lightweight-based hash algorithms are presented. Seok et al. [7] offer a lightweight, hash-based blockchain architecture for industrial IoT. In the study, using a blockchain-based IoT architecture, a lightweight hash algorithm was designed to improve performance and efficiency in blockchain structures in terms of area, throughput, and power consumption. A unique lightweight hash-based blockchain architecture is developed, which may adjust the hash algorithm used for mining to adapt to network load.

Rao et al. [12] offer a novel variant of the hashing function for authentication among IoT devices with limited resources. The proposed authentication technique employs a lightweight elliptic curve digital signature and a modified BLAKE2b (cBLAKE2b) hashing algorithm. The performance of cBLAKE2b is compared to that of regular BLAKE2b in a real-time Raspberry Pi 3 scenario. Performance factors such as hashing time, data size, signature creation time, and verification time are all considered in the proposed method. Experiments on various data sizes demonstrated that cBLAKE2b outperformed BLAKE2b. cBLAKE2b increases the signature formation and verification efficiency, resulting in lower power consumption for IoT devices. Degnan et al. [13] present Simon cipher key expansion as a compression function for hash applications as a flexible alternative solution. However, these advantages come at the expense of security, and it has been stated that the Simon cipher should not be used as a cryptographic hash without modification. It was stated that the system developed in the study would be beneficial for validating small amounts of data in low-entropy data streams.

Hirose et al. [14] present Lesamnta-LW, a novel lightweight 256-bit hash function. The security of Lesamnta-LW is reduced to that of the underlying AES block cipher. The primary purpose of Lesamnta-LW is to provide compact and fast hashing for lightweight applications in a broader range of settings, from low-cost devices to high-end servers, at the 2^{120} security level, as described in the article. On an 8-bit Central Processing Unit (CPU), Lesamnta-LW has superior speed and cost tradeoffs than SHA-256 for short message hash-

ing. Patrick et al. [15] investigate the issue of energy efficiency in cryptographic applications. The energy consumption values of many different encryption algorithms were examined and evaluated. The points that should be considered in their design are emphasized to reduce the energy consumption and delay values of lightweight encryption algorithms. Dhanda et al. [16] examine the design architectures of many different lightweight algorithms in detail. Comparisons of 54 lightweight-based block ciphers, stream ciphers, and hash functions are presented. Chip area, energy and power, hardware and software efficiency, throughput, and latency values were used as comparison parameters. Open research areas for lightweight algorithms are highlighted. Buchanan et al. [17] analyze popular lightweight encryption algorithms designed for use on devices with limited processing capacity. Advantages and disadvantages are expressed. In addition, a review was carried out on ultralight algorithms and their use in IoT applications. Abed et al. [18] examine lightweight hash algorithms widely used in the literature and hash algorithms used in blockchain applications. Well-known algorithms such as SPONGENT, PHOTON, and QUARK on Blockchain-IoT platforms are tested and evaluated in terms of memory space, power, energy, security, and efficiency. The test results indicate that SPONGENT demonstrates the best overall performance in terms of security and efficiency. At the same time, QUARK consumes the least power and energy but has the lowest security ratings. On the other hand, the PHOTON algorithm uses less memory and performs well across multiple criteria, including area, energy consumption, and security.

Jungk et al. [19] perform Field Programmable Gate Array (FPGA) applications of PHOTON, SPONGENT, KECCAK-200, and KECCAK-400, which are lightweight encryption algorithms, and realize their analysis. The comparison is made between throughput and slices, which is a measure of relative performance. The SPONGENT algorithm implementation achieved the highest throughput/area ratios. Lara-Nino et al. [20] evaluate the hardware performance in terms of resource usage, latency, and energy consumption of LHash-96 and SPONGENT-88 lightweight hash algorithms for IoT platforms. It is emphasized that fewer cycles per round and higher I/O rates are two important parameters for performance and energy consumption. The SPONGENT-88 design is stated to be the smallest and most efficient design ever. Meuser et al. [21] examine and compare PHOTON and QUARK lightweight hash algorithms in terms of security and performance. The analysis indicated that the PHOTON algorithm is less effective than QUARK in software optimization, although it outperforms QUARK in hardware applications. It is presented as an advantage that the two algorithms provide implementation with fewer hardware elements, and it is shown that it successfully provides the security and performance balance.

Padma et al. [22] propose a trust-based ensemble consensus and a Grey Wolf Optimization (GWO) approach for increasing trust levels in IoT networks, optimizing blockchain processes, and increasing scalability and throughput while maintaining IoT data security and privacy. Sakan et al. [23] propose a hash algorithm based on the symmetric block cipher of the compression function that performs with the wipe-pipe construction method. Windarta et al. [24] realized a hash function based on Saturnin lightweight block cipher that works in a sponge-based mode of operation called Beetle. Mahmoud et al. [25] present a fine-grained lightweight access control scheme containing hash function properties using attribute-based encryption (ABE) modified with Residue Number System (RNS) properties with fog computing. Li et al. [26] offer a distributed and two-layered trust management framework for IoT using blockchain architecture to provide privacy protection. Achar et al. [27] present four SHA-512-inspired LiteHash functions that reduce computational costs with approximation techniques while maintaining cryptographic security. Validated using the National Institute of Standards and Technology (NIST) PseudoRandom Number Generator (PRNG) test suite, the design demonstrated improvements in area, power, and efficiency, providing up to 2 Gbps throughput and 16.86% area and 32.48% power reductions while supporting the entire SHA-2 family with minimal changes. Some general studies explore the growing use of highly constrained devices within the IoT and the challenges they face in communication networks, such as power consumption,

limited memory, performance costs, and security. Singh et al. [28] provide a detailed review of lightweight cryptographic primitives, including block ciphers, hash functions, and stream ciphers, which are suitable for resource-limited IoT environments. Besides these, policy-based hash functions for blockchain in IoT [29], parallel implementation of some proposed lightweight hash algorithms [30], and chaos-based hash functions [31] continue to be developed in the literature.

This paper also analyzes cryptographic algorithms based on key size, block size, and structure, which will be discussed in the following sections. In addition, the authors propose a new hash construction design to address the challenges of improving the security, performance, and efficiency of cryptographic algorithms, and discuss open issues and future research directions for IoT security architectures. The main contributions of this article to the literature are as follows:

- A new hash construction design is proposed.
- A new hash algorithm (LwHash) is developed using the proposed construction design with the counter encryption mode and the SPECK algorithm.
- Performance evaluation and security tests of the developed hash algorithm were performed and compared with the algorithms in the literature. It has been demonstrated that the proposed solution meets the required security standards and has demonstrated better performance than other studies.
- A statistical analysis and cryptanalysis of the lightweight-based hash function have been conducted.

This article focuses on the development of a new lightweight hash algorithm, utilizing the SPECK algorithm [32], which is based on a lightweight encryption approach. The developed algorithm aims to achieve a design that enhances security, improves collision resistance, and increases efficiency, offering a more robust solution compared to existing algorithms. It is hypothesized that the implementation of the proposed algorithm, particularly in environments with constrained resources, such as those observed in the context of IoT blockchain applications, will confer notable benefits. The design of the algorithm employed counter (CTR) mode, a technique commonly utilized in block cipher algorithms. The padding process was implemented with the objective of creating a straight hash operation. After the introduction in this article, information about lightweight and hash algorithms is given in Section 2, and the proposed lightweight-based hash algorithm is presented in Section 3. The security and performance analyses of the new algorithm with different parameters are carried out in Section 4.

2. Background

This section presents a summary of lightweight block encryption algorithms, hash algorithms, and lightweight-based hash algorithms that are widely used in the literature, as well as the security requirements of hash algorithms.

2.1. Lightweight Algorithms

A summary of the recently developed lightweight block encryption algorithms in the literature is presented in Table 1. The design architectures of the algorithms, block size, key size, number of cycles, development years, and information about the authors are presented. It has been determined that the most important criteria in the development of these algorithms are the changes made to the block and key sizes and the number of cycles. Compared to traditional encryption algorithm designs, the SPECK cipher is designed for use on resource-constrained devices, and its processing load is reduced. The block sizes of the algorithms are generally 64 bits, and some algorithms are rarely designed as 128 bits. Key sizes are chosen quite differently according to algorithms. It is seen that the number of rounds slightly increases compared to the classical algorithms since the operations performed in the rounds are lightened. As the design architecture, Feistel Networks and Substitution-Permutation Network (SPN) structures are used. In the SPN architecture, the

entire block size is processed in each cycle. In the Fiestel structure, the block is divided into two equal parts, and the operation is performed on half a block in each cycle.

Table 1. Some recently developed lightweight algorithms.

Algorithm	Structure	Block Size	Key Size	Number of Rounds	Year	Developed by
RC6	Feistel	128	128/196/256	20	1998	Rivest et al. [33]
PRESENT	SPN	64	80/128	31	2007	Bogdanov et al. [10]
KLEIN	SPN	64	64/80/96	12/16/20	2011	Gong et al. [34]
LBLOCK	Feistel	64	80	32	2011	Wu et al. [35]
TWINE	Feistel	64	80/128	36	2011	Suzaki et al. [36]
PICCOLO	Feistel	64	80/128	25/31	2011	Shibutani et al. [37]
LEA	Feistel	128	128/192/256	24/28/32	2013	Hong et al. [38]
SPECK	Feistel	32/48/64 /96/128	64/72/96/128 /144/192/256	22/23/26/27/28 /29/32/33/34	2013	Beaulieu et al. [39]
CHASKEY	Feistel	128	128	8/16	2014	Mouha et. al. [40]
FANTOMAS	SPN	128	128	12	2014	Grosso et al. [41]
PRIDE	SPN	64	128	20	2014	Albrecht et al. [42]
ROBIN	SPN	128	128	16	2014	Grosso et al. [41]
RECTANGLE	SPN	64	80/128	25	2015	Zhang et al. [43]
MANTIS	SPN	64	128	10/12/14/16	2016	Beierle et al. [44]
SPARX	Feistel	64/128	128/256	24/32/40	2016	Dinu et al. [45]
LILLIPUT	Feistel	64	80	30	2017	Nachef et al. [46]
CRAFT	SPN	64	128	32	2019	Beierle et al. [47]

2.2. Hash Functions

Table 2 provides a summary of widely used hash algorithms in the literature. The hash size, block size, round numbers, internal state size of hash algorithms, year of development, architecture, and developers are summarized. The hash algorithms given in the table are widely used in many applications today; for instance, the lightweight authentication protocol [48]. As a result of attacks on some of these algorithms, security vulnerabilities have emerged and they have lost their reliability today [49–51]. In attacks on some algorithms, it has been demonstrated that the algorithm is resistant to collisions up to a specific round. However, some algorithms are accepted as the standard in the literature and whose attack studies continue. When the hash size of the algorithms is evaluated, it is seen that the hash size values of the algorithms first published in the literature are less than the algorithms designed today. The newly designed algorithms support different hash sizes, and they can produce an output of up to 512 bits. It has been determined that the block sizes are proportional and similar to the output size values. When the number of cycles is evaluated, it is seen that they have very high cycle numbers compared to standard block cipher algorithms, and the number of cycles differs according to the algorithm design. While the number of cycles increases the security of the algorithm, it brings extra cost. While determining the number of cycles in algorithm design, it is crucial to decide on the optimum value without reducing the level of security and performance. When the design architectures are examined, it is seen that many different architectures are used in algorithm designs. These architectures have advantages and disadvantages over each other. It can be said that Merkle–Damgard and Sponge architectures are frequently preferred in algorithm designs.

Table 2. Some recently developed cryptographic hash functions.

Algorithm	Output Size (Bits)	Block Size	Rounds	State Size	Year	Construction	Developed by
SHA-0	160	512	80	160	1993	Merkle-Damgaard	NSA [52]
SHA-1	160	512	80	160	1995	Merkle-Damgaard	NSA [53]
RIPEMD	128/160/256/320	512	64/80/80/80	128/160/256/320	1996	Merkle-Damgaard	Hans Dobbertin et al. [54]
SHA-2	224/256/384/512	256	64/80	256/512	2001	Merkle-Damgaard	NSA [55]
Whirlpool	512	512	10	512	2004	Miyaguchi-Preneel	Vincent Rijmen et al. [56]
BLAKE	512	1024	12	512	2008	HAIFA construction	Jean-Philippe et al. [57]
MD6	512	512	40 + [d/4]	512	2008	Merkle tree	Ronald Rivest et al. [58]
BLAKE2	256	512	14/16	256	2012	HAIFA construction	Jean-Philippe et al. [59]
Grøstl	256/512	512	10/14	256/512	2016	AES construction	Praveen Gauravaram et al. [60]
Skein	256/512	512/1024	72/80	256/512/1024	2016	Unique Block Iteration	Bruce Schneier et al. [61]
SHA-3	224/256/384/512	1600	24	1600/1152/1088/834/576	2016	Sponge construction	Guido Bertoni et al. [62]
BLAKE3	—	512	7	256	2020	Merkle tree	Jack O'Connor et al. [63]

2.3. Lightweight Hash Functions

This section presents information on lightweight hash algorithms commonly discussed in the literature. SPONGE construction [64] is the main component used in the design of hash functions. It has lighter operations than the Merkle–Damgard architecture, which is preferred for traditional hash functions. Sponge construction uses keyless permutations and hash operations. In this architecture, by changing the parameters of the function, input and output can be used in a wide range [21]. SPONGENT [10] is a lightweight hash algorithm developed using sponge architecture. The SPONGENT algorithm offers versions of different hash output sizes and security levels. The number of cycles differs depending on the output size for each version. Based on the SPONGENT algorithm, it uses the basic operations of the PRESENT lightweight block cipher algorithm. There are three basic operations in each loop. In the counter operation, a bit added to the bit array calculated by the polynomial is performed. Byte exchange is performed in the sBoxLayer layer, and the bits in the state matrix are permuted using the pLayer operation. The PHOTON [8] algorithm is a lightweight hash algorithm developed with sponge architecture and uses permutation operations similar to the AES algorithm. There are five different versions with outputs of 100, 144, 196, 256, and 288 bits. The algorithm uses four basic operations: addConstant, SubCells, ShiftRows, and Mixcolumns. QUARK [9] is a lightweight hash function algorithm developed with a bit permutation-based Sponge architecture. The algorithm works in three different versions. These versions offer different security levels: U-Quark (64-bit), D-Quark (80-bit), and S-Quark (112-bit). The QUARK permutation function uses three different Nonlinear Feedback Shift Registers. The design of hash functions is basically based on providing three security principles. These requirements are Pre-Image Resistance, Second Pre-Image Resistance, and Collision Resistance [1]. In evaluating the security levels of the algorithm, it is primarily checked whether it meets these conditions.

Pre-Image Resistance: This requirement states that the hash value produced from any input cannot be calculated back from this value. This means that the function works one way [65].

Second Pre-Image Resistance: In this requirement, the difficulty of producing the hash value produced from an input value from a different input value is tested. In other words, if a hash algorithm h yields the hash value h(x) for an input x, then finding any other input value y such that h(y) = h(x) is hard to find, in a computational sense [65].

Collision Resistance: The Collision Resistance means that it should be difficult to find two different inputs of any length that result in the same hash value. This requirement is also called the collisionless hash function. In other words, for a hash function h, it is difficult to find any two distinct x and y inputs such that h(x) = h(y). A hash function is second pre-image resistant if it is collision-resistant [66].

3. Proposed Lightweight Hash Algorithm

In this section, the proposed lightweight hash function is explained in detail. First, the new model used in hash function design is introduced. The model design employs a block cipher-based structure, utilizing CTR mode—one of the most commonly used operation modes in block cipher algorithms. The working principle of counter mode is briefly explained. Lightweight-based block cipher algorithms have been analyzed, and the SPECK algorithm, which stands out in its performance, is used for encryption operations. Since the algorithm must have a specific data size in order to perform the operations healthily, the padding process is applied by checking whether the padding process is needed before starting the work. Finally, the pseudo-code of the proposed hash-based lightweight algorithm is given, and the software implementation is explained. The proposed hash function construction is based on a lightweight block cipher illustrated in Figure 1. Merkle–Damgard, Wide-pipe, Sponge, etc., constructions use the chaining principle to provide hash properties such as resistance to collisions and the avalanche effect. We propose the counter-operation mode on a block cipher in the designed scheme to give properties of the chaining principle. Also, the use of CTR mode in the proposed design enables parallel processing of blocks and ensures strong cryptographic chaining properties. In CTR mode, a nonce is generated as an initialization vector (IV) to ensure that even identical messages yield different ciphertexts. This nonce is then incremented after each block encryption, allowing each sub-block to be processed independently while still maintaining the cryptographic link across the entire input message.

The SPECK lightweight block cipher is selected for the encryption scheme. The input message is divided into sub-blocks of 128 bits (M_0, M_1, M_2, to M_n). Each block is encrypted with the SPECK cipher algorithm. This division allows the SPECK cipher to encrypt each block efficiently. Each sub-block is processed independently and encrypted using the SPECK algorithm, which performs several rounds of ARX operations to secure the data. The encrypted message sub-blocks ($^eM_0$, $^eM_1$, $^eM_2$, to ^eM_n) are XORed to detect any changes in the message block. This XOR operation not only ensures the detection of any changes in the input message blocks but also acts as a lightweight integrity check. Any modification in a single block will result in a cascading change across the entire message, helping to achieve collision resistance and enhancing security.

The output of the XOR operation is encrypted to create a mini-hash of the i-th message block. The mini-hash of the block is XORed with each ciphered sub-blocks ($^cM_0$, $^cM_1$, $^cM_2$, to ^cM_n) to revise the whole block. In this way, when there is a change in the message block, the data in the entire block may change. As a result of this, each block can be encrypted in itself and provide the necessary changes. Depending on the message size, many message blocks can be formed. Also, the purpose of the mini-hash is to add another layer of verification and ensure that each block is independently validated before being combined into the final hash value. This process helps distribute changes throughout the hash and ensures that the avalanche effect is consistently applied.

Addition and XOR operations are applied to ciphertext blocks ($^cBlock_i$) successively to shuffle and compression. Thus, the modification in any block affects the entire hash output. By performing these operations, the proposed hash function achieves a high degree of collision resistance—where two different input messages are extremely unlikely to produce the same hash—and an enhanced avalanche effect—where small changes in the input drastically alter the hash output. The SPECK round function, which utilizes the ARX (Add-Rotate-XOR) structure, ensures that each message block is securely encrypted and thoroughly mixed during the compression phase. The overall result is a hash function that not only maintains strong cryptographic properties but is also optimized for performance in low-resource environments.

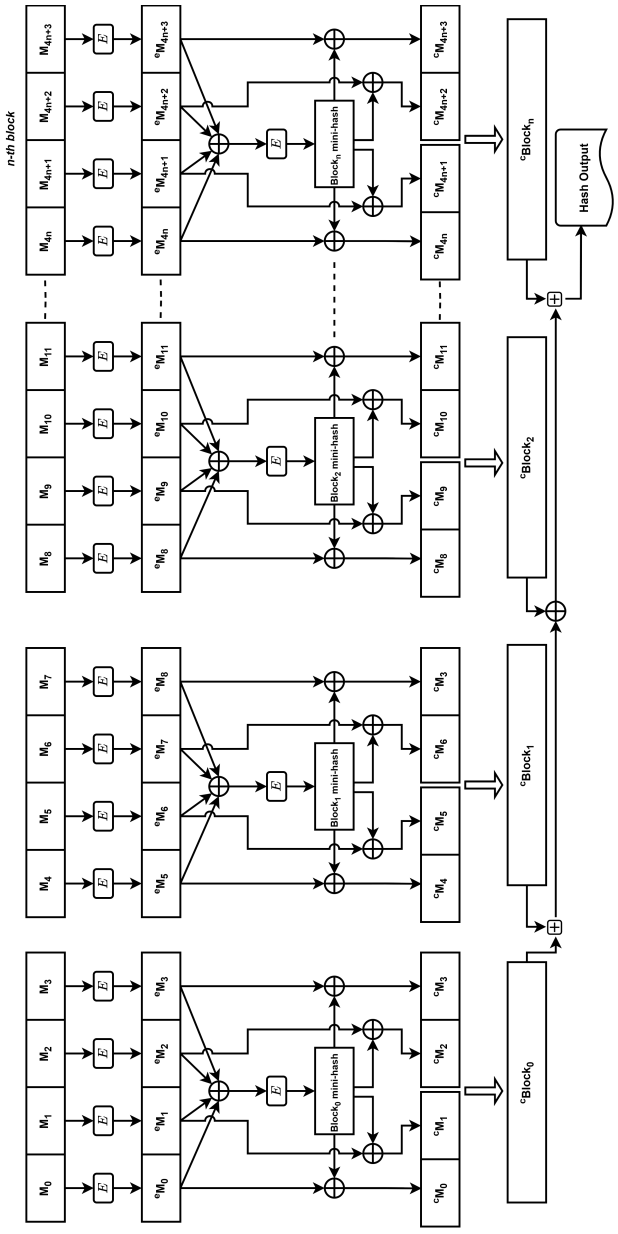

Figure 1. The general structure of the proposed hash function.

The counter-mode encryption scheme is shown in Figure 2. A nonce value is generated randomly as an initialization vector to produce different message texts. The increment-by-one counter is a simple and popular counting function used to create various outputs for repeated data. The block cipher encrypts the nonce value instead of plaintext, and the nonce value is incremented by one after each encryption operation. The encrypted text is XORed with plaintext to generate the ciphertext. The SPECK round function is based on Add–Rotate–XOR (ARX) operations. The message is divided into two branches, Xleft and Xright, and each branch affects the other after going through specific ARX operations. The left branch is shifted 8 bits to the right and added with the right branch. The result of this operation is XORed with precomputed round keys. The right branch is shifted 3 bits to the left and XORed with the output of the left branch. These operations are repeated for 32 rounds.

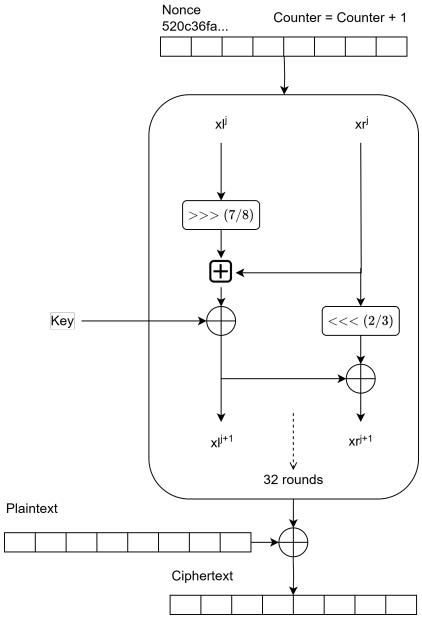

Figure 2. Counter-based block cipher implementation.

The SPECK round function definition is given in Equation (1), similar to mixing function of the THREEFISH. The x and y are 64-bit plaintext words, and the number of rounds is 32. The $S^{-\alpha}$ and S^{β} are the right and left circular shifts, which rotate 8 bits (α) right and 3 bits (β) left because of using 64-bit word size. Addition modulo 2^n is applied on the shifted x and y values and bitwise XORed with round keys. The round keys k_i are defined with $k \in F(2)^n$and generated with a key expansion function (Equations (2) and (3)) which also uses the round function. The key can be written as $K = (l_{(m-2)}, \ldots, l_0, k_0)$ where $l_i, k_0 \in GF(2)^n$ with the number of key words (m) in 2,3,4 .

$$R_k(x,y) = \left(\left(S^{-\alpha}x + y\right) \oplus k, S^{\beta}y \oplus \left(S^{-\alpha}x + y\right) \oplus k\right) \tag{1}$$

$$\ell_{i+m-1} = \left(k_i + S^{-\alpha}\ell_i\right) \oplus i \tag{2}$$

$$k_{i+1} = S^{\beta}k_i \oplus \ell_{i+m-1} \tag{3}$$

A message padding procedure is executed to generate a fixed message size, which is a multiple of 128, 256, or 512 bits in the beginning. The padded message is partitioned into sub-blocks of 128 bits $m_0, m_1, \ldots m_n$ due to the block size of the lightweight block cipher. The message padding technique is given in Algorithm 1. If the message is smaller than the requested digest size, a single set of "1" bits is added to the end of the message, and "0" bits are added until the required size is completed. The requested digest size can be multiple of the block size.

Algorithm 1 Padding Technique

Input: *Message* (M)
Block Size (B)
Output: *Padded Message*(M^d), *sizeof* (M^d) *is multiple of* B
$P = mod\ B$
$d = P - 1$
$M^d = M||1||0^d$
Return M^d

The overall hash function construction is summarized in Algorithm 2. The entire message is divided into digest sizes, which can be 512-bit, 256-bit, or 128-bit, and the

outcome message blocks are partitioned into sub-blocks of 128 bits. The number of sub-blocks formed varies according to the digest size. If the digest size is 512-bit, four sub-blocks are created, two sub-blocks are created for 256-bit, and a single block is formed for 128-bit. These sub-blocks are stored by two 64-bit array variables and XORed with encryption of nonce and round keys. These encrypted messages, the same as the number of sub-blocks, are XORed, and the outcome is encrypted for creating each block mini-hash. Block mini-hashes are XORed with ciphered text, which is the hash value of each block. Hash values are singularized by adding and XORing, respectively.

Algorithm 2 LwHash Function

Input : $M_1, M_2, \ldots, M_t$;
$M_i = (m_0^{128} || m_1^{128} || \ldots || m_r^{128})$;
$r = 0, 1, \ldots, (digest/128)$
for i = 0 to t **do**
  for j = 0 to r **do**
    $Ciphertext_i^j \leftarrow Encrypt(Nonce++, RK) \oplus m_i^j$
  end for
end for
for i = 0 to t **do**
  for j = 0 to r **do**
    $Block\ minihash_i \leftarrow Block\ minihash_i \oplus Ciphertext_i^j$
  end for
  ${}^{C}Block\ minihash_i \leftarrow Encrypt(Block\ minihash_i, RK)$
end for
for i = 0 to t **do**
  for j = 0 to r **do**
    $Ciphertext_i^j \leftarrow {}^{C}Block\ minihash_i \oplus Ciphertext_i^j$
  end for
end for
for i = 0 to t **do**
  for j = 0 to r **do**
    if i mod 2 = 0 **then**
      $Hash_i \leftarrow Hash_i + Ciphertext_i^j$
    else
      $Hash_i \leftarrow Hash_i \oplus Ciphertext_i^j$
    end if
  end for
end for

4. Security and Performance Analysis

We discuss the properties of the proposed hash function in four aspects: avalanche effect, collision resistance, security proofs, and performance efficiency. The proposed hash function is implemented in C++ (Microsoft Visual Studio 2017, version 15.9.48, Microsoft, Redmond, WA, USA) on the Windows 10 Pro platform (Edition: Windows 10 Pro, Version: 22H2, Microsoft, Redmond, WA, USA). The system specifications include a 64-bit Intel Core i7-7700HQ CPU @ 2.80 GHz processor (Intel, Santa Clara, CA, USA) with 16 GB of installed RAM (15.6 GB usable). The proposed scheme is compared with other lightweight hash functions in terms of execution time efficiency

4.1. Sensitivity of Hash Value

Six conditions on text messages were simulated to observe the changes in the hash output of the algorithm against the changes to be made on an original message. We choose the 256-bit version of our proposed hash function as an example for the message sensitivity test.

- Condition 1: The original text message is as follows: "It is clear and self-evident that aims, purposes, instances of wisdom, and benefits can only be followed through choice, will, intention, and volition, not in any other way." (as cited in [67]).
- Condition 2: The first character I is replaced with A in the original text message.
- Condition 3: The word "wisdom" is replaced with "visdom" in the original text message.
- Condition 4: The letter f in "followed" is written as "Followed" in capital letters.
- Condition 5: The extra character number "1" is added to the end of the original text message.
- Condition 6: The dot "." at the end of the message is replaced with a semicolon ";"

The corresponding hash outputs in the hexadecimal numeral system are in digest size of 256-bit version;

- Condition 1: 4E0F4D7A83EB45F7BDAEC9EC8C91289CDBC7FD9F2CA046CA0021BCBBC017B3FE
- Condition 2: 07247FBFDCF3A854418EEB922FE200451E30FE6798ED6795740DC6024DA9A7DF
- Condition 3: 79D7C7776823DA716DE9D4BC5AB07261EB3CF747A083DF2C81B4FB6BA41F8E1A
- Condition 4: 2C4B14B5F139F612E6DC34D4D4F5DBDF2DFB1B5B248D4CCBA8329AD91982D6C3
- Condition 5: AC86C653F8D9DFDE1EB38F4BFB04A0BECA5DED89BD81F475A29A88011076A9A0
- Condition 6: A08B75F9ED0A00C5B76BB6289AEEB8CD364A3E1FF1B214A0FAC2789283F72291

The graphic presentation for outputs of the presented hash function is illustrated in Figure 3 under six conditions with a 256-bit version as an instance of other 128-bit and 512-bit versions. The figures show the significant changes that the hash function makes on the output in different conditions.

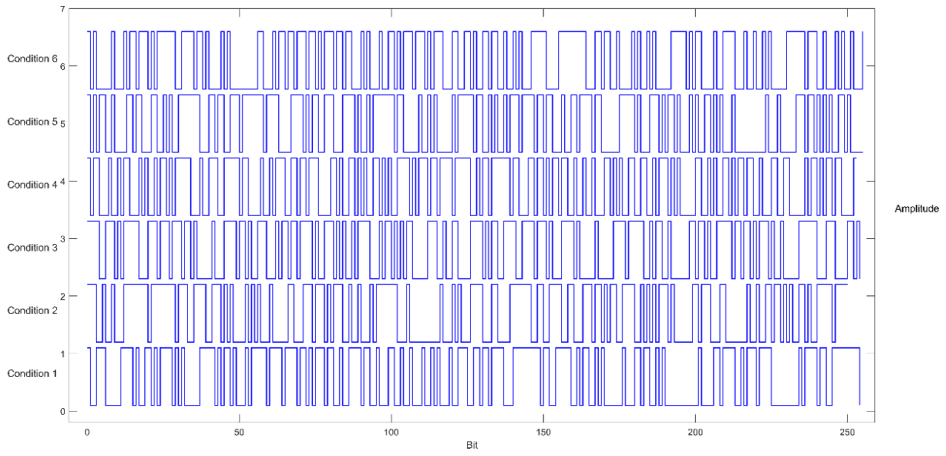

Figure 3. Graphic presentation of the 256-bit version of hash output under six conditions.

4.2. Statistic Analysis of LwHash

Confusion and diffusion are the two significant properties of the hash functions. They are two essential factors to resist the statistical attacks for a hash function. Diffusion represents a significant alteration in the bits comprising the hash value, occurring in a pseudorandom manner. The confusion hides the relationship between the key and the ciphertext in definition of Shannon. The expected change in the hash output should be a 50% probability when there is any bit change on the original message. Generally, the statistical analysis uses four qualifications to observe the changes of bits, where N is the

total test number, the number of changed bits in the i-th test is stored as H_i, and n represents the hash value length.

The number of average bit changes:

$$B_c = \frac{1}{N}\sum_{i=1}^{N} H_i \tag{4}$$

Percentage of bit change:

$$R_c = \frac{B_c}{n} \times 100\% \tag{5}$$

The standard deviation of bit change:

$$\Delta B_c = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(H_i - B_c)^2} \tag{6}$$

The standard deviation of R_c:

$$\Delta R_c = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{H_i}{n} - R_c\right)^2} \times 100\% \tag{7}$$

Firstly, we use a pseudorandom number generator called Mersenne Twister (MT19937) to create 1000 random text messages, and corresponding hash values are generated. Afterward, we change a bit on an arbitrary position in the original message and generate its corresponding hash value. The change data between these two hash values are given in Table 3, and other hash function data are added for comparison. The average number of bit changes and the percentage of bit changes reach the ideal value of 50% across all versions of the proposed hash function. The standard deviation value is desired to be small in order to keep the bit change in the required range. As can be seen from the table, the proposed hash function has a lower standard deviation of Bc and Rc values than the others.

Table 3. The statistical analysis of the proposed hash function.

	$B_{min} - B_{max}$	The Average of Changed Bits (B_c)	R_c (%)	$\triangle B_c$	$\triangle R_c$ (%)
LwHash-128	47–80	64.10	50.08	5.64	4.40
LwHash-256	103–154	128.07	50.02	7.67	2.99
LwHash-512	218–291	256.59	50.11	11.52	2.25
SHA-1	49–114	80.16	50.10	6.24	3.90
SHA-256	71–193	128.61	50.24	10.83	4.23
SHA-512 Ref [68]	221–288	255.82	49.96	11.08	2.16
KECCAK-256	80–207	129.34	50.52	10.21	3.99
PLHF	91–168	127.91	49.96	8.93	3.49
MD5	N/A	64.03	50.02	5.66	4.42
Spongent-224	89–136	111.84	49.92	N/A	N/A
Neeva	89–137	111.91	49.95	N/A	N/A
Ref. [69]	99–150	128.37	50.14	7.64	2.98
Ref. [70]	10–155	128.10	50.04	7.96	3.11
Ref. [71]	241–268	256.11	50.01	16.36	3.20

The distribution of changed bit numbers is shown in Figure 4 for all versions of the proposed hash function. The number of bit changes with the proposed hash function fluctuates within a range that is approximately consistent with the optimal value of 128, as illustrated in Figure 4. The position of the changed bit i-th or the number of repetitions does not affect on the number of bit changes. The statistical histogram of changed bit numbers is illustrated in Figure 5, and it is observed that it has a structure similar to the normal distribution, which is the ideal form. The most common values in the distribution are those close to the mean value.

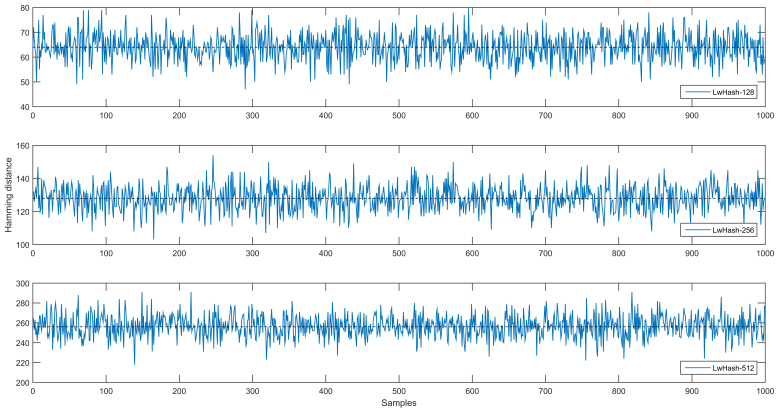

Figure 4. Distribution of changed bit numbers.

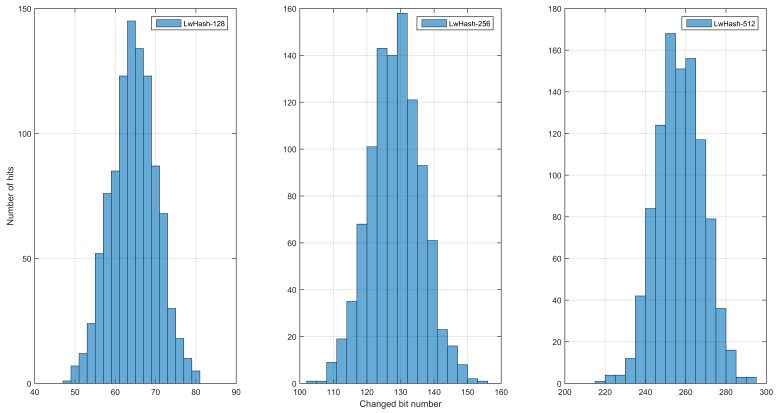

Figure 5. Statistical histogram of changed bit numbers.

4.3. Distribution of Hash Values

The hash value must have a uniform distribution to resist pre-image or second pre-image attacks. A brute-force attack can be the easiest way to find the first or second pre-image of the hash value if there is a weakness. There are some other faster pre-image attacks as a result of cryptanalysis for particular hash functions. We generated 1000 random messages to observe the frequency distribution of hash values, as demonstrated in Figure 6. The number of each hexadecimal character in the hash value is calculated to show the distribution of the hash value. It is expected that each hexadecimal character will occur in equal numbers. A chi-square test (Equation (8)) was employed as a goodness-of-fit test to determine whether the hash value is distributed uniformly across all versions of the proposed hash function. For 1000 random messages, the number of expected hexadecimal characters (E_i) is 8000 for the 512-bit version, which ranges from 7854 to 8144 (O_i). The critical chi-square value (X_c) is calculated as 11.71, which means the null hypothesis (H_0, the hash value is uniformly distributed) is true for any alpha α value. The range of hexadecimal characters is from 3919 to 4080 for the 256-bit version, and the critical chi-square value is 6.69, which has a minimal error rate. Also, the hexadecimal characters range from 1924 to 2073 for the 128-bit version, and the critical chi-square value is 11.69, which makes the null hypothesis true over again.

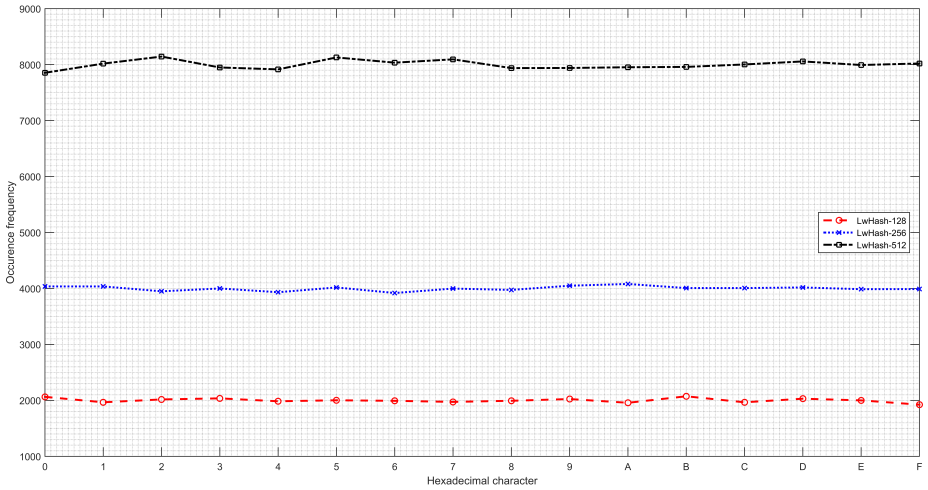

Figure 6. Frequency distribution of hash output.

$$X_c^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{8}$$

We aim at achieving an average of 50% probability of change in the hash value, and also, it is expected that change affects all bit positions. This change in bit positions also requires a uniform distribution. The bit variance test shows the uniformity of the bit change on the digest bit positions and is related to the avalanche effect of the proposed hash function. The changes in the bit positions are shown in Figure 7 for all versions of the proposed hash function. We generate 1000 random messages and corresponding hash values. Also, we use the chi-square test to check the uniform distribution of the variance on bit positions. The bit variance ranges from 454 to 556, which is expected to be 500 changes for 1000 tests. The critical chi-square value is calculated as 11.69, 6.69, and 11.71, respectively, for 128-bit, 256-bit, and 512-bit versions. All versions of the proposed hash function passed the test, and the variance on bit positions has a uniform distribution as a result.

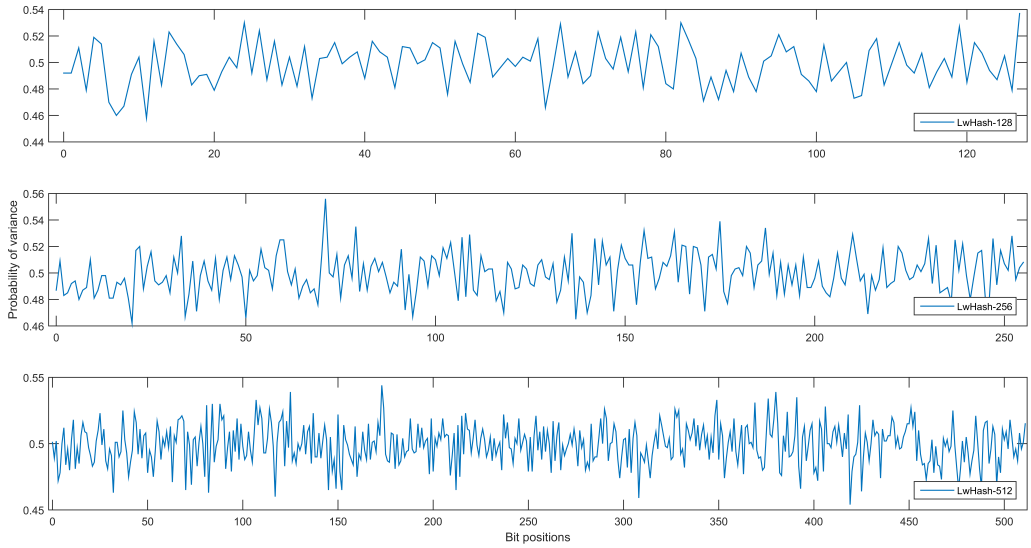

Figure 7. The variance of bit positions.

4.4. Collision Resistance

The collision test finds the same hash outputs, which have different input messages. Collision resistance is one of the main properties of cryptographic hash functions. It is

assumed that collisions are inevitable in an enormous sample space. It is unfeasible to test all possible hash values due to the considerable sample space (2^{512}, 2^{256}, etc.) for desired digest size. The method of Wong [72] is applicable instead of testing all possibilities to measure the collision resistance of hash functions. We generate 2000 random messages and corresponding hash values, which are stored in ASCII format. Then, we change a single bit in a random position and generate a new hash value for changed messages. We count the number of the same ASCII characters in the exact location (Ns) to measure the collision resistance. The number of equal entry values is desired to be as low as possible, and the proposed hash function results are shown in Figure 8. The maximum Ns value is 2 for 128-bit and 256-bit versions, and the maximum Ns value is 3 for the 512-bit version. The maximum Ns values are reasonable when compared to other hash functions. The other exhibited performance is that there are no equal entries in 93% of the testing for the 128-bit version and close results for other versions.

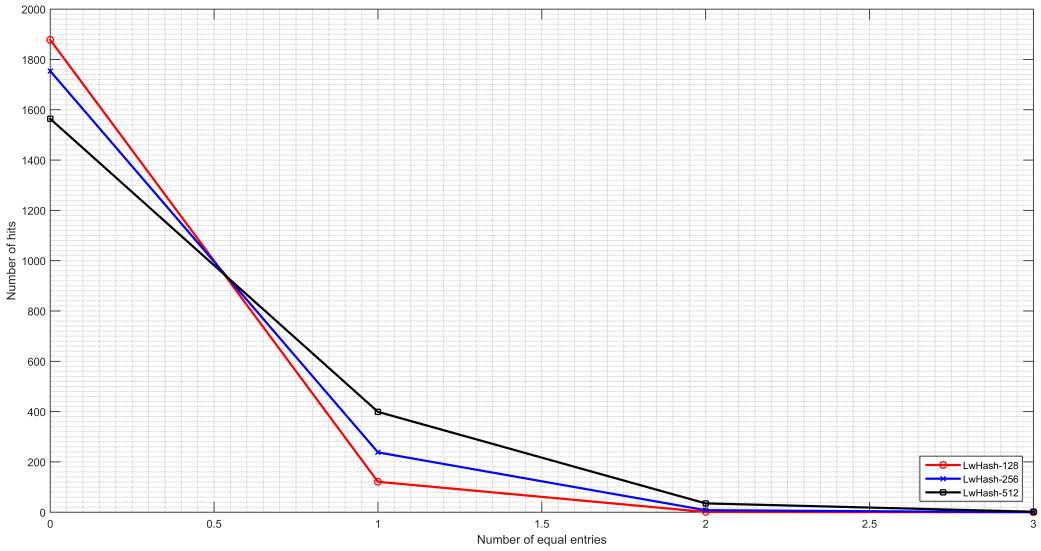

Figure 8. The number of equal entries at the same location in a hash value.

Table 4 shows the parameters that define the distances between the hash values of messages changed by a single bit. The distances in ASCII characters d_{hash} in the exact location of hash values are created by changing a single bit are calculated with Equation (9). The minimum, maximum, and mean value are calculated from the collection of d_{hash} values. The d_{avr} value is obtained by dividing the mean value by the character values it contains. The deviation in the d_{hash} values will be expected to be minor, and the d_{avr} value will be close to the ideal value of 85.33. The proposed hash function has more stable and robust collision resistance than the other compared hash functions, considering the collection of d_{hash} values.

$$d_{hash} = \sum_{i=1}^{\left(\frac{digest}{8}\right)} |dec(t_i) - dec(\tilde{t}_i)| \quad (9)$$

Table 4. The distances between the two hash values.

	Size Output	N_s *max*	D_{hash} *max*	D_{hash} *min*	D_{hash} *mean*	D_{avr}
LwHash-128	128	2	2195	613	1372.18	85.76
LwHash-256	256	2	3850	1547	2727.43	85.23
LwHash-512	512	3	7162	3825	5456.58	85.25
SHA-1	160	2	2556	857	2732.80	85.40
SHA-256	256	2	4098	1365	2732	85.37
KECCAK-256	256	2	4105	1368	2733	85.40
PLHF	256	1	4096	1364	2731	85.34
Ref. [69]	256	3	4079	1474	2733.18	85.41
Ref. [70]	256	2	3831	1695	2715.39	84.85
Ref. [70]	512	3	7062	3911	5414.34	84.59

4.5. Cryptanalysis

4.5.1. Collision Resistance Attack (Birthday Attack)

A Birthday Attack is a collision attack against hash functions. This attack uses the birthday paradox to increase the probability that two different inputs will produce the same hash output. According to the birthday paradox, the probability that two different inputs will produce the same hash output (collision) depends on the probability of the exact length of the hash function, 2^n. However, instead of a direct $2^{n/2}$ trials, the probability of finding a collision increases approximately with $2^{n/2}$ trials [66]. From another perspective, the minimum number of chosen messages required is $(q) \approx \sqrt{2m \ln \frac{1}{1-p_1}}$, where m is output bits (128, 256, 512) and p is the probability of random collision. The minimum workload required by the attackers for LwHash versions is given in Table 5. For example, for an attacker to successfully execute an attack on the hash value h for the two proposed structures with a minimum length of 256 bits ($m = 256$) and probability of 0.5, the minimum workload required would be 4.00×10^{38} attempts, which is practically infeasible.

Table 5. Birthday attack to LwHash.

	Desired Probability of Random Collision			
	0.01	0.1	0.5	0.9
LwHash-128	2.61×10^{18}	8.46×10^{18}	1.84×10^{19}	3.95×10^{19}
LwHash-256	4.82×10^{37}	1.56×10^{38}	4.00×10^{38}	7.30×10^{38}
LwHash-512	1.64×10^{76}	5.31×10^{76}	1.6×10^{77}	2.48×10^{77}

4.5.2. Pre-Image and Second Pre-Image Attacks

The goal of a pre-image attack is to find an input m that corresponds to a hash output h. That is, to find the value m that corresponds to a known hash output $h = H(m)$ [65]. If a hash function has a 256-bit output, then for each different input to that function, there are 2^{256} possible outputs. In a pre-image attack, the attacker would need to make 2^{256} attempts on average. However, the chances of success for the attacker are, on average, half as many attempts out of 2^{256}, which for our LwHasH-512 means $2^{256-1} = 2^{255}$. This means that a pre-image attack is almost impossible in practice.

The goal of the second pre-image attack is to find a different input m_2 for the hash value $h = H(m_1)$ corresponding to a known input m_1. That is, to find $m_1 \neq m2$ such that $H(m_1) = H(m_2)$ [65]. For the same LwHash-512 function, there are, on average, 2^{512} possibilities for the attacker to find a different m_2. However, the probability of success still requires, on average, $2^{512-1} = 2^{511}$ attempts. This also means that a second pre-image attack is almost impossible in practice.

4.5.3. Exhaustive Key Search Attack

This is a type of attack in which each possible key in the key space of a cryptographic algorithm is tried one by one to find the correct key. This attack can be effective if the cryptographic system has weak key lengths. The length of the key is the most important factor that determines the difficulty of the attack. The security of a cryptographic algorithm depends on the length of the key used. If the key length is k bits, then the number of possible keys is 2^k. Finding the correct key takes, on average, half of all keys to be tried. So,

the attacker needs to find the correct key in 2^{k-1} tries on average [73]. LwHash function uses a 128-bit key that requires $2^{128-1} = 2^{127}$ attempts, so this attack is also ineffective.

4.6. Efficiency Analysis

An extensive comparison was made to compare the performance efficiency of the proposed hash function with other lightweight hash functions such as Keccak-512, BLAKE-512, Grostl-512, JH-512, SipHash-64, Lesamnta-256, QUARK-256, Photon-256, and SPONGENT-256 according to execution time. We calculated the required execution time in ms to generate the hash values of messages with different sizes (1 KB to 1 MB) by taking the average of 1000 tests. The Lesamnta, QUARK, PHOTON, and SPONGENT functions have a long execution time for hardware-oriented structures. They are all based on the P-Sponge design, and only Lesamnta builds on the Merkle–Damgard design. Compared to other algorithms, the Grostl, JH, and SipHash functions have moderate execution times, placing them in the middle of the performance range. SipHash and JH have a similar structure, while Grostl uses the essential components from the AES. The Keccak winner of SHA-3 and BLAKE algorithms has the closest execution time with the proposed hash function. The proposed hash function is based on a fast and software-oriented block cipher structure that uses ARX operations. We achieved performance efficiency on the proposed hash function with a new hash construction utilizing a block cipher. Figure 9 shows the execution time comparison of lightweight hash functions, and all versions of the proposed hash function perform more efficiently than others.

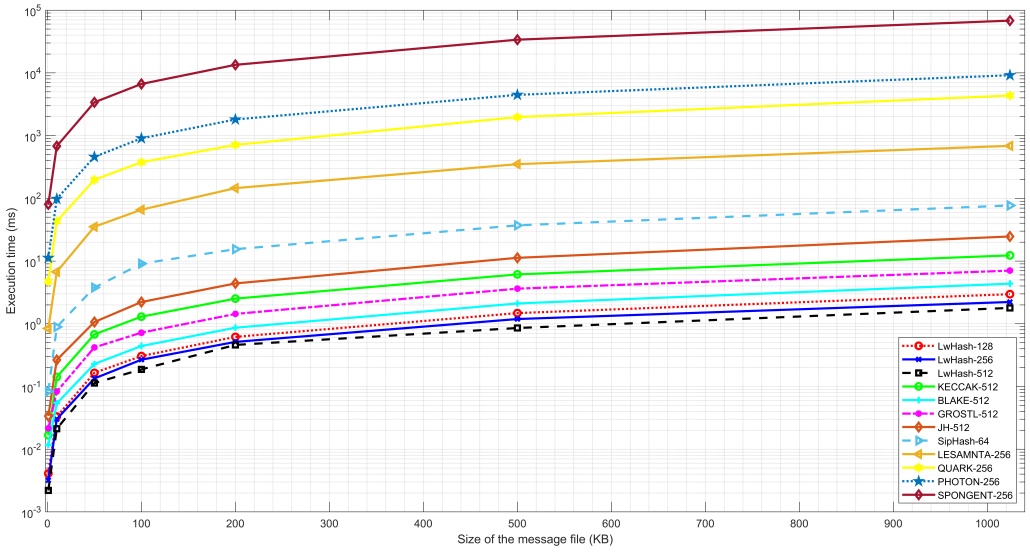

Figure 9. Comparison of execution times.

5. Conclusions and Outlook

This article proposes a new lightweight hash algorithm model based on a lightweight encryption algorithm. Firstly, some key algorithms in the literature are examined, and then the design of the proposed SPECK encryption algorithm based on the new hash algorithm is explained in detail. Security and performance tests of the proposed hash algorithm were carried out. Sensitivity of Hash value analysis was performed for the 256-bit version of the algorithm for six different cases. It has been shown that the algorithm has sufficient precision for all cases. A statistical analysis was conducted to evaluate the efficiency of the proposed algorithm, and the test results were compared with the recent algorithms in the literature. According to the comparison results, the algorithm had sufficient statistical analysis results. The frequency distribution test was applied over the input values obtained from the algorithm. The variance of bit position results was presented. It has been shown that the obtained hash values have a balanced distribution. The collision resistance test was

applied as the final security test and compared with the current algorithms in the literature. The proposed hash function has more stable and robust collision resistance than the other compared hash functions. As a performance test, a comparison with current algorithms over execution time values is presented. According to the test results, the proposed hash function has better efficiency than others. According to the security and performance test results and comparison values, the proposed lightweight hash algorithm has been shown to have good security and performance values.

In future research, we aim to test the applicability and performance of the hash function on real-world IoT devices and blockchain platforms. It would be beneficial to evaluate the performance of the algorithm in real-world scenarios, specifically in terms of energy consumption, speed, and security. To this end, the proposed lightweight hashing algorithm will be tested not only in software environments but also in hardware-based systems, such as FPGAs or Application Specific Integrated Circuits (ASICs). Parallel processing capabilities of the proposed hash function and its performance on multi-core processors are also potential research topics. In the case of parallel processing support, the efficiency of the algorithm may increase on large datasets or compute-intensive applications.

Author Contributions: Conceptualization, A.S.; Methodology, A.S. and Ü.Ç.; Software, A.S.; Validation, Ü.Ç.; Formal analysis, A.S.; Investigation, Ü.Ç.; Writing—original draft, A.S. and Ü.Ç.; Writing—review & editing, Ü.Ç. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data that support the findings of this study are available upon request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

Conflicts of Interest: The authors have no relevant financial or non-financial interests to disclose.

References

1. Rogaway, P.; Shrimpton, T. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Proceedings of the Fast Software Encryption: 11th International Workshop, FSE 2004, Delhi, India, 5–7 February 2004; Revised Papers 11; Springer: Berlin/Heidelberg, Germany, 2004, pp. 371–388.
2. Radack, S. Itl Bulletin for May 2012 Secure Hash Standard: Updated Specifications Approved and Issued as Federal Information Processing Standard (FIPS) 180–184. Available online: https://csrc.nist.gov/files/pubs/shared/itlb/itlbul2012-05.pdf (accessed on 7 August 2024).
3. Ashton, K. That 'internet of things' thing. *RFID J.* **2009**, *22*, 97–114.
4. Bamakan, S.M.H.; Motavali, A.; Bondarti, A.B. A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Syst. Appl.* **2020**, *154*, 113385. [CrossRef]
5. Wu, H.; Peng, Z.; Guo, S.; Yang, Y.; Xiao, B. VQL: Efficient and Verifiable Cloud Query Services for Blockchain Systems. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1393–1406. [CrossRef]
6. Windarta, S.; Suryadi, S.; Ramli, K.; Pranggono, B.; Gunawan, T.S. Lightweight cryptographic hash functions: Design trends, comparative study, and future directions. *IEEE Access* **2022**, *10*, 82272–82294. [CrossRef]
7. Seok, B.; Park, J.; Park, J.H. A lightweight hash-based blockchain architecture for industrial IoT. *Appl. Sci.* **2019**, *9*, 3740. [CrossRef]
8. Guo, J.; Peyrin, T.; Poschmann, A. The PHOTON family of lightweight hash functions. In *Advances in Cryptology*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 222–239.
9. Aumasson, J.P.; Henzen, L.; Meier, W.; Naya-Plasencia, M. Quark: A lightweight hash. *J. Cryptol.* **2013**, *26*, 313–339. [CrossRef]
10. Bogdanov, A.; Knežević, M.; Leander, G.; Toz, D.; Varıcı, K.; Verbauwhede, I. SPONGENT: A lightweight hash function. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2011, pp. 312–325.
11. Hirose, S.; Ideguchi, K.; Kuwakado, H.; Owada, T.; Preneel, B.; Yoshida, H. A lightweight 256-bit hash function for hardware and low-end devices: Lesamnta-LW. In *International Conference on Information Security and Cryptology*; Springer: Berlin/Heidelberg, Germany, 2010, pp. 151–168.
12. Rao, V.; Prema, K. Light-weight hashing method for user authentication in Internet-of-Things. *Ad. Hoc. Netw.* **2019**, *89*, 97–106. [CrossRef]
13. Degnan, B.; Rose, E.; Durgin, G.; Maeda, S. A modified simon cipher 4-block key schedule as a hash. *IEEE J. Radio Freq. Identif.* **2017**, *1*, 85–89. [CrossRef]
14. Hirose, S.; Ideguchi, K.; Kuwakado, H.; Owada, T.; Preneel, B.; Yoshida, H. An AES based 256-bit hash function for lightweight applications: Lesamnta-LW. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2012**, *95*, 89–99. [CrossRef]

15. Patrick, C.; Schaumont, P. The role of energy in the lightweight cryptographic profile. In Proceedings of the NIST Lightweight Cryptography Workshop, Gaithersburg, MD, USA, 17–18 October 2016.
16. Dhanda, S.S.; Singh, B.; Jindal, P. Lightweight cryptography: a solution to secure IoT. *Wirel. Pers. Commun.* **2020**, *112*, 1947–1980. [CrossRef]
17. Buchanan, W.J.; Li, S.; Asif, R. Lightweight cryptography methods. *J. Cyber Secur. Technol.* **2017**, *1*, 187–201. [CrossRef]
18. Abed, S.; Jaffal, R.; Mohd, B.J.; Al-Shayeji, M. An analysis and evaluation of lightweight hash functions for blockchain-based IoT devices. *Clust. Comput.* **2021**, *24*, 3065–3084. [CrossRef]
19. Jungk, B.; Lima, L.R.; Hiller, M. A systematic study of lightweight hash functions on FPGAs. In Proceedings of the 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), Cancun, Mexico, 8–10 December 2014; pp. 1–6.
20. Lara-Nino, C.A.; Morales-Sandoval, M.; Diaz-Perez, A. Small lightweight hash functions in FPGA. In Proceedings of the 2018 IEEE 9th Latin American Symposium on Circuits & Systems (LASCAS), Puerto Vallarta, Mexico, 25–28 February 2018; pp. 1–4.
21. Meuser, T.; Schmidt, L.; Wiesmaier, A. Comparing Lightweight Hash Functions–PHOTON & Quark. *Tech. Univ. Darmstadt*. Available online: https://download.hrz.tu-darmstadt.de/media/FB20/Dekanat/Publikationen/CDC/2015-07-06_TR_PhotonQuark.pdf (accessed on 18 August 2024) .
22. Padma, A.; Ramaiah, M. GLSBIoT: GWO-based enhancement for lightweight scalable blockchain for IoT with trust based consensus. *Future Gener. Comput. Syst.* **2024**, *159*, 64–76. [CrossRef]
23. Sakan, K.; Nyssanbayeva, S.; Kapalova, N.; Algazy, K.; Khompysh, A.; Dyusenbayev, D. Development and analysis of the new hashing algorithm based on block cipher. *East.-Eur. J. Enterp. Technol.* **2022**, *116*, 60–73. [CrossRef]
24. Windarta, S.; Suryadi, S.; Ramli, K.; Lestari, A.A.; Wildan, W.; Pranggono, B.; Wardhani, R.W. Two new lightweight cryptographic hash functions based on saturnin and beetle for the Internet of Things. *IEEE Access* **2023**, *11*, 84074–84090. [CrossRef]
25. Mahmoud, M.A.; Gurunathan, M.; Ramli, R.; Babatunde, K.A.; Faisal, F.H. Review and Development of a Scalable Lightweight Blockchain Integrated Model (LightBlock) for IoT Applications. *Electronics* **2023**, *12*, 1025. [CrossRef]
26. Li, W.; Zhang, Q.; Deng, S.; Zhou, B.; Wang, B.; Cao, J. Q-learning improved lightweight consensus algorithm for blockchain-structured internet of things. *IEEE Internet Things J.* **2023**, *11*, 2855–2869. [CrossRef]
27. Achar, S.D.; P, T.; Nandi, S.; Nandi, S. LiteHash: Hash Functions for Resource-Constrained Hardware. *ACM Trans. Embed. Comput. Syst.* 2024, *accepted*. [CrossRef]
28. Singh, S.; Sharma, P.K.; Moon, S.Y.; Park, J.H. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *J. Ambient. Intell. Humaniz. Comput.* **2024**, *15*, 625–1642.
29. Duan, P.; Wang, J.; Zhang, Y.; Ma, Z.; Luo, S. Policy-based chameleon hash with black-box traceability for redactable blockchain in IoT. *Electronics* **2023**, *12*, 1646. [CrossRef]
30. Choi, H.; Choi, S.; Seo, S. Parallel Implementation of Lightweight Secure Hash Algorithm on CPU and GPU Environments. *Electronics* **2024**, *13*, 896. [CrossRef]
31. Dai, X.; Wang, X.; Han, H.; Wang, E. N-dimensional non-degenerate chaos based on two-parameter gain with application to hash function. *Electronics* **2024**, *13*, 2627. [CrossRef]
32. Beaulieu, R.; Shors, D.; Smith, J.; Treatman-Clark, S.; Weeks, B.; Wingers, L. The SIMON and SPECK lightweight block ciphers. In Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, 7–11 June 2015; pp. 1–6.
33. Rivest, R.L.; Robshaw, M.J.; Sidney, R.; Yin, Y.L. The RC6TM block cipher. In Proceedings of the First Advanced Encryption Standard (AES) Conference, Ventura, CA, USA, 20–22 August 1998; p. 16.
34. Gong, Z.; Nikova, S.; Law, Y.W. KLEIN: A new family of lightweight block ciphers. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 1–18.
35. Wu, W.; Zhang, L. LBlock: A lightweight block cipher. In *International Conference on Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 327–344.
36. Suzaki, T.; Minematsu, K.; Morioka, S.; Kobayashi, E. Twine: A lightweight, versatile block cipher. In Proceedings of the ECRYPT Workshop on Lightweight Cryptography, Louvain-la-Neuve, Belgium, 28–29 November 2011; pp. 339–354.
37. Shibutani, K.; Isobe, T.; Hiwatari, H.; Mitsuda, A.; Akishita, T.; Shirai, T. Piccolo: An ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2011; CHES'11; pp. 342–357.
38. Hong, D.; Sung, J.; Hong, S.; Lim, J.; Lee, S.; Koo, B.S.; Lee, C.; Chang, D.; Lee, J.; Jeong, K. HIGHT: A new block cipher suitable for low-resource device. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 46–59.
39. Beaulieu, R.; Shors, D.; Smith, J.; Treatman-Clark, S.; Weeks, B.; Wingers, L. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptol. ePrint Arch.* **2013**, *2013*, 404.
40. Mouha, N.; Mennink, B.; Herrewege, A.V.; Watanabe, D.; Preneel, B.; Verbauwhede, I. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In *Selected Areas in Cryptography*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 306–323.
41. Grosso, V.; Leurent, G.; Standaert, F.X.; Varıcı, K. LS-designs: Bitslice encryption for efficient masked software implementations. In *International Workshop on Fast Software Encryption*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 18–37.
42. Albrecht, M.R.; Driessen, B.; Kavun, E.B.; Leander, G.; Paar, C.; Yalçın, T. Block Ciphers—Focus on the Linear Layer (feat. PRIDE). In *Advances in Cryptology—CRYPTO 2014*; Garay, J.A., Gennaro, R., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; pp. 57–76.

43. Zhang, W.; Bao, Z.; Lin, D.; Rijmen, V.; Yang, B.; Verbauwhede, I. RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms. *Sci. China Inf. Sci.* **2015**, *58*, 1–15. [CrossRef]
44. Beierle, C.; Jean, J.; Kölbl, S.; Leander, G.; Moradi, A.; Peyrin, T.; Sasaki, Y.; Sasdrich, P.; Sim, S.M. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 123–153.
45. Dinu, D.; Perrin, L.; Udovenko, A.; Velichkov, V.; Großschädl, J.; Biryukov, A. *Design Strategies for ARX with Provable Bounds: Sparx and LAX*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 484–513.
46. Nachef, V.; Marrière, N.; Volte, E. Differential Attacks on LILLIPUT Cipher. *IACR Cryptol. ePrint Arch.* **2017**, *2017*, 1121.
47. Beierle, C.; Leander, G.; Moradi, A.; Rasoolzadeh, S. CRAFT: Lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.* **2019**, *2019*, 5–45. [CrossRef]
48. Jebrane, J.; Lazaar, S. ILAPU-Q: An Improved Lightweight Authentication Protocol for IoT Based on U-quark Hash Function. *Recent Adv. Comput. Sci. Commun. (Former. Recent Patents Comput. Sci.)* **2024**, *17*, 78–87. [CrossRef]
49. Stevens, M. Fast Collision Attack on MD5. *Cryptology ePrint Archive*. 2006. Available online: https://eprint.iacr.org/2006/104 (accessed on 1 August 2024).
50. Preneel, B.; Oorschot, P.C.V. MDx-MAC and building fast MACs from hash functions. In *Advances in Cryptology*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 1–14.
51. Vidali, J.; Nose, P.; Pašalić, E. Collisions for variants of the BLAKE hash function. *Inf. Process. Lett.* **2010**, *110*, 585–590. [CrossRef]
52. Secure Hash Standard. *FIPS Pub 180*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 1993.
53. Secure Hash Standard. *FIPS Pub 180-1*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 1995; Volume 17, p. 15.
54. Preneel, B.; Bosselaers, A.; Dobbertin, H. The Cryptographic Hash Function RIPEMD-160. 1997. Available online: https://www.esat.kuleuven.be/cosic/publications/article-317.pdf (accessed on 2 August 2024).
55. Sklavos, N.; Koufopavlou, O. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions. In Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS'03, Bangkok, Thailand, 25–28 May 2003; Volume 5, p. V.
56. Paulo, B.S.; Vincent, R. The WHIRLPOOL Hashing Function. *NESSIE Proj. Propos.* **2000**. Available online: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=664b5286124b28abf2d30a07ba6f9e020f4138fe (accessed on 21 August 2024).
57. Aumasson, J.P.; Meier, W.; Phan, R.C.W.; Henzen, L. *The Hash Function BLAKE*; Springer: Berlin/Heidelberg, Germany, 2014.
58. Rivest, R.L.; Agre, B.; Bailey, D.V.; Crutchfield, C.; Dodis, Y.; Fleming, K.E.; Khan, A.; Krishnamurthy, J.; Lin, Y.; Reyzin, L.; et al. The MD6 hash function–a proposal to NIST for SHA-3. *Submiss. NIST* **2008**, *2*, 1–234.
59. Aumasson, J.P.; Meier, W.; Phan, R.C.W.; Henzen, L. Blake2. In *The Hash Function BLAKE*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 165–183.
60. Gauravaram, P.; Knudsen, L.R.; Matusiewicz, K.; Mendel, F.; Rechberger, C.; Schläffer, M.; Thomsen, S.S. Grøstl-a SHA-3 candidate. In *Dagstuhl Seminar Proceedings*; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Wadern, Germany, 2009; Volume 9031, pp. 1–33. [CrossRef]
61. Ferguson, N.; Lucks, S.; Schneier, B.; Whiting, D.; Bellare, M.; Kohno, T.; Callas, J.; Walker, J. The Skein hash function family. *Submiss. NIST (Round 3)* **2010**, *7*, 3.
62. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. Keccak. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 26–30 May 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 313–314.
63. O'Connor, J.; Aumasson, J.P.; Neves, S.; Wilcox-O'Hearn, Z. BLAKE3: One Function, Fast Everywhere. Available online: https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf (accessed on 15 August 2024).
64. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. On the indifferentiability of the sponge construction. In *Advances in Cryptology*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 181–197.
65. Chen, S.; Jin, C. Preimage Attacks on Some Hashing Modes Instantiating Reduced-Round LBlock. *IEEE Access* **2018**, *6*, 44659–44665. [CrossRef]
66. Flajolet, P.; Gardy, D.; Thimonier, L. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discret. Appl. Math.* **1992**, *39*, 207–229. [CrossRef]
67. Nursi, S. *From the Risale-i Nur Collection: The Flashes*; Nurpublishers: Istanbul, Turkey, 1993; Volume 1.
68. Dworkin, M.J. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*; Federal Inf. Process. Stds. (NIST FIPS); National Institute of Standards and Technology: Gaithersburg, MD, USA, 2015.
69. Abdelfatah, R.I.; Baka, E.A.; Nasr, M.E. Keyed parallel hash algorithm based on multiple chaotic maps (KPHA-MCM). *IEEE Access* **2021**, *9*, 130399–130409. [CrossRef]
70. Abdoun, N.; El Assad, S.; Manh Hoang, T.; Deforges, O.; Assaf, R.; Khalil, M. Designing two secure keyed hash functions based on sponge construction and the chaotic neural network. *Entropy* **2020**, *22*, 1012. [CrossRef] [PubMed]
71. Yang, Y.; Tian, X.; Pei, P.; He, X.; Zhang, X. Novel cryptographic hash function based on multiple compressive parallel structures. *Soft Comput.* **2022**, *26*, 13233–13248. [CrossRef]

72. Wong, K.W. A combined chaotic cryptographic and hashing scheme. *Phys. Lett. A* **2003**, *307*, 292–298. [CrossRef]
73. Bakhtiari, S.; Safavi-Naini, R.; Pieprzyk, J. Cryptographic Hash Functions: A Survey. Technical Report, Citeseer. 1995. Available online: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=07748b0e0a9c601169929a427a327a19ba478101 (accessed on 16 August 2024).