Future Technology
Open Access Journal
ISSN 2832-0379

November 2026| Volume 05 | Issue 04 | Pages 310-319

Journal homepage: https://fupubco.com/futech

https://doi.org/10.55670/fpll.futech.5.4.25

Future Publishing LLC

Article

A new hash function based on irreversible finite automata

Ghassan Salloom*, Layth Hassnawi, Karam Mohammed, Mohammed Adhab

Scientific Research Commission, Baghdad, Iraq

ARTICLE INFO

Article history:
Received 16 April 2026
Received in revised form
10 September 2026
Accepted 24 September 2026

Keywords:
Cryptographic hash function,
Irreversible automata, Avalanche effect,
Collision resistance, Diffusion, One-way function

*Corresponding author
Email address:
ghassan.kh.salloom@src.edu.iq

DOI: 10.55670/fpll.futech.5.4.25

ABSTRACT

A mathematical hash function maps inputs of arbitrary length to unique outputs (digests) of a fixed length. It is highly useful and used in almost all information security applications; it can also serve as index data in hash tables, detect duplicate data for fingerprinting or uniquely identifying files, and serve as well for checksums to identify data corruption. In this work, a novel 256-bit cryptographic hash function based on the irreversible Mealy finite automata model (IFA) is introduced. In this architecture, the irreversible transition matrix of the Mealy automaton (IFA) is constructed using a non-injective state-transition mechanism to introduce local irreversibility and ambiguous backward reconstruction. The automaton output table is precomputed using a Chebyshev-based nonlinear generator to improve output randomness and nonlinear complexity. For further diffusion, confusion, collision resistance, and one-wayness improvements, we integrate the irreversible automaton within a sponge-based construction with a 512-bit internal state (256-bit rate + 256-bit capacity), yielding a designed collision resistance of $c/2 = 128$ bits and preimage resistance of $min(d, c) = 256 - bit$ under the sponge capacity bound. Evaluation against SHA-256 shows an avalanche mean of 50.02% (σ = 3.14%), all 13 NIST SP 800-22 statistical tests passed, and zero collisions among 50,000 tests. Throughput is about 13 MB/s versus 183 MB/s for SHA-256, reflecting the sequential dependency of the Mealy transformation.

1. Introduction

Mathematical hashing functions are cryptographic algorithms that map data of any size to a fixed-length digest. These functions are crucial for security in information technology systems, such as digital signatures, message authentication codes, password protection, and blockchain technology. The main characteristics of hashing functions include preimage resistance, second-preimage resistance, and collision resistance [1]. The two most prevalent hashing algorithms in use today, for instance, SHA-2 [2] and SHA-3 [3], are based on iterative rounds of arithmetic, bitwise, and permutation operations within Merkle–Damgård or sponge constructions. In this paper, we explore a novel method that uses the self-nature property of irreversibility in finite automata as the mechanism of one-wayness. It has a non-injective transition function, i.e., each state-input pair maps to the same successor, preventing unique backward reconstruction. In this study, we adapted this property and used it in sponge construction to build a hash function that satisfies standard security requirements. Moreover, the present work uses Chebyshev polynomials in a different role: not as the hash engine itself, but as the nonlinear generator for the Mealy machine's output table, combining automata-theoretic irreversibility with chaos-theoretic nonlinearity. Nemoures' hash functions based on automata and chaos proposals rely on cellular automata or chaotic maps as mixing engines. They validate their output using only statistical tests such as NIST SP 800-22 and avalanche effects, without exploiting the structural irreversibility of finite-state transitions as a dedicated source of one-wayness, and without embedding the construction within a framework that provides formal security bounds. This gap motivates the present work. The objectives of this work are:

- To design a cryptographic hash function whose one-way property is derived from the structural irreversibility of a Mealy finite automaton, rather than from arithmetic operations;
- To integrate Chebyshev polynomial-based nonlinear output generation into the automaton's output function, providing non-linearity independent of the transition mechanism;
- To embed the irreversible Mealy transformation within a sponge construction to achieve formal collision and preimage resistance bounds; and (4) to evaluate the resulting hash function against SHA-256 using standard statistical tests, avalanche analysis, and structural irreversibility metrics.

The paper is structured as follows: Section 2 surveys related work on hash function design. In Section 3, we introduce our proposal, IFA-Hash, and present the mathematical model, data structure requirements, transformation layer, and required algorithms. Section 4 compares it against SHA-256. In Section 5, we discuss security and limitations. Section 6 concludes.

2. Related work

We divide related work into three areas: hash function development, automata-based cryptography, and chaos-based hash functions.

2.1 Hash function evolution

The development of cryptographic hash functions can be presented in three generations. The first ones, such as MD5 [4] (128-bit output) and SHA-1 [5] (160-bit output), used the Merkle–Damgård construction, processing the message in fixed-size blocks through an iterated compression function. In 2004, MD5 collisions were practically demonstrated in [6], while Stevens et al. [7] produced the first collision for full SHA-1 in 2017; hence, both algorithms are unsuitable for security applications. Whereas the second generation includes the SHA-2 family [2], such as SHA-256 and SHA-512, which strengthened the Merkle–Damgård design with larger state sizes (256 and 512 bits), more complex message expansion, and additional mixing operations. SHA-256 and SHA-512 remain widely trusted and deployed. However, both versions have the length-extension vulnerability that is intrinsic to all Merkle–Damgård constructions, i.e., given $H(M)$, an attacker can compute $H(M||M')$ for a chosen suffix M'without knowing M, because the digest reveals the full internal state [8]. Finally, SHA-3 [3] is the third generation for hashing schemes, represented and standardized by NIST in 2015, after a multi-year public competition. The digest generator uses the Keccak sponge construction [9], which utilizes a permutation-based absorb-squeeze structure to replace the compression-function model. The sponge's capacity parameter design provides a clean security argument and eliminates length-extension attacks. In contrast, to achieve throughputs exceeding 1 GB/s on modern processors, BLAKE2 [10] and BLAKE3 [10,11] offer high-speed alternatives based on different internal structures.

2.2 Automata-based cryptography

Alawida et al. [12] introduced a hash function that combines chaotic maps with deterministic finite state automata (DFA), where the structural complexity of automaton transitions (states) can improve collision resistance. Nanda et al. [13] developed a 128-bit cellular automata-based hash function for data authentication, achieving competitive performance on resource-constrained platforms. Wu and Chang [14] proposed a secure one-way hash function using cellular automata, specifically designed for IoT environments. These works show that automata can serve as viable cryptographic engines, but none exploit the specific irreversibility property of a Mealy machine.

2.3 Chaos-based hash functions

Chaos-based hash functions exploit the sensitivity to initial conditions, ergodicity, and pseudo-randomness of discrete chaotic maps to construct one-way compression functions. Early hash functions used one-dimensional maps, such as the logistic map and tent map, while more recent designs exploit higher-dimensional or hybrid maps for improved security. Akhavan et al. [15] proposed a parallel hash function based on a 3D chaotic map using Chebyshev polynomials as a building block, showing that Chebyshev maps can produce digest values with strong statistical properties. Li et al. [16] designed a chaotic complex hashing scheme based on complex quadratic maps, generating variable-length hash codes with excellent collision resistance. Tang et al. [17] suggested a hyperchaotic hash function based on a 2D linear cross-coupled map with a parallel feedback structure, improving efficiency through concurrent processing of data blocks. Salloom et al. [18] introduced a 256-bit hash function, which utilizes a static 256×256 chaotic substitution box and nonlinear logistic map transformations. The introduced hash function was evaluated in terms of security and performance. The security analysis is tested through a series of statistical and cryptanalytic methods. The results demonstrate strong cryptographic properties, including high avalanche performance, balanced bit distribution, and total resistance to collisions. The performance analysis showed that it performs poorly compared with SHA-256 because of its nonlinear operations and substitution-box overhead.

2.4 Sponge constructions

Bertoni et al. [9] introduced the sponge construction; it's a flexible framework for building hash functions and related cryptographic primitives from a fixed-length permutation or transformation. Unlike traditional hash designs, processing consists of two phases: an absorbing phase, in which message blocks are incorporated into an internal state, and a squeezing phase, in which output blocks are generated. The sponge paradigm has since been adopted by numerous lightweight and specialized designs, including ASCON (selected as the NIST lightweight cryptography standard in 2023), Xoodyak, and PHOTON.

Recently, Alawida et al. [19] proposed a hash function based on a chaotic sponge combined with DNA sequence operations, and Ma et al. [20] developed an enhanced image hash using cellular automata with sponge construction, demonstrating the versatility of the sponge framework beyond traditional permutation-based designs. The present work follows this direction by embedding an irreversible Mealy machine as the sponge transformation. In a standard sponge with a bijective permutation, the round function is invertible; any attacker who learns the full state can run it backward. The non-injective Mealy transformation provides an additional barrier: even with the full 512-bit state at round t, the predecessor at round $t-1$ cannot be uniquely determined. This complements, but does not replace, the sponge capacity bound. Bertoni et al. [9] proved generic collision $(c/2)$ and preimage $\left(min(d,c)\right)$ bounds for the sponge and subsequently proved indifferentiability from a random oracle for both permutation-based and transformation-based sponges.

3. Proposed hash function

This approach introduces a new cryptographic 256-bit hash function whose core engine is an irreversible Mealy machine automaton (IFA); for more details on irreversible automata, see [21]. The proposed irreversible automaton comprises a Chebyshev [22,24] nonlinear output matrix (output function) and a non-injective transition matrix to improve confusion and one-way characteristics, respectively. To further improve complexity and the unpredictability of internal-state evolution, we integrate an irreversible finite automaton within a sponge-based hashing framework.

3.1 Mathematical model of Mealy machine

A Mealy machine is the main component of the work introduced. It is a finite-state machine (FSM) and a variant of a transducer machine [25-27]. It is widely used in cryptography due to its compactness and ability to produce complex, context-dependent output, formally defined as a 6-tuple:

$$A = (Q, \sum, \Gamma, \delta, \lambda, q_0) \quad (1)$$

where Q is a set of non-empty finite states, Σ is a set of non-empty input signal alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function (two-dimensional matrix), $q_0 \in Q$ is the initial state, Γ is a set of non-empty output signal alphabets, and $\lambda : Q \times \Sigma \rightarrow \Gamma$ is the output function.

To adapt the Mealy automaton as an irreversible automaton, the transition function δ is required to be non-injective in every row, i.e., for each input symbol $x \in \Sigma$, the mapping $q \mapsto \delta(q, x)$ must send at least two distinct states to the same successor. Formally defined as:

$$\forall x \in \Sigma, \exists\ q_i \neq q_j \ such\ that\ \delta(q_i, x) = \delta(q_j, x) \quad (2)$$

This condition guarantees that the transition function is not invertible; i.e., given a successor state and an input symbol, there is no unique predecessor. Figure 1 illustrates an example of the irreversible Mealy machine automaton.

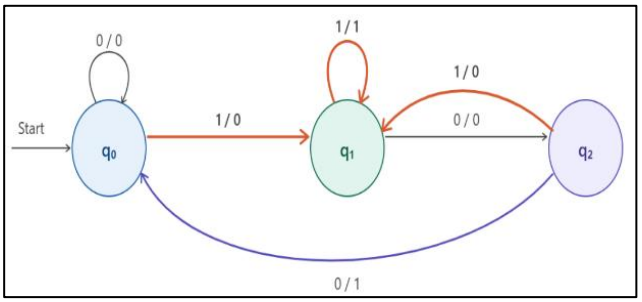

Figure 1. Non-injective transitions

Figure 1 shows that different (state, input signal) pairs can lead to outputs that make a unique reversal impossible. For input signal $x = 1$, the transition function δ maps all three states to the same successor i.e., $\delta(q_0, x) = q_1, \delta(q_1, x) = q_1$ and $\delta(q_2, x) = q_1$. Thus, given only the successor (q_1) and the input signal $x = 1$, an observer cannot determine whether the automaton (A) was previously in state q_0, q_1 or q_2. Similarly, for input signal $x = 0$, both $\delta(q_0, x) = \delta(q_2, x) = q_0$, so the predecessor of the state q_0 under the same input signal $x = 0$ is ambiguous. Tables 1 and Table 2 illustrate both transition and output functions.

Table 1. Transition function $\boldsymbol{\delta(q, x)}$

δ	$x = 0$	$x = 1$
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_1

Table 2. Output function $\lambda(q,x)$

λ	$x = 0$	$x = 1$
q_0	0	0
q_1	0	1
q_2	1	0

Practically, the irreversible transition matrix δ is generated by computing a deterministic state-dependent mapping for each state-input-signal pair. Moreover, we use a Pseudo-Random Number Generator (PRNG) with an initial seed. To enforce irreversibility conditions, controlled non-injectivity is introduced by assigning identical next-state values to selected distinct input symbols within each row. As a result, multiple state-input pairs may lead to the same next state. Thus, this prevents unique backward reconstruction and introduces local irreversible behavior in the automaton. Algorithm 1 shows the deterministic steps to generate the irreversible transition matrix δ.

Algorithm 1. Transition matrix generator

Procedure: GenerateTransitionMx
Input: $N = 256, seed = 42$
Output: $Delta[N][N]$ // Transition matrix
$For\ q = 0\ to\ N-1\ do$ // Generate deterministic transition
$For\ x = 0\ to\ N-1\ do$
$Delta[q][x] \leftarrow PRNG()\ mod\ N$
$End\ For$
$End\ For$
$For\ q = 0\ to\ N-1$ // enforce irreversibility conditions
$For\ x = 0\ to\ N-1\ step\ 2\ do$
$Delta[q][x+1] \leftarrow Delta[q][x]$ // $\Delta[q][x+1]$,
$End\ For$
$End\ For$
Return $Delta$

The second part of the irreversible automaton is the output function $\lambda : Q \times \Sigma \rightarrow \Gamma$. Where λ is represented as a precomputed 256×256 lookup table. To generate this nonlinear matrix based on Chebyshev polynomial (see Algorithm 2), the automaton state and input symbol (q, x) are first combined into a 16-bit intermediate value using bitwise mixing operations (i.e., $v = (q \ll 8) \oplus x$). The resulting value is normalized into the interval [−1, 1], using the following formula:

$$z = 2 \times \left(\frac{v}{65535}\right) - 1 \quad (3)$$

Then processed using a Chebyshev polynomial transformation to generate nonlinear output bytes for the automaton output matrix. The Chebyshev polynomial formula is defined as follows:

$$T_n(z) = cos(n \times arccos(z)),\ -1 \leq z \leq 1 \quad (4)$$

where the Chebyshev polynomial has three properties that make it a source of non-linearity in the output matrix:

- Chaotic behavior for n ≥ 2
- A uniform invariant measure
- Non-linearity over $GF(2)$.

On the other hand, the degree n was selected $(n = 7)$ based on the measured cryptographic properties of the resulting λ table. Table 3 compares λ table against the advanced encryption system (AES) S-box and random 8-bit function using standard substitution-table evaluation.

Based on the above table, the average nonlinearity (99.0) falls within the expected range for a random 8-bit function (92-100). While the algebraic degree is maximal at 7. The SAC deviation (0.035) is comparable to random tables. The differential uniformity (14) is higher than AES (4), which is expected for a non-bijective table, since bijectivity is deliberately sacrificed to achieve the irreversibility property. The distinct-per-row count (161.3) confirms the forced non-injectivity.

Table 3. Cryptographic properties of λ compared to the AES S-box and random 8-bit function

Metric	λ table	AES S-box	Random 8-bit
Nonlinearity (min/avg)	90/99.0	112	92-100
Differential uniformity	14	4	6-8
Algebraic degree	≥ 7	7	7
SAC avg deviation	0.035	0.015	0.03
Distinct per row	161.3/256	256 (bijective)	~161

Algorithm 2. Generating an output matrix using Chebyshev polynomials

Procedure: GenerateOutputMatrix
Input: $x, q, n = 7$ // input signal x, current state q, polynomial degree n=7
Output: T
For q = 0 to N−1 do
For x = 0 to N−1 do
$v = (q \ll 8) \oplus x$ // combine q and x
$z = 2 \times (v/65535) - 1$ // into the interval [−1, 1]
$T = cos(n \times arccos(z))$ // Chebyshev polynomial
$Lambda[q][x] \leftarrow floor(256 \times (T+1)/2)\ XOR\ PRNG()$
End For
End for
Return $Lambda$

The Chebyshev polynomial serves as a generation method for the λ table, not as a runtime operation. Once generated, the λ table is a fixed 256 × 256 byte array that is used exclusively through integer lookups; there is no floating-point computation that occurs during hashing. The generation procedure as described in Algorithm 2 is executed once offline to produce a deterministic, platform-independent table procedure (Algorithm 2) is executed once offline to produce a deterministic, platform-independent table. On the other hand, Table 4 shows how to verify λ, δ tables against the checksums before use.

Table 4. Checksums and hash test vectors for bit-exact verification. Seed = 42, degree = 7

Verification item	Value
SHA-256(δ)	d88b51c8abc6991206fa9c9d96f97a95 9a6958ba990bc8c6591a7206140a803a
SHA-256(λ)	b2ce53a962e501b3a743dae3f342afed dbd7be22748551f2c298a1ff981f8f18
λ [0][0]	0x83
λ [127][128]	0x99
λ [255][255]	0xCA
δ [0][0]	0x28
δ [255][255]	0xB2
H("")	44d750e27627689c3dda00749b7bf77a 271822309042c7220fae79d8f0af2977
H("abc")	76f92a7532dadb3bbda51ffb6bb92d46 dda7ed6f5c498c8f155f870371e5e6fc
H("hello")	7c5dd72942a9ebfd905295c6c1c4d8c7 d9bccd005d57b7cfd9d4e901de07b187

3.2 Sponge-based construction

The second main part of this work is a sponge-based construction. The proposed hash function exploits a sponge-based cryptographic construction to process arbitrary-length input messages and produce a fixed-length 256-bit digest. Where the irreversible Mealy automaton acts as the nonlinear transformation core within the sponge architecture. The sponge-based hashing framework includes iterative message absorption, nonlinear state mixing, secure digest extraction, and diffusion propagation. The sponge construction uses an internal state of block size $b = 512$ bits, and the sponge state b is divided into two parts: $b = r + c$, where r represents the rate portion (256 bits) and c (256 bits) is the capacity portion. The rate part is responsible for message absorption and digest extraction, whereas the capacity part improves the designed collision resistance of $c/2 = 128$ bits and the preimage resistance of $min(d, c) = 256$ bits under the random transformation model [9]. Figure 2 shows a complete pipeline, where steps are described as follows:

Step 1: Padding: The message M is padded into a multiple of 32 bytes by adding $0x80$, then $0x00$ bytes, and then the original message length as a 64-bit integer is placed at the end. $M' = M \,||\, 0x80 \,||\, 0x00 \ldots 0x00 \,||\, len(M)$, where $|M'|\ mod\ 32\ = 0$.

Step 2: Block division: The padded message M' is then divided into fixed-size 256-bit blocks as follows: $M' = B_1 \,||\, B_2 \,||\, \cdots \,||\, B_{k-1}$ where $|B_i| = 32$ bytes.

Step 3: Initialization: The 64-byte sponge state block $S[0, \ldots 63]$ is loaded with a fixed initial vector:
$S[0, \ldots 63]\ ] \leftarrow IV$, *where IV is 64 bytes*

Steps 4–7: Absorb phase: The 64-byte sponge state block is absorbed into the rate portion of the sponge state using XOR operations: $S[i] \leftarrow S[i] \oplus B[i]$, for $0 \le i < 32$
After each absorption step, the internal sponge state is processed by using multiple rounds ($R = 12$) of the irreversible automaton.

Step 8: Finalization: The transformation runs 12 additional times ($Rf = 12$) with a different set of round constants. Where each round uses its own round constant (the transformation runs 12 additional times using round constants $RC[12..23]$, which are distinct from the absorb-phase constants $RC[0..11]$): $For\ round = 0\ to\ Rf - 1$: $MealyTransform(S, RC[Rf + round]$

Step 9: Squeeze: The first 32 bytes of the state are extracted as the final 256-bit digest. The last 32 bytes (capacity) are discarded and never exposed, which prevents length-extension attacks. $D \leftarrow S[0..31]$

3.3 Mealy automaton

The Mealy machine transforms a 64-byte sponge state; it runs for 12 rounds ($R = 12$), plus 12 finalization rounds ($Rf = 12$). Each round processes all 64 bytes through the pipeline as shown in Figure 2.

Whereas Figure 3 shows the data flow within one transformation round, and as shown in Algorithm 3, for each byte from 0 to 63, six steps are performed as follows:

Step 1: Automaton state: The current state byte is read as the automaton's internal state. $q_i \leftarrow S[i]$

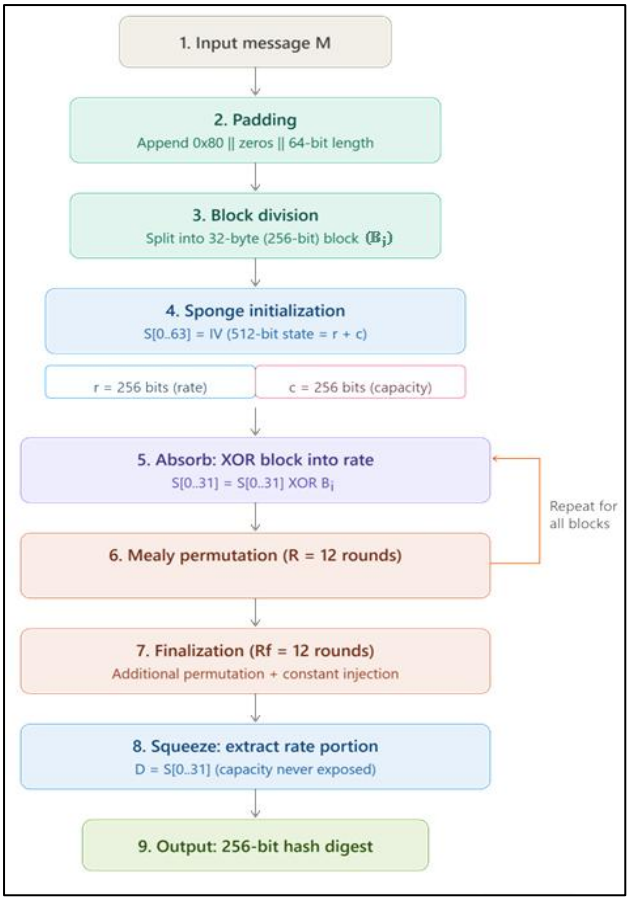

Figure 2. Sponge pipeline overview

Step 2: Input signal: The input signal to the automaton is formed by XORing with the next byte. Thus, it depends on the adjacent position.

$$x_i \leftarrow S[i]\ XOR\ S[(i+1)\ mod\ 64] \quad (5)$$

Step 3: Irreversible transition: The next automaton state is looked up from the transition table. Because δ is non-injective, different(q_i, x_i)pairs can produce the same q_{i+1}, making backward reconstruction impossible.

$$q_{i+1} = \delta[q_i][x_i] \quad (6)$$

Step 4: Chebyshev output: The output byte is looked up from the Chebyshev-based table. This table was precomputed by using the polynomial T_7 and provides non-linear substitution.

$$y_i = \lambda[q_i][x_i] \quad (7)$$

Step 5: Mixing: The state byte is replaced by XORing three values: the Chebyshev output, the next automaton state, and a left-rotated copy of a byte from 7 positions away. The offset 7 was chosen because $gcd(7,64) = 1$, which ensures all byte positions are covered.

$$S[i] = y_i \oplus q_i + 1 \oplus ROTL(S[(i+7) mod\ 64], 3) \quad (8)$$

Step 6: Diffusion: A right-rotated copy of a byte from 3 positions away is XORed in. This adds a second mixing path at a different distance, with $gcd(3,64) = 1$ ensuring a different coverage pattern from step 5.

$$S[i] = S[i] \oplus ROTR(S[(i+3) mod\ 64], 5) \quad (9)$$

Finally, after 64 bytes have been processed, a unique round constant vector is XORed into the state to prevent identical rounds.

$$S[i] = S[i] \oplus RC[round][i], for\ i = 0, 1, \ldots, 63 \quad (10)$$

Algorithm 3. Mealy automaton transform rounds

Procedure: MealyTransform
Input : IV[0..63], λ, δ, round constant RC
$For\ i = 0\ to\ 63\ do$
$S[i]\ ] \leftarrow IV[i]$
$Q_i \leftarrow S[i]$ // automaton state
$X_i \leftarrow S[i]\ XOR\ S[(i+1)\ mod\ 64]$ // input signal
$Q_{i+1} \leftarrow \delta[Q_i][X_i]$ // irreversible transition
$y_i \leftarrow \lambda[Q_i][X_i]$ // Chebyshev output
$S[i] \leftarrow y_i\ XOR\ Q_{i+1}\ XOR\ RTOL(S[i+7] mod\ 64,3)$
$S[i] \leftarrow S[i]\ XOR\ RTOR(S[i+3] mod\ 64,5)$
End for
For i=0 to 63 : $S[i] \leftarrow S[i]\ XOR\ RC[round][i]$ // round constant

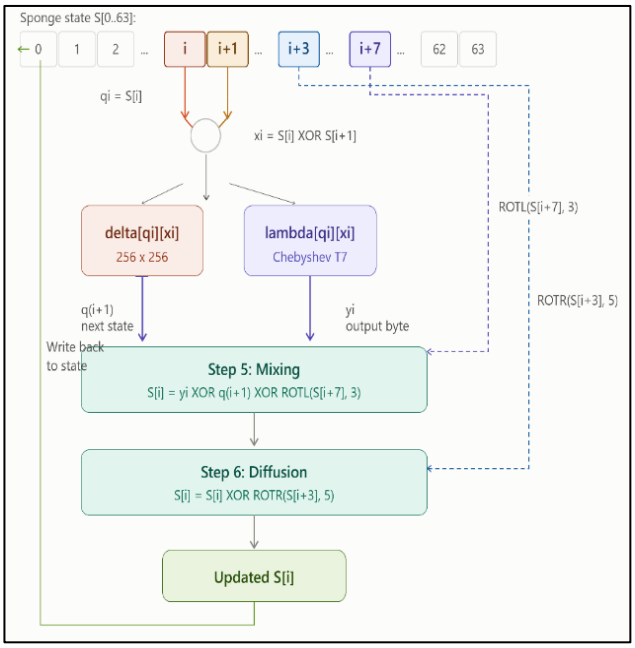

Figure 3. Mealy automaton acts as a transformation round

Finally, the complete hash algorithm as shown in Algorithm 4 includes absorb, which uses RC[0..11], and finalization, which uses RC[12..23]. The capacity S[32..63] is never exposed.

4. Comparative evaluation

We compare the proposed hash function against SHA-256. Where the algorithms are implemented in C and compiled with GCC-O2-lm under identical conditions on the same machine. The experiment has been held in MSI Notebook CYBORG, having Intel(R) Core(TM) i7-13620H (2.40 GHz) with 16 GB RAM under the 64-bit operating system Windows 11.

Determinism was verified: hashing the same message twice always produces identical digests. The reference implementation (immsh_final.c), where both lookup tables in machine-readable form (delta.bin, lambda.bin) and a known-

answer test file (kat.txt) are provided as supplementary material.

Algorithm 4. A complete hash function generator (IFA-hash function)

Input: Message M, public parameters (δ, λ, IV, RC[0..23])
Output: 256-bit digest D
Pad M: append 0x80, zeros, 64-bit big-endian length
Split into 32-byte blocks B_0, B_1, …, B_{k-1}
For i = 0 to k−1 do // absorb phase
For j = 0 to 31: S[j] ← S[j] XOR B_i[j]
For round = 0 to R−1: // R = 12
MealyTransform(S, RC[round])
End For
For round = 0 to Rf−1: //Rf=12
MealyPermute(S, RC[Rf + round])
D←S[0..31] // squeeze
Return D

4.1 Computational efficiency

As shown in Table 4, IFA-Hash achieves approximately 13 MB/s at steady state, about 14× slower than SHA-256's 183 MB/s. The per-byte cost of IFA-Hash consists of two 256×256 table lookups (δ and λ, each a single memory read), three XOR operations, two-bit rotations ($ROTL$ and $ROTR$), and one modular index computation, a total of approximately 8 operations per byte. This is repeated for R rounds ($R = 12$ and $Rf = 12$), each processing 64 bytes, giving 24 × 64 × 8 = 12,288 operations per 32-byte message block. Compared with SHA-256, it performs 64 rounds of 10 operations each on a 64-byte block (640 operations), explaining the throughput ratio. The lower throughput of 9.2 MB/s at 1 KB is due to the fixed overhead of initialization and finalization (12 additional rounds), whereas at message sizes above 10 KB, throughput stabilizes at 13.8 MB/s, confirming $O(n)$ time complexity with $O(1)$ space. Formally, for a massage of m bytes, padding generates $[m + 9]/32]$ blocks, each requiring $R = 12$ transformation rounds of 64 bytes operations, in addition to Rf=12 finalization rounds. Thus, we have total work is $12 \times 64 \times [m + 9]/32] + 12 \times 64$, which is $O(m)$ in the message length. The space requirement is the two lookup tables ($126\ KB$) plus a 64-byte sponge state, independent of message length.

Table 4. Throughput for both IFA-hash and SHA-256

Message Size	IFA-Hash (MB/s)	SHA-256 (MB/s)
1 KB	**9.2**	123.7
10 KB	**12.5**	190.7
100 KB	**13.8**	188.4
1 MB	**13.8**	182.6

4.2 Avalanche effect

The avalanche test measures the percentage of output bits that change when a single random bit of the input is flipped. For a 256-bit hash behaving as a random oracle, the expected mean is exactly 50.00% with a standard deviation of $50/\sqrt{256} \approx 3.12$ [1]. IFA-Hash achieves a mean of 50.02% ($\sigma = 3.14$), while SHA-256 achieves 50.01% ($\sigma = 3.13$). Both values are statistically indistinguishable from the random oracle expectation, indicating that the Mealy machine combined with $ROTL/ROTR$ diffusion provides the same quality of avalanche propagation as SHA-256's arithmetic compression function. The minimum values (40.62) and maximum values (59.77% and 64.06%) represent the tails of the distribution over 2,000 trials and fall within the expected range, as shown in Table 5. With more than 10,000 trials, the tail distribution matches the Binomial (256, 0.5) expectation: 0.45% of trials fall below 42% (expected 0.51%), and 0.51% exceed 58% (expected 0.51%). The tails are symmetric, confirming no directional bias. The minimum observed value (37.89%) corresponds to a z-score of −3.87 from the mean, consistent with the expected extreme of 10,000 binomial samples.

Table 5. Avalanche statistics

Metric	IFA-Hash	SHA-256
Mean bit change	**50.02%**	50.01%
Std. deviation	**3.14%**	3.13%
Minimum	**40.62%**	39.45%
Maximum	**59.77%**	64.06%

4.3 Reduced-round differential analysis

In order to search for exploitable differential characteristics, we apply a single-bit input difference at byte 0 to 10,000 random initial states and measure the output difference after 1, 2, and 3 rounds of the published δ and λ tables, as shown in Table 6. Based on the above table, after the first round, diffusion is adjusted to 3.1% of the state, confirming that the Mealy transformation achieves full diffusion gradually across multiple rounds. The maximum single-byte differential probability at one round (0.0062) is consistent with the non-bijective table structure and does not indicate an exploitable high-probability characteristic. Rotation-invariant differences (identical XOR across all 64 bytes) reach 50% Hamming in a single round, confirming that this class of structured differences does not bypass the diffusion mechanism.

Table 6. Reduced-round differential propagation

Rounds	Max diff. probability	Avg Hamming (of 512)
1	0.0062 (1.6/256)	15.9 (3.1%)
2	0.0083 (2.1/256)	35.8 (7.0%)
3	0.0145 (3.7/256)	67.7 (13.2%)
12 (full)	~1/256 (0.0039)	256.0 (50.0%)

4.4 Collision resistance

Based on a sanity check, Table 7 shows that there are no exact collisions found among 50,000 random messages with varying lengths (1–512 bytes). Bit balance measures the fraction of 1 bits in the digest across 5,000 random messages. Where the ideal value for a random oracle is 50.000%. IFA-Hash achieves 49.975%, and SHA-256 achieves 49.981%, both negligibly close to the ideal. The near-collision Hamming distance is the average number of bits that differ between the digests of two messages that differ by exactly one bit. For a 256-bit random oracle, the expected Hamming distance is 128.0 (exactly half the digest length). IFA-Hash achieves 128.0, and SHA-256 achieves 128.2, both matching the theoretical expectation. The birthday bound for finding a collision is $2^{128} \approx 3.4 \times 10^{38}$ for both algorithms. The sponge

construction was originally analyzed with a random permutation [9], but Bertoni et al. also established that the framework remains secure when the internal function is a transformation rather than a permutation [28]. In the introduced design, the Mealy round function is deliberately non-bijective, i.e., the non-injective transition function δ destroys state information at each step, which is the source of one-wayness. Based on the sponge framework, the security argument does not require the internal function to be bijective. It requires only that the capacity portion ($c = 256$ bits) remain hidden from the attacker. Although the byte-level non-injectivity of δ and λ introduces internal state collisions by design, these collisions are confined within the sponge's internal state and cannot be observed or exploited by an external attacker who sees only the rate portion (the 256-bit digest). The collision resistance therefore is bounded by $c/2 = 128$ bits, and the preimage resistance by $min(d, c) = 256$ bits, consistent with the sponge security framework. The non-bijectivity and the capacity serve complementary roles: the capacity hides the full state from the output, while the irreversible transitions destroy the state's backward history.

Table 7. Collision-resistance metrics

Metric	IFA-Hash	SHA-256
Bit balance (ones %)	49.975%	49.981%
Near-collision Hamming	128.0 / 128	128.2 / 128
Birthday bound	2^{128}	2^{128}
Exact collisions	0	0

4.5 NIST SP 800-22 statistical tests

The NIST SP 800-22 test suite [29] is the standard randomness evaluation framework for cryptographic algorithms. In this test, we generated a bitstream of 1,000,000 bits. We hashed sequential counter values (0, 1, 2, …, 3906). Then concatenated the 256-bit digests. Table 8 illustrates the following results:

- All 15 tests pass at a significance level of $\alpha = 0.01$, with P-values ranging from 0.1867 (Maurer's universal statistical test) to 1.0000 (four tests).
- The frequency test ($p = 0.4307$) confirms near-ideal bit balance.
- The runs test ($p = 0.7304$) confirms the absence of sequential correlations between bits.
- The serial test ($P = 0.6811$) and approximate entropy test ($P = 1.0000$) confirm no detectable patterns in consecutive bit sequences.
- The matrix rank test ($P = 0.3824$) confirms linear independence of output bit subsets.

4.6 Structural irreversibility

The main security strength for IFA-Hash rests on the sponge construction's proven bounds based on the random-transformation model [9]. Where Bertoni et al. proved that a sponge using a random transformation with capacity c is indifferentiable from a random oracle, yielding collision resistance of $c/2 = 128$ bits and preimage resistance of $min(d, c) = 256$ bits. These bounds depend on the internal transformation behaving indistinguishably from a random function to any computationally bounded adversary. The Mealy round function is a public, fixed transformation:

$$f: 0,1^{512} \rightarrow 0,1^{512} \quad (11)$$

Constructed from two public 256×256 lookup tables (δ and λ) and deterministic bit-rotation operations. Unlike the idealized random transformation in [28], f is fully known to the attacker. The gap between the idealized model and the concrete construction is bridged empirically rather than formally: the 15/15 NIST SP 800-22 tests as shown in Table 8, the avalanche statistics matching the random oracle expectation (Table 4), and the zero-collision result across 50,000 test messages (Section 4.3) provide evidence that f behaves as a high-quality pseudorandom transformation.

Additionally, we provide a structural analysis of the lookup tables to characterize their non-invertibility. The transition table δ maps 256 inputs to an average of 161.4 distinct outputs per row, and the output table λ maps to 161.3 distinct outputs per row, corresponding to an information loss of $-log_2(161/256) \approx 0.67$ bits per table lookup, as shown in Table 9. This per-step non-injectivity is the mechanism by which the round function resists direct algebraic inversion of individual steps.

Table 8. NIST SP 800-22 results (α=0.01)

#	Test	p-value	Result
1	Frequency	0.4307	PASS
2	Block frequency	0.5554	PASS
3	Runs	0.7304	PASS
4	Longest run	0.5516	PASS
5	Matrix rank	0.3824	PASS
6	DFT (spectral)	0.3636	PASS
7	Non-overlapping template	0.9204	PASS
8	Maurer's universal	0.1867	PASS
9	Serial	0.6811	PASS
10	Approximate entropy	1.0000	PASS
11	Cumulative sums	1.0000	PASS
12	Random excursions	0.8309	PASS
13	Random excursions variant	0.8648	PASS
14	Overlapping template	0.2762	PASS
15	Linear complexity	1	PASS

Table 9. Structural irreversibility analysis

Metric	δ (transition)	λ (output)
Avg distinct per row	161.4 / 256	161.3 / 256
Info loss per step	0.67 bits	0.67 bits
Combined per step	1.33 bits	
Total (24 rounds × 64 bytes)	2043.9 bits	
Recoverable fraction	2^{-2044}	

From Table 9, the per-step information loss quantifies the non-injectivity built into the δ and λ tables, where each row maps 256 inputs to approximately 161 distinct outputs, discarding 0.67 bits per lookup. This metric confirms that the tables are well-constructed for irreversibility.

4.7 Discussion

Based on the avalanche test, changing one input bit affected all 256 output bits with near-uniform frequency: the avalanche rate was 50.02% ($\sigma = 3.14$), and the symmetric tail distribution showed no position-specific bias that a differential attacker could use. All 15 NIST SP 800-22 tests also passed, with no detectable patterns in the output stream. Section 4.3 shows that the maximum single-byte differential probability at one round (0.0062) does not exceed the theoretical expectation for a non-bijective table, and that full diffusion is achieved well before the operational 12 rounds. Moreover, the structural analysis (Table 9) confirms that the δ and λ tables are well-constructed for irreversibility, with ~161 distinct outputs per row. However, these results collectively establish that the construction behaves as a high-quality pseudorandom transformation.

5. Security analysis

5.1 Preimage resistance

As a sanity check, we select 20 random target messages, compute their digests, and then search for any message whose digest matches the first k bits of the target. For a random oracle with an $n - bit$ output, the expected number of attempts to find a k-bit partial preimage is 2^k, and the number of attempts follows a geometric distribution with a parameter p =2^{-k}.

For k = 8 bits (expected 256 attempts): IFA-Hash finds matches for all 20 targets with a mean of 332 attempts (ratio 1.30). For $k = 16$ bits (expected 65,536 attempts), all 20 targets are matched with a mean of 90,202 attempts (ratio 1.38). Both ratios are within the expected variance of the geometric distribution. A ratio significantly below 1.0 would indicate the hash function finds preimages too easily (weak), while a ratio significantly above 2.0 would suggest the distribution is not geometric (biased). These tests verify that the output bits are not stuck or biased but do not establish the full preimage bound. The preimage resistance claim (2^{256}) rests on the sponge capacity bound $min(d, c) = min(256,256) = 256\ bits$ [9].

5.2 Second-preimage resistance

Second-preimage resistance is tested by fixing a message $m1$, computing its digest, and then searching for a distinct message $m2 \neq m1$ whose digest matches. Over 10 target messages, matching the first 8 bits requires a mean of 204 attempts versus the expected 256 (ratio 0.80). The below-unity ratio is consistent with the geometric distribution, which has high variance (standard deviation equal to the mean for a geometric random variable). The theoretical second-preimage resistance of the sponge construction is $min(2^n, 2^c)$= 2^{256} [30]. Importantly, the sponge construction is not vulnerable to the Kelsey–Schneier long-message second-preimage attack that reduces the security of Merkle–Damgård hashes below 2^n for very long messages, because the sponge's capacity is never exposed in the output.

5.3 Differential analysis

Differential analysis measures how different types of input modifications propagate to the output. A secure hash function should produce approximately 50% bit change for any type of input difference. We tested four difference types: flipping a single bit (the classic avalanche criterion), flipping an entire byte (8-bit difference), appending a byte (length change), and prepending a byte (position shift). All four types produce Hamming distances within 0.07% of the ideal 128.0 (50%), confirming that no input difference type produces exploitable output patterns. The consistency across different types indicates that the Mealy machine's sequential processing does not favor any particular position or difference magnitude; the differential propagation across 2000 trials is shown in Table 10.

Table 10. Differential propagation

Difference type	Hamming (of 128)	Bit change
Single-bit flip	128.2	50.07%
Single-byte flip	128.0	50.02%
Append one byte	128.0	49.98%
Prepend one byte	128.0	49.99%

5.4 Length-extension resistance

In a length-extension attack, an attacker who knows $H(M)$attempts to compute $H(M\|M')$ for some suffix M' without knowing M. This attack is possible against Merkle–Damgård-based hashes (including SHA-256) because the digest reveals the full internal state, which serves as the start state for processing the suffix. The sponge construction prevents this attack because the digest reveals only the rate portion (the first 32 bytes) of the 64-byte internal state. The capacity portion (bytes 32–63) remains hidden. To mount a length-extension attack, the attacker would need to determine the capacity from the rate, which requires inverting the Mealy transformation, a computation equivalent to a 2^{256} exhaustive search. Empirically, the mean Hamming distance between $H(m)$ and $H(m\|suffix)$ is 127.7 out of 128 (49.87% bit change), indistinguishable from independent random values. This confirms that knowing H(m) provides no computational advantage for computing $H(m\|suffix)$.

5.5 Bit independence

The bit independence test (closely related to the Strict Avalanche Criterion (SAC)) measures whether each individual output bit changes with a probability of approximately 50% when a random input bit is flipped. For each of the 256 output bits, we record the flip count across 1,000 trials and compute the flip percentage. A bit is considered "weak" if its flip percentage falls outside the 40–60% acceptance band. The tests show a per-bit flip range of 45.5% to 54.7%, with all 256 output bits falling within the acceptance band of zero weak bits. This confirms that every output bit behaves as an independent Bernoulli (0.5) random variable conditioned on a single-bit input change, and no output bit is stuck, biased, or correlated with another output bit.

6. Limitations and applications

6.1 Limitations

- At approximately 13 MB/s, IFA-Hash is roughly 13× slower than SHA-256.
- The security arguments are structural and empirical rather than formal reductions to standard hardness assumptions. A stronger concrete security analysis, such as reduced-round distinguishers, SAT/MILP modeling of the Mealy transformation, or systematic differential trail search. would further strengthen the confidence in the construction and is identified as future work.

- The 128 KB parameter set is larger than SHA-256's internal constants (~100 bytes), though it fits comfortably in the L1 cache.

6.2 Applications

- IFA-Hash is suited to applications where security properties are advantageous over high performance.
- The sequential dependency of the Mealy transformation, where each byte's transition depends on the previous byte's result, limits intra-block parallelism, though a measured attacker-side GPU implementation is needed to quantify this cost. Under Grover's algorithm, preimage resistance reduces from 2^{256} to 2^{128}. Under the Brassard–Høyer–Tapp algorithm, collision resistance reduces from 2^{128} classical to $\approx 2^{c/3}$ work with comparable qubit cost. More detailed post-quantum analysis with specific NIST security level mapping is left as future work.

7. Conclusion

We have presented a 256-bit cryptographic hash function that draws on three foundations: automata theory (the irreversible Mealy machine provides structural one-wayness through non-injective state transitions), chaos theory (the Chebyshev polynomial T_7 provides non-linear output with provable uniformity and sensitivity), and the sponge construction (providing formal security bounds and length-extension immunity). The transformation function uses two 256×256 lookup tables (δ and λ) combined with $ROTL/ROTR$ bit-rotation mixing at carefully chosen offsets. Comparative evaluation against SHA-256, under identical settings, confirms that IFA-Hash matches SHA-256 on every statistical security measure: avalanche effect (50.02% vs 50.01%), bit balance (49.975% vs 49.981%), near-collision Hamming distance (128.0/128 vs 128.2/128), bit independence (0 weak bits out of 256), and NIST SP 800-22 compliance (15/15 tests passed with all p-values above 0.18). The collision resistance ($c/2 = 128$ bits) and preimage resistance ($min(d, c) = 256$ bits) rest on the sponge capacity bound under the random-transformation model. The empirical tests (NIST, avalanche, collision counting) are necessary sanity checks that confirm the construction is not trivially broken, but they are not equivalent to a formal security reduction and should not be interpreted as one. Structural analysis of the lookup tables (0.67 bits lost per step, 161 distinct outputs per row) provides design-level assurance of irreversibility but is likewise a heuristic complement to the formal bounds. The throughput ($13\ MB/s\ vs\ 183\ MB/s$) reflects the sequential dependency of the Mealy transformation. The reference implementation, lookup tables, and known-answer test vectors are included as supplementary material to enable independent verification and cryptanalysis.

Ethical issue

The authors are aware of and comply with best practices in publication ethics, specifically regarding authorship (avoidance of guest authorship), dual submission, manipulation of figures, competing interests, and compliance with research ethics policies. The authors adhere to publication requirements that the submitted work is original and has not been published elsewhere. All survey participants provided informed consent before participating, and the anonymity and confidentiality of all respondents were strictly maintained throughout the research process.

Data availability statement

The manuscript contains all the data. However, additional data will be provided by the corresponding author upon reasonable request.

Conflict of interest

The authors declare no potential conflict of interest.

References

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. CRC Press, 1996. https://theswissbay.ch/pdf/Gentoomen%20Library/Cryptography/Handbook%20of%20Applied%20Cryptography%20-%20Alfred%20J.%20Menezes.pdf

[2] Federal Information Processing Standards Publication FIPS PUB 180, "Secure Hash Standard". https://www.wolfssl.com/license/fips/?gad_campaignid=916470942&gad_source=1&gbraid=0AAAAAD_TYV27qymIFqyJcU41eA0W6yfsR

[3] "SHA-3 standard :," 2015. doi: 10.6028/NIST.FIPS.202.

[4] R. L. Rivest, "The MD5 Message-Digest Algorithm," 1992. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1321

[5] R. H. Brown, M. L. Good, and A. Prabhakar, "Federal Information Processing Standards Publication: secure hash standard," 1993.

[6] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD," Cryptology ePrint Archive, Report 2004/199, 2004.

[7] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in Advances in Cryptology - CRYPTO 2017, Springer, 2017, pp. 570–596.

[8] J. Kelsey and B. Schneier, "Second preimages on n-bit hash functions for much less than 2^n work," in Advances in Cryptology - EUROCRYPT 2005, Springer, 2005, pp. 474–490.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in ECRYPT Hash Workshop, 2007. https://csrc.nist.rip/groups/ST/hash/documents/JoanDaemen.pdf

[10] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5," in Applied Cryptography and Network Security (ACNS 2013), Springer, 2013, pp. 119–135.

[11] "BLAKE3 one function, fast everywhere." [Online]. Available: https://blake3.io

[12] M. Alawida, J. Sen Teh, D. P. Oyinloye, M. Ahmad, and R. S. Alkhawaldeh, "A new hash function based on chaotic maps and deterministic finite state automata," IEEE Access, vol. 8, pp. 176774–176788, 2020.

[13] S. K. Nanda, S. Mohanty, and P. K. Pattnaik, "An optimized 128-bit cellular automata-based hash function for authentication," International Journal of Electrical and Computer Engineering, vol. 13, no. 2, pp. 1858–1866, 2023.

[14] S.-T. Wu and J.-R. Chang, "Secure one-way hash function using cellular automata for IoT," Sustainability, vol. 15, no. 4, 2023.

[15] A. Akhavan, A. Samsudin, and A. Akhshani, "A novel parallel hash function based on 3D chaotic map," EURASIP J. Adv. Signal Process., vol. 2013, 2013.
[16] Y. Li, X. Li, and X. Liu, "Chaotic complex hashing: a simple chaotic keyed hash function based on complex quadratic map," Chaos Solitons Fractals, vol. 173, 2023.
[17] J. Tang, Z. Zhang, and T. Huang, "Hyperchaotic hashing: a chaotic hash function based on 2D linear cross-coupled map with parallel feedback structure," Sci. Rep., vol. 15, 2025.
[18] G. Salloom and L. Hassnawi, "A Novel Hash Function Based on a Chaotic Substitution Box," Engineering, Technology and Applied Science Research, vol. 15, no. 5, pp. 27382–27386, Oct. 2025, doi: 10.48084/etasr.12601.
[19] M. Alawida, J. Sen Teh, W. H. Alshoura, M. Ahmad, and R. S. Alkhawaldeh, "A novel hash function based on a chaotic sponge and DNA sequence," IEEE Access, vol. 9, pp. 158995–159013, 2021.
[20] Y. Ma, S. Hassan, and F. Tahir, "Enhanced image hash using cellular automata with sponge construction and elliptic curve cryptography," Sci. Rep., vol. 15, 2025.
[21] M. Holzer and M. Kutrib, "Reversible Nondeterministic Finite Automata," in Reversible Computation, I. Phillips and H. Rahaman, Eds., Cham: Springer International Publishing, 2017, pp. 35–51.
[22] L. Kocarev and Z. Tasev, "Public-key encryption based on Chebyshev maps," in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 2003, pp. 28–31.
[23] P. Bergamo, P. D'Arco, A. De Santis, and L. Kocarev, "Security of public-key cryptosystems based on Chebyshev polynomials," IEEE Transactions on Circuits and Systems I, vol. 52, no. 7, pp. 1382–1393, 2005.
[24] D. F. Ayoub, A. H. Elghandour, and S. Hashima, "A novel chaos-based generating function of the Chebyshev polynomials and its applications in image encryption," Journal of Information Security and Applications, vol. 62, 2021.
[25] P. A. Jyotirmie, T. Surendra, A. C. Sekhar, and S. U. Devi, "Application of Mealy Machine and Recurrence Relations in Cryptography," vol. 4, no. 5, pp. 246–250, 2013.
[26] G. H. Mealy, "A Method for Synthesizing Sequential Circuits," 1955. doi: 10.1002/j.1538-7305.1955.tb03788.x.
[27] G. H. Mealy, "A method for synthesizing sequential circuits," Bell System Technical Journal, vol. 34, no. 5, pp. 1045–1079, 1955.
[28] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the Indifferentiability of the Sponge Construction." [Online]. Available: http://sponge.noekeon.org/
[29] A. Rukhin, J. Soto, and J. Nechvatal, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," 2010. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf
[30] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance." [Online]. Available: www.cs.ucdavis.edu/~rogawaywww.ece.ucdavis.edu/~teshrim