Journal of King Saud University – Computer and Information Sciences 35 (2023) 310–323

HOSTED BY

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

A novel subset-based polynomial design for enhancing the security of short message-digest with inflated avalanche and random responses

P. Karthik [a], P. Shanthibala [a], Akashdeep Bhardwaj [b], Salil Bharany [c], Heejung Yu [d,*], Yousaf Bin Zikria [e,f,*]

[a] Department of Computer Science, School of Engineering and Technology, Pondicherry University, Puducherry, India
[b] Department of Computer Science, University of Petroleum and Energy Studies, Dehradun, India
[c] Department of Computer Engineering & Technology, Guru Nanak Dev University, Amritsar, India
[d] Department of Electronics and Information Engineering, Korea University, Sejong 30019, South Korea
[e] Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, South Korea
[f] Department of Computer Science and Information Technology, Abu Dhabi University, Abu Dhabi, UAE

ARTICLE INFO

Article history:
Received 6 September 2022
Revised 27 November 2022
Accepted 1 December 2022
Available online 7 December 2022

Keywords:
Provably secure subset hash functions
One-way secure subset hash function
polynomial digest for MDC
Random oracle design with polynomial function
Polynomial digest function

ABSTRACT

The data breach and the integrity violation of remote data remain significant issues in the domain of information security. A provably-secure hash function aids in providing solutions to integrity-related issues. Nevertheless, the choice of a provably-secure hash function has to be made with caution from the perspective of security. This research study attempts to identify the weakness of contemporary keyless hash functions and proposes an algorithm called a provably secure subset hash function (PSSHF). The objectives of the studies are reinforcing the internal structure of random oracle (RO), intensifying stochastic deportment, presenting computationally infeasible conditions for reverse decoding, and forestalling block-level and differential attacks through subsets and polynomial functions. The avalanche response of PSSHF is 50.06% and is higher than that of its contemporary variants. Likewise, the Near-collision response of PSSHF is 49.94% and is the least among its other similitudes. The empirical analysis of the effect of avalanche proves the novel design modifies 93.78% of output symbols besides excelling its other counterparts on random behavior. The runtime response proves the PSSHF processes short messages with acceptable delay. Therefore, the proposed PSSHF can be considered a perfect replacement for its similitudes in respect of the short messages for higher security.

1. Introduction

Data security remains one of the most critical concerns in modern computing due to its significant impacts on social and economic domains. The arrival of Internet-based cloud services has completely transmuted the conventional data processing methods. In most cases, the data are not under the physical supervision of the owner which provides ample opportunities for malicious cyberpunk to cause integrity breaches to the stored data. This scenario signifies the fact that data security is treated as an integral part of data processing mechanisms. A hash or message digest function effectively solves the data integrity problem. It takes a discretionary input string, M^*, and generates an output, N, of limited length. Here, N is generally represented in numerical form which is considered an alternative of M^*. Consequently, it could be employed to identify the integrity violation of remote data without bandwidth overhead. The working principle of the digest function is as follows:

$$H(M^*) \rightarrow N,\ |M| > |N| \tag{1}$$

Here, $|.|$ denotes the cardinality of the given set. The hash function performs compression when the input size $|M|$ is larger than the output size $|N|$, i.e., $|M| > |N|$. This property invites the hash function for hash collisions following the pigeon-hole principle. The hash functions are aware of this fact, and they work against hash collisions. Therefore, the security of the cryptographic digest

* Corresponding authors at: Department of Electronics and Information Engineering, Korea University, Sejong 30019, South Korea (H. Yu); Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, South Korea (Y.B. Zikria)
E-mail addresses: heejungyu@korea.ac.kr (H. Yu), yousafbinzikria@ynu.ac.kr (Y. B. Zikria).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2022.12.002
1319-1578/

function relies upon how effectively it prevents hash collisions when it is trialled with distinct inputs. The collision attacks on MD5 and the SHA-160 algorithms forbade the research community not to employ these algorithms for cryptographic applications (Stevens et al., 2017; Wang et al., 2005). The partial attacks on SHA2-256, SHA2-512, and SHA3-512 showcase the structural design flaws of contemporary algorithms (Eichlseder et al., 2014; Kim et al., 2020; Guo et al., 2020). This research study proposes a novel subset-based polynomial design to enhance the security of keyless digests. It employs complex mathematical structures to reinforce the security of random oracle (RO) and enhances random behavior to effectively forestall block-level and differential attacks. It attempts to perform a rigorous empirical investigation on one-way and collision-resistant properties and comprehensively analyses the random behavior of PSSHF at the bit level. This research study has excluded MD5 and SHA-160 algorithms from the analysis as they had been identified for hash collisions.

The paper is organized as follows: Section 2 deals with the literature review. Section 3 advocates the security concerns of the hash function and Section 4 elaborates on the construction principles of PSSHF. Section 5 presents the analysis of experimental results. Section 6 pleads the discussion. Section 7 presents the conclusion.

2. Literature review

The security of the cryptographic hash function relies upon its collision resistance property. To achieve this, the research community has been trying hard to find an ideal design paradigm that performs one-to-one mapping between input and output. Merkle and Damgard independently made such attempts for the first time in history, and they mathematically proved the collision resistance property in 1989 (Damgård, 1998; Merkle, 1989). Merkle adopted a keyless approach for producing manipulation detection codes (MDC) and Damgard employed keyed design to produce the message authentication codes (MAC). Their design principles were together called MD principles and they became the de facto standard for the cryptographic hash functions. All the contemporary standard digest algorithms excluding SHA3 use the MD design principles.

The research community suggested some key guidelines for cryptographic digest functions to foresee higher security. One-way and collision resistance properties are prominent among them, and the non-correlation property also draws its significance for better protection against differential attacks. The compression and ease of computing properties were optionally suggested to enhance the scope of its widespread use (Menezes et al., 2021; Al-Kuwari et al., 2021; Rogaway and Shrimpton, 2004). The design of contemporary digest functions tries to accommodate ease of computing property to enhance efficiency. It encourages the diligent use of bit-wise operators in the internal design. The influence of bit-wise operators helps the hash functions to achieve better efficiency besides making them vulnerable to reverse decoding. The collisions/partial collisions presented in Table 1 bear witness to this fact.

The limited output nature and lack of random behavior are the two serious security concerns for the digest functions. This is because they provide ample scope for cryptanalysts to perform differential analysis. The collision attacks on digest functions successfully proved the digest size of 160-bit is no longer a secure one. Introducing random behavior in the digest output will provide solutions for differential threats (Matsui, 1993). This can be done by associating every input bit of the input string with a significant number of output bits (Kam et al., 2009). To foresee comprehensive security in compliance with the above, the digest algorithm needs to meet the strict avalanche criterion. Accordingly, a niggling change in the input bits ought to modify more than 50 % of the output bits (Sanap and More, 2021). In addition, the security of the digest function also relies upon the size of the private key (Menezes et al., 2021).

Bartkewitz demonstrated the design of the hash function using block ciphers (T., Bartkewitz, 2009). His design employed MD design principles for its construction, and it converted an arbitrary length input string as a mathematical product of its block size. The block size was decided on the size of the digest. This process was called MD strengthening (Lai and Massey, 1992). An initialization vector (IV) was applied to modify the first block of the input to produce an intermediate output H_i and was subsequently used as an IV for the following block to be processed. The procedure was continued until no more blocks were left to process. The output of the end block was considered as the digest for the given input.

Rivest devised an MD4 algorithm for the generation of a 128-bit digest (Rivest et al., 1991). The MD4 is a block cipher that divides the message into blocks of size 512-bit or 64-byte. He came up again with another 128-bit algorithm namely MD5 in 1992 (Rivest, 2021). The new MD5 algorithm was identical to MD4 but it incorporated slender modifications in its design. However, Wang identified collisions in MD4 and MD5 algorithms. Therefore, they are forbidden for cryptographic use (Wang et al., 2005). In the year 2001, Eastlake invented a secure hash algorithm (SHA)-160 or SHA-1 (“rfc3174”, 2021). The SHA-1 is a 160-bit algorithm, and it performs message padding by operating the MD principles. It employs five initialization vectors and four auxiliary functions. The algorithm is operated on 80 rounds and for every 20 rounds, the auxiliary function is changed. Similar to MD4 and MD5, SHA-1 was also identified for producing collisions. Therefore, it was restrained from cryptographic use (Stevens et al., 2017; Wang et al., 2005; Webster et al., 1985).

The SHA-2 was the successor of SHA-1 and was completely different in construction from its primitive version. The SHA-2 consisted of a family of six hash functions with digest sizes ranging from 224-bit to 512-bit. Among the variants, SHA-256 and SHA-512 were important. The remaining variants were the curtailed forms of either SHA-256 or SHA-512. The design used

Table 1
Comprehensive analysis of cryptographic keyless hash functions.

DIGEST FUNCTION	INPUT BLOCKSIZE (Bits)	HASH O/P SIZE (Bits)	STATE VECTORS	STATE VECTOR SIZE (Bits)	COLLISION ATTACKS	NATURE OF ATTACK	TYPE OF CONSTRUCTION
MD5	512	128	4	32	Broken	Differential	RO
SHA-0	512	160	5	32	Broken	Differential	RO
SHA-160	512	160	5	32	Broken	Differential	RO
SHA2- 224/256	512	224	8	32	Partially Broken	Differential	RO
SHA2- 256	512	256	8	32	Partially Broken	Differential	RO
SHA2- 384/512	1024	384	8	64	Partially Broken	Differential	RO
SHA2- 512	1024	512	8	64	Partially Broken	Differential	RO
SHA2- 224/512	1024	224	8	64	Partially Broken	Differential	RO
SHA2- 256/512	1024	256	8	64	Partially Broken	Differential	RO
SHA3- 512	576	512	25	64	Partially Broken	Differential	Sponge

for SHA2-512 was identical to SHA2-256 except that; the size of the IV element used in the SHA-512 was 64-bit. This algorithm has not been broken yet and for this reason, it is widely being employed for the generation of MDC. The collisions found on SHA-160 forced the American National Institute of Standards and Technology (NIST) to go for a modern standard for cryptographic use. The NIST conducted an open contest to introduce a novel standard, and Keccak was declared the winner. The SHA3 engages the sponge principle as a replacement for RO (Bertoni et al., 2013). The SHA3 algorithm is the contemporary standard for digest functions. The urge for finding an ideal design template for the digest function continues and recent research tries to implicate mathematical structures in the design of digest functions to strengthen security. The chaotic function and elliptic curve are the other alternative techniques for the design of a strong one-way digest function (Meshram et al., 2019; Teh et al., 2018; Yu and Wang, 2021). However, the absence of standards in the aforesaid techniques forced this work to exclude them from the analysis.

Table 1 analyzes cryptographic hash functions using other metrics like input block size, size of the output, number of state vectors used, size of the state vector, collision attack remarks, nature of the attack, and the type of construction used. The entries prove all the standard digest functions are vulnerable to differential attacks irrespective of the type of construction employed. In addition, it emphasizes the need for strengthening the random behavior of modern digest functions by following the Matsui claim to counter the differential analysis attack (Matsui, 1993). The entries also prove that any digest of size<160-bit is unsafe for cryptographic applications. Therefore, the output size should be defined to a minimum of 224 bits, to ensure better security.

3. Hash function: Security concerns

The hash function performs the mapping from M to N, such that$|M| > |N|$. Therefore, collision becomes unavoidable in the hash function. If t_1 and t_2 are the numbers of the bits present in the input and output respectively, then there would be $2^{t_1-t_2}$ input messages that create hash collisions. Let n be the size of the digest, then the probability for any two randomly chosen messages to create a second pre-image collision would be $1/2^n$. Therefore, the large value of n reduces the probability of hash collision. It is recommended that the size of the hash output should be chosen to a minimum of 112 bits to survive the Brute-force and birthday attacks (Preneel et al., 1993).

3.1. Structural weakness of MD construction

The structural weakness of MD construction was proved by appending the trailing bits after the terminal block (Bartkewitz, 2009). Accordingly, the added message $m\prime$ would be created with additional blocks $m_{t+1}, m_{t+2}, \cdots, m_{t+n}$ for the given message m. The padded message was then eventually used to launch a pre-image collision attack on MD construction. The application of bit-wise operators at the input blocks generates bizarre intermediate output symbols for the concerned block besides restricting their size to less than or equal to the block size. At this point, preserving one-to-one mapping between input and output would become an extremely difficult task at the block level. The successful 2^k multi-collision attacks launched on MD hashes with a time complexity of $O\left(K2^{\frac{n}{2}}\right)$ proved that the MD construction has structural design flaws (Joux, 2004). The wide-pipe idea suggested by Lucks helps to solve the internal collision problem of MD construction and it enables the design to perform a one-to-one mapping between input and output at the blocks (Lucks, 2004).

3.2. Analysis of crypto attacks

The cryptographic hash functions evince concerns about one-way and collision resistance properties. Many attacks could be launched against these algorithms of which some are generic, and others are algorithm-specific. Generic attacks refer to general attacks and could be launched against all cryptographic hash functions (Bellare et al., 2004; Preneel, 1993). Table 2 gives details about popular generic attacks. Among the generic attacks, the Birthday attack is extremely effective to break any algorithm with reduced iterations.

The hash output ought to contain a minimum of 112 bits to envision strong security attributes (Merkle, 1989). It was also suggested that the hash function demands a lower limit of 2^{80} hash comparisons to survive the brute-force and birthday attacks (Preneel, 1993). Contrarily, the continuous growth in the hardware industry makes all the above claims invalid. At present, an antagonist could launch a successful attack on the 160-bit hash algorithm within a reasonable time. In the year 2005, Wang detected collisions on the SHA-1 algorithm with a theoretical bound of 2^{80} operations (Wang et al., 2005). Over again, he had detected collisions on the SHA-1 algorithm with a complexity of fewer than 2^{69} operations (Webster et al., 1985). Identically, several collisions had been detected on 128-bit digest functions (Wang et al., 2005). The aforesaid attacks prove the output size heavily impacts the security of the hash function. The analysis of generic attacks in Table 2 reveals a fact that higher digest size makes the digest functions less vulnerable to hash collisions. Further, the application of bit-wise operators improves the efficiency of digest functions by demanding fewer clock cycles. However, they will give room for the cryptanalyst to decode the intermediate digests at the blocks. On the contrary, the RO design that implements the other alternative techniques like polynomial function, chaotic function, and elliptic curve would pose more challenging conditions for the cryptanalysis to decode the intermediate digest at the block level.

3.3. Design of RO

The security of the cryptographic hash function depends on the efficient design of the RO model. Merkle remarked on the desirable properties of the RO model. Accordingly, an RO model should be designed like a black box such that, the output returned by the model should be completely different from the given input

Table 2
Various generic attacks on cryptographic hash functions.

Attack Type	No of Trials	Collision Probability	Remarks
Brute Force	2^n	$1/2^n$	n: Size of the input bits.
Birthday Attack	$2^{n/2}$	$q^2/\ 2^n$	q: Number of queries n: Size of the output bits
Random attack	1	$1/2^n$	n: Size of the output bits
Exhaustive Key Search	M+ ($2^{\ k}$-1) / ($1-2^{-n}$)	$\left(1-\frac{1}{2^n}\right)\sum_{i=1}^{M}\frac{i}{2^{n(i-1)}} < \frac{1}{1-2^{-n}}$	M: Message size N: Digest size K: Key size

(Merkle, 1989). It was successfully proved for the first time that Oracle access could be replaced with the RO model. Consequently, any iterative hash function that follows the RO model could be proven as a provably secure hash function (Bellare and Rogaway, 1993). J S Coronchallenged Bellare's claim, and he proved that the argument of no structural flaw in the design of the iterative hash model was false (Coron et al., 2005). Many algorithms that were considered secure in the RO model were proven to be insecure in their concrete effectuation (Bellare et al., 2004; Cannetti et al., 2015).

4. Provably secure subset hash functions (PSSHF)

The conventional iterative hash functions process the input strings as blocks. They perform bit manipulation using bitwise operators. These operators are simple to use, and they typically demand fewer clock cycles. They provide ample scope for the cryptanalyst to conduct as many trials as possible to perform correlation analysis on the intermediate hash outputs to launch block-level attacks. To mitigate this issue, the current design proposes a subset-based polynomial paradigm. PSSHF employs 11 subsets. Among the subsets, one subset contains all the elements and will act as a modifier. The remaining 10 subsets are formed as a pair of disjoint subsets. This algorithm works in three stages as follows:

- Pre-processing
- Hash generation of subsets
- Grouping

4.1. Pre-processing

The hash generation of PSSHF begins with pre-processing. It is composed of three levels namely parsing, MD-strengthening, and crippling. Pre-processing helps the PSSHF in two ways. First, it converts the subset elements into unintelligible byte values. Second, it modifies the input size as a mathematical product of the block size.

- **Parsing**

The PSSHF forms subsets through parsing. At this level, the input string is parsed as an individual byte and is added to independent subsets by agreeing on the predefined conditions as given in Table 3. The elements of the subsets are stored in autonomous arrays to make them suitable for independent processing. Let S represents the set of input symbols. The elements of S are presented as

$$S = \{x_1, x_2, x_3, x_4, \cdots, x_n\} \tag{3}$$

$$|S| = n.$$

Parsing forms 11 subsets namely A, Φ, Φ', Θ, Θ', Π, Π', Δ, Δ',Ω, and Ω'. Among the subsets, set A contains all the elements of S. The remaining 10 subsets are formed as pairs of 5 disjoint subsets namely {Φ,Φ'}, {Θ, Θ'}, {Π, Π'}, {Δ, Δ'}, and {Ω, Ω'}. The subsets Θ, Θ', Π and Π' are formed through patterns using the position of the input elements as illustrated in Fig. 1. Table 4 presents the mathematical conditions involved in parsing the subset elements from the given input string. The sets Θ, Θ', Π, and Π' follow the sine wave pattern. The set Θ is constructed using the recurrence pattern {3,1} and it starts with index 0. The elements of the set Θ' are formed using the recurrence pattern {1,3} and it starts with index position 1. The sets Π and Π' follow the recurrence patterns {3,2,1,2} and {1,2,3,2} respectively and they have additional indices 0 and 1, respectively. The subsets Δ and Ω are generated through odd and even indices that start with even and odd positions respectively. The subsets Δ' and Ω' are the left-out elements of the sets Δ and Ω, respectively. The sample indices generated by the 11 subsets are given in Table 5.

- **MD-strengthening**

This remains the second level of pre-processing in which the individual subsets are applied MD-strengthening. The conventional standard message digests, namely MD4, MD5, SHA-160, and the variants of SHA2, hold 64/128 bits from the rear end to record the length value of the input data. This causes the aforesaid algorithms to fail to process a message when its size surpasses $2^{64}/2^{128}$ bits. The PSSHF accepts the arguments put forth by Preneel on MD strengthening (Preneel, 1998). However, it overcomes the structural flaw through a dynamic reservation policy for storing length attributes. The dynamic reservation enables the PSSHF to litigate any size arbitrary message. In addition, it prevents the PSSHF to work on an extra block when the size of the terminal block contains 448-bit for a 512-bit block or 896-bit for a 1024-bit block. In the proposed design, the t^{th} block, M_t, would appear as shown in Fig. 2.

- **Crippling**

At this level, the elements of subsets are fed into the crippler that transforms the given input elements into an unintelligible byte array through IVs. It employs 11 IVs for 11 subsets such that each IV involves separate 8 elements of each 64-bit in size. The IVs are applied to the individual subsets to convert the subset elements into an unintelligible format as given in Fig. 3.

The crippler divides the subset elements into blocks of 512-bit. These blocks are then subdivided into 8 sub-blocks of 64-bit in size. Here, the blocks $B1$ to $B8$ represent the sub-blocks and the blocks from A to H represent the 8 IV elements. The individual array elements of the sub-blocks are XORed with the elements of

Table 3
Formation of subsets for the PSSHF.

Subsets with Mathematical Conditions	Remarks
A= {X | A ⊆ S, |A| = |S|}	Modifier
Φ= {X| Φ ⊂ S, |Φ|=*n*/2,Φ ∩ Φ'= ∅,ΦUΦ'=S,Φ ≠ Θ,Φ ≠ Θ',Φ ≠ Π,Φ ≠ Π',Φ ≠ Δ,Φ ≠ Δ',Φ ≠ Ω,Φ ≠ Ω'}	Disjoint Pair
Φ'= {X|Φ'⊂S, |Φ'|=*n*/2,Φ ∩ Φ'=∅, ΦUΦ'=S, Φ'≠Θ, Φ'≠Θ', Φ'≠Π, Φ'≠Π', Φ'≠Δ, Φ'≠Δ',Φ'≠Ω, Φ'≠Ω'}	
Θ= {X| Θ ⊂ S, |Θ|= *n*/2, Θ ∩ Θ'=∅, ΘUΘ'=S, Θ ≠ Φ, Θ ≠ Φ', Θ ≠ Π, Θ ≠ Π', Θ ≠ Δ, Θ ≠ Δ', Θ ≠ Ω, Θ ≠ Ω'}	Disjoint Pair
Θ'= {X| Θ'⊂S, |Θ'|=*n*/2, Φ ∩ Φ'=∅, ΘUΘ'=S, Θ'≠Φ, Θ'≠Φ', Θ'≠Π, Θ'≠Π', Θ'≠Δ, Θ'≠Δ', Θ'≠Ω, Θ'≠Ω'}	
Π= {X| Π ⊂ S, |Π|=*n*/2, Π ∩ Π'=∅, ΠUΠ'=S, Π ≠ Φ, Π ≠ Φ', Π ≠ Θ,Π ≠ Θ', Π ≠ Δ, Π ≠ Δ',Π ≠ Ω, Π ≠ Ω'}	Disjoint Pair
Π'= {X| Π'⊂S, |Π'|= *n*/2, Π ∩ Π'=∅, ΠUΠ'=S, Π'≠Φ, Π'≠Φ', Π'≠Θ, Π'≠Θ', Π'≠Δ, Π'≠Δ', Π'≠Ω, Π'≠Ω'}	
Δ= {X| Δ ⊂ S, |Δ|=*n-n/3*, Δ ∩ Δ' = ∅, ΔUΔ'=S, Δ ≠ Φ, Δ ≠ Φ', Δ ≠ Θ, Δ ≠ Θ', Δ ≠ Π, Δ ≠ Π', Δ ≠ Ω, Δ ≠ Ω'}	Disjoint Pair
Δ'= {X| Δ'⊂ S, |Δ'|=*n*/3, Δ ∩ Δ'=∅, ΔUΔ'=S, Δ'≠Φ, Δ'≠Φ', Δ'≠Θ, Δ'≠ Θ', Δ'≠Π, Δ'≠Π', Δ'≠Ω, Δ'≠Ω'}	
Ω= {X| Ω ⊂ S, |Ω|=*n-n/3*, Ω ∩ Ω'=∅, ΩUΩ'=S, Ω ≠ Φ, Ω ≠ Φ', Ω ≠ Θ, Ω ≠ Θ', Ω ≠ Π, Ω ≠ Π', Ω ≠ Δ, Ω ≠ Δ'}	Disjoint Pair
Ω'= {X| Ω'⊂S, |Ω'|=*n-n/3*, Ω ∩ Ω'=∅, ΩUΩ'=S, Ω'≠Φ, Ω'≠Φ', Ω'≠Θ, Ω'≠Θ', Ω'≠Π, Ω'≠Π', Ω'≠Δ, Ω'≠Δ'}	

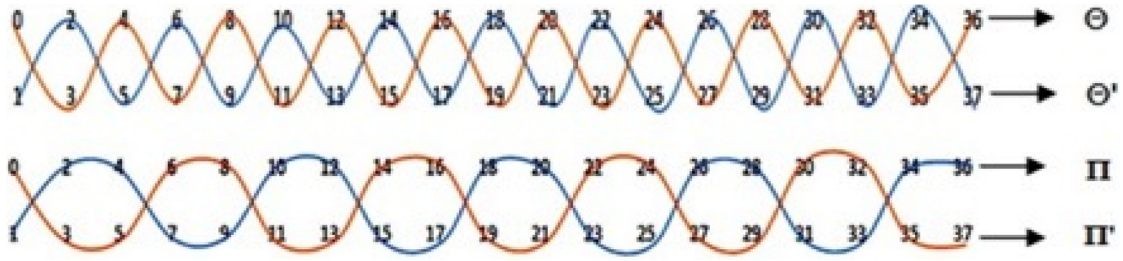

Fig. 1. Formation of sets Θ,Θ', Π and Π' using index positions.

Table 4
Generation of indices through mathematical functions.

S. No	Subset Name	Mathematical Functions Employed	Initial Condition
1	A	$a_n = a_1+(n-1)$	$a_1 = 0$
2	Φ	$a_n = a_1+(n-1)^* 2$	$a_1 = 0$
3	Φ'	$a_n = a_1+(n-1)^* 2$	$a_1 = 1$
4	Θ	$f(a_n, b_n) = \begin{cases} a_n = +a_1(n-1)*4 \\ b_n = +b_1(n-1)*4 \end{cases}$	$a_1 = 0$ $b_1 = 3$
5	Θ'	$f(a_n, b_n) = \begin{cases} a_n = +a_1(n-1)*4 \\ b_n = +b_1(n-1)*4 \end{cases}$	$a_1 = 1$ $b_1 = 2$
6	Π	$f(a_n, b_n, c_n, d_n) = \begin{cases} a_n = +a_1(n-1)*8 \\ b_n = +b_1(n-1)*8 \\ c_n = +c_1(n-1)*8 \\ d_n = +d_1(n-1)*8 \end{cases}$	$a_1 = 3$ $b_1 = 5$ $c_1 = 6$ $d_1 = 8$
7	Π'	$f(a_n, b_n, c_n, d_n) = \begin{cases} a_n = +a_1(n-1)*8 \\ b_n = +b_1(n-1)*8 \\ c_n = +c_1(n-1)*8 \\ d_n = +d_1(n-1)*8 \end{cases}$	$a_1 = 2$ $b_1 = 4$ $c_1 = 7$ $d_1 = 9$
8	Δ	$f(a_n, b_n) = \begin{cases} a_n = +a_1(n-1)*3 \\ b_n = +b_1(n-1)*3 \end{cases}$	$a_1 = 0$ $b_1 = 1$
9	Δ'	$a_n = a_1+(n-1)^* 3$	$a_1 = 2$
10	Ω	$f(a_n, b_n) = \begin{cases} a_n = +a_1(n-1)*4 \\ b_n = +b_1(n-1)*4 \end{cases}$	$a_1 = 1$ $b_1 = 2$
11	Ω'	$a_n = a_1+(n-1)^* 3$	$a_1 = 0$

IV to produce an unintelligible chunk of 512 bits. If the input data exceeds 512-bit, then circular right shift and left shift operations would be performed on IV at both the array level and element level respectively for the separate block to be processed. The independent element in the IV performs 3 circular left shift operations and afterwards, the IV array elements circularly shifted one position right. This would ensure the adjacent block receives a completely different IV than the former block. The aforementioned process is continued till no more blocks are left out. At the end of this step, the subset elements are converted as unintelligible bytes. The output of the crippler is then given to round function $f(\cdot)$ of n^{th} degree two-variable polynomial function (Chaudhry et al., 2022; Bharany et al., 2022; Akram, 2021; Bharany et al., 2022; Javed et al., 2022).

4.2. Hash generation of subsets

The hash generation of subsets takes the pre-processed subsets and processes them separately using the n^{th} degree polynomial function. The application of the polynomial function benefices the PSSHF in the following ways. The polynomial outputs cannot be decoded. Therefore, the PSSHF naturally acquires the one-way property. Similarly, there are no universal methods available to solve a polynomial function having a degree greater than 4 (Pan, 1997). Therefore, it is computationally infeasible to solve the polynomial function of degree 64 as given in the following equation.

$$f(x) = a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4 + \cdots + a_nx^n \tag{4}$$

The PSSHF further complicates this process by adding one more polynomial variable y of degree n as follows:

$$f(x,y) = a_1x^ny^1 + a_2x^{n-1}y^2 + a_3x^{n-2}y^3 + a_4x^{n-3}y^4 + \cdots + a_nx^1y^n \tag{5}$$

where n denotes the block size in bytes and the elements of the block play the role of the coefficients between a_1 and a_n. The function f is a round function and it decides the output of the current block using the former block. This process is continued till no more blocks are left out. The values for x and y are distinct and are assigned values using the prime numbers between 100 and 1000. It was observed that the power of the prime numbers used in Eq. (5) introduces some unpredictable random behavior in the hash output. This helps the PSSHF to provide formidable resistance to differential analysis.

The proposed design adopts the wide-pipe idea proposed by Lucks (Lucks, 2004). However, it differs from the conventional design in deriving the required bits for the digest. The algorithm processes the input block by block and allows each block to produce an intermediate output greater than the required size. The output size of the intermediate blocks is typically produced between 150 and 190 nibbles. The PSSHF finally generates a 256-bit hash from the intermediate result of the terminal block. To do so, the PSSHF finds the midpoint of the intermediate hash output

Table 5
Sample indices generated for various subsets.

S. No	Subset	Sample Indices Generated																	
1	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
2	ϕ	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	...
3	ϕ'	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	...
4	Θ	0	3	4	7	8	11	12	15	16	19	20	23	24	27	28	31	32	...
5	Θ'	1	2	5	6	9	10	13	14	17	18	21	22	25	26	29	30	33	...
6	∏	0	3	5	6	8	11	13	14	16	19	21	22	24	27	29	30	32	...
7	∏'	1	2	4	7	9	10	12	15	17	18	20	23	25	26	28	31	33	...
8	Δ	0	1	3	4	6	7	9	10	12	13	15	16	18	19	21	22	24	...
9	Δ'	2	5	8	11	14	17	20	23	26	29	32	35	38	41	44	47	50	...
10	Ω	1	2	4	5	7	8	10	11	13	14	16	17	19	20	22	23	25	...
11	Ω'	0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	...

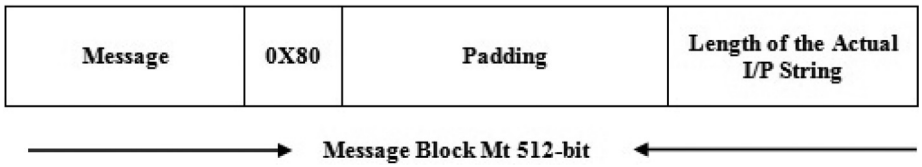

Fig. 2. The structure of the last block M_t after MD strengthening.

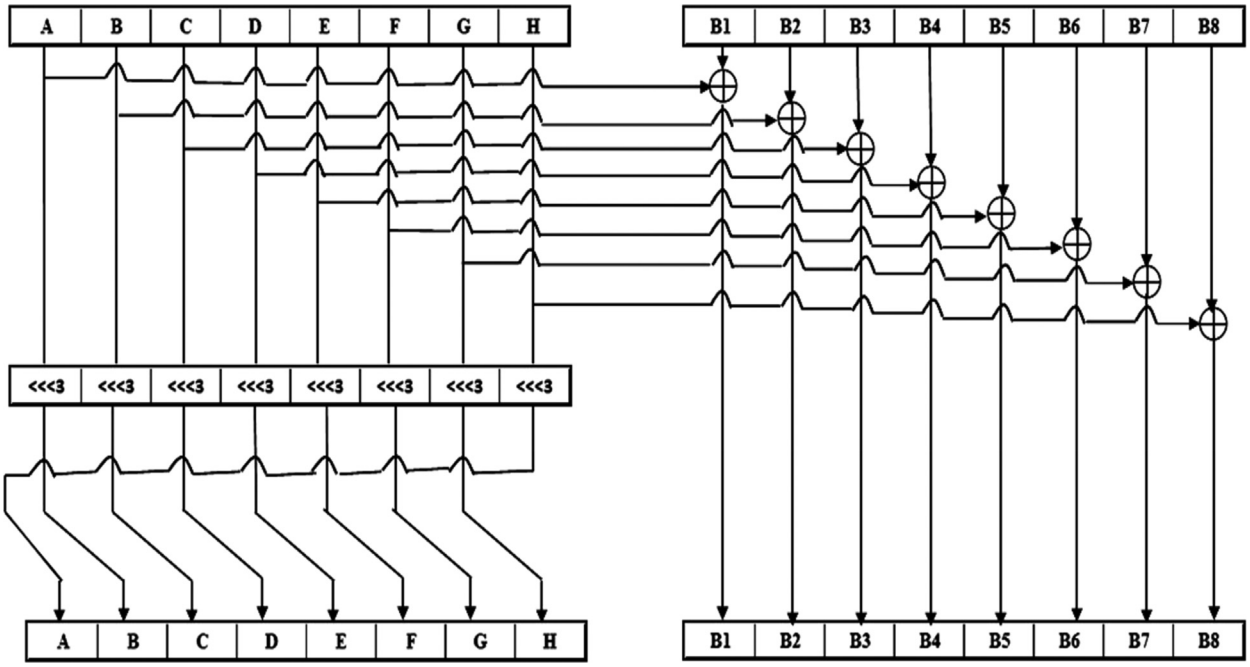

Fig. 3. Processing the block elements using IV.

using its length. It later chooses a pair of 64 nibbles $h1$ and $h2$ by taking the first 64 nibbles in either direction from the midpoint. The final intermediate block output is the XOR of $h1$ and $h2$. The sequence of operations performed at the terminal block is illustrated in Fig. 4.

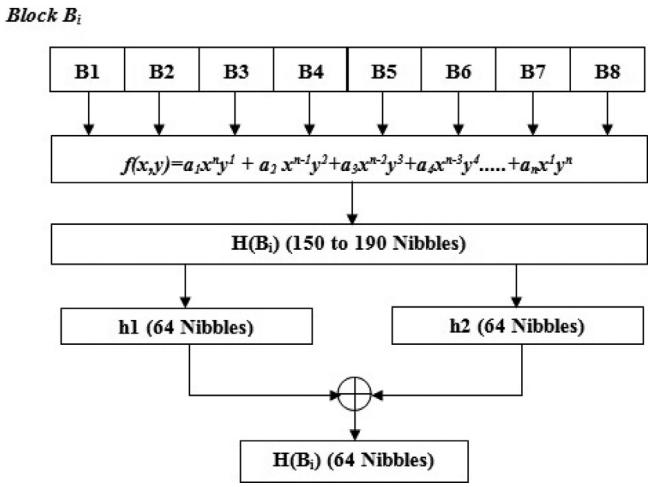

Fig. 4. Hash Generation of block B_i

4.3. Grouping

At this stage, the required 512-bit digest is formed by grouping the subsets. Grouping is performed such that the disjoint subsets fall into a distinct group.

The hash outputs of the subsets in the distinct group are XORed with each other to produce a pair of 256-bit intermediate hash outputs. These outputs are then individually XORed with the hash output of the subset A to produce the final pair $H(T1)$ and $H(T2)$ of 256-bit each. The concatenation of the $H(T1)$ and $H(T2)$ would be the final hash $H(S)$ and it is 512-bit in size. Fig. 5 illustrates the working principle of the PSSHF. The objective of the current design to employ 11 subsets is to present computationally infeasible conditions to the cryptanalyst to find a hash collision through an alternate message A'. Under these circumstances, the only feasible way left for the cryptanalyst to perform the aforementioned task is to generate as many $A\prime$ by modifying the elements of A at distinct locations. However, each of such attempts has to be performed without affecting the subsets A, Φ,Φ', Θ, Θ', Π, Π', Δ, Δ', Ω, and Ω'. This is practically infeasible because the PSSHF is designed in such a way that it would strike the outputs to a minimum of 6 subsets for an input bit-change at any location. Further, the current design forces the cryptanalyst to perform an input–output correlation on 11 subsets for every single try with $A\prime$. This task is computationally infeasible as the subsets produce unpredictable responses for any trivial change in the input. So, it is believed that the current design is going to present extremely challenging conditions for the cryptanalyst to decode the digest (Bharany et al., 2022; Ashraf, 2022; Bharany et al., 2022).

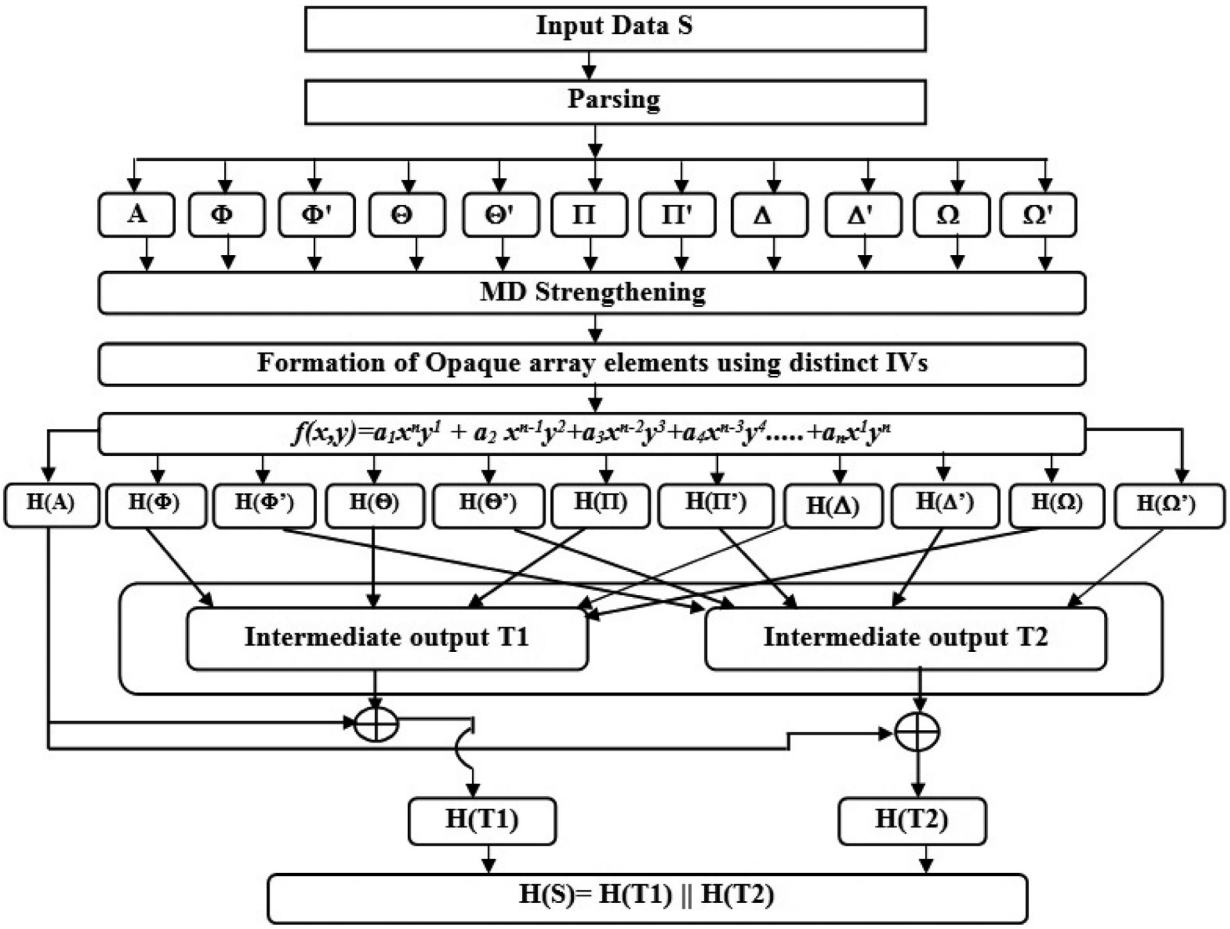

Fig. 5. Working principle of PSSHF.

5. Analysis of experimental results

5.1. Empirical analysis

PSSHF is a 512-bit digest function. Therefore, launching generic attacks against the algorithm is computationally impracticable. However, the empirical analysis of the PSSHF on collision resistance, pre-image resistance, and second pre-image resistance was extensively performed within the physical limits of the system. The analysis was equally extended to examine the random behavior of the PSSHF. All the analyses include only the standard digest functions and the non-standard digest functions were excluded from the comparative analyses. To perform the analyses, Intel(R) Core(TM) i3 6006U CPU @ 2.00 GHz processor was used. Windows-8 64-bit OS and JDK 10.0.1 were employed for the design and analyses of the PSSHF.

- **Analysis of collision/pre-image resistance through bit manipulation**

PSSHF was precisely analyzed for its resistivity on collision/pre-image collisions with more than 3.78 million hash values. To generate the huge volume of hashes the input data was chosen up to 64 blocks such that the processing requirements for the task lie within the physical limits of the system used. Every single byte of the sample data was modified for all the possible values and the corresponding PSSHF responses were compared with the reference value for the possible collision/pre-image collision as given in Table 6. The extensive experimental analysis establishes the fact that the PSSHF vigorously defies the Collision/Pre-image collision. The analysis was equally extended to contemporary standard digest functions. The result proves the PSSHF behavior is identical to any other contemporary standard digest function.

- **Analysis of collision/pre-image collision through byte interchanges**

PSSHF was again analyzed for its resistivity on collision/pre-image collisions through byte interchanges. To generate a vast number of PSSHF responses the input data was chosen in up to 48 blocks such that the individual byte of the sample data was exchanged with other bytes as given in Table 7. The PSSHF response corresponding to the separate interchange was compared with the reference value for the possible collision/pre-image collision. The careful investigation with over 11.14 million hash comparisons proves the fact that the PSSHF robustly resists the collision/pre-image collision. The analysis was also extended to standard digest functions and the results prove the PSSHF behavior is indistinguishable from its other standard counterparts. The combined analyses of bit manipulation and byte interchanges prove the efficacy of PSSHF on Collision and Pre-image resistance for all sorts of generic data manipulation attempts.

- **Analysis of second pre-image resistance**

This experiment was conducted to identify a possible hash collision between any two arbitrarily selected distinct messages namely $M1$ and $M2$. To do so, the input sample was chosen with

Table 6
A comparative analysis of PSSHF on Collision and pre-image resistance properties.

S No	Size of the sample input (Blocks)	Size of the data (Bytes)	Total Hash Comparisons made (Worst Case)	Collisions Count on PSSHF and SHA variants											Research outcome
				PSSHF-512	SHA2						SHA3				
					224/256	256/256	224/512	256/512	384/512	512/512	224/512	256/512	384/512	512/512	
1	< 1	46	11,730	0	0	0	0	0	0	0	0	0	0	0	PSSHF strongly resists Collision and Pre-image Collisions and its behavior is identical to standard digest functions.
2	1	64	16,320	0	0	0	0	0	0	0	0	0	0	0	
3	2	128	32,640	0	0	0	0	0	0	0	0	0	0	0	
4	4	256	65,280	0	0	0	0	0	0	0	0	0	0	0	
5	6	384	97,920	0	0	0	0	0	0	0	0	0	0	0	
6	8	512	130,560	0	0	0	0	0	0	0	0	0	0	0	
7	10	640	163,200	0	0	0	0	0	0	0	0	0	0	0	
8	12	768	195,840	0	0	0	0	0	0	0	0	0	0	0	
9	14	896	228,480	0	0	0	0	0	0	0	0	0	0	0	
10	16	1024	261,120	0	0	0	0	0	0	0	0	0	0	0	
11	18	1152	293,760	0	0	0	0	0	0	0	0	0	0	0	
12	20	1280	326,400	0	0	0	0	0	0	0	0	0	0	0	
13	24	1536	391,680	0	0	0	0	0	0	0	0	0	0	0	
14	32	2048	522,240	0	0	0	0	0	0	0	0	0	0	0	
15	64	4096	1,044,480	0	0	0	0	0	0	0	0	0	0	0	
Total Hash Comparisons			**3,781,650**												

Table 7
Relative analysis of PSSHF on Collision/Pre-image resistance.

S No	Size of the sample input (Blocks)	Size of the data (Bytes)	Total Hash Comparisons made (Worst Case)	Collisions Count on PSSHF and SHA variants											Research outcome
				PSSHF-512	SHA2						SHA3				
					224/256	256/256	224/512	256/512	384/512	512/512	224/512	256/512	384/512	512/512	
1	< 1	46	1035	0	0	0	0	0	0	0	0	0	0	0	PSSHF strongly resists Collision and Pre-image Collisions and its behavior is identical to standard digest functions.
2	1	64	2016	0	0	0	0	0	0	0	0	0	0	0	
3	2	128	8128	0	0	0	0	0	0	0	0	0	0	0	
4	4	256	32,640	0	0	0	0	0	0	0	0	0	0	0	
5	6	384	73,536	0	0	0	0	0	0	0	0	0	0	0	
6	8	512	130,816	0	0	0	0	0	0	0	0	0	0	0	
7	10	640	204,480	0	0	0	0	0	0	0	0	0	0	0	
8	12	768	294,528	0	0	0	0	0	0	0	0	0	0	0	
9	14	896	400,960	0	0	0	0	0	0	0	0	0	0	0	
10	16	1024	523,776	0	0	0	0	0	0	0	0	0	0	0	
11	18	1152	662,976	0	0	0	0	0	0	0	0	0	0	0	
12	20	1280	818,560	0	0	0	0	0	0	0	0	0	0	0	
13	24	1536	1,178,880	0	0	0	0	0	0	0	0	0	0	0	
14	32	2048	2,096,128	0	0	0	0	0	0	0	0	0	0	0	
15	48	3072	4,717,056	0	0	0	0	0	0	0	0	0	0	0	
Total Hash Comparisons			**11,145,515**												

Table 8
Analysis of Second pre-image resistance with standard digest functions.

S No	No. of Samples Taken	Hash Comparisons made	Collisions Count on PSSHF and SHA variants											Research outcome
			PSSHF-512	SHA2						SHA3				
				224/256	256/256	224/512	256/512	384/512	512/512	224/512	256/512	384/512	512/512	PSSHF produces zero
1	1000	500	0	0	0	0	0	0	0	0	0	0	0	collision and exhibits formidable
2	2000	1000	0	0	0	0	0	0	0	0	0	0	0	Second pre-image resistance
3	3000	1500	0	0	0	0	0	0	0	0	0	0	0	
4	4000	2000	0	0	0	0	0	0	0	0	0	0	0	
5	5000	2500	0	0	0	0	0	0	0	0	0	0	0	
6	6000	3000	0	0	0	0	0	0	0	0	0	0	0	
7	7000	3500	0	0	0	0	0	0	0	0	0	0	0	
8	8000	4000	0	0	0	0	0	0	0	0	0	0	0	
9	9000	4500	0	0	0	0	0	0	0	0	0	0	0	
10	10,000	5000	0	0	0	0	0	0	0	0	0	0	0	
11	15,000	7500	0	0	0	0	0	0	0	0	0	0	0	
12	20,000	10,000	0	0	0	0	0	0	0	0	0	0	0	
13	25,000	12,500	0	0	0	0	0	0	0	0	0	0	0	
14	30,000	15,000	0	0	0	0	0	0	0	0	0	0	0	
Total	**145,000**	**72,500**												

varying lengths between 1 and 100,000 bytes. The samples were generated programmatically and stored in separate files. A random generator was operated to pick any two files randomly from the sample space. The selected files were then given to the PSSHF for a Second pre-image collision. The experiment was conducted with 145,000 random samples with 72,500 hash comparisons. The experiment was also extended to contemporary digest functions. The results presented in Table 8 prove the PSSHF exhibits a strong second pre-image resistance and it behaves like contemporary digest algorithms.

- **Analysis of confusion and diffusion on avalanche response**

The confusion and diffusion analysis is intended to study the avalanche demeanor of the PSSHF at the bit level. Accordingly, a digest function should unvaryingly affect the output bits even for a trivial change in the input (Motara and Irwin, 2016). To examine the avalanche behavior of the PSSHF, an input string of size 128 bytes was employed and was alternated a bit at distinct locations. The positional bit changes in the PSSHF response to the reference output were marked for the avalanche analysis. Identically, the transposed output-bit locations were marked to further investigate the avalanche response. The outcome of the analysis proves the following observations:

- The avalanche response of the PSSHF remains uniform all over the digest.
- The output of the PSSHF is unpredictable and random.
- PSSHF fulfils the strict avalanche condition.

The analysis was further extended to the standard digest functions for performing a comparative analysis as given in Table 9 using 3 K data. The individual table entry represents the average avalanche response derived from 500 hash comparisons. The result proves the average avalanche response of the PSSHF for distinct input bit/byte changes is 50.06 % and is the level best value among the other counterparts. Fig. 6 proves the consistency of the PSSHF response on confusion and diffusion irrespective of the changes in the input data. The relative analysis for a single input bit-change is shown in Fig. 7. It proves the mean response of PSSHF is 50.01 % and is the second-highest value among the other counterparts. However, the PSSHF excels in the 512-bit category and is presented in Fig. 8. Fig. 9 presents the relative analysis of PSSHF with other short digest functions. The analysis proves the PSSHF excels on the avalanche property and it levels the performance of the SHA2-384-bit algorithm. The result also proves the PSSHF meets the strict avalanche criteria by consistently flipping more than

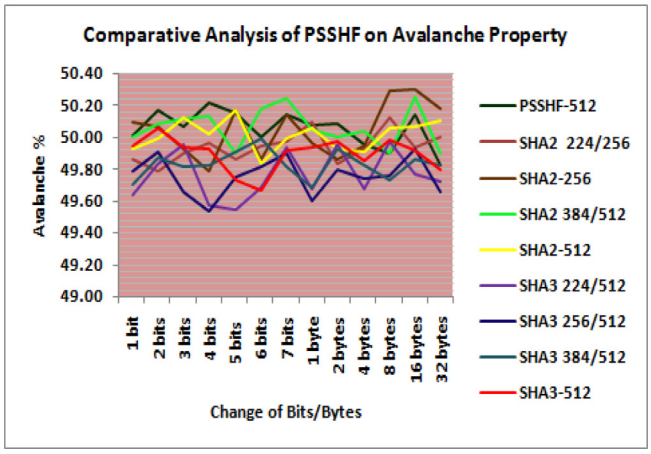

Fig. 6. Comparative analysis of PSSHF on Avalanche response for 3 K data.

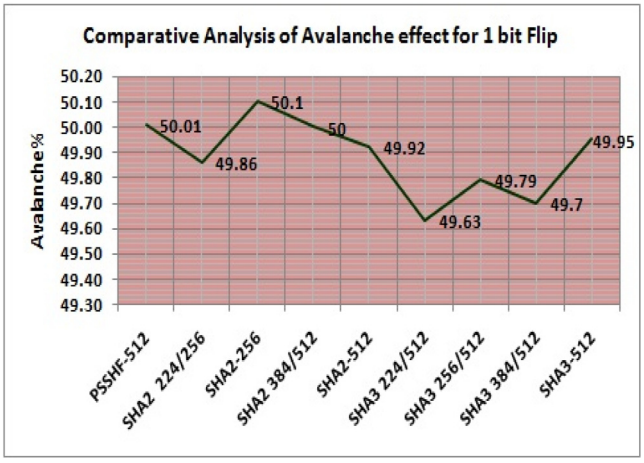

Fig. 7. Comparative study of avalanche response for an input bit-change.

50 % of output bits (Sanap and More, 2021; Yi, 2005). Therefore, launching a differential attack against the PSSHF is extremely hard.

- **Analysis of Confusion and diffusion on near-collision Response**

The objective of the experiment is to analyze the PSSHF response to near-collision at the 128-bit level. However, a compre-

Table 9
Comparative analysis of Confusion and Diffusion for 3 K data.

S.No	Input Bits/Bytes Changed	PSSHF-512	SHA2 Variants				SHA3 Variants			
			224/256	256/256	384/512	512/512	224/512	256/512	384/512	512
1	1 bit	50.01	49.86	50.1	50	49.92	49.63	49.79	49.7	49.95
2	2 bits	50.17	49.79	50.07	50.08	49.99	49.83	49.91	49.87	50.06
3	3 bits	50.07	49.9	49.93	50.11	50.12	49.95	49.65	49.81	49.94
4	4 bits	50.22	49.96	49.79	50.13	50.02	49.57	49.53	49.82	49.93
5	5 bits	50.15	49.86	50.17	49.9	50.17	49.54	49.75	49.91	49.73
6	6 bits	50.00	49.95	49.86	50.18	49.83	49.68	49.82	49.99	49.66
7	7 bits	50.14	49.97	50.14	50.24	49.99	49.93	49.9	49.81	49.92
8	1 byte	50.08	50.09	49.97	50.05	50.06	49.67	49.6	49.68	49.94
9	2 bytes	50.09	49.84	49.86	50	49.92	49.95	49.8	49.92	49.97
10	4 bytes	49.95	49.93	49.95	50.04	49.91	49.67	49.74	49.82	49.85
11	8 bytes	49.90	50.12	50.29	49.9	50.06	49.97	49.76	49.73	49.98
12	16 bytes	50.14	49.94	50.3	50.25	50.07	49.76	49.93	49.86	49.92
13	32 bytes	49.82	50	50.18	49.9	50.1	49.72	49.65	49.82	49.79
Average Avalanche (%)		**50.06**	**49.94**	**50.05**	**50.06**	**50.01**	**49.76**	**49.76**	**49.83**	**49.90**

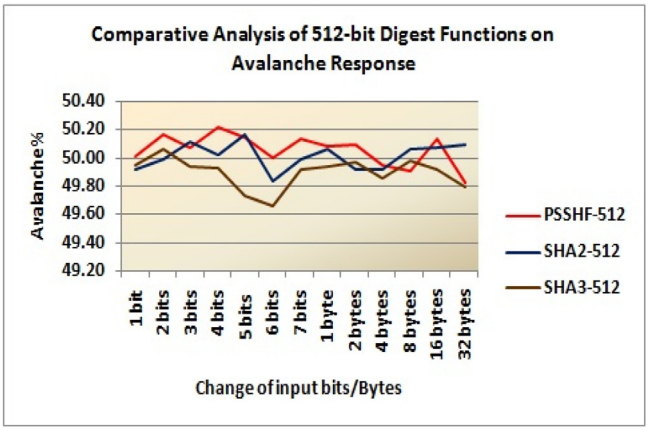

Fig. 8. Analysis of 512-bit digest function at bit-level.

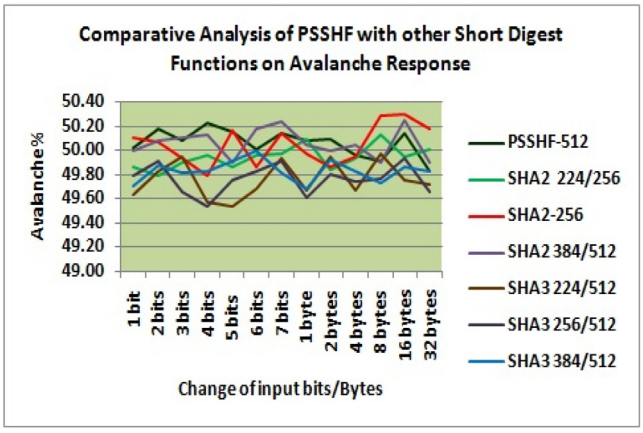

Fig. 9. Comparative study of PSSHF with other short output digest functions.

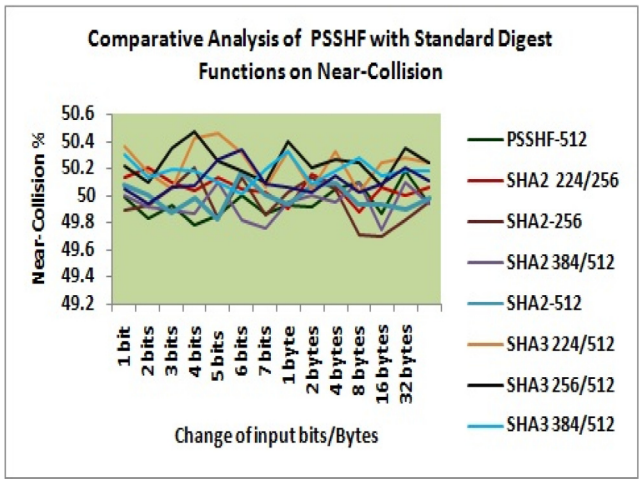

Fig. 10. Comparative analysis of the PSSHF with standard digest functions on near-collision.

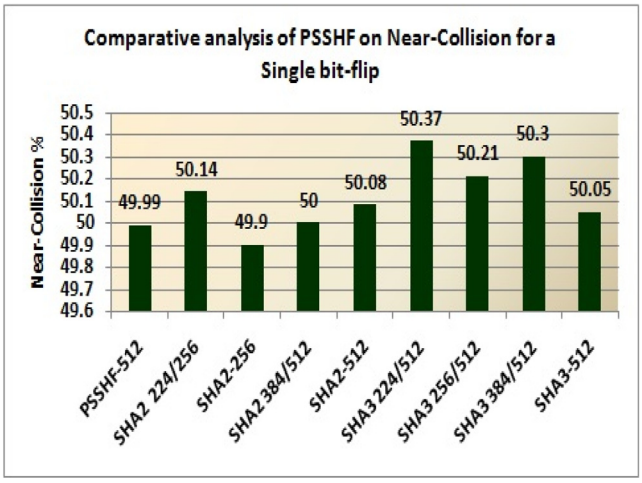

Fig. 11. Near-collision analysis for a single bit-flip.

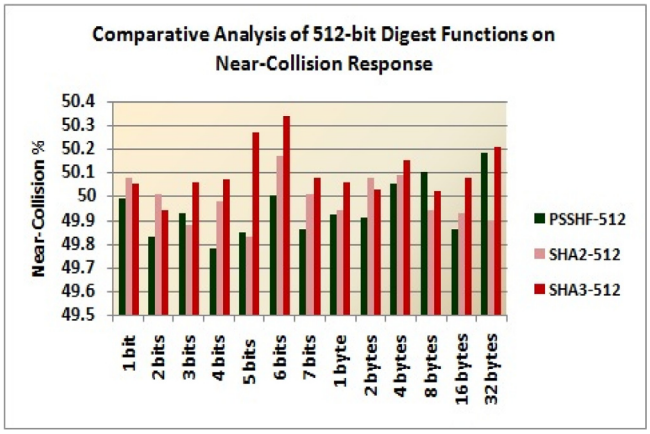

Fig. 12. Near-collision analysis of 512-bit digests.

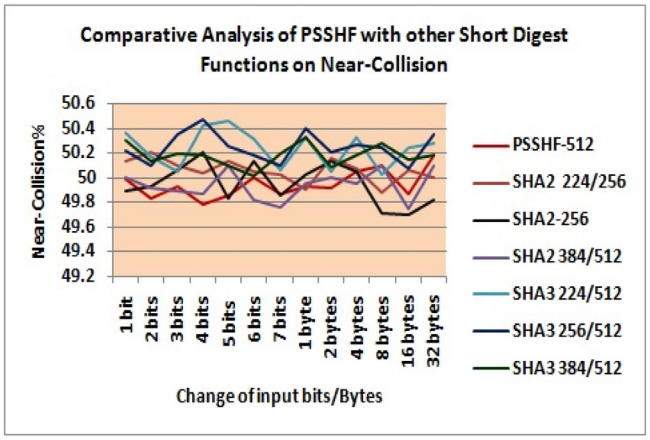

Fig. 13. Comparative analysis of PSSHF with short digest functions.

hensive analysis was performed with 3 K data using 500 hash comparisons obtained by modifying the input data at distinct locations. The changes in the PSSHF response corresponding to the change in the input bits/bytes were recorded and matched with the reference output using the bit-locations. The results evidence the average near-collision response of the PSSHF is 49.94 %. It levels with the performance of SHA2-384 and is the least among its other counterparts. The result proves the near-collision probability is minimal for the PSSHF. Fig. 10 graphically proves the consistent performance of the PSSHF in minimizing the near-collision effect irrespective of the changes in the input data. The PSSHF performs marginally lower than the SHA2-256 for a single input-bit change and is shown in Fig. 11. However, it outperforms the other contemporary digest functions in the 512-bit category which is shown in Fig. 12. Fig. 13 graphically illustrates the consistent performance of

the PSSHF with short digest functions. The result witnesses that the near-collision response of the PSSHF is comparatively minimal than its other counterparts. Therefore, launching pre-image and second pre-image attacks against the PSSHF is extremely difficult.

- **Analysis of the effect of Avalanche on PSSHF output**

The objective of the experiment is to study the avalanche effect on PSSHF output nibbles. To do so, the PSSHF was provided with different input data of varying lengths. The inputs are randomly varied between 1 bit and greater than 10 Bytes. The Hexadecimal responses of the PSSHF corresponding to the input changes were marked and matched with the reference output. Table 10 shows the result of the experiment such that the separate entry is derived from the average of 25 hash comparisons. The analysis proves the PSSHF modifies 93.77 % of the output symbols for a niggling change in the input data. The experiment was equally extended to standard digest functions for performing a comparative study as given in Table 11. The separate entry of the table is calculated from the average of 500 hash comparisons using 3 K input data. The result proves the PSSHF outperforms the contemporary digest functions with a record average avalanche effect of 93.78 % changes in the output nibbles. It shows the PSSHF maintains its consistency irrespective of the input data and performs exceedingly well in producing distinct output symbols. Therefore, launching differential attacks against the PSSHF becomes a grueling task for cryptanalysts.

- **Runtime analysis**

At this point, the PSSHF was subjected to relative runtime analysis with the contemporary digest functions. Runtime(R_t)was calculated by taking the average execution time of 25 samples from each data set. Table 12 shows the outcome of the analysis.

Fig. 14 visually illustrates the relative performance of the PSSHF with contemporary digest functions. The result proves R_t of PSSHF linearly increases with the data size D_s. This happens due to the application of higher polynomial powers and the application of 11 subsets in the PSSHF design. Simultaneously, the entries show the PSSHF relatively performs well for short messages with acceptable delay. Therefore, it could be considered as an alternative for short message digests from the perspective of security. However, research efforts have been going in place to minimize R_t without jeopardizing security.

5.2. Statistical analysis of PSSHF on random response

The statistical analysis mathematically proves the random behavior of the PSSHF output. To perform the analysis the NIST-approved frequency (Monobit) test was conducted on the PSSHF (Bassham, et al., 2021). Accordingly, a sample output of the PSSHF was chosen to calculate S_n, S_{abs}, error function (*ERF*), and complementary error function (*ERFC*). The response of the PSSHF is considered random if the *ERFC* value is greater than 1 %, i.e., 0.01. The random behavior of the PSSHF is calculated as follows:

Table 10
Effect of Avalanche on PSSHF output-nibbles.

Data size (bits/Bytes)	Bits & Bytes Changed								Average Effect (%)
	1Bit	1Byte	3Bytes	5Bytes	7Bytes	9Bytes	10Bytes	greater than10Bytes	
<512b	93.75	93.75	94.53	92.19	92.97	97.65	91.41	92.97	94.21
512b	92.97	93.76	93.75	94.53	93.75	92.97	96.09	92.97	93.88
128B	92.97	96.88	91.41	95.31	92.97	98.43	89.06	93.75	94.08
256B	92.19	92.18	95.31	91.41	94.53	95.31	96.09	91.41	93.62
512B	98.44	90.63	97.66	94.53	92.19	90.63	97.66	92.97	94.27
1 KB	92.19	95.31	92.97	89.06	92.19	92.97	92.19	96.09	93.03
2 KB	93.75	97.66	94.53	96.31	89.06	92.97	90.62	94.53	93.51
4 KB	91.41	93.75	96.09	92.19	92.97	96.88	93.75	96.09	93.88
8 KB	96.09	93.75	93.75	91.41	96.88	92.97	92.97	94.53	93.69
20 KB	92.99	94.53	93.75	92.19	93.75	93.75	91.41	98.44	93.75
50 KB	93.75	92.97	93.75	96.88	92.97	90.63	91.41	95.31	93.56
The average effect of Avalanche on output nibbles (%)									**93.77 %**

Table 11
Relative analysis of avalanche effect on the output nibbles.

Change in Input Bits/Bytes	Change in the Output Nibbles								
	PSSHF-512	SHA2 Variants				SHA3 Variants			
		224/256	256/256	384/512	512/512	224/512	256/512	384/512	512/512
1 bit	93.89	93.53	93.3	93.28	93.26	93.72	93.83	93.53	93.62
2 bits	93.62	94	92.97	93.14	93.31	93.49	93.68	93.8	93.6
3 bits	93.81	93.75	93.16	93.24	93.35	93.65	93.46	93.68	93.58
4 bits	93.57	93.75	93.04	93.39	93.46	93.68	93.47	93.71	93.62
5 bits	93.86	93.72	92.92	93.28	93.47	93.41	93.7	93.75	93.59
6 bits	93.85	93.81	93.1	92.94	93.42	93.46	93.55	93.79	93.71
7 bits	93.58	93.61	93.32	93.29	93.52	93.49	94.04	93.43	93.81
1 byte	93.92	93.96	93.08	93.36	93.21	93.6	93.6	93.66	93.7
2 bytes	93.7	93.49	93.14	93.24	93.24	93.56	93.78	93.67	93.74
4 bytes	93.78	93.8	93.2	93.17	93.61	93.55	93.6	93.72	93.57
8 bytes	93.92	93.75	93.04	93.24	93.51	93.38	93.8	93.69	93.66
16 bytes	93.81	93.59	93.12	93.18	93.36	93.74	93.78	93.76	93.77
32 bytes	93.88	93.66	92.88	93.12	93.31	93.77	93.61	93.59	93.72
Average %	**93.78**	**93.72**	**93.1**	**93.22**	**93.39**	**93.58**	**93.68**	**93.68**	**93.67**

The given sample output is:

d21555ae8ab254b2f2403c6109673a19282bab45484772d8498dfb81d5622a029ac688f53375cd6b780ade66ff81afa49b8d0c86228d9c 7238ef8dffd78d41e7

The binary equivalent of the digest is.

1101001000010101010101011010111010001010101100100101010010110010111100100100000000111100011
0000100001001011001110011101000011001001010000010101110101011010001010100100001000111011100
1011011000010010011000110111111101110000000111010101011000100010101000000001010011010110001101
0001000111101010011001101110101110011010110101101111000000010101101111001100110111111111000
0001101011111010010010011011100011010000110010000110001000101000110110011100011100100011100
01110111110001101111111111101011110001101010000001111100111

The size of the digest $N = 512$, calculate $S_n = X_1 + X_2 + X_3 + \ldots + X_{512}$ such that $X_i = 2\varepsilon_i - 1$ and $|S_n| = 10$.

Calculate $S_{abs} = \frac{|S_n|}{\sqrt{N}} = 0.442$.
Calculate $\mathrm{ERF}(Sabs) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)dt = 0.4681$
Compute $ERF(\frac{S_{abs}}{\sqrt{2})}) = 0.3415$
Calculate $ERFC = 1 - ERF = 0.6585$

The value of *ERFC* corresponding to the PSSHF output is 0.6585. The value is greater than 0.01. Therefore, the PSSHF output produces a random sequence of output bits. The experiment was also extended to 50 sample outputs of the PSSHF to precisely identify its random behaviour. The average value obtained for *ERFC* in the statistical analysis is 0.3936. The statistical analysis mathematically confirms the sequence of output bits produced by the PSSHF is random.

6. Discussions

- **Significance of Random property against Security**

The PSSHF is a 512-bits algorithm. Therefore, launching Brute-force and birthday attacks against the PSSHF is computationally infeasible. This is because they demand 2^{512} and 2^{256} worst-case attempts respectively to find a collision. In a like manner, the probability of launching a successful Random attack on PSSHF would be $1/2^{512}$. However, the value is extremely low and could be neglected. Therefore, the only feasible way available for the cryptanalyst to find a hash hit is through differential attacks (Bassham, et al., 2021; Dinur et al., 2013). The known attacks against the standard digest functions also reinforce this fact. Consequently,

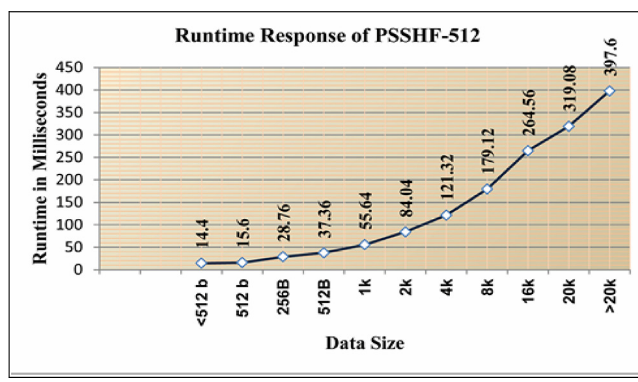

Fig. 14. PSSHF-Runtime response for distinct data.

Table 12
Relative runtime analysis of PSSHF.

Data Size	Runtime Performance of Keyless Hash functions (In Milliseconds)								
	SHA2-Variants				SHA3 Variants				PSSHF-512
	224/256	256/256	384/512	512/512	224/512	256/512	384/512	512/512	
<512b	1.24	0	0	0	9.4	0	0	0	14.4
512b	1.24	0	0.64	0	10	0.64	0	0.64	15.6
256B	2.48	0	0	l.28	8.72	0	0	0.6	28.76
512B	l.28	0	0.64	0	10.64	0.32	0	0	37.36
1 k	l.28	l.28	0	0.64	9	0	0	0	55.64
2 k	1.88	0	0.6	0.64	12.52	0.6	0.6	0.16	84.04
4 k	l.28	0.64	1.2	0.6	11.92	0.6	0.6	0.64	121.32
8 k	1.24	0.6	1.2	0.6	9.36	0.6	0	0	179.12
16 k	1.84	0.76	1.24	0	9.36	0.64	0.6	1.84	264.56
20 k	l.84	0.64	1.2	1.92	8.8	0.6	0.64	0.64	319.08
greater than 20 k	1.88	1.88	0	0.64	10.64	1.88	1.84	2.52	397.6

Table 13
Response of PSSHF for the small changes in the input data.

INPUT		Received from Sierra.Stanford.EDU by gloworm.Stanford.EDU wi	Received SromSierra.Stanford.EDU by gloworm.Stanfprd.EDU wi
S.No	SET	SUBSET OUTPUT	SUBSET RESPONSE FOR A SMALL CHANGE IN THE INPUT
1	A	3f6548456650acb81f89458b347ee0dd97d5b4ddf7bbd37d5aa5e7e795050f67	6b20be7eee2b3831f0b5738d8b79935c16d06a7e4c32d92feb33b6cd26152d5da3
2	Φ	3235734d9f898b2d495ecd8c95fd749f59a9efd2147b2928dbafd8d8d09b3e03	91e4b77baf66f1fc05a1d27cd41a82230d640f8a9f2573f3aa6c2de1fbfec3f
3	Φ∋	759dcd175fc88231fbcef8f86ba0a8dcd7092f4e174ad7113280da47b3fa6bf34	50e5d30306b6813d0da9921cedb92d1fb7b0b39694b4b81e94e8b4043fe050bf
4	Θ	876fae51db177464d5af261ab28e82dc3056012d9172a80f327c4344b059facc	dec398beca2bca0087f333e93f013e0ddd0d4410b7148c7c8bbf5d06406c1320
5	Θ∋	ea3b25561dca7625dcaab06b3183d5d0860504ae42fcd6011ea288d688a71ea4	3cfe9861997a982fbf008995e333cfd4cc7319618ef4325221e97bd913292dc6
6	Π	24fa2334e32384092ef51a5c15d9ed1d9427c91ff823573b2d0e49cbfd11f9ac	e063d37c22a3009722056156c04f905727b6438c84542458df1ed96bd9c2344e
7	Π∋	be976284614dd6b44eaeed985658896fec20712329f78215b4876fe9f8a5338d0	3a38e6b0227fa00c6ad9e295adbb38423b1655ceddd5c5ebd1de33db9745d73c
8	Δ	39c34433960 6acafec7007c220f0392f207206e53c0fbb1c6f09b97ab5ba21c2	f7d6141e24f4d6bb1e1e3d2c4983441df78eef90df65876c220fec2647a8f76
9	Δ∋	3e4bd9025f84dcba7f8a159ed5368420be929bf03d6eac0e8de7e723ce31cad1	9eb9ca70d6de6b0344b6544763551367 6e8c3fc1199ba65edf7922ea8dee67f9
10	Ω	40a0caf9df8a2ad48389d936d697ccb74b06814dc9d1b2e0660466e6c7ad189	17f2c8af2c2b2e2f8adf2a70cea0a90f98ac4f2542dbec551bfa70c9221f30c4
11	Ω∋	63246a6d1800d3fd490ed985ab8c1af958ef6002e820c4baf02d25c97dae2b8b	63246a6d1800d3fd490ed985ab8c1af958ef6002e820c4baf02d25c97dae2b8b
HALF 1		f42298334feba8aa08f36a0668da8d8551290f5da5ebe7ac7015ca903c0c2384d	30c7b5e7461a0321961448442488744d69d90b7d34dbd4c3c6fbc509e8ae8320f
HALF 2		2415172d8763f00106d01ff351f0c9444740c65509ee415a530a05ee78ea8d7f	b4481173a96dc40d1d4ebaa077573cf7669ff191ae0c2a4793ea63030150a0db

enhancing the random behavior of the digest function is necessary for the security of digest functions.

- **Static vs Dynamic Memory Allocation**

The digest functions like MD4, MD5, SHA-160, and the variants of the SHA2 allocate fixed memory to record the length attribute of the input data. This property disables the digest functions to process a message when its size surpasses $2^{64}/2^{128}$ bits. The proposed design overcomes this design flaw through a dynamic reservation policy. Dynamic reservation enables the PSSHF to process any message without length restriction as mentioned in Table 13 is the response of PSSHF for small changes in the input of the data that is performed.

- **Subsets vs Security**

Finding the input–output relation is a crucial step in making a successful differential attack. The application of 11 subsets and the formation of subsets through parsing make this process a computationally infeasible task. The sample response of the subsets for a trivial change in the input is presented in Table 13 to illustrate this fact. The responses of the subsets are random and any single bit-change in the input affects the responses of a minimum of 6 subsets. Identically, the changes that happen in consecutive locations affect the responses of all the subsets. Under these circumstances, performing input–output correlation would demand the most grueling efforts from the cryptanalyst.

7. Conclusion

PSSHF offers better solutions for remote data integrity issues from the perspective of security. The use of subsets and n^{th} degree polynomial function unusually hardens the task of an antagonist to detect a collision. The experimental analysis with more than 15 million hash comparisons proves the PSSHF as a strong one-way and collision-resistant function. The average avalanche response of PSSHF is 50.06 %. This is the highest value to its other counterparts. The statistical analysis of the Frequency (Monobit) test also mathematically confirms the same as the mean *ERFC* value of 0.3936 is well ahead of the minimum threshold value of 0.01. The analysis of the effect of the avalanche proves the PSSHF modifies 93.78 % of the output nibbles. This value is superior to other standard digest algorithms. The average Near-collision response of PSSHF is 49.94 %. This value is the least among the conventional hash algorithms. Therefore, launching differential attacks against the PSSHF would become a most grueling task for cryptanalysts. Contrarily, the use of 11 subsets, higher-order prime numbers, and the n^{th} degree polynomial function affects the performance of the algorithm to a substantial level and this situation is analogous to public cryptography. Considering the reality and the nature of applications, it is comprehended that the keyless hash functions are typically applied to produce MDC on limited data. The experimental result of runtime analysis shows that the proposed system works reasonably well for the limited data with acceptable delay. Therefore, this algorithm could be considered an ideal choice for applications involving limited data with core significance on security. In addition, we have been currently working on the performance issue of the PSSHF to envision broader applications without jeopardizing security.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by "Regional Innovation Strategy (RIS)" through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (MOE) (2021RIS-004), by Basic Science Research Program through NRF funded by MOE (2021R1I1A3041887), by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2022- 0-00704, Development of 3D-NET Core Technology for High-Mobility Vehicular Service) and by Korea University Research Grant.

References

"rfc3174", 2021. Datatracker.ietf.org. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3174. [Accessed: 03- Jul- 2021].

Akram, M.W. et al., 2021. A Secure and Lightweight Drones-Access Protocol for Smart City Surveillance. In: IEEE Transactions on Intelligent Transportation Systems. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–10. https://doi.org/10.1109/tits.2021.3129913.

Al-Kuwari, S., Davenport, J. and Bradford, R. "Cryptographic Hash Functions: Recent Design Trends and Security Notions". Eprint.iacr.org, 2021. [Online]. Available: https://eprint.iacr.org/2011/565. [Accessed: 14- Jul- 2021].

Ashraf et al., 2022. A Survey on Cyber Security Threats in IoT-Enabled Maritime Industry. In: IEEE Transactions on Intelligent Transportation Systems. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–14. https://doi.org/10.1109/tits.2022.3164678.

Bartkewitz, T., 2009. Building hash functions from block cypher, their security and implementation properties. Ruhr-University Bochum.

Bassham, L. et al., 2021. "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", NIST, [Online]. Available: https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic. [Accessed: 07- Aug- 2021].\.

Bellare, M. and Rogaway, P. 1993. "Random oracles are practical". In: Proceedings of the 1st ACM conference on Computer and communications security - CCS '93. Available: 10.1145/168588.168596 [Accessed 2 July 2021].

Bellare, M., Boldyreva, A., Palacio, A. "An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem". In: Cachin, C., Camenisch, J.L. (Eds.), Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT 2004. Lecture Notes in Computer Science, vol 3027. Springer, Berlin, Heidelberg.

Bellare, M., Kohno, T. "Hash Function Balance and Its Impact on Birthday Attacks". In: Cachin, C., Camenisch, J.L. (Eds.), Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027. Springer, Berlin, Heidelberg.

Bertoni, G., Daemen, J., Peeters, M., van Assche, G. 2013. "Keccak". In: Annual international conference on the theory and applications of cryptographic techniques, 2013 (pp. 313-314). Springer, Berlin, Heidelberg.

Bharany, S. et al., 2022. "Wildfire Monitoring Based on Energy Efficient Clustering Approach for FANETS," Drones, vol. 6, no. 8. MDPI AG, p. 193, Aug. 02, 2022. doi: 10.3390/drones6080193.

Bharany, S. 2022. et al., "Efficient Middleware for the Portability of PaaS Services Consuming Applications among Heterogeneous Clouds," Sensors, vol. 22, no. 13. MDPI AG, p. 5013, Jul. 02, 2022. doi: 10.3390/s22135013.

Bharany, S. et al., 2022. "A Systematic Survey on Energy-Efficient Techniques in Sustainable Cloud Computing," Sustainability, vol. 14, no. 10. MDPI AG, p. 6256, May 20, 2022. doi: 10.3390/su14106256.

Bharany, S. et al., 2022."Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy," Sustainable Energy Technologies and Assessments, vol. 53. Elsevier BV, p. 102613, Oct. 2022. doi: 10.1016/j.seta.2022.102613.

Cannetti, R., Goldreich, O.,Halevi, S., 2015. "The random oracle methodology, Revisited (Preliminary version)". 30th Annual ACM Symp. On Theory of Computing, Perugia, Italy, ACM Press.

Chaudhry, S.A., Yahya, K., Garg, S., Kaddoum, G., Hassan, M., Zikria, Y.B., 2022. LAS-SG: An Elliptic Curve based Lightweight Authentication Scheme for Smart Grid Environments. In: IEEE Transactions on Industrial Informatics. Institute of Electrical and Electronics Engineers (IEEE), p. 1. https://doi.org/10.1109/tii.2022.3158663.

Coron, J., Dodis, Y., Malinaud, C., Puniya, P., 2005. "Merkle-Damgård revisited: How to construct a hash function". In: Annual International Cryptology Conference (pp. 430-448). Springer, Berlin, Heidelberg.

Damgård, I. 1998. "A design principle for hash functions". In: Conference on the Theory and Application of Cryptology, Springer, New York, Pp. 416-427.

Dinur, I., Dunkelman, O., Shamir, A. 2013. "Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials". International Workshop on Fast Software Encryption, Springer, Berlin, Heidelberg. pp. 219-240.

Eichlseder, M., Mendel, F., Schläffer, M. 2014. "Branching heuristics in differential collision search with applications to SHA-512". International Workshop on Fast Software Encryption, Springer, Berlin, Heidelberg. pp. 473-488.

Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L., 2020. Practical collision attacks against round-reduced SHA-3. Journal of Cryptology 33 (1), 228–270.

Javed, R. et al., 2022. "Future smart cities: requirements, emerging technologies, applications, challenges, and future aspects," Cities, vol. 129. Elsevier BV, p. 103794, Oct. 2022. doi: 10.1016/j.cities.2022.103794.

Joux, A. 2004. "Multicollisions in iterated hash functions. Application to cascaded constructions". In: Annual International Cryptology Conference, Springer, Berlin, Heidelberg. pp. 306-316.

Kam, J., Davida, G., 2009. Structured design of substitution-permutation encryption networks. IEEE Transactions on Computers 28 (10), 747–753.

Kim, Y., Choi, H., Seo, C., 2020. "Efficient implementation of SHA-3 hash function on 8-bit AVR-based sensor nodes". International Conference on Information Security and Cryptology, Springer, Cham. pp. 140-154.

Lai, X., Massey, J., 1992. Hash functions based on block ciphers. In: Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, pp. 55–70.

Lucks, S. 2004. "Design Principles for Iterated Hash Functions". IACR Cryptol. ePrint Arch., 253.

Matsui, M., 1993. Linear cryptanalysis method for DES cipher. In: Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, pp. 386–397.

Menezes, A., van Oorschot, P. and Vanstone, S. "Handbook of Applied Cryptography", Cacr.uwaterloo.ca, 2021. [Online]. Available: https://cacr.uwaterloo.ca/hac. [Accessed: 2-August-2021].

Merkle, R., 1989. One way hash functions and DES. In: Conference on the Theory and Application of Cryptology. Springer, New York, pp. 428–446.

Meshram, C., Li, C., Meshram, S., 2019. An efficient online/offline ID-based short signature procedure using extended chaotic maps. Soft Computing 23 (3), 747–753.

Motara, Y., Irwin, B., 2016. "Sha-1 and the strict avalanche criterion". In: IEEE Information security for South Africa (ISSA), pp. 35-40.

Pan, V., 1997. Solving a polynomial equation: some history and recent progress. SIAM review 39 (2), 187–220.

Preneel, B., 1993. Analysis and design of cryptographic hash functions. KatholiekeUniversiteitte Leuven. Doctoral dissertation.

Preneel, B., 1998. The state of cryptographic hash functions. In: School Organized by the European Educational Forum. Springer, , Heidelberg, pp. 158–182.

Preneel, B., Govaerts, R., Vandewalle, J., 1993. Hash functions based on block ciphers: A synthetic approach. In: Annual International Cryptology Conference. Springer, Berlin, Heidelberg, pp. 368–378.

Rivest, R.L. 1991. The MD4 Message Digest Algorithm. In: Menezes, A.J., Vanstone, S. A. (Eds.), Advances in Cryptology-CRYPTO' 90. CRYPTO 1990. Lecture Notes in Computer Science, vol. 537. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-38424-3_22.

Rivest, R. 2021. "rfc1321", Datatracker.ietf.org. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1321. [Accessed: 19- Mar- 2021].

Rogaway, P., Shrimpton, T., 2004. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, pp. 371–388.

Sanap, S.D., More, V., 2021. Performance Analysis of Encryption Techniques Based on Avalanche effect and Strict Avalanche Criterion. In: 2021 3rd International Conference on Signal Processing and Communication (ICPSC), pp. 676–679.

Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y. 2017. The First Collision for Full SHA-1. In: Katz J., Shacham H. (Eds.), Advances in Cryptology – CRYPTO 2017. CRYPTO 2017. Lecture Notes in Computer Science, vol 10401. Springer, Cham.

Teh, J., Tan, K., Alawida, M., 2018. A chaos-based keyed hash function based on fixed point representation. Cluster Computing 22 (2), 649–660. https://doi.org/10.1007/s10586-018-2870-z [Accessed 2 August.

Wang, X., Yin L. and Yu, H. 2021. "Collision Search Attacks on SHA1", Cryptome.org, [Online]. Available: https://cryptome.org/sha1-attacks.htm. [Accessed: 07- Jun- 2021]. Wang, X., Yin, Y.L., Yu, H. 2005. Finding Collisions in the Full SHA-1. In: Shoup, V. (Eds.), Advances in Cryptology – CRYPTO 2005. CRYPTO 2005. Lecture Notes in Computer Science, vol 3621. Springer, Berlin, Heidelberg.

Wang, X., Lai, X., Feng, D., Chen, H., Yu, X. 2005. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R. (Eds.), Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science, vol 3494. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11426639_1.

Webster A.F., Tavares S.E. 1986. On the Design of S-Boxes. In: Williams H.C. (Eds.), Advances in Cryptology — CRYPTO '85 Proceedings. CRYPTO 1985. Lecture Notes in Computer Science, vol 218. Springer, Berlin, Heidelberg.

Yi, X., June 2005. Hash function based on chaotic tent maps. IEEE Transactions on Circuits and Systems II: Express Briefs 52 (6), 354–357.

Yu, H. and Wang, X. 2021. "dblp: Near-Collision Attack on the Compression Function of Dynamic SHA2.", Dblp.org. [Online]. Available: https://dblp.org/rec/journals/iacr/YuW09.html. [Accessed: 10- Sep- 2021].