A Novel Hash Function Based on a Chaotic Substitution Box

Ghassan Salloom
Scientific Research Commission, Baghdad, Iraq
ghassankhaleel@gmail.com (corresponding author)

Layth Hassnawi
Scientific Research Commission, Baghdad, Iraq
laythmail@yahoo.com

Received: 8 June 2025 | Revised: 28 July 2025 and 3 August 2025 | Accepted: 11 August 2025

 DOI: https://doi.org/10.48084/etasr.12601

ABSTRACT

A hash function is a mathematical model that maps inputs of arbitrary size to unique outputs of a fixed length in bits. Hash functions are highly useful and appear in almost all information security applications. In addition to information security applications, it can also serve as index data in hash tables, aiding in the detection of duplicate data for fingerprinting or uniquely identifying files, as well as for checksums to identify data corruption. This research introduces an innovative 256-bit hash function that utilizes a chaotic substitution box using a non-linear logistic map. Unlike MD5 or SHA-family hash functions, which rely on modular arithmetic, logical operations, and bitwise shifts for diffusion and non-linearity, the proposed method incorporates a chaotic substitution box to introduce an additional nonlinear transformation layer and high diffusion. The avalanche rate, statistical analysis, pre-image resistance, second pre-image, collision resistance, and performance are examined to evaluate the cryptographic strength and the performance of the proposed method.

Keywords-hash function; MD5; collision; chaotic; substitution box

I. INTRODUCTION

A cryptographic hash function is a mathematical model that transforms input data blocks into a fixed-length digest through iterative processing steps. Depending on the specific algorithm, block sizes typically range from 128 to 512 bits. The computation is often structured in many rounds, resembling block cipher operations, where each round processes a combination of the current message block and the previous hash state. An effective hash function must exhibit critical properties, including preimage resistance, second preimage resistance, and collision resistance. The preimage resistance ensures that it is computationally infeasible to reconstruct the original input from a hash. The second preimage resistance protects against the discovery of alternative inputs that produce the same hash as a given input. Collision resistance also ensures that it is difficult to determine two distinct inputs that produce identical hash values [1].

Cryptographic hash functions, including classical algorithms such as MD5 [2], SHA-1 [3], and SHA-256 [4], are widely used and essential for information security, data integrity, and authentication purposes. The Message Digest Algorithm (MD5) was once one of the most widely used cryptographic hash functions. The MD5 algorithmic process involves padding, initializing variables, processing in 512-bit blocks, and producing the final 128-bit hash. Each block undergoes 64 transformation steps, including bitwise operations, modular addition, and non-linear functions to achieve diffusion. However, it was later found to be vulnerable to preimage and collision attacks [5]. Moreover, MD5 was one of the earliest and fastest hash functions in terms of CPU performance, which contributed to its widespread early adoption [6]. Although MD5 was once valued for its computational efficiency, its known vulnerabilities limit its suitability for modern cryptographic applications, especially those requiring high security. However, some lightweight or legacy systems that prioritize speed over security may find low resource demands attractive.

The Secure Hash Algorithm 1 (SHA-1), developed in 1995, produces a 160-bit message digest and shares design principles with MD4 and MD5. SHA-1 derives its name from the Secure Hash Algorithm family and reflects a series of iterative operations to compute the hash. Although originally considered secure, advances in cryptanalysis have demonstrated that SHA-1 is vulnerable to collision attacks with a complexity significantly lower than the birthday bound [7]. In addition, machine learning-based cryptanalysis can employ pattern recognition algorithms to detect that a cipher fails to fully randomize input differences [8]. SHA-256 [4], a member of the SHA-2 family, produces a 256-bit hash value and includes several internal hash functions with varying output sizes, each constructed using different prime number permutations. It

performs a series of 64-bit operations and is generally faster than older algorithms such as MD5 and SHA-1. Currently, SHA-256 and the other SHA-2 variants are considered secure and are currently recommended for use in most cryptographic applications.

In parallel with standardized hash functions, recent research has explored alternative approaches, including hash designs based on chaotic maps, neural networks, DNA computing, and lattice-based constructions. Among these, chaos-based hash functions have attracted particular interest due to their sensitivity to initial conditions, inherent non-linearity, and adaptability to lightweight and secure applications. This work introduces a novel 256-bit cryptographic hash function based on a two-dimensional (256×256) substitution box ($Sbox$). Traditional hash functions, such as MD4 and MD5, depend on numerous rounds of bitwise operations, modular addition, and non-linear functions, and are susceptible to preimage and collision attacks. The modified SHA-1 method integrates the strong confusion and diffusion properties of chaos theory to enhance resistance against cryptanalytic attacks, such as preimage and collision attacks. A key innovation of this work is the construction of a two-dimensional (256×256) substitution box (*Sbox*), generated through a hybrid method that combines the multiplicative inverse over $GF(2^{16})$ with a transformation based on the logistic map. This design ensures high nonlinearity, unpredictability, and resistance to preimage and collision attacks. To evaluate the robustness of the proposed hash function, a comprehensive set of analyses was performed, including bit distribution tests, cryptographic strength assessments (such as avalanche effect, collision resistance, preimage attack, and second preimage attack), and performance benchmarking.

II. PROPOSED HASH FUNCTION

The proposed approach differs from conventional hash functions in its method of generating hash values (digests). The core concept is based on the construction of a chaotic substitution box. Specifically, this study develops a non-linear chaotic 256×256 *Sbox* that maps each of the 65,536 possible 16-bit input combinations to a distinct 8-bit output. This design is conceptually inspired by the 8×8 AES *Sbox*, which is widely recognized for its strong confusion properties and low collision probability [9, 10]. To achieve these cryptographic objectives, the principles underlying the AES *Sbox* construction are extended with several modifications tailored for hash function design, as detailed below:

- Inversion of a finite field: For each input $P \in [65536]$, compute the multiplicative inverses in $GF(2^{16})$, Let P^{-1} be this value. For more details on finding the multiplicative inverse based on $GF(2^{16})$ see [11-13].
- Map to chaotic system: Normalize $P \in [65536]$ to a real number, $r \in [0, 1]$, by the relation:

  $r = \frac{P^{-1}}{65536}$
- Apply a chaotic map using the logistic map:

  $r_{n+1} = \mu \,.\, r_n \,.\, (1 - r_n)$

  where μ controls the behavior of the logistic map, $\mu \in (3.57, 4)$, to make the system chaotic, which is useful for cryptography because it introduces unpredictability and high sensitivity to initial conditions [14, 15]. r_n is the initial seed of the iteration, which should be in the interval $[0, 1]$. A small change in r_n leads to large changes in subsequent outputs (high sensitivity), making it suitable for cryptographic *Sbox* generation. This helps in improving confusion, diffusion, and non-linearity [16, 17].
- Convert result to integer: Convert the real number r back to an integer in $[0, 65536]$, such that:

  $output = [r.65536]$
- Store the result in a $256\ x\ 256$ two-dimensional $Sbox$

  $Sbox[Raw][Col] = output$

The process of constructing the proposed hash function can be divided into two main phases: initialization and operation. During the initialization, a two-dimensional 256×256 *Sbox* is generated, as described previously. In addition, the initialized hash state vector *H* is set with 256 bits (4 words of 64 bits, i.e., H_0, H_1, H_2, H_3). The construction procedure of generating a chaotic *Sbox* based on a non-linear chaotic map is described in Algorithm 1.

```
Algorithm 1: Generate a Chaotic Sbox
Procedure Generate_Sbox
Input: μ ∈ (3.57, 4)  // μ > 3.57 and < 4
Output: 256 × 256 Sbox
1.  For i = 0 to 255 do
2.    For j = 0 to 255 do
        //Mix i and j to form 16-bit input
3.      k ← (i << 8) | j
        //inverse in GF(2^16)
4.      In ← GFinverse(k)  //multiplicative
5.      r_n ← In/65536 // Normalize to [0,1]
       //Apply logistic map
6.      r_{n+1} ← μ . r_n . (1 − r_n)
       //convert to integer
7.      output ← (ushort)(r_{n+1} × 65536)
8.      Sbox[i][j] ← output
9.    End For
10. End For
Return 256 × 256 Sbox
```

In the operation phase, the input message is first padded to a multiple of 256 bits. For each 256-bit block, labeled as B_0, B_1, ..., B_{31}, the data is split into 32 bytes and grouped into 16 words, $W_0, W_1, W_2, \ldots, W_{15}$, where each word consists of two consecutive bytes. This grouping is defined as

$W_i = (B_{2i} \ll 8) | B_{2i+1}$

Each word W_i is then transformed using the chaotic *Sbox* as follows:

$S_i = Sbox[\mathrm{W_i} >> 8][\mathrm{W_i}\ \&\ 0\mathrm{xFF}]$

This step maps 16 input values to 16 substituted values. To ensure further diffusion, nonlinear mixing (e.g., logistic map) is used, for instance:

$$M_i = S_i \oplus (S_{(i+1)\%16} + S_{(i+2)\%16})$$

Next, the current hash state is updated through a compression function that mixes selected M_i values with the intermediate hash values H_0, H_1, H_2, H_3 as follows:

$$H_0 = H_0 \oplus M_0 \oplus M_4 \oplus M_8 \oplus M_{12}$$

$$H_1 = H_1 \oplus M_1 \oplus M_3 \oplus M_9 \oplus M_{13}$$

$$H_2 = H_2 \oplus M_2 \oplus M_6 \oplus M_{10} \oplus M_{14}$$

$$H_3 = H_3 \oplus M_3 \oplus M_7 \oplus M_{11} \oplus M_{15}$$

Finally, after processing all message blocks, the final 256-bit hash value is obtained by concatenating the updated state values as follows:

$$H = H_0 \parallel H_1 \parallel H_2 \parallel H_3$$

Figure 1 shows the stages of initialization and operation involved in generating hashing values based on the introduced version.

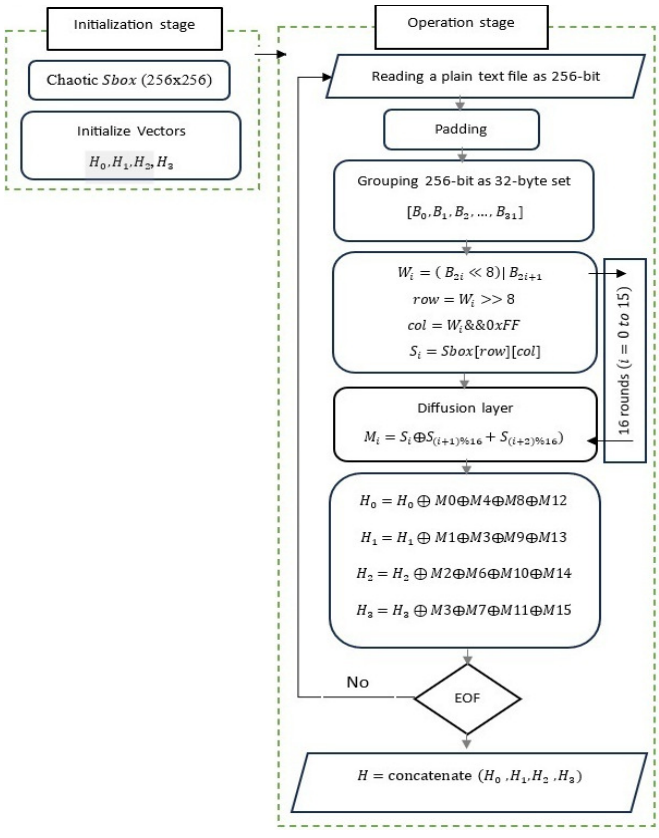

Fig. 1. Diagram of the initialization and operation stages.

III. STATISTICAL ANALYSIS OF THE PROPOSED HASH FUNCTION

Statistical analysis is crucial to evaluating the cryptographic strength. This section evaluates the avalanche effect, randomness, collision resistance, preimage attack, and second preimage attack of the proposed hash function. The avalanche rate is the primary method used to quantify the diffusion of the cryptosystems [18]. The cryptosystem has an avalanche effect when a one-bit change in the plaintext causes a significant change in the corresponding ciphertext block, and also a one-bit change in the ciphertext results in a significant change in the corresponding plaintext block. Moreover, if a block cipher does not exhibit the avalanche effect to a significant degree, then it has poor randomization. To estimate this property of the introduced work, using plaintext 32-byte blocks, the digest of them is calculated through the introduced hash function and then modify a bit of the plaintext (the second byte 73 in the ASCII sequence is changed to 72), calculate the digest again, and finally compute the number of distinct bits in the output digest blocks of each digest by the formula:

$$avalanche\ rate = \frac{number\ of\ flipped\ bits\ in\ ciphertext}{number\ of\ bits\ in\ ciphertext}$$

Table I shows that the proposed hash function has an appropriate avalanche rate and reached 0.55. Therefore, it is protected against differential cryptanalytic attack, for instance, chosen plaintext – chosen ciphertext. Moreover, the proposed design has strong diffusion.

TABLE I. AVALANCHE RATE OF THE PROPOSED METHOD

Plaintext	Hashed	Avalanche rate
54 68 69 73 20 **73** 65 63 74 69 6F 6E 20 64 69 73 63 75 73 73 65 73 20 73 65 76 65 72 61 6C 20 21	25 67 A9 12 34 1A FD B8 75 24 5C 34 A0 82 28 D7 5D CD 6E 3C 6D 37 2F 8A 8A A1 74 5D F1 5A 77 01	
54 68 69 73 20 **72** 65 63 74 69 6F 6E 20 64 69 73 63 75 73 73 65 73 20 73 65 76 65 72 61 6C 20 21	7F F0 BE CE C8 AD 57 B4 D3 ED BA 4D 71 DC 5A B2 F7 6B 3F C3 0D 6C CA 17 7F 1C DF 97 31 A4 11 43	0.55

Moreover, statistical analysis demonstrates that the proposed hash function achieves a stronger avalanche effect, with an average bit-change rate of 55%, compared to SHA-256, which achieves approximately 49.6% under identical plaintext size and bit-flip scenarios.

A. Randomness Test

To evaluate the randomness and uniformity of the proposed hash function, a bit distribution analysis was performed on 10,000 hash outputs, each measuring 256 bits. This yielded a total of 2,560,000 bits for the statistical assessment. The primary objective of this analysis was to determine if the bits are uniformly distributed between 0s and 1s, as an optimally designed cryptographic hash function should produce outputs that have no bias towards either bit value. The findings showed 127766 1s (49.91%) and 128234 0s (50.09%), indicating a near-perfect balance. This distribution closely aligns with the expected 50/50 ratio, confirming the absence of systematic bias

in the output. A comparison with SHA-256 under identical testing conditions with 1,279,964 1s (49.99%) and 1,280,036 0s (50.001%) revealed that both functions exhibit similarly strong bit balance characteristics. Moreover, to test bit uniformity, 320,000 bytes were analyzed, generated from 10,000 hash outputs of both SHA-256 and the introduced hash function. The proposed function showed a marginal improvement in uniformity, with a lower chi-square statistic (259.8 vs. 267.1), reduced standard deviation of byte occurrences, and slightly higher Shannon entropy (7.9985 vs. 7.9972). These results illustrate stronger byte-wise diffusion and randomness characteristics in the proposed hash algorithm. This test was performed based on the ENT 2008 program.

B. Collision Test

Collision resistance is an essential property for any cryptographic hash function [19], making it computationally infeasible to find two different inputs that produce the same hash output. An experimental collision test based on 10,000 distinct input messages was carried out to verify the collision resistance of the proposed hash function. During the testing process, all 10,000 outputs were analyzed for duplicates. The experiment reported zero collisions, demonstrating a high level of resistance to hash collisions within the tested space. This outcome highlights the effectiveness of the foundational chaotic *Sbox* design and non-linear transformations (logistic map) in distributing input entropy evenly over the output space.

C. Preimage Attack

A preimage attack involves the attempt to discover an input x such that $H(x) = y$, where y is a known hash output. The attacker aims to invert the hash function to obtain the original message [20]. For a hash function producing an n-bit output, the complexity of a successful preimage attack should be approximately 2^n operations. In the case of the proposed 256-bit hash function, this translates to 2^{256} operations, making brute-force preimage attacks computationally infeasible.

D. Second-Preimage Attack

In this type of attack, given an input message x_1, the attacker tries to find a different message x_2 such that $H(x_1) = H(x_2)$ [21, 22]. This is a stronger form of attack than the preimage attack and is especially dangerous in authentication, digital signatures, and data integrity systems. A secure hash function with an n-bit output should offer second-preimage resistance of 2^n complexity. Again, the introduced design ensures 2^{256} complexity, ensuring a high level of security.

Moreover, a 256×256 *Sbox* has approximately 65,536 hash values, so the number of possible settings of the *Sbox* is: $256^{65536} = 2^{8\times65536} = 2^{524288}$. This is an unimaginably large number of options, so it is infeasible to find two different inputs that produce the same hash output.

IV. PERFORMANCE TEST

To evaluate the performance of the proposed hash function, it was compared with the classic industrial standard cryptographic hash function SHA-256, which belongs to the SHA-2 series, using three key performance measures: time consumption (time to digest an input), throughput (data volume processed per time unit, MB/s), and resource usage (memory and CPU resources dedicated to hashing). The performance tests were executed on an MSI notebook having an Intel(R) 13th Gen Intel(R) Core(TM) i7-13620H CPU 2.40 GHz with 16 GB of RAM under the 64-bit operating system Windows 11, using a .NET 6.0 environment with Visual Studio 2017 C# implementation.

To estimate the performance, input sizes of 1 MB and 10 MB were selected to represent different sizes of data. Each test was repeated 100 times, and averages were computed. A stopwatch-based benchmarking method was utilized in C# using the system diagnostics stopwatch class for time measurement. For SHA-256, the built-in implementation of System Security Cryptography was used. For the proposed hash, a custom class was utilized, which implemented the 256×256 *Sbox* and chaotic transformation logic. The results in Table II can be used to extract the following facts:

- The proposed hash function involves execution time higher than in SHA-256 due to its nonlinear operations and the overhead of the *Sbox* lookup.
- Throughput is lower than SHA-256, which is acceptable considering the added security advantages (e.g., increased diffusion/confusion).
- In addition, memory usage increases marginally due to the storage of the larger *Sbox* and non-linear intermediate states.

Through these trade-offs, the proposed hash function illustrates acceptable performance for many security applications, particularly in environments where security is prioritized over low performance.

TABLE II. PERFORMANCE RESULTS

Hash function	Input size	Time (ms)	Throughput (MB/s)	Memory usage
SHA-256	1 MB	5	200	7.8
Proposed hash	1 MB	6.2	161	8.1
SHA-256	10 MB	48.8	204	8.7
Proposed hash	10 MB	60	166	9.4

V. CONCLUSION

The proposed 256-bit hash function, which utilizes a static 256×256 chaotic *Sbox* and non-linear logistic map transformations, was rigorously evaluated through a series of statistical and cryptanalytic tests. The results demonstrate strong cryptographic properties, including high avalanche performance, balanced bit distribution, and total resistance to collisions within the tested input space. Moreover, the introduced hash function shows theoretical resistance to preimage and second-preimage attacks due to its 256-bit output length, large *Sbox*, and non-linear logistic internal structure. Finally, the integration of chaotic *Sbox* and non-linear systems introduces non-linear behavior that is difficult to reverse or predict, strengthening one-wayness and overall unpredictability. On the other hand, the performance tests show that it is slower than SHA-256 due to its non-linear operations and *Sbox* overhead. Future work will investigate improving the

execution time, bitwise manipulation, and parallel processing to enhance the proposed hash function's performance and throughput. Using efficient fixed-point arithmetic for nonlinear chaotic maps can reduce floating-point computation overhead.

REFERENCES

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 2018.

[2] R. L. Rivest, "The MD5 Message-Digest Algorithm," Internet Engineering Task Force, Request for Comments RFC 1321, Dec. 1992. https://doi.org/10.17487/RFC1321.

[3] "Secure hash standard," National Institute of Standards and Technology (U.S.), Washington, D.C., USA, NIST FIPS 180-1, 1995. https://doi.org/10.6028/NIST.FIPS.180-1.

[4] "Secure hash standard," National Institute of Standards and Technology (U.S.), Washington, D.C., USA, NIST FIPS 180-4, 2015. https://doi.org/10.6028/NIST.FIPS.180-4.

[5] T. Xie, F. Liu, and D. Feng, "Fast Collision Attack on MD5," Cryptology ePrint Archive 2013/170, 2013.

[6] M. Bellare and D. Micciancio, "A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost," in *Advances in Cryptology — EUROCRYPT '97*, 1997, pp. 163–192, https://doi.org/10.1007/3-540-69053-0_13.

[7] M. Ambrona, G. Barthe, and B. Schmidt, "Generic Transformations of Predicate Encodings: Constructions and Applications," in *Advances in Cryptology – CRYPTO 2017*, vol. 10401, J. Katz and H. Shacham, Eds. Springer International Publishing, 2017, pp. 36–66.

[8] M. M. Alani, "Applications of machine learning in cryptography: a survey," in *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, Kuala Lumpur, Malaysia, Jan. 2019, pp. 23–27, https://doi.org/10.1145/3309074.3309092.

[9] J. Daemen and V. Rijmen, *The Design of Rijndael: The Advanced Encryption Standard (AES)*. Springer, 2020.

[10] G. Jakimoski and L. Kocarev, "Chaos and cryptography: block encryption ciphers based on chaotic maps," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 48, no. 2, pp. 163–169, Oct. 2001, https://doi.org/10.1109/81.904880.

[11] H. S. Yoo, S. U. Yoon, and E. S. Kim, "An efficient algorithm for computing inverses in GF(2m) using dual bases," in *Proceedings of the 2003 International Conference on Computational Science*, Mar. 2003, pp. 994–999.

[12] J. Luo, K. D. Bowers, A. Oprea, and L. Xu, "Efficient software implementations of large finite fields *GF* (2 ^{n}) for secure storage applications," *ACM Transactions on Storage*, vol. 8, no. 1, pp. 1–27, Feb. 2012, https://doi.org/10.1145/2093139.2093141.

[13] W. Dong-Mei, "A fast implementation of modular inversion over GF (2m) based on FPGA," in *2010 2nd IEEE International Conference on Information Management and Engineering*, Chengdu, China, 2010, pp. 465–468, https://doi.org/10.1109/ICIME.2010.5477629.

[14] I. El Gaabouri, M. Senhadji, M. Belkasmi, and B. El Bhiri, "A new S-box pattern generation based on chaotic enhanced logistic map: case of 5-bit S-box," *Cybersecurity*, vol. 7, no. 1, Nov. 2024, Art. no. 59, https://doi.org/10.1186/s42400-024-00254-4.

[15] J. Arif *et al.*, "A Novel Chaotic Permutation-Substitution Image Encryption Scheme Based on Logistic Map and Random Substitution," *IEEE Access*, vol. 10, pp. 12966–12982, 2022, https://doi.org/10.1109/ACCESS.2022.3146792.

[16] X. Qian, Q. Yang, Q. Li, Q. Liu, Y. Wu, and W. Wang, "A Novel Color Image Encryption Algorithm Based on Three-Dimensional Chaotic Maps and Reconstruction Techniques," *IEEE Access*, vol. 9, pp. 61334–61345, 2021, https://doi.org/10.1109/ACCESS.2021.3073514.

[17] W. Alexan, M. Elkandoz, M. Mashaly, E. Azab, and A. Aboshousha, "Color Image Encryption Through Chaos and KAA Map," *IEEE Access*, vol. 11, pp. 11541–11554, 2023, https://doi.org/10.1109/ACCESS.2023.3242311.

[18] D. Upadhyay, N. Gaikwad, M. Zaman, and S. Sampalli, "Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications," *IEEE Access*, vol. 10, pp. 112472–112486, 2022, https://doi.org/10.1109/ACCESS.2022.3215778.

[19] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," in *Fast Software Encryption*, vol. 3017, B. Roy and W. Meier, Eds. Springer Berlin Heidelberg, 2004, pp. 371–388.

[20] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Advances in Cryptology – EUROCRYPT 2005*, vol. 3494, R. Cramer, Ed. Springer Berlin Heidelberg, 2005, pp. 19–35.

[21] J. Kelsey and B. Schneier, "Second Preimages on n-Bit Hash Functions for Much Less than 2nWork," in *Advances in Cryptology – EUROCRYPT 2005*, 2005, pp. 474–490, https://doi.org/10.1007/11426639_28.

[22] N. Bagheri, "Security Analysis of Zipper Hash Against Multicollisions Attacks," *Engineering, Technology & Applied Science Research*, vol. 2, no. 3, pp. 226–231, Jun. 2012, https://doi.org/10.48084/etasr.17.