Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol.45 No. 1 (2024)

Design and Implementation of an Efficient Collision Resistant Novel Hash Function for IOT

Chaithanya S[1], Siddesh G K[2], L Rangaiah[3], Sunitha R[4]

[1,3,4]*Department of Electronics and Communication Engineering, RajaRajeswari College of Engineering, Bangalore, Visvesvaraya Technological University-Belagavi, Karnataka, India*

[2] *Department of Electronics and Communication Engineering , Alva's Institute of Engineering & Technology, Moodubidire, Visvesvaraya Technological University-Belagavi, Karnataka, India*

Abstract:- The ever-growing proliferation of communication facilities in modern times has led to a widespread adoption of Internet of Things. Such networks deal with a number of difficulties in maintaining user data security, privacy, and trust. It becomes difficult to provision the aforementioned services, particularly when working with real-time streaming data. When handling sensor network contents and streaming data, the consequences of privacy and confidentiality violations become more severe. A crucial idea in offering Internet of Things as a service is secure authentication while transmitting data between sender and recipient nodes. Such an authentication technique for IoT-based applications can be provided via hash algorithms. Secure data retrieval and effective access control on large-scale IoT networks need the integration of authentication technologies with IoT and cloud computing. It may not be wise to rely on a single cryptographic protocol or algorithm for all applications. If a flaw is found in one algorithm, the danger of a catastrophic failure is decreased by diversity provided by new algorithms. In this study, we suggest, put into Implement and evaluate the Novel Hash Function designed on the base of Bellman Equation. This function takes mutable length input and produces a fixed length hash 256 bits and 128 bits are the two different lengths for hash values. The performance of the suggested approach is compared to that of the traditional hash algorithms, including SHA-2, SHA-3 and other existing hash functions. The results of the statistical tests and cryptanalytic analyses show how reliable the suggested unkeyed hash functions. This collision Résistant Hash Function.can be applied for secure signature schemes, pseudorandom number generation, session key generation, etc.

Keywords*:* Authentication, Bellman Equation, IoT, Cryptography, Unkeyed Hash.

1. Introduction

With the advancement of Internet technology, electronic communications are becoming increasingly vital in daily communication. The e-business and e-commerce industries, where authenticity and integrity are vital, are pertinent to these digital information interchange processes. This increases the need for privacy and integrity for private and privileged data, including passwords, user IDs, and encryption keys. In the direction of fulfill this objective, cryptographic hash functions are a perception that has been generated and used. Hash operations are among the essential elements of contemporary cryptography for secure communication. They are essential to many different applications, including criminalistics analysis, password management, generating random numbers for special session keys, and establishing a unified view in block chains. They also play a solid protagonist in verifying the authenticity and integrity of the data. To strengthen basic cryptographic methods like hash functions and symmetric and asymmetric ciphers, advancements in the science of cryptography were necessary. Due to technology shifted to compact designs, thus there is a great deal of optimism surrounding the upcoming event from a security standpoint. This paper addresses an approach for an efficient hash algorithm for authentication.
The hash algorithm helps computers to quickly authenticate and decode particular shared information, such as SSL certificates. A unique digest of a certificate and its signature is produced by SHA, preventing fraudsters from

revealing the confidential data. Hashing is a function that converts an input of any dimension into a string of specified dimension that serves as a ”fingerprint” String. An index into a hash table is typically created using this kind of method. Keyed and unkeyed hash schemes are the two categories into which hash functions fall. Unkeyed hash functions just require a message; however keyed hash methods demand an input message and an authentication key as input parameters. [1] Without knowing the secret key, it will be adversary finds it challenging to create the same hashes as the process of an encrypted hash function. Thanks to improvements in cryptanalysis tools and the increased processing power of modern computer systems, it is simpler to break the past hash functions. Therefore, new and more powerful cryptographic algorithms are constantly needed to stay abreast on hazards and security requirements.[2] The security protocol must have hash functions; Message authentication codes, encryption and digital signatures, and the creation of pseudo-random numbers are some examples of its applications. [3] The cryptographic hash value [4] , with slight modification to the data—whether deliberate or unintentional—will almost certainly alter the hash result. The hash was commonly denoted as the message digest or just the digests, however the data you wanted to encrypt was often referred to as the message. These techniques mimicked the block cypher modes of operation that were typically engaged for encryption. All Notable hashing operations, that is MD4, MD5, SHA-1, and SHA-2, are constructed from specially created block-cipher-like components with feedback to guarantee the final function is noninvertible. Block cipher-like components were included in some of the SHA-3 finalist functions, but the function that was ultimately chosen was built on a cryptographic sponge. [5]Hashing algorithms have various other characteristics that distinguish them suited for use in digital signatures schemes and as a way to verify the message’s integrity. This system consists of two functions: Sign and Verify. Which provides a Boolean response true designating, if the given Sign is a legitimate signature for message. However, it ought to be difficult to build a fake signature. There are two kind of forgeries that may be distinguished: existential and universal forgeries. Using a 128-bit message that the user provides to produce a fingerprint, MD5 measures the data integrity. Although the message’s size may fluctuate, its irreversibility is its most important feature. The 32-bit and 16-bit machines can use this algorithm the best. Although this approach is also compatible with 64-bit processors, the architecture of this category of scheme may make it fairly slow. The MD5 is an extension of the MD4 scheme, which has three rounds and is relatively faster, MD4 has four rounds. SHA was published by NIST and the NSA. SHA was once meant to be a component of the Digital Signature Standard (DSS), a data signatureing system that required a hash function in order to operate. SHA-0 A term used to refer to the original release of the 160-bit hash function, in 1993, which was titled ”SHA”. It had a ”significant flaw” that wasn’t reported; therefore it was pulled from circulation soon after release. It was replaced with the slightly updated SHA-1[6]. The creation of the MD4 and MD5 message digest algorithms, Ronald L. Rivest [7] of MIT used similar design concepts. However, SHA-1 employs a more conservative approach. The compression technique used by SHA-1 differs from SHA-0 by a single bitwise rotation in the message schedule. The difficulty of this attack is still 2^{69}, even though SHA-1 proved to be considerably more challenging. In comparison to a brute-force attack on SHA-1, this collisional method only needs a fraction of the 2^{80} calculations [8]. Majority of the classical hash schemes rely on basic operations like modular arithmetic, algebraic and logical operations for practical commitments. These techniques are currently considered to be unsafe. Owing to lack of resilience to collision assaults for practical purposes. A collection of more intricate hash functions, whose output [9] sizes vary from 224 to 512 bits. Because non-linear were incorporated into the compression function, the hash algorithms SHA-2 series are more complicated. SHA 2 Newfangled hash algorithms, SHA-256 and SHA-512, are considered using 32-bit and 64-bit words, correspondingly. Although they employ different shift extents and additive coefficients, other than the number of rounds, their architectures are nearly identical. Simply abbreviated forms of the first two[10], calculated with various beginning values, are SHA-224 and SHA-384. Four security blemishes were discovered in SHA-1, more particularly, that a mathematical flaw might exist, highlighting the need for a better hash functionSHA-512/224 and SHA-512/256 are shorter versions of SHA-512 as well, however the starting values are created in a similar way to that in FIPS PUB 180.. Even though SHA-2 and SHA-1 have certain similarities, various assaults on SHA-2 have not been successful. A transposed duplicate of the input block is inserted after which round constants were XORed, before each ChaCha round in the cryptographic hash function known as BLAKE[11], Similar to SHA-2, there are two variants with different word lengths. The foundation of ChaCha is a 44-word array. By mixing an 8-word hash value with 16 message words, BLAKE has been diminished the ChaCha effect to obtain the

subsequent hash value. Its successors, BLAKE2 and BLAKE3 are Parallel variations were created to perform better on multi-core platforms. In 2009, Guo, J [12] there were reported collisions counter to 2.5 rounds in 2^{112} BLAKE2s processes, 2.5 rounds in 2^{224} BLAKE2b operations.

Data is ”absorbed” into the sponge used by SHA-3, and the extracted result is then ”squeezed” out. A part of the state is XORed with message blocks during the absorption phase, and then changed as a whole. In the ”squeeze” phase, state transformations alternate with reading output blocks from the same subset of the state. The state in SHA-3 is made up of a 5 by 5 array of 64-bit words. Additionally, Keccak is specified for power-of-2 word sizes as tiny as 1 bit[13]. It is possible to evaluate cryptanalytic attacks using small state sizes, and practical, lightweight applications can make use of intermediate state sizes, Using xor, and not operations, the block transformation is a permutation. Despite the fact that cryptographic hash algorithms are well-devise and often employed, there is consistently a possibility of collision, when two distinct messages will produce the same MD. As a result, there is a chance of an attack where a tamper can produce fake messages. Attacks have specifically been developed to take advantage of how hash functions are computed, in which logical functions are frequently used, in order to achieve collisions. Consequently, it is difficult to build an effective cryptographic hashing algorithm. The function ought to be collision-resistant and one-way efficient. An entirely different hash value will result from a small modification to the input. Interestingly, chaotic maps also exhibit these features. To benefit from these qualities [2], numerous academics have included chaotic maps into hash functions. Chaos-based hash functions have remained of academic curiosity despite a variety of novel designs over the years, as they are typically not used for real-world applications. The absence of adequate security verification through cryptanalytic efforts is one of the causes of this. [2],[14] In several chaotic hash functions, real number computation which is necessary for iterating chaotic maps is typically done employing the floating-point format. Certain hash functions based on chaos select fixed-point representation, [15] else are created in the integer field [16] because the usage of floating-point format causes operation issues in regarding efficiency, reliability, and simplicity of analysis. Additionally, the linearity of hash functions makes cryptanalytic assaults similar to those utilized in the aforesaid techniques easier to execute. Hash functions are operations that condense an adaptable length input into a standard-length output. Hash functions are a very effective tool in the construction of methods to secure the validity of data if they meet other requirements. Hashing is the process of transforming a mathematical expression H using a hash function. An input array, key, or message is another name for an input data M. A hash function’s cryptographic strength is assessed by how difficult it is to solve, these primary obligations. Properties of one-way hash function.
A function h is considered to be a one-way hash function if it meets the specified requirements.

- The range of the argument X is not limited, while the range of the result h(X) is fixed at n bits.
- The computation of h(X), given h and X, ought to be "simple."
- The hash function needs to be one-way in the perception that it is ”hard” to root out a message X such as h(X) = Y given a message Y in the image of h and ”hard” to discover a message X’= X such as h(X’) = h(X) given a message X and h(X).
- Collision-resistant hash function must be resilient to collisions and reduce output value duplication since it must be "hard" to distinguish two distinct messages that hash to the same result.

2. Methodology

The Bellman Equation based operation are devised on Matrix with logical functions used to generate hash algorithm is a recent idea. In this approach the message of any size will be divided in to blocks of 128 bit/256 bit, and it will be the input to matrix. The message block M will be endured following process.

2.1 Initialize State matrix

Internally First the state matrix has to be built, that is the algorithm’s operations are implemented on a two-dimensional array of bytes. There are n columns, and m rows, where m and n may be same or different with respect to the output hash code. The State comprises of N Rows and N columns. Each containing L bytes, where

L is the block length divided by 32/64 respectively for128/256 hash. This Every single byte/word in the State matrix denoted by the symbol m has two indices: a row r within the range 0 < r < N and a column c within the range 0 < c < N. N=4 for 128 bit standard, i.e., 0 < r, c < 4

In this approach we are taking N as 4 for the state matrix, since we try to develop 128/256 bit hash. Fill the data in byte/word, from most significant bit to least significant bit to the state array. Initially the state matrix is copied with the input array Di, of bytes di1,di2. . . .. according to the structure.

$$m[r,c] = Di[r * N + c], 0 < r < N, 0 < c < N \quad (1)$$

Upon completion of the procedure, to the output array Hout, the State is copied as follows.

$$Hout[r * N + c] = m[r,c], 0 < r < N, 0 < c < N \quad (2)$$

2.2 Calculate hash factor

The hashing factor δ is thought to be an important factor required for hashing in case of simple massages. δ=l, where l is the span of the input message. To achieve security in sensitive data, and the hashing factor will be a sum of all element of the state matrix or secret key if same function is changed to keyed hash. i.e. δ=Sk. Hence we can reduce collision to a great extent. Here we can call δ as a key value, so we can call the hash function as key based hash function particularly in this case.

$$\delta = \sum_{i=0}^{N-1} . \sum_{j=0}^{N-1} m_{ij} \quad (3)$$

In case of unkeyed hash to get collision resistance hash values The hashing factor δ is calculated by using equation(1), in case of repeated random numbers, we can use it as a block count so every hash will vary even if same number is repeated.

2.3 Round Function

The Round Function is mainly designed on the base of Bellman Equation. The cell value will be updated by adding the compound factor to the current value.

$$mt + 1 = (mt + \chi) \bmod 256 \quad (4)$$

The updated cell value of the state mt+1[r, c] is given by the current value of state mt[r, c] is summed by the modulo 256 addition with the compound factor χ. Addition of a round function value K to ξ yields a compound factor χ.

$$\chi = K + \xi \quad (5)$$

β is the maximum element present in the row pointed by the column of the particular cell.

$$\beta = \max_{\forall j \in [0,..N-1]} m_t(c_t, j) \quad (6)$$

The hashing factor δ is multiplied with the maximum element β results the product ξ, will be added with the round function K to gives an compound factor χ.

$$\xi = \delta * \beta + \sum_{i=1}^{n} m_t(r_t, c_i) \quad (7)$$

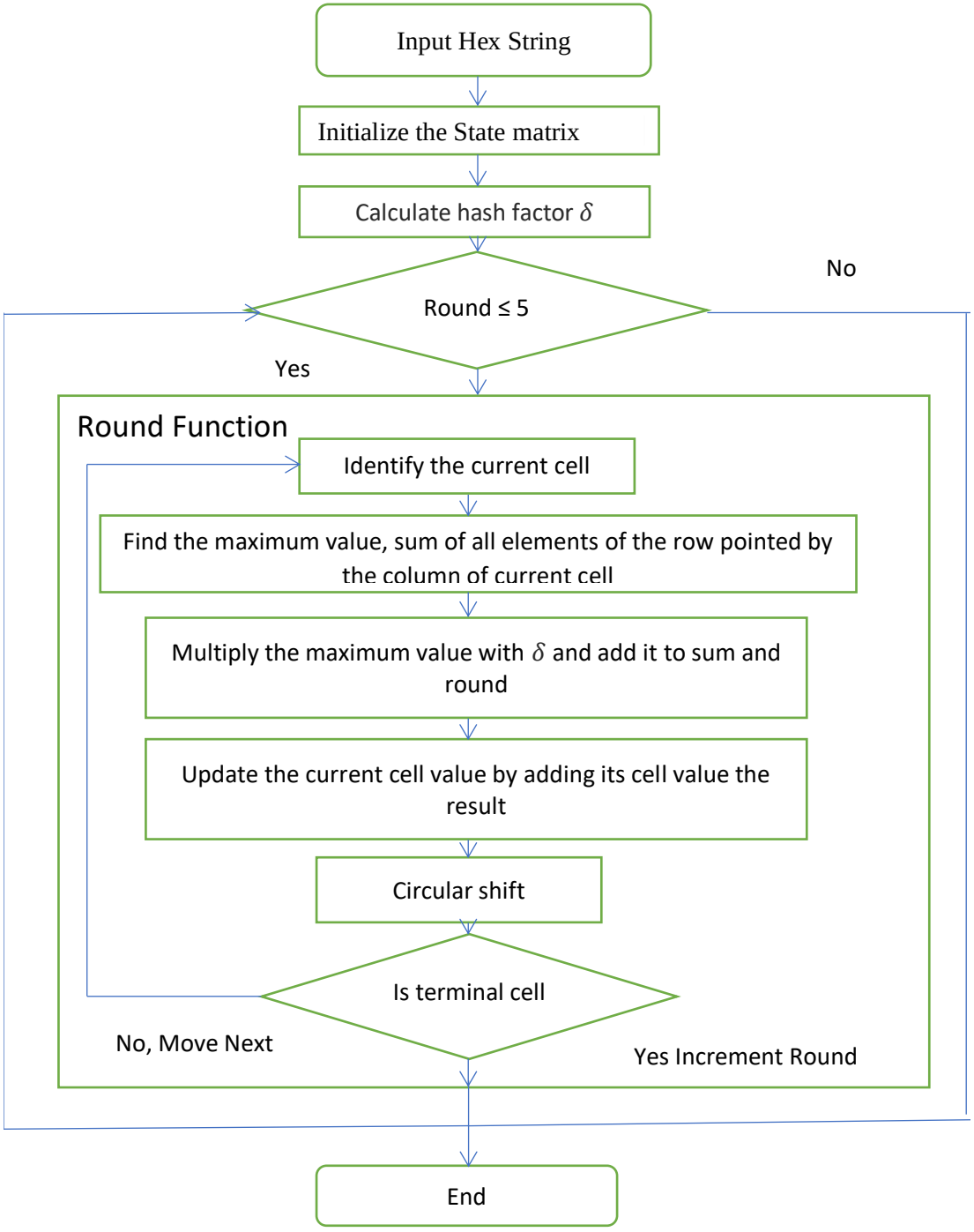

Figure 1:FlowChart

2.4 Circular shift

The circular shift of υ of the n entries in the tuple of data will make the data more random. υ of the n entries in the tuple is such that,

$$\upsilon i = (i + 1) \bmod n, \quad \forall\, i = 0, \ldots n - 1 \tag{8}$$

The figure 1 shows the flow of the proposed algorithm. Number of Rounds we considered here as 5 to attain necessary security and optimal performance.

3. Experimental Evaluation

3.1. *Hash Value Distribution*

For various input messages, a hash function ought to produce equally dispersed hash results. If information is not distributed consistently there might be information leaks that an adversary might utilise to detect collisions or launch forgery attacks. We chose 1,000 Message inputs at random in an effort to judge the level of incidence of the suggested hash function. To analyse their distribution, each of these message's hash values is converted into hexadecimal values. These input messages are counted according to how often each hexadecimal value, follow-on in the uniform spreading seen in Figure 2.

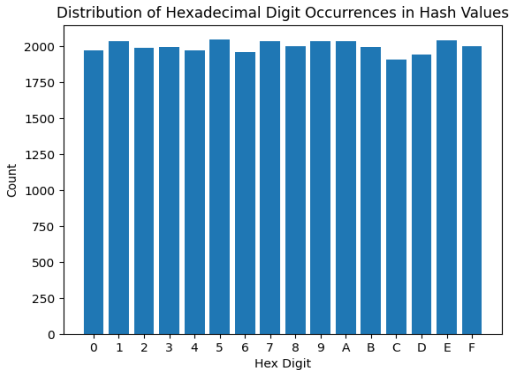

Figure 2: Distribution of Hexadecimal Digit Occurrence in Hash Value.

10,000 distinct input messages were hashed, and we counted the amount of changed bits for respective bit location in direction to further confirm the scattering of the recommended hash function is uniform. Theoretically, the bits must vary 5,000 times for a perfect distribution. The average value obtained by the recommended hash algorithm is 4989, which is close to ideal. Figure 3 depicts the outcomes of the distribution test for N = 10,000. All bit locations can be realised to be near to Hypothetical value demonstrates that the suggested hash function can withstand statistical attacks.

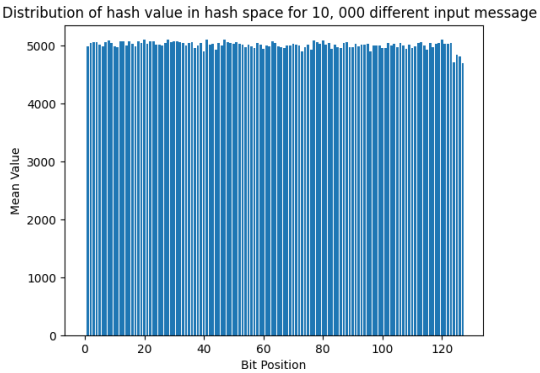

Figure 3: The dispersion of hash values inside the hash space

Another statistic to measure the variations flanked by ones and zeroes in a hash function output is linear convolutedness. For various inputs messages, we tally the quantity of ones and zeroes of the hash value in direction to absorb the linear convolutedness. The quantity of 1s should equal the quantity of 0s. Figure 4 displays the recommended hash algorithm's linear complexity for N different texts. The suggested hash function, as can be seen, has linear complexity that is almost optimal.

3.2. *Distribution Analysis*

An input message was created using indiscriminate characters and then converted to its corresponding decimal values, which were then shown in Figure 4 to demonstrate how crucial it is for hashing schemes to allowing for distribution examination of the hash value. The algorithm's hash values are evenly dispersed across the scope of

hash values shown in Figure5. This made sure the hash distribution was sufficient uniformity to act as both a robust .

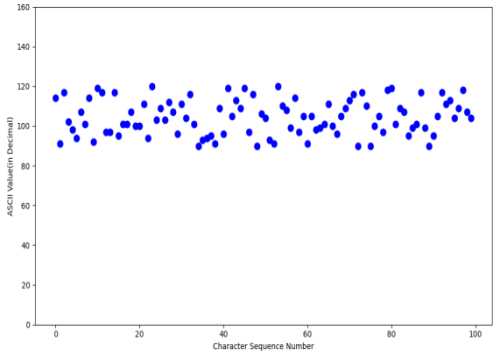

Figure 4: Distribution of ASCII Input .security measure and to conceal information.

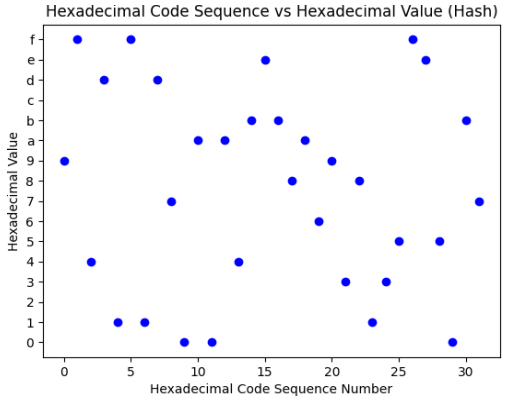

Figure 5: Distribution of Hexadecimal code sequence in hash.

3.3 Sensitivity Test

A reliable hashing algorithm needs to be extremely sensitive to the secret key or message. This test makes use of a 128 bit input message. The input message is then subjected to a series of minor alterations, for which hash values are created. These changes comprise,

1. A 128-bit input message M made up entirely of hexadecimal integers
2. Tilt M's first bit.
3. Tilt the last M bit.
4. Tilt the M's central segment.
5. Tilt the M random bit.
6. Tilt the last capacity bit.

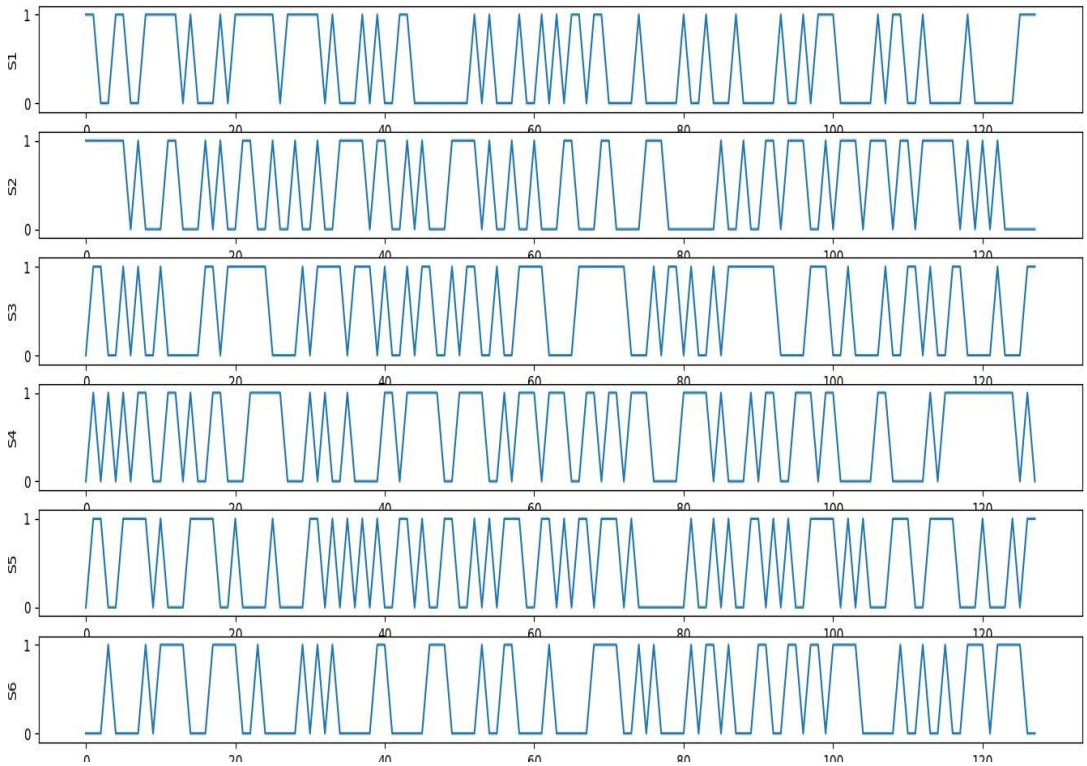

Figure 6: Binary Representation of Hash for different cases.

The output hash for all the above samples is generated. Figure 6 displays six resultant hash values in binary. The hash values produced show a great level of sensitivity to minor modifications because they are completely different despite the changes modest adjustments to the message input. Six hash values for different kinds of situations are shown in hexadecimal form in Table 1. The table shows the high input sensitivity and avalanche properties by reporting the fraction of bits that transformed for each sample in comparison to the starting hash.

Table 1: 128-bit hash values in 6 distinct circumstances, with a varying bit-to-bit ratio

Message	Modification to message	Hash	Ratio of no. of bits changed
Sample1(S1)	Input	ccfa2fdf45300a256c209104b82c8207	
Sample2(S2)	First	fd18a6493d947a48c61c049b9776faa0	0.5078125
Sample3(S3)	Last	6520df85ee96593c3f8b4bf872134c23	0.546875
Sample4(S4)	Middle	559a63e290df3cbb9b70f459d8305ffa	0.5546875
Sample5(S5)	Randam bit	67a3c8435534cae6b7404a6a7a8e788b	0.4765625
Sample6(S6)	Last capacity	10bc7905418384c20f285a336f0493bc	0.5
Average			0.5171875

3.4 Diffusion and Confusion Test

Two strategies confusion and diffusion for threatening a statistical study are suggested by Shannon CE [17]. Building the association between the cipher text and the key as intricate and complicated as imaginable is stated to as confusion. Diffusion describes the characteristic where redundancy in the plain text statistics is ”dissipated” in the cipher text measurements. The statistical structure of M that results in its redundancy is ”dissipated” in the diffusion method into far-off statistics, or into statistical structure requiring lengthy arrangements of data bytes in cipher. The result is that, because the structure is only visible in blocks with extremely low individual probabilities, the opponent must intercept a huge amount of material to take it down. Additionally, even when he has enough data, the amount of analytical work needed is substantially higher because the duplication has been dispersed among a huge number of distinct statistics. A hash function should have either confusion and diffusion or both. The following procedures are employed in a way that analyse the suggested hash function for these properties:

- To create H1, the selected hashing the chosen input message M.
- In order to generate M1, a single haphazard bit is altered in M.
- Hash M1 to produce H2.
- Examine the quantity of bits were altered from H1 to H2.
- Run the experiment N times.

The following measures that help us gauge diffusion and confusion are then calculated using the formula.

- Bits that have changed the least $l_b = \min_{\forall\, i\in[1..N]} b_i$
- Greatest amount of bits changed $g_b = \max_{\forall\, i\in[1..N]} b_i$
- Modified bit number means $\mu_c = \sum_{i=1}^{N} \frac{b_i}{N}$
- Average Modified probability $P_c = (\sum_{i=1}^{N} \frac{b_i}{N})/(N * l_h)$ where $l_h =$ bit length of hash value
- Standard deviation of the revised bit number $\sigma_c = \sqrt{\frac{\sum_{i=1}^{N}(b_i-\mu_c)^2}{N}}$
- Standard deviation of the revised probability $\sigma_p = \sqrt{\frac{\sum_{i=1}^{N}(b_i-P_c)^2}{N}}$

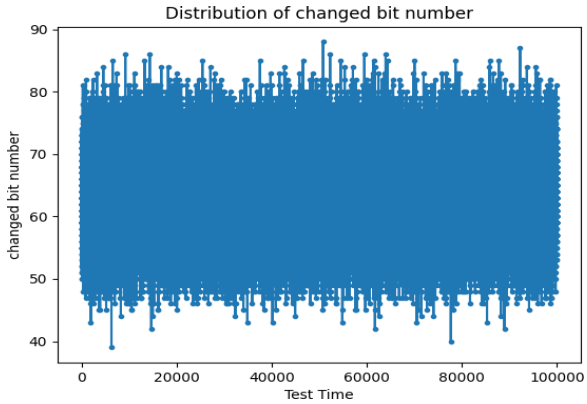

Figure 7: Distribution of the Bi values obtained with n = 128 and N = 10000

The experiment is run N times, By choosing random input message of 128 bits with the suggested hash technique, The hash value of the changed data is then calculated by selecting one bit from the plain data, by flipping its value. Comparing the two hash values where Bi is a measurement of the quantity of altered bits. The same first string is

reused throughout all N iterations. where N is equal to 128, 256, 512, 1024, 2048, 10,000 and 1,00,000, respectively. The figure.7. exhibits the Distribution of the Bi values obtained with n = 128 and N = 10000 and figure.8 shows its histogram.

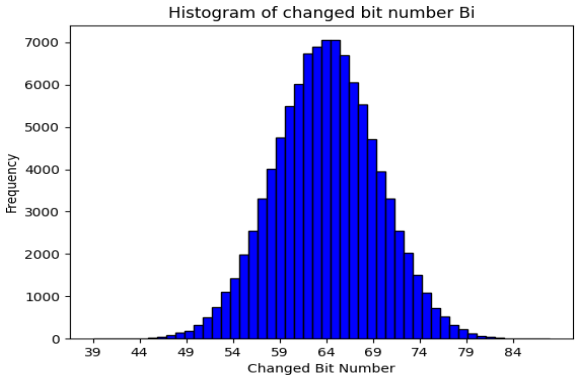

Figure 8: Histogram of changed bit number Bi

Table2 lists the comparable values for lb,gb,μc,Pc,σc,σp the average of the statistics with the changed bits is also tabulated.

Table 2: Statistics with the changed bits

N	l_b	g_b	μ_c	P_c(%)	σ_c	σ_p
100000	49	78	64.2398	50.18734	5.40341	4.22141
10000	49	78	64.2546	50.12078	5.40678	4.22405
2048	49	78	64.33789	50.26398	5.32187	4.15771
1024	49	78	64.35156	50.27466	5.3542	4.18297
512	49	78	64.43359	50.33875	5.36877	4.19435
256	49	78	64.51563	50.40283	5.70223	4.45486
128	49	78	64.08594	50.06714	5.32392	4.15932
AVG	49	78	64.3170	50.2365	5.411597	4.22781

Table 3: Statistical efficiency evaluation with 128-bit and N = 2048

Algorithms	μc	Pc(%)	σc	σp
Amine Zellagui [18]	64,12	50.05	5.16	4.22
Teh [19]	64.02	50.01	5.64	4.4
Li, Y [15]	63.94	49.96	5.61	4.38
Ren [20] Composite multi-scroll attractor	63.99	49.99	5.68	4.46
Ren [20] Double-scroll attractor	63.89	49.91	5.66	4.42
Ren [20] Single-scroll attractor	63.94	49.95	5.67	4.43
Mihaela [21]	64.11	49.99	5.51	4.33
Lin's [22]	63.84	49.87	5.76	4.5
Ahmad's [23]	63.87	49.9	5.85	4.36
Li, Y., Li, X.[24]	64.27	50.21	5.59	4.36
Kanso[25]	64.01	50.01	5.61	4.38
Xiao D[26]	63.8501	49.88	5.7898	4.52
Yang, H[27]	64.14	50.11	5.55	4.34
Li, Yantao[28]	63.89	49.91	5.64	4.41
Wang, Y[29]	63.98	49.98	5.53	4.33

Meliolla, G[3]	63.62	49.703	5.343	4.175
Akhavan, A[30]	63.9102	49.9298	5.6978	4.4514
M. Alawida[31]	63.941	49.832	5.689	4.413
M. Todorova[32]	63.95	49.96	5.68	4.43
Ahmad, M[2]	64.095	50.074	5.736	4.481
Chankasame, W[14]	62.65	48.94	5.479789	4.281085
Lin, Z., Yu, S[16]	63.95	49.96	5.62	4.39
Proposed	64.23789	50.16398	5.13187	4.15771

Table 4: Statistical efficiency evaluation with 256-bit and N = 2048

Algorithms	**μc**	**Pc(%)**	**σc**	**σp**
SHA2 256	127.98	50.00	8.10	3.16
SHA3 256	128.01	50.00	8.07	3.13
Amine Zellagui [18]	128.66	50.06	7.27	3.12
Hongjun Liu,[33]	128.13	50.05	7.94	3.1
Yijun(SHA256-ISH)[34]	128.01	50	8.17	3.19
Mihaela [21]	128.98	50.38	8.08	3.15
Chankasame, W[14]	126.93	49.58	7.904845	3.08783
Proposed	128.6736	50.08033	7.18397	3.02061

The Table 3 shows the Statistical evaluation of outcome for N = 2048 with 128-bit hash. The Statistical evaluation of outcome for N = 2048 with 256-bit hash is tabulated in Table 4.

3.5 Collision Resistance Analysis

Collision occurs when the same hash value is created by two distinct messages, whereas collision attack attempts to locate comparable hash values from two distinct inputs. . It is basically a probability issue in which two arbitrary inputs are discovered To give away a common hash value. Collision resistance is an important quality of an efficient hashing algorithm because It is challenging to locate pair of unlike messages with the identical hash value. We initially compute the hash digests for a pair of input messages that differ by one bit in the direction to analyse the suggested hash function's ability to resist collisions. A pair of hash values is shown as ASCII inscriptions for evaluation sake. Consider a hit refer to the quantity of identical ASCII inscriptions that appear at similar spot in the hash value.

Consider the hash of a random message M as h, the new hash $\hat{h}$ which is computed by modifying individual bit of the comparable message M, divide the n bit hash in to x sub blocks of size m bit each. The amount of m-bit sub blocks at the identical place and with the identical value in both hash values is counted as,

$$\alpha = \sum_{i=1}^{m} g\left(a(x_i), a(\hat{x}_i)\right), \qquad where\ g(u,v) = \begin{cases} 0 & u \neq v \\ 1 & u = v \end{cases} \tag{9}$$

The function a (·) translates the entries into their corresponding ASCII values. Smaller α exemplifies the hash function's greater resilience to collisions.

Since we are using the ASCII characters for comparison, m is equal to 8 bits.

The hypothetical number of α with different values throughout N distinct investigations, represented by$K_N(\alpha)$, is therefore assumed founded on the supposition that hashes are distributed randomly and uniformly.

The amount of hits when N samples are used in the test is denoted by $K_N(\alpha)$. For different N values, the theoretical $K_N(\alpha)$ calculated as,

$$K_N(\alpha) = N.\frac{x!}{\alpha!\,(x-\alpha)!}.(\frac{1}{2^m})^{\alpha}.(1-\frac{1}{2^m})^{x-\alpha} \quad (10)$$

where α = 0, 1,..., x and a collision occurs if $\alpha = x$ = n/m. since m= 8 bits , x=16, the theoretical value of $K_N(0) =$ 9392.9809, $K_N(1)$ =589.3635, $K_N(1)$= 17.3342.. $K_N(15)$= 1.199e-31, $K_N(16)$= 2.9387 e-35

The practical value of $K_N(0) = 9427.46$, $K_N(1)$ = 572.54 and $K_N(\alpha)$=0 for α =2, 3,..., 16

The experimental and theoretical findings might be seen as are fairly close. In addition, comparisons of KN (α) between theoretical values, experimental values of suggested algorithm are provided. Figure 9 and Figure 10 shows the Logarithmic scale and decimal scale of Concentration of the quantity of identical ASCII values in hash values, correspondingly, at the same place.

Thus, the ultimate hash value's each bit is guaranteed to be connected to every other bit of message. A alteration of individual bit in the data will be diffused, resulting in major modifications to the ultimate hash value. Furthermore Table4 compares the resistance to collision of several hash algorithms, Take note that there can only be one identical character at most, and the likelihood of collision is quite minimal. Table 5 gives the comparison of number of hits for N=10000 between Ideal and Proposed Hash.

To create hash values, two distinct input messages that are of only one bit difference are utilized. Then comparing both hash values and stored as ASCII inscriptions.
The formula for calculating the absolute difference d is,

$$d = \sum_{i=1}^{m} |(t(x_i) - t(\hat{x}_i))| \quad (11)$$

While the function t(.) translates a character in the ASCII code into its equivalent decimal value, x_i and $\hat{x}_i$ are the ith hash code in ASCII for both the initial and updated input messages, respectively.

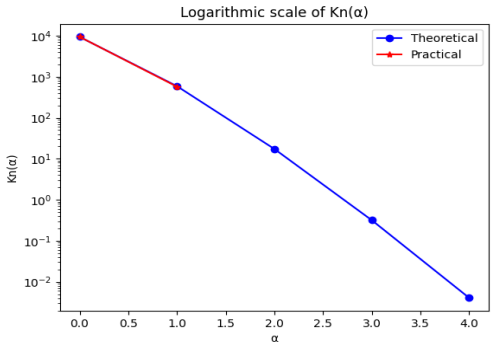

Figure 9: Frequency of identical ASCII values spread at the identical place in hash values [Logarithmic scale of Kn(α)]

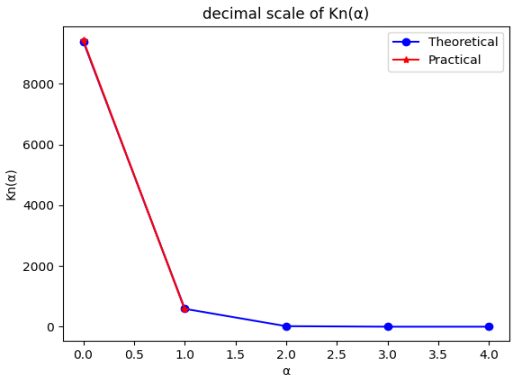

Figure 10. Frequency of identical ASCII values spread at the identical place in hash values [decimal scale of kn(α)]

The suggested hashing is vital to compare the average mean of absolute difference as calculated in theory. Utilizing 16 characters of ASCII code (128-bit hash length), the discrete uniform distribution H has a 0 to 255 range.

A uniform distribution's greatest value is divided in half by its mean value, which is the same. The sum of all possible characters is 255 x 16 = 2, 040.

Table 5: Hit count for N = 10,000

N=10000					
HITS	**0**	**1**	**2**	**3**	**4**
Ideal	9392.98	589.36	17.33	0.3172	0.00404
proposed	9427.46	572.54	0	0	0
Li, Y[35]	9414	569	17	0	0
Ahmad's [23]	9387	586	27	0	0
Li, Y., Li, X.[24]	9554	404	42	0	0
Yang, H[27]	9381	605	14	0	0
Wang, Y[29]	9777	607	16	0	0
Akhavan, A[30]	9434	553	13	0	0
M. Alawida[31]	9365	600	35	0	0
M. Todorova[32]	9373	607	20	0	0
Ahmad, M[2]	9396	583	21	0	0

The hash digest's optimum hypothetical value is used in order to calculate the optimal theoretical absolute difference. Under the hypothesis that both hash values H and $\hat{H}$ are ideal uniform distributions, the "Sum of Absolute Difference" of the dual aforementioned Digests must align with two thirds of the uniform distribution's average mean value.

The ideal average of absolute difference's mean among H and $\hat{H}$ is comparable to (2/3) X2, 040 = 1, 360 Since the average 2,040 is the average value for a uniform distribution with 128-bit hash digest properties.

Table 6. Absolute distance for 128-bit hash with N=512,1024,2048

N	Minimum	Maximum	Mean	Mean/char
512	692	2123	1361.461	85.09131
1024	692	2123	1368.383	85.52394
2048	692	2123	1375.7	85.98125

Table 7. Comparing absolute distances of other algorithms with proposed128-bit hash for N = 2048

Algorithms	Minimum	Maximum	Mean	Mean/char
Proposed	692	2123	1375.7	85.98125
Amine Zellagui[18]	673	2261	1364.8	85.3
Teh[19]	584	2321	1366	N/A
Li, Yantao[28]	577	2089	1335	83.41
Ahmad's[23]	537	2399	1364.7	85.29
Li, Y., Li, X.[24]	646	2096	1429	89.29

Xiao D[26]	696	2221	1506	94.125
Wang, Y[29]	689	2295	1526	N/A
Li, Y[35]	688	2019	1350	84.38
Akhavan, A[30]	744	2431	1371	N/A
M. Alawida[31]	N/A	N/A	1,353.5	N/A
M. Todorova[32]	573	2450	1362	N/A
Ahmad, M[2]	554	2522	1361	85.06
Chankasame, W[14]	544	2400	1348	N/A

The mean 1375.7 for N = 10,000 tests, which is extremely near to optimal value. This recommends that the diffusion of suggested hash values is uniform and is almost ideal and that all of the hash value's characters have an equal chance of occurring.Table 6 displays the mean, maximum, and minimum distance between hash values for various N values in a 128-bit hash. Table 7 compares this with alternative algorithms.

3.6 Hamming Distance Test

This test's objective was to determine how little deviations in the input data affected the hash value.

When the T-Student test statistic |t| falls within the range of (0,1.96), the hash function passes the test. Half of Hash size is the expected value. A 5% significance level was chosen. This is the T-Student formula:

$$|t| = \left| \frac{sample\ mean - expected\ value}{sample\ standard\ deviation} \left(\sqrt{sample\ size}\right) \right| \quad (12)$$

For every pair of input and hash, we created an additional pair of input and hash, where Hash was defined as a digest computed from input, where input was the original data modified by a single arbitrary bit to its opposite. There are two inputs and two hashes that are quite comparable. The experiment's goal is to calculate the Hamming distance among these hashes. This procedure was repeated for all 10000 created inputs and hashing routines. You can define the hamming distance as follows.

The amount of 1s within the S3 string equals the Hamming distance between S1 and S2. If S1 is the primary bit series and S2 is the secondary bit series, then lengths of S1 and S2 are equivalent. Similar to the discrepancy among S1 and S2, it is the amount of sites where S1 and S2 diverge in terms of value. Table 8. show the experiment's findings for the proposed hashing function. The predicted distance is 64, so the horizontal red line is fixed to 64 is shown in figure. 11. Despite 5377 of 10,000 data points crossing the red line, nearly 50% was the result. According to the acute readings shown in Table 8, the middling is close to 50%. The T-Student test was successful because the |t| statistic was equal to 0.7282. For 256 bit hash, the 5289 of 10,000 readings crossing the red line, the cut was near to 50% as shown in figure 12. The T-Student test was successful because the |t| statistic was equal to 0.6864.

Table 8: Hamming distance test for proposed Hash

Proposed algorithm bit size	**Min**	**Max**	**Average**	**Standard deviation**
128	42.52	85.4	64.01086	5.647574
256	96.6	159.1	127.9651	7.966312

It was successful in the Hamming distance test since small changes in the input data result in proposed hashes that are at least 50% different.

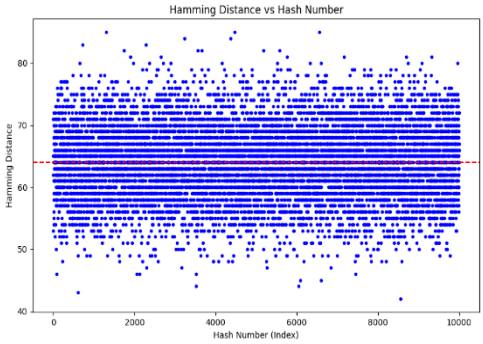

Figure 11: Hamming Distance vs Hash number(128 bit hash)

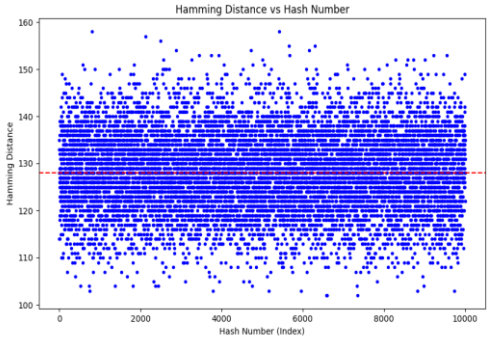

Figure 12: Hamming Distance vs Hash number (256 bit hash)

3.7 Bits Probability Test

The goal was to determine whether or not The digest's bits might be expected. We had to calculate the likelihood of 1's in each bit position in order to quantify it. The optimal scenario is when there is a Equal number of zeros and ones present.

$$P1(i) = 50\%, \forall\, i = 1 \ldots n \quad (13)$$

Here n is the hash length and i represent the bit position. We calculated the probability of '1' for 10,000 produced digests for each hashing function and the anticipated result is 50%, also calculated the |t| statistic by using the equation (12). When |t| <1.96 for significance level = 5%, the test is considered to be successful. Figure 13 and Figure 14 presents the experiment's findings for the proposed hashing function for 128 bit and 256 bits respectively.

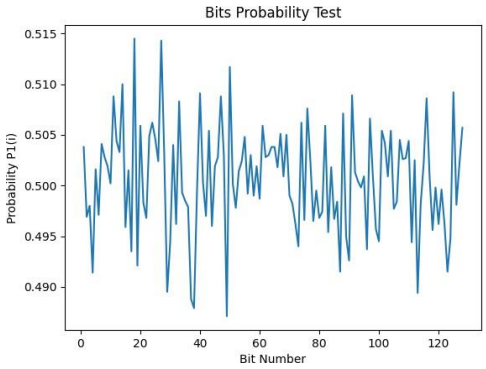

Figure 13: Bits Probability Test(128 bit hash)

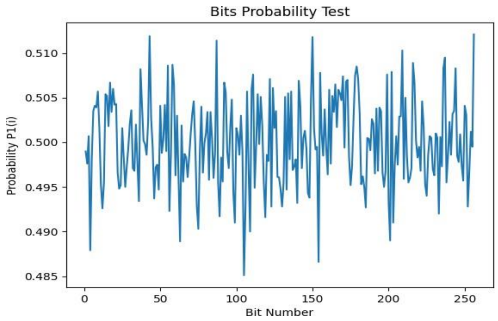

Figure 14: Bits Probability Test(256 bit hash)

As can be seen, the standard deviation is small and the average value is extremely near to 50%. The fluctuations are relatively modest even though none of the bits have a probability of 50% or greater. Given that the |t| statistic equals 0. 7282, it may be concluded that none of the 128 hash bits can be anticipated.

4. Conclusion

Proper implementation of the hash algorithm should ensure both cryptographic integrity and security. Here, a novel hash function based on bellman equation is proposed.

We have designed and realized novel cryptographic hash function with strong collision resistance assets. We conducted examination of the security performance.

The suggested hash function was evaluated based on a number of factors, including the distribution of hash values, susceptibility to minute shifts in the message, characteristics of perplexity, and dissemination, defiance to collisions, and efficiency. Results reveal that, in comparison to existing hash functions, the suggested function has almost perfect statistical features. The randomness in novel Hash has been successfully verified. And we also achieves a strong collision resistance hash with zero probability of getting more than one hits and has only 0.05 probability of getting single hit.

References

[1] P. Puniya, New design criteria for hash functions and block ciphers. PhD thesis, New York University, 2007.

[2] M. Ahmad, S. Singh, and S. Khurana, “Cryptographic one-way hash function gener- ation using twelve-terms 4d nonlinear system,” International Journal of Information Technology, vol. 13, pp. 2295–2303, 2021.

[3] G. Meliolla, K. A. Nugroho, and F. I. Hariadi, “Implementation of hash function on embedded-system platform using chaotic tent map algorithm,” in 2016 International Symposium on Electronics and Smart Devices (ISESD), pp. 179–183, IEEE, 2016.

[4] Sahu and S. Ghosh, “Review paper on secure hash algorithm with its variants,” 05 2017.

[5] W. Penard and T. van Werkhoven, “On the secure hash algorithm family,” Cryptography in context, pp. 1–18, 2008.

[6] B. Preneel, “The first 30 years of cryptographic hash functions and the nist sha-3 competition,” in Cryptographers’ track at the RSA conference, pp. 1–14, Springer, 2010.R. Rivest, “The md5 message-digest algorithm,” tech. rep., 1992.

[7] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full sha-1,” in Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25, pp. 17–36, Springer,

[8] H. Handschuh, H. C. van Tilborg, and S. Jajodia, “Sha-0, sha-1, sha-2 (secure hash algorithm),” Cryptologia, vol. 24, no. 2, 2000

[9] C. Dobraunig, M. Eichlseder, and F. Mendel, "Analysis of sha-512/224 and sha- 512/256," in Advances in Cryptology–ASIACRYPT 2015: 21st International Confer- ence on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21, pp. 612–630, Springer, 2015.

[10] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "Sha-3 proposal blake," Submission to NIST, vol. 92, pp. 1–79, 2008.

[11] J. Guo, P. Karpman, I. Nikoli´c, L. Wang, and S. Wu, "Analysis of blake2," in Topics in Cryptology–CT-RSA 2014: The Cryptographer's Track at the RSA Conference2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings, pp. 402–423, Springer, 2014.

[12] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output func- tions," 2015.

[13] W. Chankasame and W. San-Um, "A chaos-based keyed hash function for secure pro- tocol and messege authentication in mobile ad hoc wireless networks," in 2015 Science and Information Conference (SAI), pp. 1357–1364, IEEE, 2015.

[14] J. S. Teh, K. Tan, and M. Alawida, "A chaos-based keyed hash function based on fixed point representation," Cluster Computing, vol. 22, pp. 649–660, 2019.

[15] Z. Lin, S. Yu, and J. Lu¨, "A novel approach for constructing one-way hash function based on a message block controlled 8d hyperchaotic map," International Journal of Bifurcation and Chaos, vol. 27, no. 07, p. 1750106, 2017.

[16] C. E. Shannon, "Communication theory of secrecy systems," The Bell system technical journal, vol. 28, no. 4, pp. 656–715, 1949.

[17] Zellagui, N. Hadj-Said, and A. Ali-Pacha, "A new hash function inspired by sponge construction using chaotic maps," Journal of Discrete Mathematical Sciences and Cryp- tography, pp. 1–31, 2022.

[18] J. S. Teh, M. Alawida, and J. J. Ho, "Unkeyed hash function based on chaotic sponge construction and fixed-point arithmetic," Nonlinear Dynamics, vol. 100, no. 1, pp. 713– 729, 2020.

[19] H.-P. Ren, C.-F. Zhao, and C. Grebogi, "One-way hash function based on delay-induced hyperchaos," International Journal of Bifurcation and Chaos, vol. 30, no. 02, p. 2050020, 2020.

[20] M. Todorova, B. Stoyanov, K. Szczypiorski, and K. Kordov, "Shah: Hash function based on irregularly decimated chaotic map," arXiv preprint arXiv:1808.01956, 2018.

[21] Z. Lin, C. Guyeux, S. Yu, Q. Wang, and S. Cai, "On the use of chaotic iterations to design keyed hash function," Cluster Computing, vol. 22, pp. 905–919, 2019.

[22] M. Ahmad, S. Khurana, S. Singh, and H. D. AlSharari, "A simple secure hash function scheme using multiple chaotic maps," 3D Research, vol. 8, pp. 1–15, 2017.

[23] Y. Li, X. Li, and X. Liu, "A fast and efficient hash function based on generalized chaotic mapping with variable parameters," Neural Computing and Applications, vol. 28, no. 6, pp. 1405–1415, 2017.

[24] Kanso and M. Ghebleh, "A structure-based chaotic hashing scheme," Nonlinear Dynamics, vol. 81, no. 1-2, pp. 27–40, 2015.

[25] D. Xiao, X. Liao, and S. Deng, "One-way hash function construction based on the chaotic map with changeable-parameter," Chaos, Solitons & Fractals, vol. 24, no. 1,pp. 65–71, 2005.

[26] H. Yang, K.-W. Wong, X. Liao, Y. Wang, and D. Yang, "One-way hash function con- struction based on chaotic map network," Chaos, Solitons & Fractals, vol. 41, no. 5, pp. 2566–2574, 2009.

[27] Y. Li, D. Xiao, and S. Deng, "Keyed hash function based on a dynamic lookup table of functions," Information Sciences, vol. 214, pp. 56–75, 2012.

[28] Y. Wang, X. Liao, D. Xiao, and K.-W. Wong, "One-way hash function construction based on 2d coupled map lattices," Information Sciences, vol. 178, no. 5, pp. 1391– 1406, 2008.

[29] Akhavan, A. Samsudin, and A. Akhshani, "Hash function based on piecewise non- linear chaotic map," Chaos, Solitons & Fractals, vol. 42, no. 2, pp. 1046–1053, 2009.

[30] M. Alawida, A. Samsudin, N. Alajarmeh, J. S. Teh, M. Ahmad, et al., "A novel hash function based on a chaotic sponge and dna sequence," IEEE Access, vol. 9, pp. 17882– 17897, 2021.

[31] M. Todorova, B. Stoyanov, K. Szczypiorski, W. Graniszewski, and K. Kordov, "Bentsign: keyed hash algorithm based on bent boolean function and chaotic attractor," Bulletin of the Polish Academy of Sciences Technical Sciences, pp. 557–569, 2019.

[32] H. Liu, A. Kadir, and J. Liu, "Keyed hash function using hyper chaotic system with time-varying parameters perturbation," IEEE Access, vol. 7, pp. 37211–37219, 2019.

[33] Y. Yang, F. Chen, J. Chen, Y. Zhang, and K. L. Yung, "A secure hash function based on feedback iterative structure," Enterprise Information Systems, vol. 13, no. 3, pp. 281– 302, 2019.

[34] Y. Li and G. Ge, "Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security," Multimedia Tools and Applications, vol. 78, no. 13, pp. 17973–17994, 2019