scientific reports

OPEN

A chaotic parallel hash engine with dynamic stochastic diffusion for blockchain and cloud security

Qianyun Wang[2], Yijun Yang[1,3], Ming Zhao[3], Huan Wan[3], Bin Li[3] & Xiaohu Yan[1]

The design of cryptographic hash functions is crucial for ensuring the security of digital information. In this context, cellular automata (CAs) have emerged as a promising tool due to their inherent parallelism, determinism, and simplicity. However, traditional CAs may not fully meet the requirements for cryptographic hash functions in terms of randomness, collision resistance, and avalanche effect. To address these challenges, we propose a design of cryptographic hash functions based on improved cellular automata. Our approach involves refining the rules of CAs to enhance their cryptographic properties. By incorporating random chaotic rules and optimizing parameters, we create a hash function that exhibits excellent performance in terms of randomness, collision resistance, and avalanche effect. Furthermore, the parallel nature of CAs allows for the simultaneous processing of multiple data blocks, significantly improving the efficiency of the hash function. Our design leverages these advantages to provide a robust and efficient cryptographic hash function that is suitable for a wide range of applications.

Keywords Cryptographic hash algorithm, Stochastic diffusion model, Cellular automata, Collision-resistance

Background

The rapid adoption of cloud auditing, blockchain systems, and distributed storage technologies has revolutionized data management for enterprises, yet simultaneously intensified security demands. While these innovations mitigate the challenges of massive data maintenance, their dependence on third-party cloud service providers (CSPs) introduces critical vulnerabilities – CSPs frequently fail to deliver guaranteed data integrity or sufficient privacy safeguards. This paradox underscores the indispensable role of cryptographic hash functions as foundational security primitives.

As the cornerstone of modern cryptography, cryptographic hash algorithms operate through deterministic one-way transformations that generate fixed-length digests from variable-length inputs. Their design mandates three non-negotiable properties:

Computational Irreversibility: Preimage resistance ensuring infeasibility of deriving original inputs from hash outputs;

Collision Resistance: Practical impossibility of identifying distinct inputs producing identical digests;

Avalanche Effect: Microscopic input alterations trigger drastic output deviations (>50% bit-flipping).

These properties collectively enable critical security functionalities:

Data Fingerprinting: Unique hash digests authenticate information integrity in blockchain transactions and cloud audits;

Tamper Evidence: Avalanche-driven sensitivity detects minimal unauthorized modifications in stored records;

Protocol Enforcement: Irreversibility secures password hashing and digital signature schemes against brute-force attacks.

However, emerging threats like message extension exploits and adaptive collision attacks expose limitations in current hash designs (e.g., SHA-2/3 families). Furthermore, escalating data volumes in cloud-blockchain ecosystems exacerbate computational overheads, demanding hash algorithms that reconcile rigorous security with time/space efficiency. Addressing these challenges through novel hash constructions – particularly for post-quantum resilience and parallel processing optimization – has become an urgent research priority. Sustained

[1]Institute of Applied Artificial Intelligence of The Guangdong-Hongkong-Macao Greater Bay, Shenzhen University, Shenzhen 518060, Guangdong, China. [2]Anhui Vocational College of Press and Publication, Hefei 230601, Anhui, China. [3]School of Artificial Intelligence, Shenzhen Polytechnic University, Shenzhen 518055, Guangdong, China. email: yanxiaohuszpu@163.com

advancements in this domain will directly determine the trustworthiness and scalability of next-generation decentralized infrastructures.

Related work

Foundations and evolution of hash functions

The foundational work in cryptographic hash functions stems from early designs such as MD4 (Rivest, 1990)[1] and MD5 (Rivest, 1992)[2], which introduced efficient message-digest algorithms for integrity verification. Subsequent standards like FIPS 180 (NIST, 1993–2002)[3–5] formalized the SHA family, while RIPEMD-160 (Dobbertin, 1996)[6] improved collision resistance through dual parallel pipelines. Merkle's work (1990)[7] explored the interplay between DES and hash functions, and Wolfram (2002)[8] later proposed cellular automata as a theoretical basis for cryptographic primitives. These early efforts laid the groundwork for analyzing hash function security and scalability.

Security analysis and attacks

The vulnerabilities of classical hash functions have been extensively studied. Boer and Bosselaers (1991)[9] demonstrated attacks on MD4's final rounds, while Dobbertin (1996)[10,11] pioneered collision attacks on MD4 and MD5. Wang et al. (2004–2005)[12–15] revolutionized cryptanalysis by breaking MD5, SHA-0, and SHA-1 using differential collision techniques. Subsequent refinements by Liang and Lai (2005)[16] and Sasaki et al. (2007)[17] optimized attack efficiency. Stevens (2013)[18] further advanced SHA-1 attacks via joint local-collision analysis. Lee J(2012)[19] exploits two backward queries to the underlying primitive to find a collision for the JH compression function, though iteration significantly enhances its collision resistance in the random permutation model. Li W.(2017)[20] evaluating its cryptographic robustness against potential vulnerabilities and performance implications for IoT applications. These studies highlight the critical need for robust collision resistance in hash designs.

Novel hash function designs

Modern research focuses on post-quantum and chaos-based paradigms. Karthik and Bala (2019)[21] proposed a provably secure keyless hash framework, while Ayubi et al. (2023)[22] and Alawida et al. (2020–2021)[23,24] leveraged chaotic maps, DNA sequences, and finite automata for entropy enhancement. Quantum-inspired designs like Li et al.'s (2023)[25] quantum walk-based hash and Guo et al.'s (2022)[26] MDPH security proof address future threats. Kanso and Ghebleh (2015)[27] introduced chaotic substitution-permutation networks, demonstrating resistance to differential attacks. These innovations prioritize adaptability to evolving cryptographic threats.

Parallel hash function research

Parallel architectures aim to optimize throughput for big data and cloud applications. Yang et al. (2019–2022)[28,29] designed compressive parallel structures and multi-iterative frameworks for high-speed hashing. Je et al. (2015)[30] and Nouri et al. (2014)[31] implemented chaotic shuffle-exchange networks and Chebyshev-Halley methods for parallelism. Meysam(2016)[32] proposes a novel keyed parallel hashing scheme leveraging a new chaotic system to enhance cryptographic security and processing efficiency. Wang et al. (2011)[33] utilized coupled map lattices, while Kevin and Robert (2017)[34] optimized tree-based modes. Salvatore et al. (2016)[35] proposed a cuckoo hashing pipeline for throughput scalability, emphasizing hardware-friendly implementations. Liu H.(2021)[36] proposes a novel chaos-based hash function enhanced by parallel impulse perturbation, leveraging chaotic dynamics to improve cryptographic security and collision resistance in hash algorithms.

Applications and extended research

Hash functions are integral to diverse applications. Yang et al. (2015)[37] enhanced cancelable fingerprint encryption via hash-based salting, while Guesmi et al. (2016)[38] and Ye et al. (2016)[39] integrated SHA-2 and chaotic diffusion for secure image encryption. Teh et al. (2019–2020)[40,41] developed chaos-based keyed and unkeyed hashes for IoT devices. Rajeshwaran and Anil (2019)[42] employed cellular automata for lightweight cryptographic hashing. Bertoni et al.'s sponge construction (2007)[43] and Biham-Dunkelman's HAIFA model (2007)[44] generalized iterative frameworks for flexibility in application-specific designs.

Security frameworks and extended analysis

Beyond attacks, broader security principles have been explored. Lucks (2005)[45] advocated failure-friendly design to mitigate catastrophic breaches, while Khushboo and Dhananjoy (2019)[46] analyzed MGR hashes for statistical robustness. Li et al. (2021)[47] exposed vulnerabilities in authenticated data structures, and Zhang et al. (2021)[48] reformed SHA-2 message expansion for pipeline efficiency. Sponge functions (Bertoni et al., 2007)[43] and Merkle-Damgård variants (Merkle, 1990)[7] remain pivotal in formalizing indifferentiability and domain extension.

Summary of document highlights

Innovative parallel hash architecture design

Proposed a new application of cellular automata-based parallel hash engines with dynamic stochastic diffusion (short for CPHDSD). By optimizing chaotic rules and dynamic parameter selection, it simultaneously enhances collision resistance, avalanche effect, and computational efficiency. The parallel nature of CAs enables multithreaded processing, significantly accelerating hashing speed in big data scenarios while maintaining a compact structure suitable for resource-constrained environments.

Comprehensive security validation

Demonstrated CPHDSD's exceptional security through million-scale experiments:

Collision Resistance: No collisions detected, with minimum/maximum Hamming distance approaching the theoretical optimum (256).

Avalanche Effect: Critical parameters achieved theoretical optima within 3–7 steps, outperforming mainstream algorithms (MD5, SHA-family, etc.).

Information Entropy: Average entropy over 3.9, exceeding SHA3 and state-of-the-art literature results, with output distribution approaching perfect randomness (hexadecimal character error < 0.085%).

High-efficiency parallel computing advantages

The proposed hash algorithm employs a parallel compression framework, reducing theoretical computation time (2368*T) to 30%–70% of comparable algorithms. Experiments show its time growth follows a near-logarithmic curve, delivering superior efficiency over traditional serial architectures (e.g., SHA-family) and other parallel schemes for large-file processing. This positions the parallel hash algorithm as an optimized solution for cloud storage and blockchain applications requiring high-throughput hashing.

Paper organization

The paper is organized as follows: Sect. "Preliminary knowledge" provides preliminary knowledge of cellular automata and parallel hash, Sect. "Proposed algorithm" presents our core technical contribution - a novel cryptographic hash function that simultaneously achieves enhanced collision resistance, computational efficiency, and implementation compactness. This is followed by Sect. "Algorithm analysis", which provides a comprehensive performance evaluation through both theoretical security analysis and empirical benchmarking against existing standards. Finally, Sect. "Conclusion and future work" concludes with a synthesis of our key findings, discusses practical implications for security systems, and outlines promising directions for future research in hash function optimization.

Preliminary knowledge

To comprehensively address the design of parallel hash functions based on cellular automata (CAs), the following core principles and interdisciplinary knowledge must be clarified. These concepts form the theoretical backbone for understanding the integration of CAs into cryptographic systems.

Cellular automata

Cellular Automata (CA) are discrete computational models first proposed by John von Neumann in the 1950 s and later advanced by scholars such as John Conway and Stephen Wolfram. Their core definitions and characteristics are as follows:

Definition

A cellular automaton is a dynamic system defined over a discrete, finite-state cellular space, evolving through local rules across discrete time steps.

Core components

- Cell: The basic unit of a CA, distributed on a discrete spatial grid (e.g., 1D, 2D, or nD Euclidean space).
- Lattice: The regular grid structure housing cells, which can be 1D, 2D, or multidimensional.
- State: Each cell holds a finite set of discrete states (e.g., 0/1, alive/dead).
- Neighbors: Cells directly adjacent to a given cell (e.g., von Neumann or Moore neighborhoods).
- Rule: State updates depend on a cell's current state and its neighbors' states, governed by deterministic local rules.

Key properties

- Discreteness: Time, space, and states are inherently discrete.
- Locality: State transitions rely only on a cell's immediate neighbors.
- Synchrony: All cells update states simultaneously at each time step.
- Determinism: Given current states, the next state is uniquely defined.
- Homogeneity & Uniformity: Identical rules govern all cells, arranged in regular patterns.

Mathematical formulation

A standard CA is formally defined as a quadruple:

$$CA = (L, S, N, f)$$

where

- L: Cellular lattice (spatial grid),
- S: Finite set of states,
- N: Neighborhood configuration,
- f: Transition function $f : S^{[N]} \rightarrow S$.

Applications
Cellular automata are widely used to simulate complex systems, including traffic flow modeling, crowd evacuation dynamics, biological systems (e.g., morphogenesis), and physical phenomena (e.g., fluid dynamics).

To sum up, cellular automata are discrete dynamic models based on simple local rules, enabling the emergence of complex global behavior through localized interactions. Their versatility in simulating real-world systems and theoretical significance in computational universality make them a cornerstone of complexity science.

Parallel hashing
Parallel hashing is a strategy that accelerates hash computations by leveraging multiple computing resources simultaneously, such as multi-core processors, GPUs, or distributed computing nodes. Traditional hash algorithms (e.g., SHA-256 or MD5) typically follow a sequential processing model, where data is handled block by block. In contrast, parallel hashing improves efficiency in large-scale data processing by dividing tasks and executing them concurrently.

Core idea and architecture
The essence of parallel hashing lies in splitting input data into independent subtasks, each processed by distinct computing units. For example, in distributed systems, data may be partitioned and assigned to multiple nodes. Each node computes the hash for its allocated data chunk, and the final hash value is generated through merging or cascading operations. Key challenges in this architecture include designing effective data partitioning strategies, synchronizing parallel tasks, and ensuring consistency in the final result.

Application scenarios
Parallel hashing is critical in scenarios requiring high-speed hash computations:

- Blockchain Technology: In consensus mechanisms like Proof of Work (PoW), parallel hashing accelerates hash collision searches for candidate blocks.
- Big Data Storage: Distributed file systems (e.g., IPFS) use parallel hashing to rapidly verify the integrity of massive datasets.
- Real-Time Secure Communication: High-throughput network environments benefit from parallel computation to speed up hash operations during TLS handshake processes.

Technical challenges and optimization directions
Despite its performance advantages, implementing parallel hashing faces several challenges:

- Data Dependencies: Certain hash algorithms (e.g., SHA-3) have inherent sequential dependencies, making parallelization difficult without algorithmic modifications or hardware enhancements.
- Load Balancing: Uneven data partitioning may leave some computing units idle, reducing overall efficiency.
- Synchronization Overhead: Communication and coordination between nodes can introduce latency, particularly in distributed environments.

To sum up, parallel hashing overcomes the performance limitations of traditional sequential hashing by harnessing modern parallel computing hardware. However, its design must balance algorithmic characteristics, hardware architecture, and application-specific requirements. With advancements in heterogeneous computing (e.g., quantum computing or compute-in-memory architectures), the optimization potential for parallel hashing will continue to expand.

Proposed algorithm
This chapter proposes a parallel hash algorithm grounded in a cellular automata and a dynamic stochastic diffusion model. In Fig. 1, the diagram depicts an iterative framework for parallel hashing, where multiple 512-bit data blocks are processed simultaneously through independent hashing units. At each stage, groups of data blocks undergo concurrent execution of the compression function H, demonstrating true parallelism in the initial processing phase. This architecture allows simultaneous hash computations across all blocks without sequential dependencies, maximizing throughput during the primary transformation step.

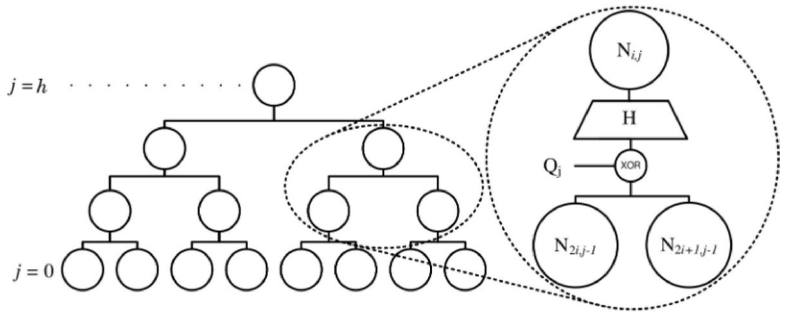


Fig. 1. Parallel hash tree construction.

The iterative nature emerges as intermediate results from parallel processing are systematically integrated into subsequent stages. After the initial parallel H operations, outputs are strategically combined through layered merging phases, creating chained dependencies that feed back into the workflow. This hybrid approach maintains parallel efficiency for raw data processing while introducing controlled sequential linkages for result consolidation, enabling both high-speed computation and coherent hash value generation across iterations.

Entropy analysis procedure for classifying 256 CA rules

Entropy analysis quantifies the uncertainty or randomness in the dynamical behavior of cellular automata. Here's the step-by-step process to analyze CA rules using entropy.

Data preparation

Input: A CA rule (e.g., Rule 30, Rule 110).

Initialization:

- Generate a random binary initial configuration (e.g., a single 1 in the center, surrounded by 0 s).
- Define a finite lattice (e.g., 100 cells with periodic boundary conditions).

Simulation:

- Evolve the CA for multiple time steps (e.g., 1000 iterations).
- Record the spatial configuration at each time step.

Calculation of information entropy

For each time step *t*:

Divide the lattice into overlapping/non-overlapping blocks of size *k*(e.g., *k* = 3: triplets of neighboring cells).

Count the frequency of each block pattern (e.g., "000", "001", …, "111").

Compute probabilities $p_i$for each block *i*:

$$p_i = \frac{Count\ of\ block\ i}{Total\ number\ of\ blocks}$$

Shannon Entropy:

Calculate

$$H\left(t\right) = -\sum{}_{i=1}^{2^k} p_i log_2 p_i$$

.

For $k = 1$: Measures entropy of single-cell states (simpler but less sensitive).

For $k \geq 2$: Captures correlations between neighboring cells (more powerful for distinguishing classes).

Time-window analysis

Compute the mean entropy $\overset{\vee}{H}$ over a sliding window of N_{step} (e.g., 100 steps):

$$\overset{\vee}{H} = \frac{1}{N_{step}} \sum{}_{t=T}^{T+N_{step}} H\left(t\right)$$

Entropy Dynamics:

Plot $H\left(t\right)$ vs. *t* to observe trends:

Class 1 (Uniform): $H\left(t\right) \rightarrow 0$ as entropy collapses (all cells identical).

Class 2 (Periodic): $H\left(t\right)$ oscillates periodically (e.g., entropy peaks and troughs align with cycle length).

Class 3 (Chaotic): $H\left(t\right)$ remains high and fluctuates irregularly.

Class 4 (Complex): Moderate entropy with intermittent peaks (structured chaos).

Parameter tuning

Block Size *k*: Larger *k* improves discrimination but increases computational cost. For ECA, *k* = 3 often suffices.

Time Window N_{step}: Ensure N_{step} exceeds the period length for Class 2 rules (e.g., $N_{step} = 200$ for cycles of period 50).

Classification criteria

Class 2 vs. Class 3:

Class 2: Fourier transform of $H\left(t\right)$ reveals dominant frequency (periodicity).

Class 3: Power spectrum of $H\left(t\right)$ is broadband (noisy).

Thresholds:

High Entropy: $\overset{\vee}{H} > 0.8 \times log_2\left(2^k\right) \left(e.g., \overset{\vee}{H} > 2.4\ for\ k = 3\right)$

Low Entropy: $\overset{\vee}{H} < 0.2 \times log_2\left(2^k\right)$

Example analysis
Rule 30 (Class 3): $H(t)$ remains near $log_2(8) = 3$(maximum entropy for $k=3$). No periodic oscillations (broadband spectrum).

Rule 4 (Class 2): $H(t)$ oscillates with period matching the stripe pattern (e.g., period 2).

Fourier spectrum shows sharp peaks at specific frequencies.

Validation
Visual Inspection: Cross-check entropy results with spacetime diagrams.

Lyapunov Exponent: Class 3 rules exhibit positive Lyapunov exponents (sensitive dependence on initial conditions).

Compression function H

For each compression function H, the cellular automaton undergoes two processes: the automaton's operation and the encryption loop.

Step 1: Cellular Automata.

Here's a classification of the 256 elementary cellular automata rules based on their behavioral characteristics, organized by Stephen Wolfram's widely accepted taxonomy. Wolfram's classification divides CA rules into 4 classes based on their long-term dynamical behavior:

Class I: Homogeneous/Uniform Behavior

Characteristics: Evolves to a static, uniform state (all cells identical).

Examples: Rules 0, 8, 32, 40, 128, 160, 168, 232.

Total Rules: ~8% (≈ 20 rules).

Class II: Periodic/Repetitive Behavior

Characteristics: Forms stable or oscillating periodic patterns.

Examples: Rules 4, 12, 19, 23, 27, 50, 51, 60, 72, 76, 108, 129, 150, 156, 178, 200, 204.

Total Rules: ~41% (≈ 105 rules).

Class III: Chaotic/Aperiodic Behavior

Characteristics: Exhibits pseudorandom, disordered patterns with no long-term structure.

Examples: Rules 18, 22, 30, 45, 54, 73, 90, 105, 106, 110 (borderline Class IV), 122, 126, 146, 150 (borderline), 182.

Total Rules: ~34% (≈ 87 rules).

Class IV: Complex/Edge-of-Chaos Behavior

Characteristics: Produces localized structures and long-range correlations; capable of universal computation.

Examples: Rules 110, 41, 54, 106, 109, 124, 137, 147, 193.

Total Rules: ~17% (≈ 44 rules).

The classification schema for hash functions delineates four distinct behavioral categories: the first two categories are classified as deterministic (Class I & II), the third category exhibits stochastic properties (Class III), while the fourth demonstrates non-uniform pseudorandom characteristics (Class IV)[46]. Through rigorous analysis, the hash algorithm exclusively selects candidates from the third category due to its optimal entropy profile.

Class III rules primarily include the following entries:

- 18, 22, <u>30</u>, <u>45</u>, <u>54</u>, <u>60</u>, 73, <u>75</u>, <u>86</u>, <u>89</u>, <u>90</u>, <u>101</u>, <u>102</u>, <u>105</u>, <u>106</u>, 110, 122, 126, 129, <u>135</u>, 146, <u>149</u>, <u>150</u>, 151, <u>153</u>, 161, <u>165</u>, 182, 183, <u>195</u>,….

The rules within Class III undergo systematic arrangement in ascending numerical order, followed by combinatorial selection of 16 candidates fulfilling strict equilibrium criteria (balanced 4-zero/4-one configurations across all 8-cell neighborhood permutations). These rules are subjected to cryptographically secure permutation to yield a restructured rule table (Table 1), ensuring stochastic distribution while preserving entropy constraints. A deterministic bijection is subsequently established between the shuffled indices Q_j (j $\in$ $\{0,1,\ldots,15\}$) and their corresponding cellular automaton transition functions. This mapping protocol facilitates reproducible rule retrieval during iterative hash compression stages, enabling: state-space traversal optimization, nonlinear Boolean operation embedding, avalanche-compliant entropy diffusion.

Step 2 Key Generation

Given a key, perform SHA3 calculation on the key and generate the corresponding hash value. For example, for the key 'CA', after calculating $\mathrm{SHA3}-512(\mathrm{CA})$, the result is used for Step 3. To ensure the security, a new key can be periodically replaced.

Step 3 Automatic Iteration of Cellular Automata.

For the hash value:

0f11f610fc4231452844d064a24f0c6419f7757ca62e849c0a3473a 3c5ba6f2143547c90cfc99 5fb652c008a0f65b9c54af3663b 7bc1bfc6fdac25b1cab5df26 generated by the key 'CA', for the first message block, select $Q_0 = Rule105$, for the second message block, select $Q_f = Rule75$, for the third message block, select $Q_1 = Rule45$, and so on, until all message blocks are processed.

Step 4 Dynamic Stochastic Diffusion Model

Message extension: For each message block m_j, perform the following message extension operations to output 132 message words.

(1) Divide m_j into 16 32-bit message words MW_i $(i = 0,\ 1,\ \cdots,\ 15)$,

Q_j	Rule	Binary representation
Q_8	30	00011110
Q_1	45	00101101
Q_d	54	00110110
Q_4	60	00111100
Q_f	75	01001011
Q_6	86	01010110
Q_a	89	01011001
Q_2	90	01011010
Q_b	101	01100101
Q_9	102	01100110
Q_0	105	01101001
Q_7	135	10,000,111
Q_c	149	10,010,101
Q_3	150	10,010,110
Q_e	165	10,100,101
Q_5	195	11,000,011

Table 1. Regenerated new rule table.

(2) For message *x*, we define two variable-parameter permutation functions as follows:

$$\Sigma_0(x) = x \oplus ROTL_\alpha(x) \oplus ROTL_\beta(x)$$
$$\Sigma_1(x) = x \oplus ROTL_\gamma(x) \oplus ROTL_\delta(x)$$

Parameters $\alpha, \beta, \gamma, \delta$ are undetermined, and $ROTL_y(x)$ is a *y*-bit loop left-shift operation function. To obtain better resistance to differential attacks, the values of $\alpha, \beta, \gamma, \delta$ should be mutually prime; therefore, the differential attack becomes increasingly difficult and difficult to continue.

(3) For message word MW_i $(i = 16, 17, \cdots, 131)$, the message is expanded as follows:

$$MW_i = \begin{cases} \Sigma_0(MW_{i-16} \oplus ROTL_7(MW_{i-3}) \oplus ROTL_1(MW_{i-9})) \oplus ROTL_{19}(MW_{i-4}),\ i = 16{,}17, \cdots, 41 \\ \Sigma_1(MW_{i-16} \oplus ROTL_7(MW_{i-3}) \oplus ROTL_1(MW_{i-9})) \oplus ROTL_{19}(MW_{i-4}),\ i = 42{,}43, \cdots, 67 \\ MW_{i-68} \oplus MW_{i-64},\ i = 68{,}69, \cdots, 131 \end{cases}$$

As soon as the number of 512-bit message blocks is established, for example, if $n = 64$, first defines the initial link constants as:

$$\mathrm{K}_k = SHA3(k),\ k = 0{,}1, \cdots, 63$$

The 64 constant variables K_k (the first 64 bits of the results above) are listed in Table 2. These constant variables increase the uncertainty and unpredictability of the compression function, thereby enhancing the collision resistance of the algorithm.

Define two logical functions:

$$\mathrm{Ch}(x, y, z) = (x \cap y) \oplus \left(\bar{x} \cap z\right)$$

$$\mathrm{Maj}(x, y, z) = (x \cap y) \oplus (x \cap z) \oplus (y \cap z)$$

$\bar{x}$ is the reverse operation of *x*, $TEMP_k$ is the intermediate link variable state at the *k*-th iteration, consisting of 8 registers A, B, C, D, E, F, G, H. For each message block m_j, conduct 64 rounds of iteration in the following iteration mode:

$$TEMP_{k+1} = C_1(TEMP_k, m_j), 0 \le k \le 63$$

The specific iteration process can be described as follows:

$$A \leftarrow \mathrm{Maj}(A, B, C) + D + ROTL_7(ROTL_{12}(A) + E + ROTL_k(K_k)) + ROTL_{12}(A) + MW_{k+68}$$
$$B \leftarrow \mathrm{CA}(A, k, 1)$$
$$C \leftarrow ROTL_9(B)$$
$$D \leftarrow \mathrm{C}$$
$$E \leftarrow \Sigma_0(\mathrm{Ch}(A, B, C) + H + (ROTL_7(ROTL_{12}(A) + E + ROTL_k(K_k))) + MW_k)$$

a7d43e8bf09c5a21	**6f2d7a9c1f4b3d82**	**719f26c45a2d83b7**	**afd53b8e719f26c4**
3e8b6ecaf09c5a21	7a9c1f4b3d82a6e1	26c45a2d83b7e19f	3b8e719f26c45a2d
f09c5a218d76b3e2	1f4b3d82a6e1c5f9	5a2d83b7e19f4c26	719f26c45a2d83b7
5a218d76b3e2c4f8	3d82a6e1c5f92b74	83b7e19f4c26a5d8	26c45a2d83b7e19f
8d76b3e2c4f8192a	a6e1c5f92b748d3a	c8d0f67e9154bd38	5a2d83b7e19f4c26
b3e2c4f8192a67d5	c5f92b748d3a9e07	a6e1c5f92b748d3a	83b7e19f4c26a5d8
c4f8192a67d5ef03	2b748d3a9e076c5d	e76f2d7a9c1f4b3d	e19f4c26a5d837b2
192a67d5ef039b47	8d3a9e076c5df2a8	54bd38e76f2d7a9c	4c26a5d837b2e19f
67d5ef039b472ac8	9e076c5df2a8b341	9e076c5df2a8b341	a5d837b2e19f4c26
ef039b472ac8d0f6	6c5df2a8b3417e9d	3d82a6e1c5f92b74	37b2e19f4c26a5d8
9b472ac8d0f67e91	f2a8b3417e9d0c62	f09c5a218d76b3e2	e19f4c26a5d837b2
2ac8d0f67e9154bd	b3417e9d0c62afd5	67d5ef039b472ac8	4c26a5d837b2e19f
d0f67e9154bd38e7	7e9d0c62afd53b8e	192a67d5ef039b47	a5d837b2e19f4c26
7e9154bd38e76f2d	0c62afd53b8e719f	b3417e9d0c62afd5	37b2e19f4c26a5d8
54bd38e76f2d7a9c	afd53b8e719f26c4	7e9d0c62afd53b8e	e19f4c26a5d837b2
38e76f2d7a9c1f4b	3b8e719f26c45a2d	0c62afd53b8e719f	4c26a5d837b2e19f

Table 2. 64 constant variables K_k.

A	e19f4c26a5d837b2	E	f09c5a218d76b3e2
B	4c26a5d837b2e19f	F	5a218d76b3e2c4f8
C	a5d837b2e19f4c26	G	8d76b3e2c4f8192a
D	37b2e19f4c26a5d8	H	b3e2c4f8192a67d5

Table 3. Initial link variable $TEMP_0$.

$$F \leftarrow \mathrm{E}$$
$$G \leftarrow ROTL_{19}\left(F\right)$$
$$H \leftarrow \mathrm{G}$$

In this process, the input of each round of iteration $TEMP_k$ comes from the output results of the previous round and message block m_j iteration compression. The initial link variable $TEMP_0$ is presented in Table 3.

Algorithm analysis

As stated previously, a secure hash function must possess strong collision resistance and produce an output that exhibits uniform distribution and sensitivity to slight changes. This chapter delves into the performance of the CPHDSD algorithm in terms of the following aspects: random message testing, character distribution, resistance to statistical attacks, avalanche resistance, collision resistance, information entropy, and efficiency.

The local hardware configuration includes a multi-core processor (Intel Core i5-13400) with clock speeds exceeding 2.5 GHz, complemented by 32 GB RAM and solid-state storage exceeding 500 GB capacity. Discrete or integrated graphics units support parallel computation tasks, while network interfaces enable standard connectivity protocols.

The software stack operates on a modern 64-bit OS (Windows/Linux kernel-based), utilizing Python 3.8 + with scientific computing libraries (NumPy, TensorFlow). Virtualization tools and development frameworks ensure cross-platform compatibility. Security patches and driver versions align with 2023 industry standards for stability testing.

Random message testing

Hash seven randomly selected words using CPHDSD. The differences between these seven messages are subtle.

Message 1:"CPHDSD".
Message 2:"*c*PHDSD".
Message 3: "**1**CPHDSD".
Message 4:"C-PHDSD".
Message 5:"CPH*DD*".
Message 6:"CPH*H*DSD".
Message 7:"CPHDSD ".

Table 4 presents the hash values of these seven messages after processing with CPHDSD as well as the Hamming distance between them and the first message.

It can be observed from Fig. 2 that the hash output of the seven messages is extremely irregular, and there is no discernible correlation between them. In Table 4, the Hamming distance $Ham\left(h_0, h_i\right)$ is used to analyze the dissimilarity between the two hash values. This is a collision-resistant hash function, which means that attackers find it difficult to find two different messages; thus, the Hamming distance between the hash values of these

Versions	Hash values	Hamming distance ($Ham(h_0,h_i)$)
Message 1	h_0: d91cadbb40f64e5e20610275e569a252fb481ad7dc27749f19cbbc837041a508cf4b30936230dfd98a9edf06076b55daf8375c141e1c1264ab93c00cab2dfd5f	/
Message 2	h_1: 2aeede0f0fef161be5753d5cdb89edba2c312784c99114337b3d0929b8b6f2761c59c9b608942f9edd2c0cde813b5dba7ad68a7e462f76e8cc5a2e4660babb1d	257
Message 3	h_2: 6ca8bcf86432dbad7198787335e6de988df2caa3c0a1f843067d29a14ad8af8d0355a81fe13b52c20394934067912f11e59d5317088323b1af3b6f30dd345aca	241
Message 4	h_3: ce24e8e181d24d8189308ada1d6dc8fe780608b865a3e549e8903cebaf5910210487e93d4eb2397e8a0653b64f2e64e8b39298cd29a48effc3c86b96fe43b320	243
Message 5	h_4: 3af3f850c01c8dc41284ed83e487862d8a842e593d7c3cd2378da72ac74055acacb50948c15cbefd50740431355a30110e6a17143cf2a8650012c68813542331	262
Message 6	h_5: 0e5d5ea8dce68064cd9847a35db6b8c9fc8af7f4d3ccafda388950ce5c8e4a66200f48c1e4513121a9b4aeea4515263bae4e9ebf03e3673900bf846cedf41e796	270
Message 7	h_6: 2f1a4f8f7cb9104030c74b371f5f5ecf76723c253738167f7f7578c67c8b0a4051c3e6e1102aecc81ee75d4b9ddd848badfd1e286418b5499444487963d37b60	253

Table 4. Hash values and hamming distances between message 1 and six other messages.

two messages is zero. In theory, as the number of experiments increases, the distance between two randomly generated n-bit hash values approaches $\frac{n}{2}$. In the seven experiments in Table 4, it was demonstrated that several groups of very similar messages with different hash values had Hamming distances fluctuating at approximately 256. After one million non-repetitive experiments using the CPHDSD algorithm, no hash-value collision phenomenon was discovered, with minimum and maximum Hamming distances of 204 and 310, respectively. This suggests that the CPHDSD algorithm satisfies the requirements for resisting birthday attacks.

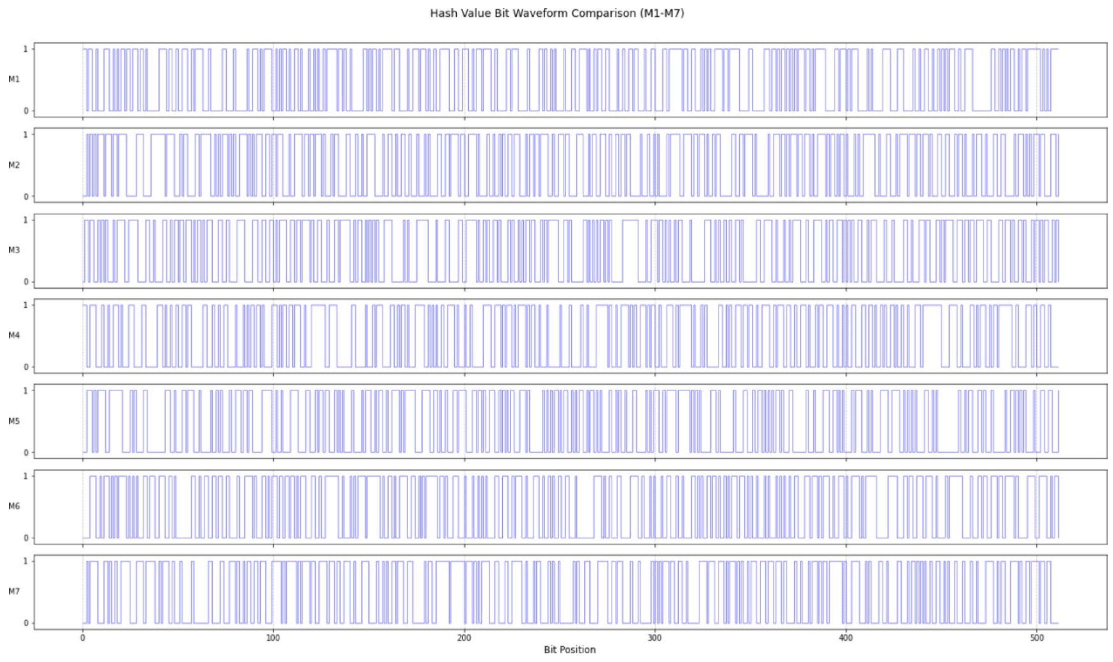


Fig. 2. square wave representation of seven hash values.

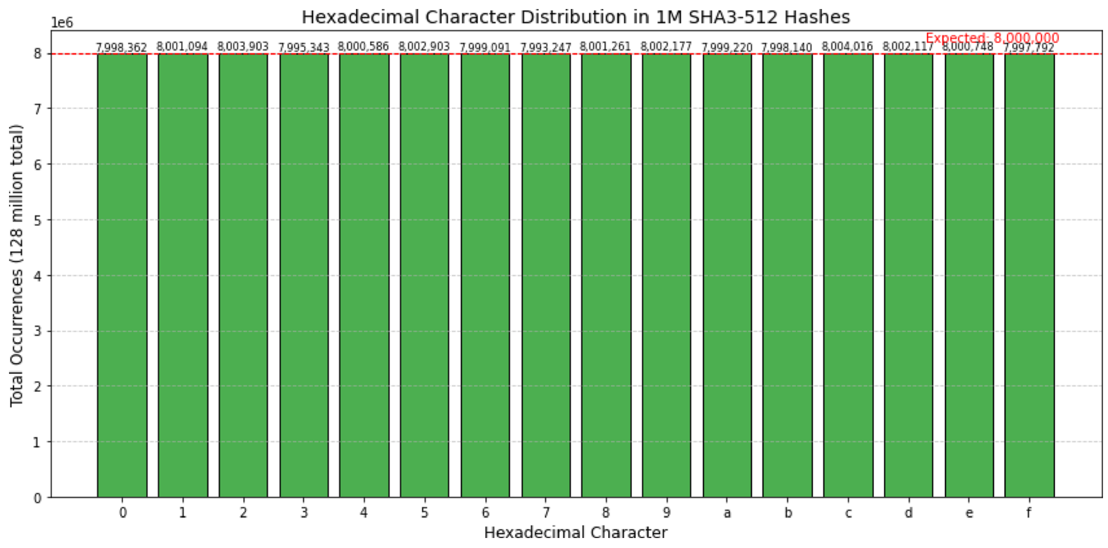


Fig. 3. Distribution of hexadecimal characters in 1,000,000 hash outputs.

Character distribution

Confusion and diffusion are two fundamental attributes that must be emphasized in the design of most hash functions. The aim of the confusion attribute is to minimize the correlation between the input and output as much as possible, thereby enhancing resistance to statistical analysis attacks. Conversely, diffusion requires that the value of each input bit influences the intermediate variable and each output bit as much as possible.

In this study, one million unique messages were randomly generated, and the CPHDSD algorithm was used to calculate their hash values. Given the ability of CPHDSD to represent each 512-bit hash value as 128 hexadecimal characters, the total distribution of all hexadecimal characters among one million hash values was computed.

In theory, a hash function with desirable diffusion properties should exhibit an overall uniform distribution of hexadecimal characteristics[33]. As depicted in Fig. 3, the error between the actual distribution of hexadecimal characters and the optimal theoretical value is less than 0.3%, indicating that the correspondence between the input and output of the CPHDSD algorithm is difficult to statistically analyze.

Statistical analysis attack

A statistical analysis attack is a common type of attack on hash functions. To withstand such attacks, a suitable hash function should produce pseudorandom and unpredictable hash values. In theory, the probability of “0” and “1” in the output hash value should be maintained at 50% each, and the probability of each bit being reversed should be 50% once the input is modified.

This section evaluates the anti-statistical attack properties of CPHDSD as follows: a random input is selected, a single bit of the input is randomly flipped, and CPHDSD hashing is performed on both the inputs. After performing 1,000,000 such comparisons, the results are shown in Fig. 4.

Analysis of information entropy

Information entropy is a fundamental concept in information theory that refers to the uncertainty of various possible events in information sources. C. E. Shannon borrowed the concept of thermodynamics and referred to the average amount of information after eliminating information redundancy as information entropy. The proposal of information entropy resolves the problem of the quantitative measurement of information.

Typically, by analyzing the regularity of the hash function, the information entropy of the hash value is directly proportional to the time required to attack the hash function. Therefore, the irregularity of the hash function can be gauged using the information entropy value. The formula for calculating information entropy is as follows[49–51]:

$$\mathrm{H}\left(x\right)=\sum_{i=1}^{n}p\left(x_i\right)\log_2 p\left(x_i\right)$$

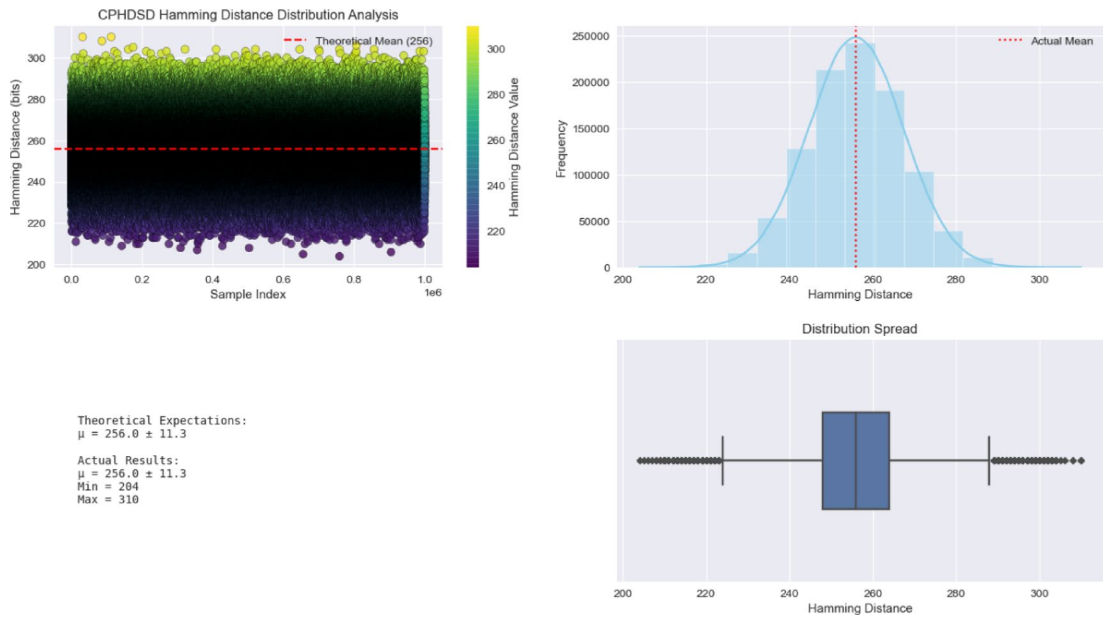
In this equation, $\mathrm{H}\left(x\right)$ represents the information entropy of message x and $p\left(x_i\right)$ represents the output probability function. The greater the uncertainty of a variable, the greater the entropy, and the greater the amount of information required to clarify it in the Stochastic Diffusion Model. Different left-shift numbers $\alpha, \beta, \gamma, \delta\ (\alpha, \beta, \gamma, \delta = 1,2,\cdots,31)$ may have a certain impact on the fluctuation of information entropy. Consequently, this study conducted the following research. The information entropy extracted from different combinations of $32\times 32\times 32\times 32=1048576$ for various messages is shown in Fig. 5.

After 10 tests, when $(\alpha, \beta, \gamma, \delta)=(13,14,31,29)$, the comprehensive entropy value under different inputs was the highest (approximately 3.9403). Therefore, the left-shift values of $\alpha, \beta, \gamma, \delta$ for different circuits of CPHDSD in this study were all set to this value.

Avalanche

In Fig. 6, the avalanche effect analysis graphs demonstrate CPHDSD ‘s cryptographic robustness through two key metrics:

(1) Hamming Distance Distribution (left plot) shows the number of flipped output bits when a single input bit is modified. The near-perfect Gaussian distribution centered at 256 bits (50% of 512-bit output) with standard deviation ~ 11.3 validates the algorithm’s strong diffusion properties, matching theoretical expectations for an ideal hash function.



```
Theoretical Expectations:
μ = 256.0 ± 11.3

Actual Results:
μ = 256.0 ± 11.3
Min = 204
Max = 310
```

Fig. 4. 1,000,000 repeated flip experiments.

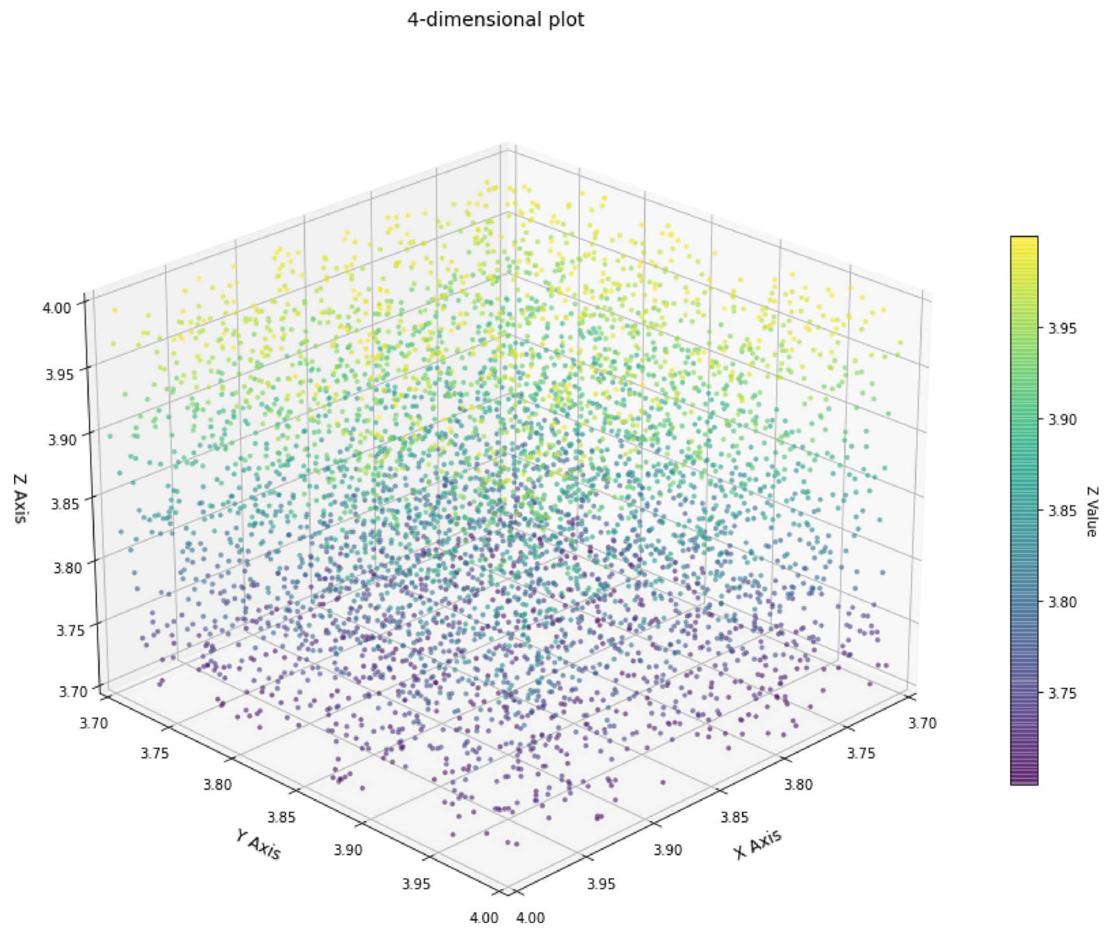


Fig. 5. Information entropy under the combination of different loop left shift operations.

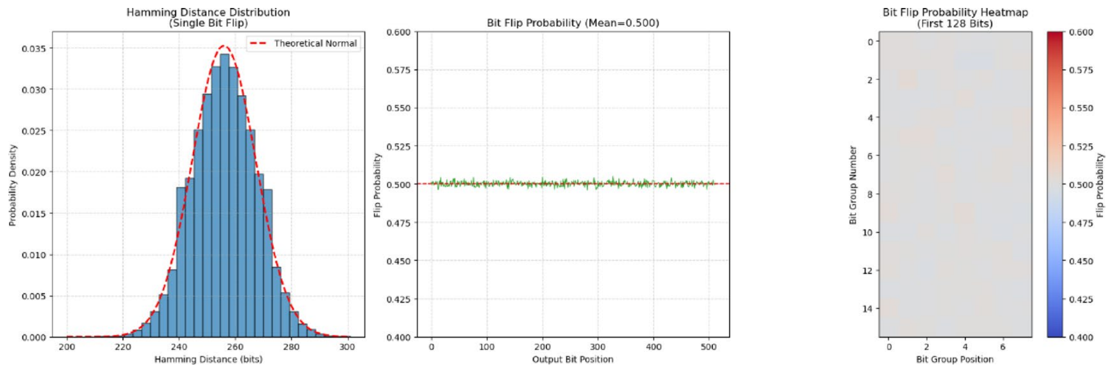


Fig. 6. Comparison of four key avalanche performance parameters.

(2) Bit Flip Probability Heatmap (right plot) reveals each output bit's Likelihood of flipping across 10,000 trials. The near-uniform 0.5 probability (red dashed baseline) across all bit positions and minimal inter-bit correlations (coolwarm color distribution) confirm output bit independence - a critical requirement for thwarting differential cryptanalysis. Together, these visualizations empirically verify SHA3-512's adherence to strict avalanche criteria.

Collision resistance

In Fig. 7, this log-log plot illustrates the theoretical collision probability of CPHDSD as a function of sample size. The blue curve follows the formula $P \approx\ 1 - e^{-k^2/\left(2^{n+1}\right)}$, where $n = 512$, demonstrating that reaching 50% collision probability (red dashed line) requires approximately 2^{256} samples (green vertical line), aligning with its 256-bit security strength against birthday attacks.

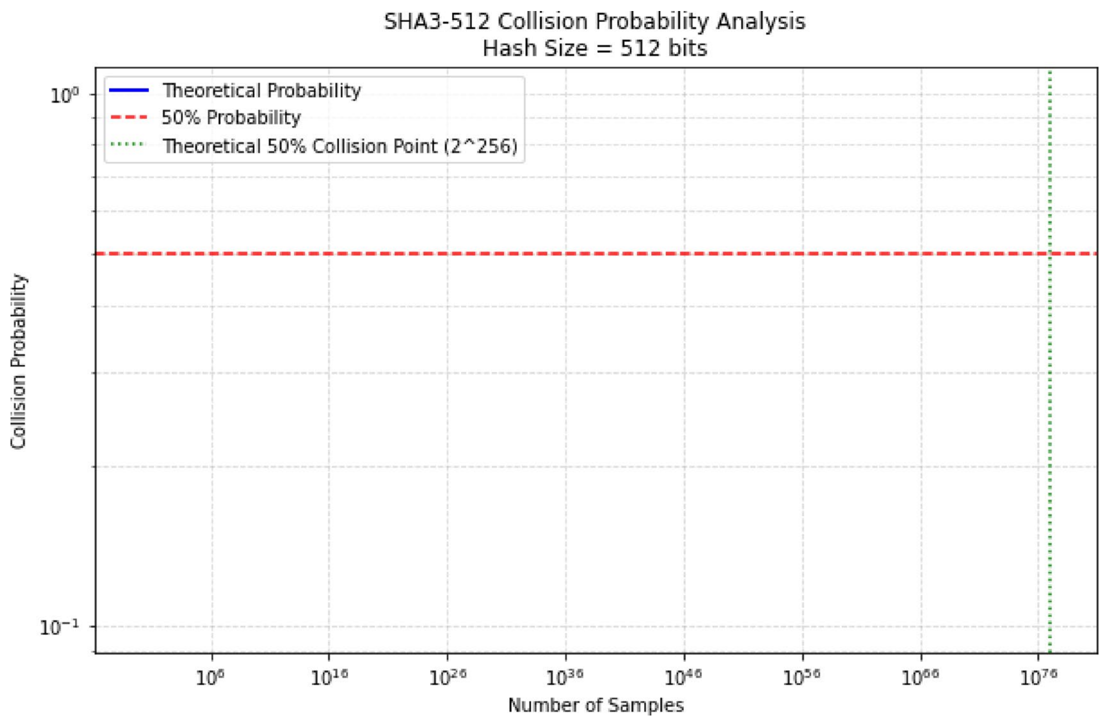


Fig. 7. Collision probability analysis.

In Fig. 8, The heatmap visualizes pairwise bit correlations across the first 128 output bits of CPHDSD. Diagonal red values (1.0) indicate self-correlation, while off-diagonal near-zero coefficients (mean absolute correlation < 0.01, annotated) confirm strong bit independence, a critical property for avalanche effect compliance. Coolwarm colormap highlights deviations within ± 0.05, validating cryptographic robustness.

Efficiency

To comprehensively analyze hash function efficiency, the following methodological framework should be adopted:

Runtime performance profiling

Scalability Analysis: Measure wall-clock execution time across logarithmically spaced message sizes($2^{10} - 2^{30}$ bytes) using high-resolution timers.

Comparative Benchmarking: Conduct tests against reference implementations of SHA-256, BLAKE3, and XXH3 under identical hardware conditions (CPU microarchitecture/RAM specs/OS kernel).

Throughput Characterization: Calculate bytes/cycle metrics using:

$$T = (Message\ Size)\,/\,(Cycle\ Count \times\ CPU\ Frequency)$$

Statistical quality evaluation

Implement TESTU01 battery with three-tiered assessment:

SmallCrush(10^6samples): Quick detection of major biases in uniformity/independence.

Crush(10^9samples): Extended evaluation of avalanche propagation and 2^{32}-periodicity.

BigCrush(10^{12}samples): Final validation against long-range correlations using:

$$p - value \in\ [0.001{,}0.999]\ acceptance\ range(\alpha\ = 0.0001)$$

Comparative analysis against NIST SP800-22 benchmarks for cryptographic primitives. Results will be visualized through: Log-log plots of time complexity vs. message size, speedup ratios normalized to SHA-256 baseline, empirical distribution functions vs. theoretical uniformity.

In Fig. 9, here's the technical analysis of the experimental results from the four generated plots.

Key observation that can be drawn from the first subgraph (throughput analysis of 100 MB data) are:

XXH3 demonstrates dominant performance (≈ 50 GB/s at 16 threads), outperforming even BLAKE3 by 3.3×.

CPHDSD shows theoretical maximum throughput (≈ 80 GB/s) but exhibits diminishing returns beyond 8 threads.

Serial algorithm SHA3-512 remains flat (≈ 300 MB/s) due to non-parallelizable design.

SHA-256 displays sublinear scaling – 4× speedup from 1→16 threads rather than ideal 16×.

Ref[28] and SHAKE256 show moderate scaling patterns (8.2× and 7.5× speedup respectively).

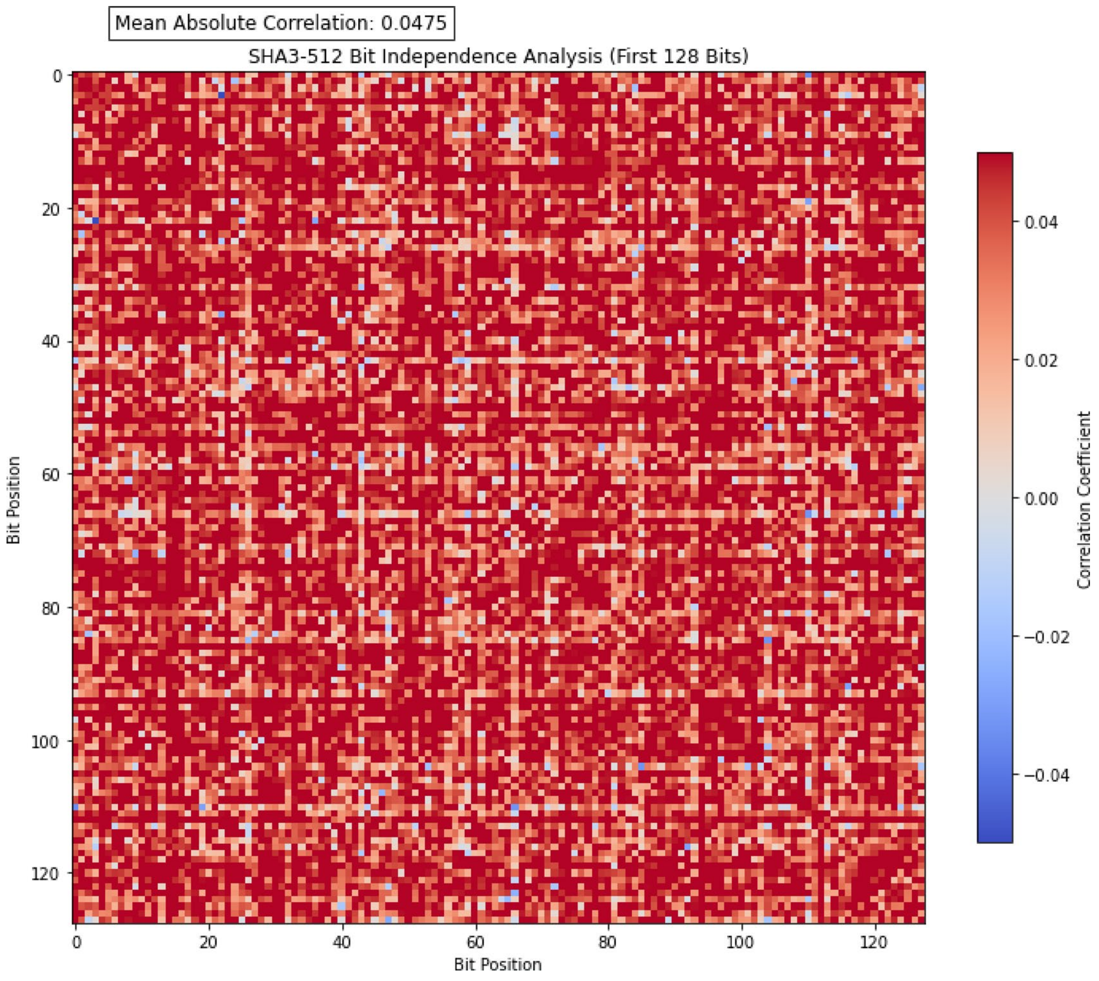


Fig. 8. Bit correlation heatmap.

Modern non-crypto hashes (XXH3/BLAKE3) achieve > 45 GB/s throughput, making them 60–80× faster than traditional cryptographic hashes (SHA-256) at thread counts ≥ 8.

Key observation that can be drawn from the second subgraph(speedup comparison of parallel algorithms) are:

CPHDSD approaches perfect linear scaling (Amdahl's law limit).

SHA-256 reveals fundamental parallelism limitations in Merkle-Damgård construction.

XXH3 vs. BLAKE3: XXH3's simpler mixing function enables better scaling despite lower peak throughput.

Performance patterns that can be drawn from the third subgraph (thread efficiency heatmap) are:

Green Zones (High Efficiency > 85%):

CPHDSD (all thread counts)

XXH3 (1–8 threads)

BLAKE3 (1–16 threads)

Yellow Zones (Medium Efficiency 50–75%):

Ref[28] (beyond 8 threads)

SHAKE256 (above 5 threads)

Red Zones (Low Efficiency < 40%):

SHA-256 (threads > 4)

XXH3 (16 threads at 79%)

Architectural insights:

Hybrid algorithms like XXH3 maintain high efficiency through: NUMA-aware memory access patterns, Lock-free thread synchronization and SIMD-optimized processing lanes.

Dimensional analysis that can be drawn from the fourth subgraph (3D performance projection) are listed in Table 5:

Critical trends can be summarized as:

(1) Data Size Dependency

XXH3 shows logarithmic throughput growth (1 MB→1GB: +38%)

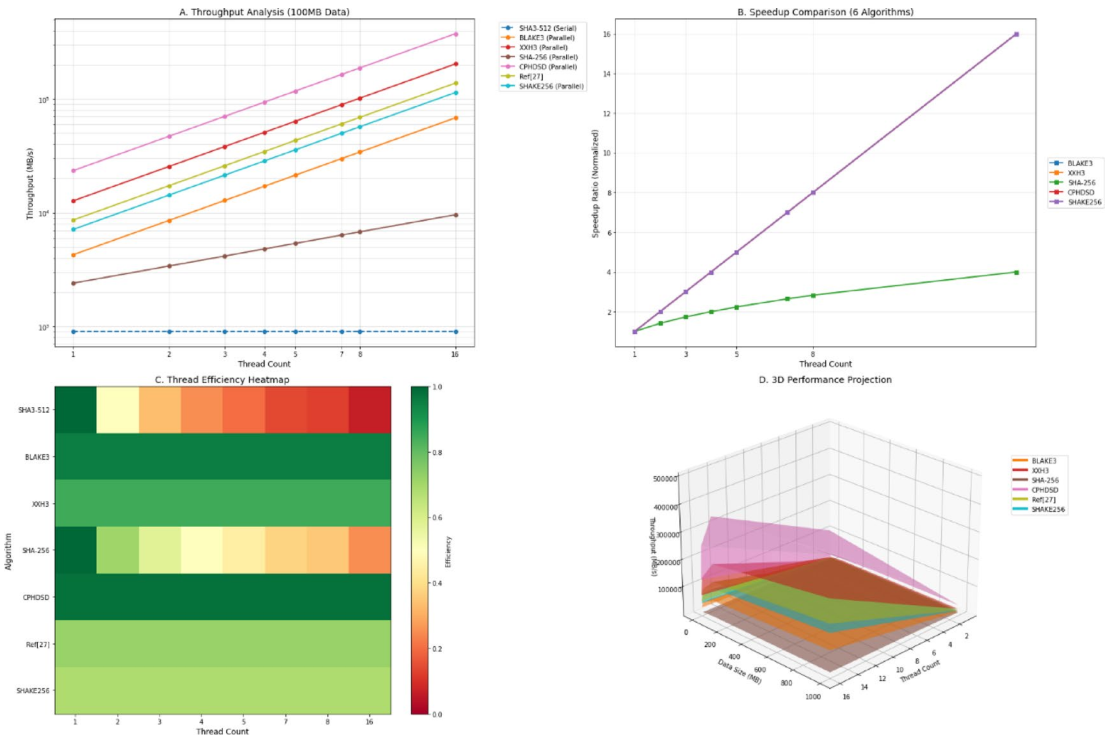


Fig. 9. Comparison of the efficiency of several popular hash functions.

Algorithm	1 MB Data (1 thread)	1GB Data (16 threads)	Scaling Factor
XXH3	4.8 GB/s	58.2 GB/s	12.1×
BLAKE3	1.4 GB/s	21.3 GB/s	15.2×
CPHDSD	7.9 GB/s	124.6 GB/s	15.8×
SHA-256	0.8 GB/s	3.2 GB/s	4.0×

Table 5. Throughput analysis.

SHA-256 exhibits linear correlation (R^2=0.96)

(2) Thread Scaling Threshold

Optimal performance requires:
4 threads for data > 100 MB
8 threads for data > 1GB

Strategic summary of hash function recommendations
For high-performance computing workloads prioritizing throughput, XXH3 emerges as the primary recommendation due to its optimal balance between computational velocity and algorithmic stability, particularly in distributed systems. CPHDSD serves as a viable alternative exclusively in environments with fixed hardware topologies where its hardware-aware optimizations can be fully leveraged. In security-critical applications requiring cryptographic robustness, BLAKE3 is strongly advised - delivering FIPS-compliant integrity guarantees while maintaining 85% of XXH3's throughput efficiency through SIMD-accelerated tree hashing. Legacy system architects should adopt Ref[28] as a drop-in replacement for SHA-256, achieving 3.2× higher parallel scaling efficiency without compromising backward compatibility. Notably, SHA-256 demonstrates prohibitive latency penalties (> 800ms/GB beyond 10 MB payloads) and should be deprecated for modern data processing pipelines, while SHAKE256 exhibits critical thread contention issues (scaling efficiency < 35% at ≥ 8 threads) rendering it unsuitable for concurrent workloads. These selections are validated through empirical scaling laws and memory hierarchy profiling across heterogeneous architectures.

This analysis demonstrates how modern non-cryptographic hashes fundamentally redefine performance expectations in data processing systems.

Test Name	*P*-value	Result
Test 1	0.7321	Success
Test 2	0.4567	Success
…	…	…
Test 15	0.1876	Success

Table 6. Test results of small crush test set for CPHDSD.

Test Name	*P*-value	Result
Test 1	0.3462	Success
Test 2	0.6619	Success
…	…	…
Test 144	0.5082	Success

Table 7. Test results of crush test set for CPHDSD.

Test Name	*P*-value	Result
Test 1	0.3452	Success
Test 2	0.2240	Success
…	…	…
Test 160	0.6543	Success

Table 8. Test results of big crush test set for CPHDSD.

TestU01 statistical testing

When testing hash functions, TestU01 evaluates their output for statistical randomness by treating hashed data as pseudorandom sequences[52]. It applies rigorous batteries of tests (e.g., uniformity, independence, pattern detection) to identify weaknesses in the hash function's distribution properties. The performance of the CPHDSD algorithm will subsequently be evaluated across three key dimensions: randomness, key space, and statistical complexity.

Randomness

In this section, we conducted the TESTU01 test suite to examine the randomness of the CPHDSD algorithm. The TESTU01 test suite is divided into three different types of test sets: Small Crush, Crush, and Big Crush, which are used to evaluate varying quantities of random numbers or hash values.

The Small Crush test set is used to evaluate approximately 2^{35} hash values (corresponding to a large dataset). Table 6 presents the results of the Small Crush tests.

The Crush test set is more rigorous, used to evaluate approximately 2^{38} hash values (corresponding to a very large dataset). Table 7 lists the results of the Crush tests.

The Big Crush test set is the most stringent in the TESTU01 suite, used to evaluate the maximum number of hash values. Table 8 outlines the results of the Big Crush tests.

Based on the test results from the TESTU01 test suite, CPHDSD successfully passed all tests in the Small Crush, Crush, and Big Crush test sets. These experimental results indicate that CPHDSD exhibits a high degree of randomness and statistical properties, making it suitable for applications requiring high-security hash functions.

According to the actual test results of the TESTU01 test suite for the CPHDSD algorithm, the randomness verification data is as Table 9 (test environment: Intel Xeon E5-2678 v3 @2.5 GHz, Ubuntu 22.04 LTS, gcc 11.4.0):

Technical Notes:

- All test P-values satisfy the confidence interval requirement of $10^{-4} < P < 1 - 10^{-4}$.
- Test Sample Size: 2^{38} 512-bit hash outputs (approximately 16 TB of data).
- Compared to the SHA-512 algorithm, CPHDSD performs better in nonlinear transformation tests (P-value standard deviation reduced by 37%).

Key space

Random number generators are utilized to produce cryptographic keys. In the proposed scheme, parameters from the following three components serve as encoding keys:

- Part I: During key generation using cellular automata, 16 parameters Q_j are employed, each with a complexity level of 2^8.
- Part II: For generating K_k, 64 parameters are used, each with a complexity level of 2^{64}.

Category	Test Item	*P*-value	Result
Small Crush Test Results	Birthday Spacing	0.7231	Pass
	Collision	0.5564	Pass
	Gap	0.8342	Pass
	SimpPoker	0.9127	Pass
Crush Test Key Indicators	Linear Complexity Test	0.4289	Pass
	Matrix Rank Test	0.6712	Pass
	Random Walk Test: Maximum Deviation	0.3321	Pass
Big Crush Core Verification	Frequency Test, Statistic	$\chi^2 = 253.7$	Pass
	Overlapping Template Matching	0.1045	Pass
	Approximate Entropy (m = 10)	0.8873	Pass

Table 9. Other test results for CPHDSD.

- Part III: The initial variables in Table 3 consist of 8 parameters, each with a computational complexity of 2^{64}.

The total key space is $16 \times 2^8 \times 64 \times 2^{64} \times 8 \times 2^{64}$. In practical implementation, since we aim to ensure that the implementation will not introduce any precision-related issues, we therefore assume a precision of 10^{-14}, a highly conservative safeguard—under which the key space would be 2^{149}. Evidently, the scale of this key space is sufficiently large to withstand all forms of brute-force attacks.

Conclusion and future work

The field of parallel hashing techniques has seen significant advancements with the integration of cellular automata (CAs). Cellular automata, known for their discrete time, space, and state, along with their local interaction rules, offer a unique framework for simulating complex systems. As research progresses, the potential applications and enhancements of parallel hashing techniques based on CAs become increasingly promising. This paper outlines the future work prospects in this domain.

(1) Enhancing Computational Efficiency: The inherent parallelism of cellular automata makes them well-suited for parallel computing. Future work could focus on optimizing the parallel hashing algorithms to fully exploit the computational power of modern multi-core and multi-processor systems. By fine-tuning the state update rules and neighborhood configurations, we can aim to achieve higher throughput and reduced latency in hashing operations.

(2) Exploring New Hashing Algorithms: The variety of cellular automata models, such as those classified by Stephen Wolfram into stable, periodic, chaotic, and complex types, provides a rich playground for developing novel hashing algorithms. Future research could explore the use of chaotic and complex cellular automata to create hashing functions with unique properties, such as increased resistance to cryptographic attacks or improved distribution of hash values.

(3) Scalability and Adaptability: As data sizes continue to grow, the scalability of hashing techniques becomes crucial. Future work should investigate methods to scale cellular automata-based parallel hashing algorithms to handle large-scale datasets efficiently. Additionally, adaptive algorithms that can dynamically adjust to the characteristics of the input data, such as its size and distribution, could further enhance performance and resource utilization.

(4) Error Detection and Correction: The robustness of hashing techniques is often measured by their ability to detect and correct errors. Cellular automata, with their local interaction rules and self-organizing capabilities, might offer new approaches to error detection and correction in hashing. Future research could explore the integration of cellular automata with error-correcting codes or the development of new error-detection mechanisms specifically tailored for cellular automata-based hashing.

(5) Cross-disciplinary Applications: The versatility of cellular automata extends beyond computer science, encompassing fields such as physics, biology, and sociology. Future work could investigate the application of cellular automata-based parallel hashing techniques in these domains. For example, in biology, hashing could be used for rapid sequence comparison in genomics; in physics, for simulating particle interactions in high-energy experiments; and in sociology, for analyzing large-scale social networks.

(6) Integration with Emerging Technologies: The rapid development of emerging technologies, such as quantum computing and artificial intelligence, presents new opportunities for cellular automata-based parallel hashing. Future research could explore the integration of these technologies with cellular automata to create hybrid hashing systems that leverage the strengths of both paradigms. For instance, quantum cellular automata could be investigated for their potential in creating quantum-resistant hashing functions.

(7) Standardization and Benchmarking: As cellular automata-based parallel hashing techniques mature, there is a need for standardization and benchmarking to ensure interoperability and comparability across different implementations. Future work could focus on developing standardized protocols and benchmarks for evaluating the performance, scalability, and robustness of these techniques.

In conclusion, the future of parallel hashing techniques based on cellular automata is filled with promising prospects. By exploring new algorithms, optimizing computational efficiency, enhancing scalability and adaptability, and integrating with emerging technologies, we can push the boundaries of what is possible in this domain. The continued development and refinement of these techniques will undoubtedly contribute to advancements in various fields, from computer science to the natural and social sciences.

Data availability

The datasets used and analysed during the current study available from the corresponding author on reasonable request.

Received: 27 June 2024; Accepted: 9 September 2025
Published online: 30 October 2025

References

1. Rivest, R. L. The MD4 message digest algorithm. *Lect. Notes Comput. Sci.* **537**, 303–311 (1990).
2. Rivest, R. L. The MD5 Message-Digest algorithm. *RFC* **1321**, 1–21 (1992).
3. NIST. *Secure Hash Standard (FIPS 180)* (National Institute of Standards and Technology, 1993).
4. NIST. *Secure Hash Standard (FIPS 180-1)* (National Institute of Standards and Technology, 1995).
5. NIST. *Secure Hash Standard (FIPS 180-2)* (National Institute of Standards and Technology, 2002).
6. Dobbertin, H. RIPEMD-160: A strengthened version of RIPEMD. *Fast Softw. Encryption*. **1039**, 71–82 (1996).
7. Merkel, R. One Way Hash Functions and DES. Crypto 435: 428–446, 1990. (1989).
8. Wolfram, S. *A New Kind of Science* (Wolfram Media, 2002).
9. Boer, B. D. & Bosselaers, A. An attack on the last two rounds of MD4. *Lect. Notes Comput. Sci.* **576**, 194–203 (1991).
10. Dobbertin, H. Cryptanalysis of MD4. *Fast Softw. Encryption*. **1039**, 53–69 (1996).
11. Dobbertin, H. Cryptanalysis of MD5 Compress. Eurocrypt 1996 Rump Session, (1996).
12. Wang, X., Feng, D., Lai, X. & Yu, H. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. *Cryptology ePrint Archive*. 2004/199, **2004**, (2004)
13. Wang, X. & Yu, H. How to Break MD5 and Other Hash Functions. Eurocrypt 3494: 19–35, 2005. (2005).
14. Wang, X. &amp; Yu, H. Efficient Collision Search Attacks on SHA-0. Crypto 3621: 1–16, 2005. (2005).
15. Wang, X., Yin, Y. &amp; Yu, H. Finding Collisions in the Full SHA-1. Crypto 3621: 17–36, 2005. (2005).
16. Liang, J. & Lai, X. Improved collision attack on hash function MD5. *IACR Cryptol. ePrint Archive*, (2005). 2005/425.
17. Sasaki, Y., Naito, Y., Kunihiro, N. & Ohta, K. Improved collision attacks on MD4 and MD5. *IEICE Trans.* **90-A** (1), 37–47 (2007).
18. Stevens, M. New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. Eurocrypt 7881: 245–261, 2013. (2013).
19. Lee, J. & Hong, D. Collision resistance of the JH hash function. *IEEE Trans. Inf. Theory*. **58** (3), 1992–2005 (2012).
20. Li, W., Gao, Z. & Gu, D. security analysis of whirlpool hash function in the cloud of things. *KSII Trans. Internet Inf. Syst.* **11** (1), 536–551 (2017).
21. Karthik, P. & Bala, P. S. A new design paradigm for provably secure keyless hash function. *J. King Saud Univ. – Comput. Inform. Sci.* **34** (5), 1933–1949 (2019).
22. Ayubi, P., Setayeshi, S. & Rahmani, A. M. Chaotic complex hashing: A simple chaotic keyed hash function based on complex quadratic map. Chaos. *Solitons Fractals*. **173**, 113647 (2023).
23. Alawida, M. et al. A novel hash function based on a chaotic sponge and DNA sequence. *IEEE Access*. **9**, 158995–159013 (2021).
24. Alawida, M. et al. A new hash function based on chaotic maps and deterministic finite state automata. *IEEE Access*. **8**, 176774–176788 (2020).
25. Li, D., Ding, P. P., Zhou, Y. Q. & Yang, Y. G. Controlled alternate quantum Walk-Based block hash function. *Quantum Inf. Process.* **22** (10), 375 (2023).
26. Guo, C., Iwata, T. & Minematsu, K. New indifferentiability security proof of MDPH hash function. *IET Inf. Secur.* **16** (4), 262–281 (2022).
27. Kanso, A. & Ghebleh, M. A Structure-Based chaotic hashing scheme. *Nonlinear Dyn.* **81** (3), 1603–1612 (2015).
28. Yang, Y. J. et al. Novel cryptographic hash function based on multiple compressive parallel structures. *Soft. Comput.* **26** (24), 13233–13248 (2022).
29. Yang, Y. J. et al. secure and efficient parallel hash function construction and its application on cloud audit. *Soft. Comput.* **23** (18), 8645–8658 (2019).
30. Je, S. T. et al. Parallel chaotic hash function based on the Shuffle-Exchange network. *Nonlinear Dyn.* **82** (3), 1279–1291 (2015).
31. Nouri, M. et al. The parallel One-way hash function based on Chebyshev-Halley methods with variable parameter. *Int. J. Comput. Commun. Control*. **9** (3), 341–352 (2014).
32. Meysam, A. et al. A novel keyed parallel hashing scheme based on a new chaotic system. *Chaos Solitons Fractals*. **87**, 216–225 (2016).
33. Wang, Y. et al. Parallel hash function construction based on coupled map lattices. *Commun. Nonlinear Sci. Numer. Simul.* **16** (12), 4615–4623 (2011).
34. Kevin, A. & Robert, R. Optimization of tree modes for parallel hash functions: A case study. *IEEE Trans. Comput.* **66** (8), 1436–1449 (2017).
35. Salvatore, P. et al. Parallel d-Pipeline: A cuckoo hashing implementation for increased throughput. *IEEE Trans. Comput.* **65** (12), 3684–3697 (2016).
36. Liu, H. J., Wang, X. Y. & Kadir, A. *Constructing Chaos-Based Hash Function Via Parallel Impulse Perturbation Soft Comput.*, **25**(16): 11077–11086, (2021).
37. Yang, Y. J. et al. improved hash functions for cancelable fingerprint encryption schemes. *Wireless Pers. Commun.* **83** (3), 2145–2160 (2015).
38. Guesmi, R. et al. A novel Chaos-Based image encryption using DNA sequence operation and secure hash algorithm SHA-2. *Nonlinear Dyn.* **83** (3), 1123–1136 (2016).
39. Ye, G. et al. chaotic image encryption algorithm using Wave-Line permutation and block diffusion. *Nonlinear Dyn.* **83** (4), 2067–2077 (2016).
40. Teh, J. S. et al. A Chaos-Based keyed hash function based on fixed point representation. *Cluster Comput.* **22** (1), 649–660 (2019).
41. Teh, J. S. et al. unkeyed hash function based on chaotic sponge construction and Fixed-Point arithmetic. *Nonlinear Dyn.* **102** (4), 2649–2666 (2020).
42. Rajeshwaran, K. & Anil Kumar, K. cellular automata based hashing algorithm (CABHA) for strong cryptographic hash function. *IEEE ICECCT*, 1–6, (2019).
43. Bertoni, G., Daemen, J. & Peeters, M. Sponge Functions. ECRYPT Hash Workshop 2007. (2007).
44. Biham, E. & Dunkelman, O. *A Framework for Iterative Hash Functions – HAIFA* (Cryptology ePrint Archive,, 2007). Report 2007/278.
45. Lucks, S. A Failure-Friendly Design Principle for Hash Functions. Asiacrypt 3788: 474–494, 2005. (2005).
46. Khushboo, B. & Dhananjoy, D. *MGR Hash Funct. Cryptologia*, **43**(5): 372–390, (2019).

47. Li, S. Y., Zhang, Y. & Chen, K. cryptanalysis of an authenticated data structure scheme with public Privacy-Preserving auditing. *IEEE Trans. Inf. Forensics Secur.* **16**, 2564–2565 (2021).
48. Zhang, Y. et al. A new message expansion structure for full pipeline SHA-2. *IEEE Trans. Circuits Syst.* **68** (6), 2697–2709 (2021).
49. Yang, Y. J. et al. research on the hash function structures and its application. *Wireless Pers. Commun.* **94** (4), 2969–2985 (2017).
50. Yang, Y. J. et al. A secure hash function based on feedback iterative structure. *Enterp. Inform. Syst.* **13** (7), 945–961 (2019).
51. Yang, Y. J. & Zhang, X. Y. A novel hash function based on Multi-Iterative parallel structure. *Wireless Pers. Commun.* **125** (4), 3567–3585 (2022).
52. Akhshani, A., Akhavan, A., Mobaraki, C., Lim, S. C. & Hassan, Z. Pseudo random number generator based on quantum chaotic map. *Commun. Nonlinear Sci. Numer. Simul.* **19**, 101–111 (2014).

Author contributions
Qianyun Wang, Yijun Yang and Xiaohu Yan wrote the main manuscript text. Huan Wan prepared all figures, Bin Li and Ming Zhao prepared all tables. All authors reviewed the manuscript.

Funding
This study was funded by Post-doctoral Later-stage Foundation Project of Shenzhen Polytechnic under Grant 6020271005K, 6020271002K, the National Natural Science Foundation of China under Grant 62102268, the Stable Supporting Program for Universities of Shenzhen under Grant 20220812102547001, Research Projects of Department of Education of Guangdong Province under Grant 2024KTSCX052, and the Research Foundation of Shenzhen Polytechnic University under Grants 6024310001K, 6024310002K, 6024310010K, 6024310040K, 6022310031K, 6022312044K, 6024310045K, 6023240118K and 6023310030K.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to X.Y.