Panamerican Mathematical Journal
ISSN: 1064-9735
Vol 35 No. 1s (2025)

Design and Implementation of Innovative Statistically Secured Hash Algorithm using Decentralized Hash Function

Bhagvant Ram Ambedkar[1], Kapil Joshi[2], Roosha Shamoon[2], Priya Thomas[3], Konda Hari Krishna[4], Harishchander Anandaram[5], Anil Kumar Lamba[6], Sakshi Tewari[7]

[1]Department of Computer Science and Information Technology, MJP Rohilkhand University Bareilly, U.P., India.

[2]Department of Computer Science & Engineering, Uttaranchal Institute of Technology, Uttaranchal University, Dehraudn 248007, Uttarakhand, India.

[3]Sr. Assistant Professor, Department of MCA, New Horizon College of Engineering, Bangalore, India.

[4]Associate Professor, Dept. of CSE, School of Computing, Mohan Babu University, Tirupati, A.P-517102, India.

[5]Assistant Professor (Senior Grade) , Amrita School of Artificial Intelligence, Coimbatore, Amrita Vishwa Vidyapeetham, India.

[6]Professor School of Computer Science and Engineering, Geeta University, Delhi NCR, Panipat India.

[7]Assistant Professor, Symbiosis Law School, Noida, Symbiosis International (Deemed University), Pune, India.

brambedkar@mjpru.ac.in[1], Kapilengg0509@gmail.com[2], rooshashamoon@uumail.in[2], priya.t_mca_nhce@newhorizonindia.edu[3], khk396@gmail.com[4], a_harishchander@cb.amrita.edu[5], anil.lambain@gmail.com[6], sakshiji.tewari@gmail.com[7]

Article History:

Received: *28-08-2024*

Revised: *28-09-2024*

Accepted: *16-10-2024*

Abstract:

Information technology is in high demand today due to the expanding global use of the internet for information sharing. Every user wants to secure communication over a public network and to confirm the accuracy of their data. Hash algorithms are used to translate the fixed-length hash code from the input message, which is used to verify the integrity of the input message or any information exchanged through the web. Numerous hash algorithms are proposed by researchers but many algorithms use high consumption time during hash function processing time and they are using centralized hash function processing. Therefore we are designing an Innovative hash algorithm to generate a 96-bit hash code using a decentralized hash function.

Keywords— Integrity; Key-Constant; One-time Padding; Security; SHA; Statistical Testing

Introduction

In this research we are designing 96 bit hash code by using the decentralization function processing. The exiting hash algorithms are using centralized function processing to produce hash code. The hash functions are confirmed the performance of message sent. How the hash function is processed and the hash-based techniques that were utilized depend on the hash codes [1]. Many hash algorithms are developed dynamic area to reduce the low power consumption and efficient application [2]. The lot of originations is using the basic cryptographic tools to provide authentication and [3]. We have very big challenge to designing a digital signature [4]. The processing of Hash functions is responsible to provide secure one-way hash code [5]. The demand of social media, secure transmission of information over public networks is one of the prime challenges [6]. Cryptographic hashing approaches are has a big challenge for sensitive data, and information [7]. We can use hill cipher technique increases the complexity of generating hash [8]. The hash function is a very sensitive to change input message to change one-way hash code [9]. The one way hashing is giving good performance compared [10]. By using a decentralized hash function technique, the time taken by the hash function is short [11]. The hashing algorithms are challenging to implement on smart devices such as debit card, credit card that has a low memory spaces [12]. We can improve the complexity of hashing by design new model [13]. The hash code based indexing are widely used in fast query services [14]. The well-defined three innovative version of SHA are produced by NIST to revised version of the standard [15]. The hash function is used to check the integrity of sensitive user data and information leaks from various cloud computing sources [16]. To ensure that digital information is valid, an authentication system is made using hashing [17]. The properties of a "good" cryptographic hash function are having the three preimage resistances to provide the security strength hash code [18]. Hash methods for sender and image authentication are used to construct the electronic signature, which is subsequently, encrypted using the RSA method [19]. Hashing algorithms SHA-1, SHA-2, MD-5, RIPEMD, and Keccak respecting a set of their basic operations [20]. The communication of information through public network hashes very big issue because it is controlled by some third-party entity [21]. The hash function used to store passwords securely is to protect confidential information from users with malicious intentions, prevent data leakage, and prevent data from being changed during transmission [22]. The present hashing methods are ineffective to be engaged for large-scale [23]. The main challenge is to design sparse hash functions, where variables can be chosen with smaller probability and lead to smaller-sized XOR constraints [24]. A set of algorithms that input a user-chosen password and provide a stream of pseudorandom bits presenting enough entropy and an adequate length to be used as a key in real-world applications [25]. Circular shifts are introduced to improve the randomness of hash values [26]. A brute force attack that applies the hash functions to all items that are likely to be in the input set [27]. The innovative DHF-96 is used fifty-four step to generate 96-bits hash code without using any publically known key constant. It is taking very low power consumption during decentralized function processing because it is not using any key constant. In the decentralized hash functions processes, we are using logical, arithmetic and circular left shift operations. It satisfies all security requirements and is statistically strong.

I. Objectives

A. *Hash Function*

The hash function produces fixed-length hash code to verify the integrity of digital data, or information of users communicated through public networks [28]. Many existing hash algorithms are producing hash code with centralized hash function processed and publically known key-constant. Because there is no need for any key constants because we cannot decode the hash code to decode the original message [29]. So we designed an innovative secured hash algorithm to produce 96-bit hash code which is processed by five decentralized hash functions without the need of any key constant.

B. *Padding*

The one-time padding increases the security strength of user information and is also used by hash functions to secure hash algorithms [30]. Padding is made up of all zero-field elements, with one field element being one [31]. We are computing the total number of padding bits for the "xyz" message as follows:

Total number of bits in input message = 24

Padding Bit =

100000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Length of Padding bit = 1000

Input Block with Padding Bit=
01111000011110010111101010000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000

Length of Input Block with Padding Bits= 1024

II. Basic Design of Innovative Statistically Secured Hash Algorithm

The innovative statistically secured hash algorithm is being developed in this part using decentralized hash function processing. The basic design of the Innovative hash algorithm is shown in Fig. 1.

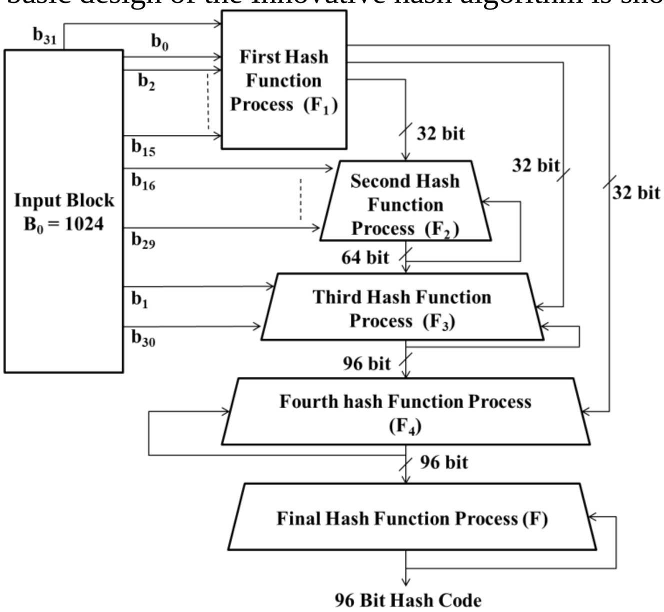

Fig. 1. Basic Design of Innovative Statistically Secured Hash Algorithm

The hash function processing of DHF96 is computed in five decentralized function processes. A 32-bit hash code is generated by the first hash function, a 64-bit hash code by the second, and a 96-bit hash code by the third. These hash codes are used by hash function four, which then generates a 96-bit hash code, and then hash function five uses the 96-bit hash code that was previously formed by hash function four to produce a 96-bit hash code that is mapped from a variable-length input message.

III. USING THE TEMPLATE

In this algorithm, we are using a variable length input message and it converts into a fixed length block of 1024 bits by padding bits. We are describing all function processing with the following steps.

1) First, we select any variable length input message = M
2) Convert M in binary using ASCII code
3) Maximum input message length (L) = 128 bits
4) If M <= L, then compute the P
5) If M > L, compute the sub-blocks M, Mi <= L,   where i = 0,1,……..
6) Padding bit (P) = (896 - Mi) mod 1024+L
7) Now compute the length of input blocks (Bi) = M + P = 1024 bits, where i = 0, 1,…………
8) Then we split input blocks Bi into 32 blocks (b0, b1, ………., b31), each block size will be 32 bits..

A. *Details of Decentralized Hash Function Processing (F)*

First Hash Function Process (F1): F1 produces a 32-bit hash code from the input message and the processing of F1 is computed as follows:

$P_1 = (\neg b_0 * \neg b_{31})$ where: ¬ = inverse of given variable and * = product of input blocks

P = output of logical NOR operation

$P_i = (\neg P_{i-1} * \neg\, b_{i+1})$,   where i = 2, 3,..........,15, and CLS_{11} = 11 bit circular left shift

CLS11= Pi , where i = 2, 3, 6, 9, 12, 15

Second Hash Function Process (F_2): F_2 produces 64-bit hash code from the output of F_1 and the processing of F_2 is computing as follows:

$P_{16} = \{\neg(P_{15} + b_{16})* \neg(b_{17} + P_1)\}$ Where "+" = bitwise join operation

$P_{17} = \{\neg(P_{16} * \neg(b_{18} + P_2)\}$

$S_1 = \{(P_{17} \oplus(b_{19} + P_3)\}$ where Si = output of modulo-2 addition

$S_i = \{(S_{i-1} \oplus(b_{18+i} + P_j)\}$,   where i = 2,3,………,11 and j = 4,………….,14. CLS_{11}= S_{i+1} where i = 0, 3, 6, 9

Where $\oplus$= Exclusive-OR logical operation

Third Hash Function Process (F_3): F_3 produces 96-bit hash code from output of F_2 and processing of F_3 is computing as follows:

$S_{12} = \{(P_{28} + b_1) \oplus (b_{30} + P_{15} + P_8)\}$

Fourth Hash Function Process (F_4): F4 is producing 96 bit hash code from output of F_1 and F_3 hash code and processing of F4 is computing as follows:

$S_{13} = CLS_{37}$ { $(S_{12}) \oplus (P_{16} + P_1)$}

$S_i = CLS_{37}$ {$(S_{i-1}) \oplus (P_{4+i} + P_j)$}, where i = 13,…..,24, and j = 3, 2, 4, 6, 5, 7, 9, 8, 10, 11, 13, 14

Final Hash Function Process (F): F produces 96-bit hash code from the output of F4 and the processing of F4 is computing as follows:

F_1 = {$(S_{13} \oplus S_{14})$}

F_i = {$(F_{i-1} \oplus S_j)$} where i = 2, 3, ….…….,12 and j = 15,……………..25

F_i = output of modulo-2 addition.

B. Statistical Testing of DHF-96

Frequency (Mono-bit) Test: By frequency mono-bit statistical testing we are verifying the randomness of DHF-96, for good randomness the probability of 0 and 1 should be near 50%. The frequency mono-bit test of DHF-96 and its experimental results executed by Python programming language are shown in Table 1.

TABLE I. STATISTICAL TESTING OF DHF 96 BY FREQUENCY (MONO-BIT) TEST

Length of Input Message in Bits	**Hash Code of DHF-96 in Hexadecimal Digit**	**Probability of 1's in %**	**Probability of 0's in %**
32	d907bd116f7dc04ec4c149e2	48.958	51.04
2000	7277b8a97ab29f116a274366	52.08	47.91
1288	e6d272dd9830bc6673f7ebd0	56.25	43.75
480	20ff52ecf1ce830754469dec	51.04	48.96
336	485e3cd83e58f2fd2dac1f0b	53.125	46.875
128	b1e6a2e003cf8712bc74f203	46.875	53.125
72	355a7370cba40d7ed1c75ca3	52.08	47.92
48	dc88e0996c33d47ce51829ec	47.91	52.08
16	1124889d7720584ff54803ff	45.83	54.16
8	961c436c08d8d761f9021fd1	45.83	54.16
8	2cd3ab20e04ba9a3a4f5af15	48.95	51.041

The test focuses mostly on the ones-to-zero ratio throughout the entire sequence. Using a random sequence as an example, this test determines whether a given sequence contains approximately an equal number of ones and zeros. The test measures how closely the proportion of one is to one-half. To take any further tests in the future, you must pass this one [32]. Seeks to find the relationship between the zeros and ones in a binary sequence of a given length. The ratio of zeros to ones is virtually identical in a fully random binary sequence. The test determines how close the unit is to 0.5 and the "random" sequence generators, that is, with a high probability to determine whether the created sequence is statistically secure [33]

Panamerican Mathematical Journal
ISSN: 1064-9735
Vol 35 No. 1s (2025)

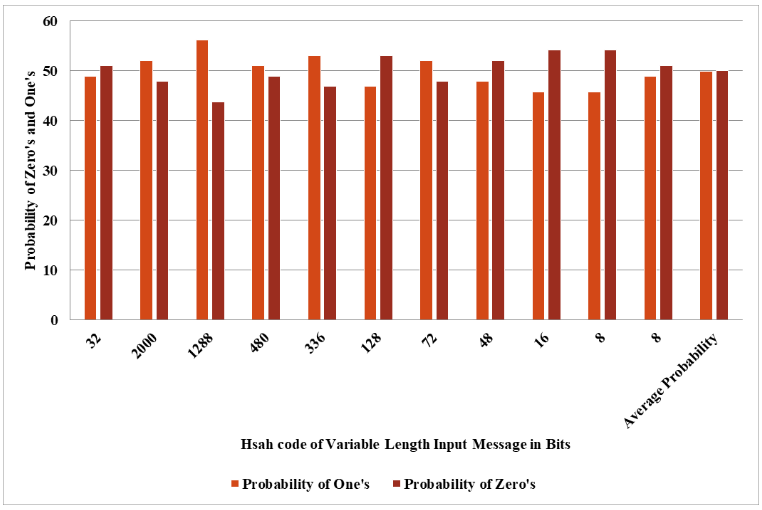

Fig. 2. Frequency (Mono-Bit) Statistical Testing of DHF-96

Sensitivity Analysis Test: A little change in the input message will result in a significant change in the hash value, as shown by the sensitivity of the hash function to modifications in the input message while comparing the effects of seven variations [32]. For instance, while computing a slight change in the input message for three structures, we identify the change bits by comparing the hash code to the initial input message for the structures.

Structure 1

Input message I1: hash

Small change in I1: “hash*”.

Small change in I1: “hAsh”.

Small change in I1: “Hash”

Small change in I1: “7hash”

Small change in I1: “haSh”

Small change in I1: “hasH”

Structure 2

Input message I1: ibtu

Small change in I1: “ibtu*”.

Small change in I1: “iBtu”.

Small change in I1: “Ibtu”

Small change in I1: “7ibtu”

Small change in I1: “ibTu”

Small change in I1: “ibtU”

Structure 3

Input message I1: jcuj

Small change in I1: “jcuj*”.

Small change in I1: “jCuj”.

Small change in I1: “Jcuj”

Small change in I1: “7jcuj”

Small change in I1: “jcUj”

Small change in I1: “jcuJ”

The resulting hash values are listed in hexadecimal format for all cases, followed by the number of changed bits compared with the hash value for Input message I1. The experimental comparative result of statistical testing of the frequency (mono-bit) test is shown in Table 2 and sensitivity tests are shown in Table 3.

TABLE II. EXPERIMENTAL RESULTS OF DHF 96 FOR STATISTICAL TESTING OF SENSITIVITY

Small change in Input Message	**Hash Code in Hexadecimal Digit**	**Total No. of Change Bits in Hash Code**
I_1=hash	eefa0dd2d4299312e1baa80d	with input message I_1
hash*	b058b097aedc86f935ed00a2	55
hAsh	2a40c23a98c1006c23e75fbd	53
Hash	4437c1deb432fdd09a095858	48
7hash	5380284a6cba5faf3d4a2f24	51
haSh	6e0d42caf08024a763ae5f0b	45
hasH	a2bcaa9945eb03ef596c040a	46
I_1=ibtu	d6660ef688393b0b61d1655	with input message I_1
ibtu*	53c4ddaa1276865b624abefa	55
iBtu	386cfc62c7c2ce52797d85b1	47
Ibtu	b7a988a38010ed72ebeaa691	53
7ibtu	c184d48886714c8fabfe76ea	59
ibTu	88d72deb442210619458f3d2	39
ibtU	4304872c7151036b4a8b2b42	51
I_1=jcuj	c0bb7c9bf3baf18889fba7ce	with input message I_1
jcuj*	9e19c1de894fe4635dac0f61	55
jCuj	64a1e2321c73247a46bd706a	54
Jcuj	6a76b09793a19f4af248579b	48
7jcuj	6b6adc17f875c71ddc826c85	49
jcUj	404c3383d713463d0bef50c8	45
jcuJ	8cfddbd062786175312d0bc9	46

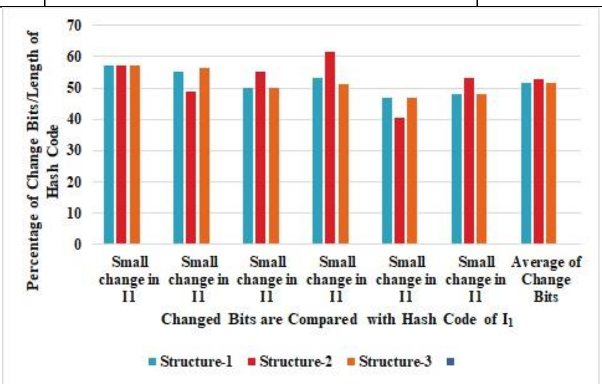

Fig. 3. Statistical Sensitivity Testing of DHF96

TABLE III. Experimental Results of DHF96 for Statistical Sensitivity Testing of Exiting Hash

Exiting Hash Algorithm	Small change in Input Message	Hash Code in Hexadecimal Digit	Total No. of Change Bits in Hash Code
SHA3_224	I_1=hash	cb00e2de2cdd8e1 295163b190b23c2ada469ca99e35df18c65354c50	with input message I_1
SHA3_224	hash*	023256e35488f460f9152e2863a3f81ef47e61bca05165 09eb621ed8	97
SHA3_224	hAsh	fba46e62783add2a682276978a5cc68817cbed0ada05 8a27c733f0f3	108
SHA3_224	Hash	646e9151b2d2997d72c67db133caf6bea04ac01f11609 61a0a92ba19	117
SHA3_224	7hash	62a23173c74cb33d590403779c970d8bfc681aca4a3d d5bf9629be84	109
SHA3_224	haSh	4c7bbac6810e8ab5ce9077bbb1c5f8ccbedbd6a26e24f 601c6869b22	112
SHA3_224	hasH	3c6b884d583155d83a4717a1d1d67820d18a017848db 1987669be888	123
SHA-256	I_1=hash	d04b98f48e8f8bcc15c6ae5ac050801cd6dcfd428fb5f 9e65c4e16e7807340fa	with input message I_1
SHA-256	hash*	e6bdb5333f3c2c007ae00feb6cd5520497ab0804763e1 d5f8e459ee947ef55d4	130
SHA-256	hAsh	cdc78e70f366eaf14276f7f37ab603b5f6fd8967c42d7 13bde55975a363d4fe3	119
SHA-256	Hash	a91069147f9bd9245cdacaef8ead4c3578ed44f179d7e b6bd4690e62ba4658f2	119
SHA-256	7hash	40233445a7b9c8ca813c89e0e5520f3483940325035fb 62ba85400ba63e1911f	123
SHA-256	haSh	b80b240f2cf7eef8659cb3e41e56e64086b0d57ae5bf55 8ce719fae4d85a0370	118
SHA-256	hasH	0eeabbb52bba2036ba295d51256fefaea2ee3a8b1af2c 553d93e99d57d84dc5a	143
SHA-512	I_1=hash	30163935c002fc4e1200906c3d30a9c4956b4af9f6dca ef1eb4b1fcb8fba69e7a7acdc491ea5b1f2864ea8c01b 01580ef09defc3b11b3f183cb21d236f7f1a6b	with input message I_1
SHA-512	hash*	6ace89974dc8dc306dd1c390aa95db1c6fbde49f4225 251922aded0044a0d76de8dd7b24e293b3551060a455 2a156674d04f992f47d7af2da1538884823950d1	271
SHA-512	hAsh	c7a70b01837d98e5d04cd8ab73a2c4c6aac9db940df4 ad65ce05afa331543a41761522c210f464a86635e9a97 633839be550d774af4f827844b881618de0258d	258
SHA-512	Hash	4efc2666ba48eae4a5b2b6db4b46ceebcff2e3070c6d1 430fbb94173e1d15e8ea7124c1e8f7b499e86503c9c1a 0ee9d15dc5c39664ef81b7da3567f1ef6ce8b8	272
SHA-512	7hash	cc854aafa7f79f165ebbe433457f62b140915bf84e356e f89bf432dda03aca622259608ff6a1d0ca9bc09ccb857	249

		F3F8E9DC7421A63CB9B0D6E385DEAE8BA7E5F	
SHA-512	HASH	76D295F3725D5D52C7B8A4364EE60AA5E45FE23EB5A0C 31523CA307FF24AF7EF6FF2CF2DF19F823E6AEBA2B6FA1 771974C1CBAA4DCC895A3356EDD0F3D11537C	251
SHA-512	HASH	52E56FD600156023636249C8F832FC4938C9E551E87E96 F3AE19DB8913C0B0504DA9A6081CDB82AC858471B2FA 6A055813A9E83B6EC1B28A32EEB72383503F1B	245

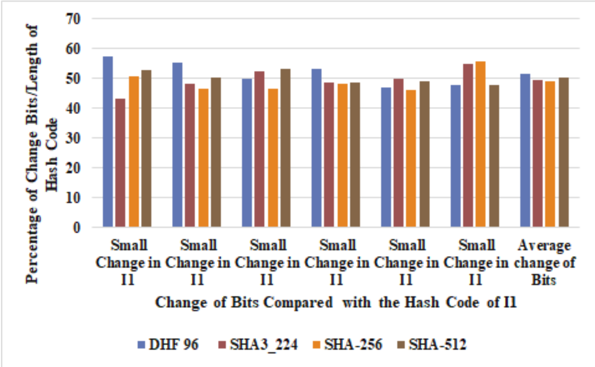

Fig. 4. Comparative Statistical Sensitivity Testing of Hash Algorithms

To verify the security requirements of a hash function it passes through three security parameters that are first preimage resistance, second preimage resistance, and collision resistance.

First Preimage: It is impossible to find any other preimage or input message of hash code, for example, any input message D and M,

hash code (D) = D

It is impossible to find

hash code (D) = M

Second Preimage resistance: It is impossible to find any two messages (D&M) that have

hash code (D) = hash code (M)

Collision Resistance: hash code (D) ≠ hash code (M)

It means that all hash codes of the above-executed example are unique and they satisfy all requirements of hash code [15].

For a map of the n-bit size of the hash code, the length of the input message will be < 2n to prevent the preimage and second preimage attacks. The brute-force attack is to pick values of x at random and try each value until a collision occurs. For an n-bit hash value, the level of effort is proportional to 2n and it tries, on average, 2n-1 values of x to find one that generates a given hash value h [30].

For a collision-resistant attack, the length of the input message can't exceed 2n/2. If we take the random variables in the range 0 through M - 1, then the probability that a repeated element is encountered exceeds 0.5 after √M choices have been made. Thus, for an n-bit hash value, if we pick a block of input message at random, we can expect to find two data blocks. If collision resistance is required then the value 2n/2 determines the strength of the hash code against brute-force attacks with an identical hash value within √(2m) = √(2m/2 attempts [30].

IV. RESULTS AND DISCUSSION

The results of the Innovative decentralized hash function algorithm are executed by Python 3.9.5 open source programming language. Comparative experimental results for statistical frequency (mono-bit) testing of DHF96 are producing good randomness because it has an average probability of one and zero is near 50% shown in Fig. 5.

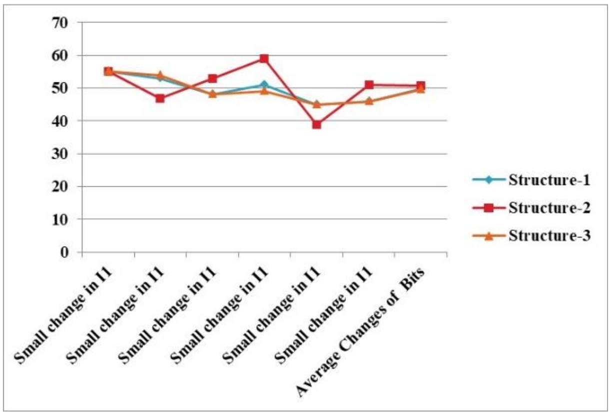

Fig. 5. ComparativeFrequency Testing of DHF-96 with Three Structure

The statistical sensitivity testing of DHF96 and comparative statistical sensitivity testing with existing hash algorithms is shown in Fig. 6.

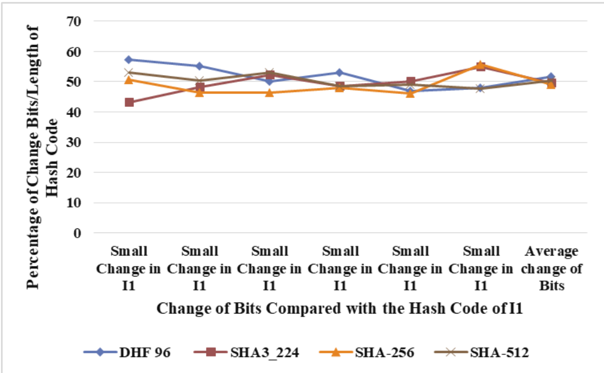

Fig. 6. Coparative sensstivity Testing of DHF-96 with Existing Hash Algorithms

The experimental results of DHF96 for frequency (mono-bit) testing are shown in Table 1 and sensitivity testing is shown in Table 2. Comparative experimental results of exiting algorithms with DHF96 for sensitivity testing are shown in Table 3.

V. CONCLUSION

This research introduced an Innovative secured hash algorithm which is mapped hash code by decreolization of hash function processed. The new design of the secured hash algorithm is statistically secured because it satisfies the statistical test and verifies all security requirements of the hash code. The main advantage of DHF96 is secured because its function processing is decentralized and the disadvantage of this algorithm is that it uses limited logical and arithmetic operations during hash function processing. It generates very fast hash code because it uses only 54 steps processed by all decentralized hash functions.

References

[1] A. A. Karcioglu and H. Bulut, "q-frame hash comparison based exact string matching algorithms for DNA sequences," Concurrency and Computation: Practice and Experience, vol. 34, no. 9, p.e6505, 2022, doi: 10.1002/cpe.6505.

[2] X. Zheng, X. Hu, J. Zhang, J. Yang, S. Cai, and X. Xiong, “An Efficient and Low-Power Design of the SM3 Hash Algorithm for IoT,” Electronics, vol. 8, no. 9, Art. no. 9, Sep. 2019, doi: 10.3390/electronics8091033.
[3] A. A. Yavuz and M. O. Ozmen, “Ultra Lightweight Multiple-time Digital Signature for the Internet of Things Devices,” IEEE Transactions on Services Computing, pp. 1–1, 2019, doi: 10.1109/TSC.2019.2928303.
[4] P. Santini, M. Baldi, and F. Chiaraluce, “Cryptanalysis of a One-Time Code-Based Digital Signature Scheme,” in 2019 IEEE International Symposium on Information Theory (ISIT), Jul. 2019, pp. 2594–2598. doi: 10.1109/ISIT.2019.8849244.
[5] A. Mohammed Ali and A. Kadhim Farhan, “A Novel Improvement With an Effective Expansion to Enhance the MD5 Hash Function for Verification of a Secure E-Document,” IEEE Access, vol. 8, pp. 80290–80304, 2020, doi: 10.1109/ACCESS.2020.2989050.
[6] M. Samiullah et al., “An Image Encryption Scheme Based on DNA Computing and Multiple Chaotic Systems,” IEEE Access, vol. 8, pp. 25650–25663, 2020, doi: 10.1109/ACCESS.2020.2970981.
[7] L. Singh, A. K. Singh, and P. K. Singh, “Secure data hiding techniques: a survey,” Multimed Tools Appl, vol. 79, no. 23, pp. 15901–15921, Jun. 2020, doi: 10.1007/s11042-018-6407-5.
[8] F. E. De Guzman, B. D. Gerardo, and R. P. Medina, “Implementation of Enhanced Secure Hash Algorithm Towards a Secured Web Portal,” in 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Feb. 2019, pp. 189–192. doi: 10.1109/CCOMS.2019.8821763.
[9] J. S. Teh, M. Alawida, and J. J. Ho, “Unkeyed hash function based on chaotic sponge construction and fixed-point arithmetic,” Nonlinear Dyn, vol. 100, no. 1, pp. 713–729, Mar. 2020, doi: 10.1007/s11071-020-05504-x.
[10] S. Z. Dadaneh, S. Boluki, M. Yin, M. Zhou, and X. Qian, “Pairwise Supervised Hashing with Bernoulli Variational Auto-Encoder and Self-Control Gradient Estimator,” in Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR, Aug. 2020, pp. 540–549. Accessed: Dec. 17, 2021. [Online]. Available: https://proceedings.mlr.press/v124/zamani-dadaneh20a.html
[11] M. Xu, G. Xu, H. Xu, J. Zhou, and S. Li, “A decentralized lightweight authentication protocol under blockchain,” Concurrency and Computation: Practice and Experience, vol. 34, no. 13, p. e6920, 2022, doi: 10.1002/cpe.6920.
[12] H. Cheng, D. Dinu, and J. Großschädl, “Efficient Implementation of the SHA-512 Hash Function for 8-Bit AVR Microcontrollers,” in Innovative Security Solutions for Information Technology and Communications, J.-L. Lanet and C. Toma, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 273–287. doi: 10.1007/978-3-030-12942-2_21.
[13] L. Han, P. Li, X. Bai, C. Grecos, X. Zhang, and P. Ren, “Cohesion Intensive Deep Hashing for Remote Sensing Image Retrieval,” Remote Sensing, vol. 12, no. 1, Art. no. 1, Jan. 2020, doi: 10.3390/rs12010101.
[14] Z. Chen, Y. Hua, B. Ding, and P. Zuo, “Lock-free Concurrent Level Hashing for Persistent Memory,” presented at the 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020, pp. 799–812. Accessed: Dec. 17, 2021. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/chen
[15] S. William, Cryptography and Network Security - Principles and Practice | Seventh Edition | By Pearson, Seventh edition. Uttar Pradesh, India: Pearson Education, 2017.
[16] H. Al-Balasmeh, M. Singh, and R. Singh, “Framework of data privacy preservation and location obfuscation in vehicular cloud networks,” Concurrency and Computation: Practice and Experience, vol. 34, no. 5, p. e6682, 2022, doi: 10.1002/cpe.6682.
[17] K. Singh and N. Singh, “Multilevel authentication protocol for enabling secure communication in Internet of Things,” Concurrency and Computation: Practice and Experience, vol. 34, no. 3, p. e6578, 2022, doi: 10.1002/cpe.6578.
[18] B. Madhuravani and D. S. R. Murthy, “Cryptographic hash functions: SHA family,” International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 2, no. 4, pp. 326–329, 2013.
[19] [19] M. Gafsi, R. Amdouni, M. A. Hajjaji, J. Malek, and A. Mtibaa, “Improved chaos-RSA-based hybrid cryptosystem for image encryption and authentication,” Concurrency and Computation: Practice and Experience, vol. 34, no. 23, p. e7187, 2022, doi: 10.1002/cpe.7187.
[20] V. Melnyk and A. Kit, “Basic operations of modern hashing algorithms,” undefined, 2013, Accessed: Dec. 17, 2021. [Online]. Available: https://www.semanticscholar.org/paper/Basic-operations-of-modern-hashing-algorithms-Melnyk-Kit/f423b1e8a5365b713e14a2d7d95b8c2a94c9aebf
[21] M. Dabbagh, K.-K. Choo, A. Beheshti, M. Tahir, and N. Safa, “A survey of empirical performance evaluation of permissioned blockchain platforms: Challenges and opportunities,” Computers & Security, vol. 100, Jan. 2021, doi: 10.1016/j.cose.2020.102078.
[22] J. Polpong and P. Wuttidittachotti, “Authentication and password storing improvement using SXR algorithm with a hash function,” IJECE, vol. 10, no. 6, p. 6582, Dec. 2020, doi: 10.11591/ijece.v10i6.pp6582-6591.
[23] K. D. Doan, S. Manchanda, S. Badirli, and C. K. Reddy, “Image Hashing by Minimizing Discrete Component-wise Wasserstein Distance,” arXiv:2003.00134 [cs], May 2020, Accessed: Dec. 17, 2021. [Online]. Available: http://arxiv.org/abs/2003.00134

[24] K. S. Meel and S. Akshay, “Sparse Hashing for Scalable Approximate Model Counting: Theory and Practice,” in Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, in LICS ’20. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 728–741. doi: 10.1145/3373718.3394809.
[25] A. Visconti and F. Gorla, “Exploiting an HMAC-SHA-1 Optimization to Speed up PBKDF2,” IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 4, pp. 775–781, Jul. 2020, doi: 10.1109/TDSC.2018.2878697.
[26] Y. Li and X. Li, “Chaotic hash function based on circular shifts with variable parameters,” Chaos, Solitons & Fractals, vol. 91, pp. 639–648, Oct. 2016, doi: 10.1016/j.chaos.2016.08.014.
[27] B. Pinkas, T. Schneider, and M. Zohner, “Scalable Private Set Intersection Based on OT Extension,” ACM Trans. Priv. Secur., vol. 21, no. 2, pp. 1–35, Feb. 2018, doi: 10.1145/3154794.
[28] B. R. Ambedkar, P. K. Bharti, and A. Husain, “Enhancing the Performance of Hash Function Using Autonomous Initial Value Proposed Secure Hash Algorithm 256,” in 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT), Apr. 2022, pp. 560–565. doi: 10.1109/CSNT54456.2022.9787561.
[29] B. R. Ambedkar, P. K. Bharti, and A. Husain, “Design and Analysis of Hash Algorithm Using Autonomous Initial Value Proposed Secure Hash Algorithm64,” in 2021 IEEE 18th India Council International Conference (INDICON), Dec. 2021, pp. 1–6. doi: 10.1109/INDICON52576.2021.9691602.
[30] W. Liang, S. Xie, J. Long, K.-C. Li, D. Zhang, and K. Li, “A double PUF-based RFID identity authentication protocol in service-centric internet of things environments,” Information Sciences, vol. 503, pp. 129–147, Nov. 2019, doi: 10.1016/j.ins.2019.06.047.
[31] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems,” presented at the 30th {USENIX} Security Symposium ({USENIX} Security 21), 2021, pp. 519–535. Accessed: Nov. 23, 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi
[32] A. Rukhin et al., “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” NIST Special Publication 800-22 (revised May 15.” 2002.
[33] A. Kuznetsov, M. Lutsenko, K. Kuznetsova, O. Martyniuk, V. Babenko, and I. Perevozova, “Statistical Testing of Blockchain Hash Algorithms,” p. 13..