American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)
ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

http://asrjetsjournal.org/

Enhanced Parallel Hash Function Algorithm Based on 3C Construction (EPHFA-3C)

Roayat Ismail Abdelfatah[a], Esraa Abdelkhalek B.[b*], M. E. Nasr[c]

[a,b,c]*Electronics and Electrical Communications Engineering Dept., Faculty of Engineering, Tanta University, Tanta, Egypt*

[a]*Email: roayatismail2016@gmail.com*

[b]*Email: israa.baqa@f-eng.tanta.edu.eg*

[c]*Email: menasr2001@yahoo.com*

Abstract

The hash function is a function that can convert data from variable size to fixed-size data that can be used in security of communication like, authentication, digital signature and integration. In this paper, a parallel, secure and fast hash function algorithm that is based on 3C construction is proposed. It is an enhancement for the MD construction. This enhancement makes the construction more resistant to the extension and multi-blocks attacks. The parallel structure of the algorithm improves the speed of hashing and reduces the number of operations. The simulation analysis such as hashes distribution, confusion and diffusion properties, and collision resistance are executed. Based on the results, our proposed hash algorithm is efficient, simple, and has strong security compared with some recent hash algorithms.

Keywords: Hash function; parallel hashing; 3C construction; collision resistance.

1. Introduction

As the significant growth of computer and Internet technology, multimedia communication plays an important role in many fields in our social community. Multimedia data security is becoming extremely significant in wired and wireless communications, such as file sharing, online payments, message authentication, and watermarking [1−5]. Hash algorithm is already proved to solve these problems accurately and efficiently. The hash function can be divided into two classes: 1) Unkeyed hash function: hash determines a single input parameter, message; and 2) Keyed hash function: has two different inputs, a message and a secret key [6-8].

----

* Corresponding author.

To be an efficient cryptographic algorithm, the hash function needs to achieve following properties: 1) Pre-image resistance (one-way): this means that a hash function would be difficult to reverse computationally. In other words, for any hash function H, generates a hash value D, it must be a hard process to find an input value X that hashes to D. This feature protects against the attacker who has a hash value only and trying to get the input. 2)Second pre-image resistance: this means; given an input and its corresponding hash, it would be difficult to get -another input with the same hash. In other words, for an input X and a hash function H that generates hash value H(X), it should be hard to find any different input value Y such that H(Y) = H(X). This feature of hash function protects against the attacker that has an input and its hash, and trying to substitute another value as valid value instead of the original input value. 3) Collision resistance: this means it should be difficult to find two distinct inputs of any length, which have the same hash. In other words, it is difficult to find two distinct inputs X and Y that achieve H(X) = H(Y). This feature of collision free ensures that these collisions would be difficult to detect, and makes the hash very hard for the attacker to get two different input values with the same hash. In addition, if a hash function supposes the collision-resistant property, then it supposes second pre-image resistant [9]. The conventional hash functions such as MD5 [10] and SHA-1 [11] are based on logical operations, multi-round operations, and digital algebraic operations that significantly affect the security, as attacks on these algorithms have been found in. Multi-block collision attacks (MBCA) were discovered on the Merkle-Damgard (MD)-structure that traditional hash functions MD5, SHA-0 and SHA-1 based on it [12 - 14]. SHA-2 hash algorithms family that announced in 2002 by NIST replaces the SHA-1 [١٥]. SHA-2 variants were analyzed and found to contain certain inefficiencies versus the attacks investigated in[16,17]. One of the reasons for this is that the popular structure of Merkle–Damgard was the basic framework behind the creation of these hash algorithms. The (MD) structure as shown in Fig. 1 takes the input message then divide it to N-1 fixed-sized block of m bits each, and padding the last block to m bits. In addition, the last block contains the length of the hash function input, this makes the task of the attacker more difficult. In this case, the attacker should find two different messages of equal length that have the same hash or find two messages of different lengths, which, when their length added to the message, have the same hash value. The hash algorithm uses a repeated compression function f, which deals with two inputs; input from the previous stage of n-bit called chaining variables, and m-bit block, and generates output of n-bit. The chaining variable has an initial value at the beginning of hashing which considered as portion of the algorithm and the hash value is the final value of the variable chaining [18, 19] so the hash function can be defined as:

$$CV_0 = IV = initial\ value\ of\ n - bit \quad (1)$$

$$CV_i = f(CV_{i-1}, M_{i-1}) \quad 1 \le i \le N \quad (2)$$

$$\mathrm{H}(M) = CV_N \quad (3)$$

Where a message m involving the blocks M_0, M_1,…, M_{N-1} is the input of the hash function. Note that there must be collisions for any hash function, because size of message at least similar to the block length m is converted into a hash value of length n, where m > n [19].

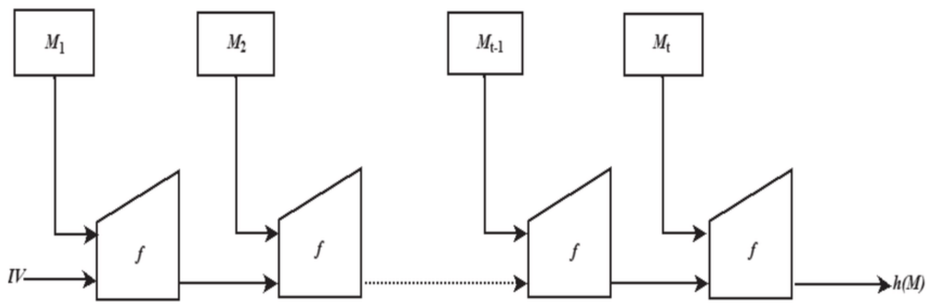

Figure 1: Merkle-Damgard iterated construction [18]

Hash function cryptographic analysis shows that the M-D hash structure is not resistant to fix point attack, multi-blocks attack, and extend attack; furthermore, there are some small impairment in compression function may cause failure in the hash algorithm. Hence, some enhanced structures such the generic 3C construction is very significant [18, 19]. The generic 3C hash function structure as appeared in Fig 2 formed of two compression functions: the function f, which repeated in the series and the function f' that iterated in the accumulation series. The function f' is a common function that can be similar to the compression function f applied in the cascade chain.

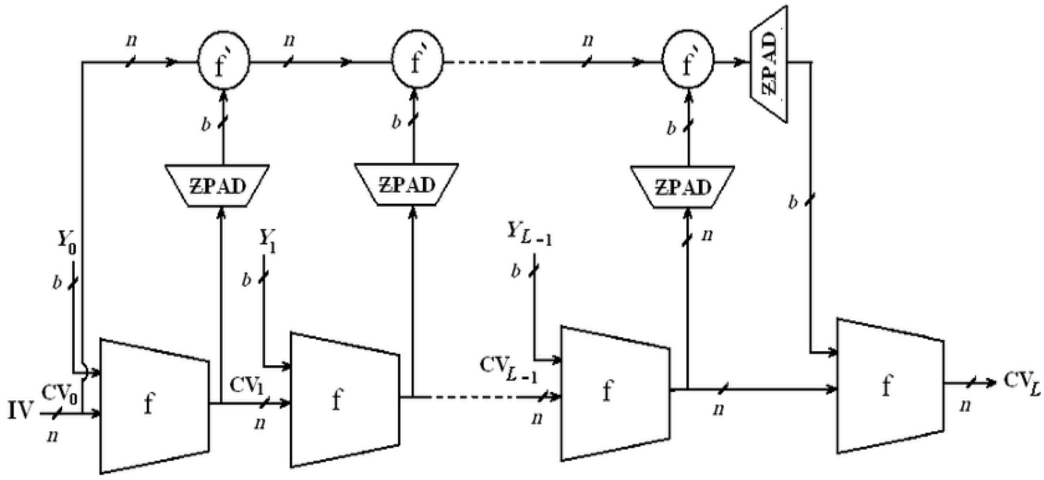

Figure 2: The generic 3C hash function construction [19]

Firstly, based on the M-D construction the input message is handled in repeated way in the compression function f. Then padding the output of this function using the usual padding method Z-PAD (attaching a bit 1 and some 0's then add the encoded binary of z-length representation) to make the length of the last block equal to the block length of m bits. Finally, the cumulative of the function f' output is also padded and entered to the external application f as input. Note that, the padding function is applied twice for the 3C structure: firstly, in the cascaded structure and secondly, on Z for the block of accumulative chain. This padding is indicated as ZPAD operation in Fig 2. The structure is called 3C construction because at least three compression function applications are needed to handle the message when it has only one block. The single block is processed by first application the, the second deal with the padded block and the last compression function application deal with the accumulative chain function block [19].

The main contribution in this paper is to enhance 3C construction from two sides:

- Information of message length is attached to the padded message so the proposed scheme is resistant to length-extension attack and meet-in-the-middle attack.
- The standard MD sequential iteration structure is changed to parallel iteration structure. In this case, the enhanced hash algorithm will have more advantage in term of speed when dealing with large files.
- New design for step function that is simple and easy to computation so the proposed algorithm has great advantage in speed.

After this introduction, the remaining of the paper is arranged as follow: Sect. II illustrates our proposed hash algorithm, Sect. III presents implementation analysis, Sect. IV covers the comparative analysis, and finally Sect. V includes the conclusion.

2. Proposed algorithm

In this section, we briefly illustrate the five phases of the proposed hash algorithm:

2.1. Message padding

Assuming that the message size is l_1, and the message block size is m=512 bits. First, padding the input message M as follow: a bit of "1" is added to the end of the message, then, attaching some of "0" after the "1" bit that makes the padding message size satisfies 448 modulo 512. Finally, l_1 is converted to binary representation and inserted at the ending of the message. The message block after padding is shown in Fig. 3.

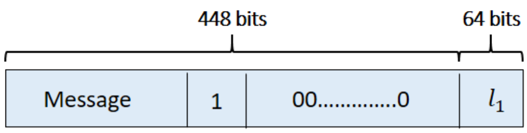

Figure 3: Message padding

2.2. Message blocking

After padding the message, the input message M is divided into N blocks each of 512-bit with $M = M_0\|M_1\|M_2\|\dots.\|M_{N-1}$. During message block partition processing, every message block is divided into sixteen 32-bit messages with $M_i = M_i^0\|M_i^1\|M_i^2\|\dots\|M_i^{15}$.

2.3. Initialization of chaining variables

We define eight chaining variables with S, T, U, V, W, X, Y, and Z. The length of each chaining variable is 32 bits. The initial value of the eight chaining variables is listed as:

S_0= 6a09e667 T_0= bb67ae85 U_0= 3c6ef372 V_0 = a54ff53a

W_0= 510e527f X_0= 9b05688c Y_0= 1f83d9ab Z_0= 5be0cd19

These are defined by having the fractional portion of square roots for the first eight primes in hexagonal representation.

2.4. Parallel iterative structure design

The basic structure of hash algorithm is the iteration structure. As illustrated in Fig. 4, the parallel iteration structure is consisted of two procedures: message preprocessing (illustrated in Sect. II-a), compression function (will be illustrated in Sec. II-e).

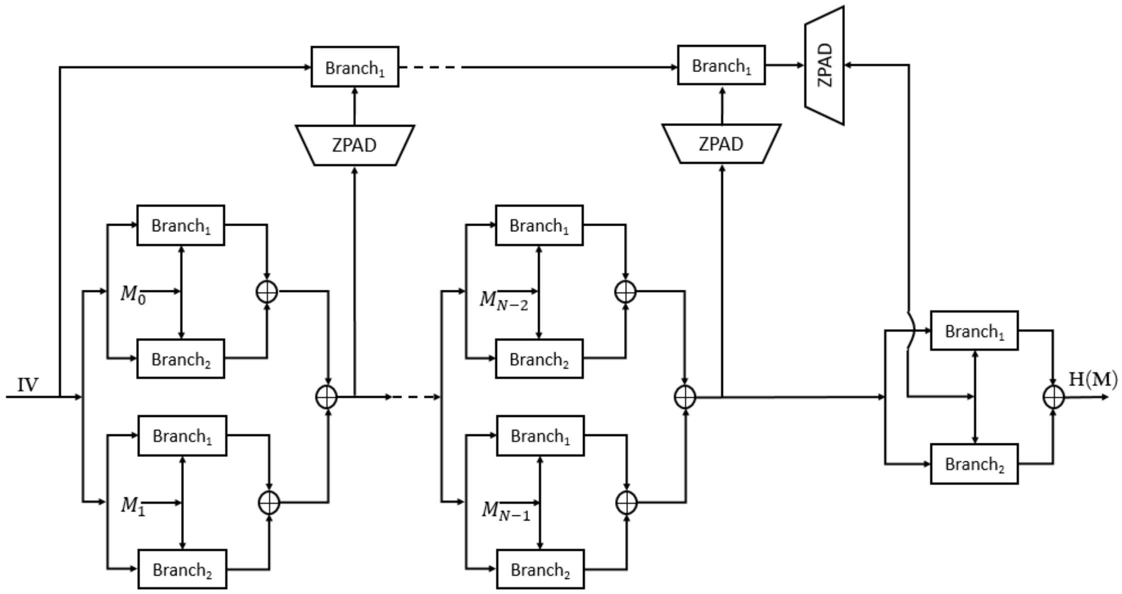

Figure 4: Parallel iterative structure

2.5. Compression Function

Compression function of our proposed algorithm is consists of three-compression functions: two of the parallel compression functions f, which iterated in the cascade series and the function f' that is iterated in the accumulative series. For the two parallel compression functions, the input of the first function is the first message block and the next message block is the input of second compression function and hence. The function f has two parallel branch function; Branch1 and Branch2 as shown in Fig. 5 therefore the attacker who attemps to fracture the function must target the two branches simultaneously. The two branches message words ordering are:

Branch1: $M_i = M_i^0 \| M_i^1 \| M_i^2 \| \ldots \| M_i^{15}$

Branch2: $M_i = M_i^{15} \| M_i^{14} \| M_i^{13} \| \ldots \| M_i^{0}$, where the order is reversed.

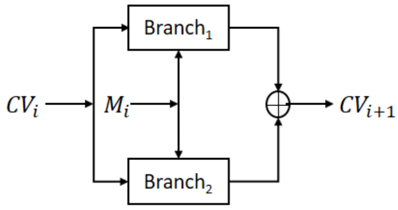

Figure 5: Compression function (f)

2.5.1. Constants

For compression function, there are sixteen constants will be defined to use. For Branch1 the constants are ordered as following:

β_0= 428a2f98 β_1= 71374491 β_2= b5c0fbcf

β_3= e9b5dba5 β_4= 3956c25b β_5= 59f111f1

β_6= 923f82a4 β_7= ab1c5ed5 β_8= d807aa98

β_9= 12835b01 β_{10}= 243185be β_{11}= 550c7dc3

β_{12}= 72be5d74 β_{13}= 80deb1fe β_{14}= 9bdc08a7

β_{15}= c19bf174

The order of constants are reversed for Branch2.

The aim of using these constants is to upset the attacker that attempt to get the best differential characteristics with high relative possibility. Therefore, we choose the constants which form the first thirty-two bits for the fractional portions of the cube roots for the first sixteen four prime numbers.

2.5.2. Step function

The compression function f is iterated four times. For k^{th} step, the input registers is divided into eight 32-bit words : S_k, T_k, U_k, V_k, W_k, X_k, Y_k, and Z_k. The inputs of $k+1$ step calculated as :

$$S_{k+1} = [[(Y_k \oplus Z_k) \oplus M_{2k+9}]^{<<<r_7}] \oplus Z_k^{<<<r_8} \quad (4)$$

$$T_{k+1} = [(S_k \oplus T_k) \oplus M_{2k}] + \beta_{4k} \quad (5)$$

$$U_{k+1} = [[(S_k \oplus T_k) \oplus M_{2k}]^{<<<r_1}] \oplus T_{4k+1}^{<<<r_2} \quad (6)$$

$$V_{k+1} = [(U_k \oplus V_k) \oplus M_{2k+1}] + \beta_{4k+1} \quad (7)$$

$$W_{k+1} = [[(U_k \oplus V_k) \oplus M_{2k+1}]^{<<<r_3}] \oplus V_k^{<<<r_4} \quad (8)$$

$$X_{k+1} = [(W_k \oplus X_k) \oplus M_{2k+8}] + \beta_{4k+2} \quad (9)$$

$$Y_{k+1} = [[(W_k \oplus X_k) \oplus M_{2k+8}]^{<<<r_5}] \oplus X_k^{<<<r_6} \quad (10)$$

$$Z_{k+1} = [(Y_k \oplus Z_k) \oplus M_{2k+9}] + \beta_{4k+3} \quad (11)$$

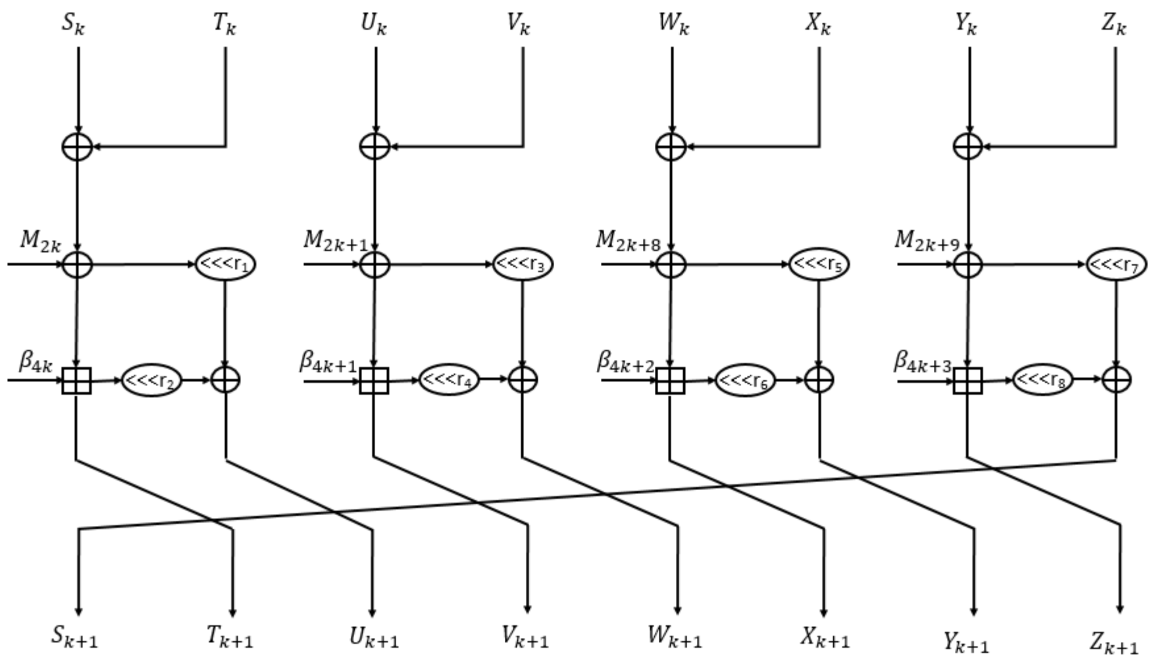

Figure 6: Step function structure

In Fig. 6,($<<< x$) donates x bits left shift rotation, $\oplus$ is XOR logic function, and (+) is addition mod 2^{32}.

2.5.3. Shift rotation variables

For $k = 0$, the initial values are defined as:

$r_1 = 5$ $r_2 = 9$ $r_3 = 3$ $r_4 = 11$

$r_5 = 8$ $r_6 = 13$ $r_7 = 7$ $r_8 = 10.$

For $k = 1, 2, 3$ the shift rotations values calculated as:

$r_1 = S_k + T_k + U_k$ $r_2 = T_k + U_k + V_k$ $r_3 = U_k + V_k + W_k$

$r_4 = V_k + W_k + X_k$ $r_5 = W_k + X_k + Y_k$ $r_6 = X_k + Y_k + Z_k$

$r_7 = Y_k + Z_k + S_k$ $r_8 = Z_k + S_k + T_k$.

3. Experimental analysis and simulation results

In this section, We estimate our parallel hash function in forms of hash values uniform distribution, hash value sensitivity to delicate changes in the original message, properties of confusion and diffusion, and collision tests.

3.1. Hash sensitivity

We randomly choose a text and the simulation of sensitivity is done under nine conditions:

Condition1: The original message "*A hash function is any function that can be used to map data of arbitrary size to fixed-size values. These values called digests.* ".

Condition2: The first character changed to lowercase.

Condition3: Add a number '1' before the sentence.

Condition4: Change the word 'arbitrary' to 'variable'.

Condition5: Remove dash between 'fixed-size'.

Condition6: Change the dot after the word 'values' to comma.

Condition7: Change the first letter of the word 'digests' to capital letter.

Condition8: Change the last character after the word 'digests' from '.' to '!'.

Condition9: Add a whitespace before the last character '.'.

The corresponding 256-bit hash values in hex format are the following:

Condition1: 4B1633D0FF7BB26695ADA088D65F5E9658A37883AF16A746B41E5FBED6377001

Condition2: EF5BD59DFA138F9A009D8A83A17F2F466D8BAA5EA0825E01CD60D828903D7AB2

Condition3: E43DC0208B12E4F7762E242DC92523A9F3A70934B3303FDEA685CD8C656A3A03

Condition4: 8E328BE1A2BD0D8EA43445260B1F7347C7F9406BBEA7FD80D88CD279AA8CEC74

Condition5: 55D91FEF938D0BF8B21E6CB28E60473AA1A3B7AA4C106F8BC9427E0969CD58C5

Condition6: A5A664AA74A739FA693DDC239E337D9C7ADD615D8BE127497E8197F34C2D9AC9

Condition7: 353232064471AED3130270F27CB7CF68C1F871D8948514781633F3DF020E5F71

Condition8: 3D53F6BAF23ABB616B89AF7157819A642CFF8E21666C607F15779F0A3CFB3359

Condition9: DE1EE0887813BA48FCC324528DCF3D8D6DC0D3263CF0E48E4468204F2EF2452A

The corresponding hash values graphic in binary format is illustrated in Fig. 7:

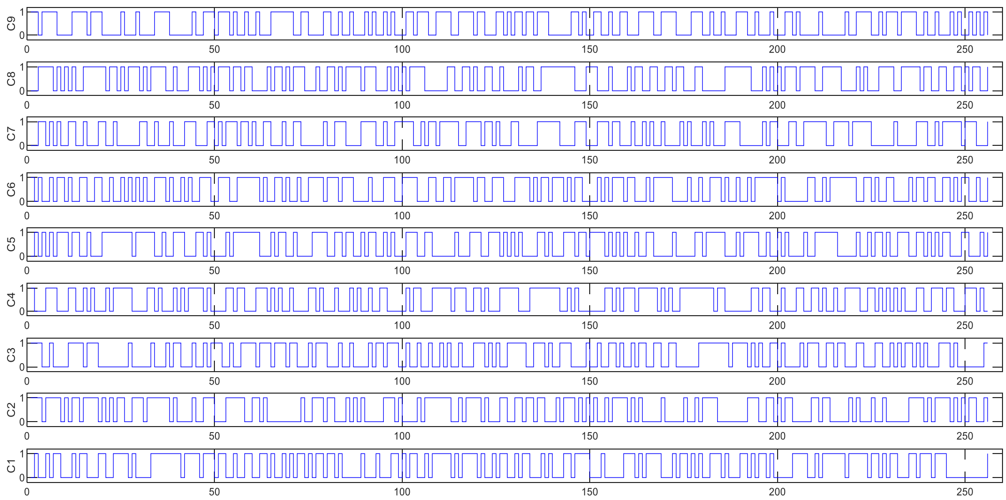

Figure 7: Hash values in binary format under nine condition

The hex codes of the hashes and their binary representations graphics show that even a single bit of modification in the original message results in disastrous variation in the hash code. Based on these graphics, our proposed hash satisfies the one-way cryptographic hash feature.

3.2. Confusion and diffusion statistical analysis

Confusion and diffusion are identified by Claude Shannon as the main features of cipher security that are considered as the main design requirements of any cryptographic hash algorithm [20]. The feature of confusion indicates to the relationship between the message and its hash value should be unpredictable and complex; whereas diffusion indicates to the hash value should be extremely based on the message. The following test is implemented in order to catch the qualitative features of the diffusion and confusion for our proposed hash: First, select a message randomly and its corresponding hash value is produced; then change a bit from the message randomly and produce the new hash value. Finally, compare the two hash values and count the number of various bits in the two hash values that placed at the same location. Generally, statistical analysis is based on the following equations:

Number of average bit change:

$$B_c = \frac{1}{N} \sum_{i=1}^{N} B_i \quad (12)$$

Standard deviation of bit change:

$$\Delta B_c = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (B_i - B_c)^2} \quad (13)$$

Percentage of bit change:

$$P = \frac{B_c}{N} \times 100\% \quad (14)$$

Standard deviation of P:

$$\Delta P = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \left(\frac{B_i}{n} - P\right)^2} \times 100\% \quad (15)$$

The test is applied on a sample size of 1024 and 2048 bits and the results are tabulated in Table 1.

Table 1: Statistical results of changed bits for n=1024 and 2048 bits

n	n = 1024	n = 2048	Mean
B_{min}	122	118	120
B_{max}	136	144	140
$\overline{B_c}$	128.25	134.5	131.375
P (%)	50.0977 %	52.5391 %	51.3184 %
ΔB_c	5.06388	7.91021	6.847
$\Delta P(\%)$	1.97808 %	3.08993 %	2.534 %

The test is performed on our proposed scheme N times, where N = 256, 512, 1024 and 2048. The test messages are 2048 bits in the length and the result is listed in Table 2 for 256-bit hashes.

Table 2: Statistical results of changed bits for N=256, 512, 1024 and 2048 times

N	N = 256	N = 512	N = 1024	N = 2048	Mean
B_{min}	110	106	106	106	107
B_{max}	148	148	150	150	149
$\overline{B_c}$	128.203	128.129	127.975	127.899	128.0515
P (%)	50.0793 %	50.0504 %	49.9901 %	49.9607 %	50.0201 %
ΔB	7.90469	7.8881	8.02607	7.9303	7.8235
$\Delta P(\%)$	3.08777 %	3.08129 %	3.13518 %	3.09778 %	3.0407 %

As shown in Table 2, the mean changed bit number for the proposed hash algorithm is $\overline{B_c} = 128.0515$ and the mean changed probability $P = 50.02\%$ these results of our scheme are around to the ideal values of 128 bits and 50%, respectively. Also, the values of standard deviation ΔB and ΔP are very small that indicates to a high capability for confusion and diffusion. Fig. 8 illustrate the behavior of distribution of changed bit number as (a) the B_i graphic indicates to its value is equally distributed, and as illustrated in Fig. 8(b), the normal distribution of B_i centered at the ideal value of 128. The proposed hash function results shows that it has close-ideal statistical characteristics in form of confusion and diffusion capability, where even one bit change from the plaintext will result in a totally distinct message digest.

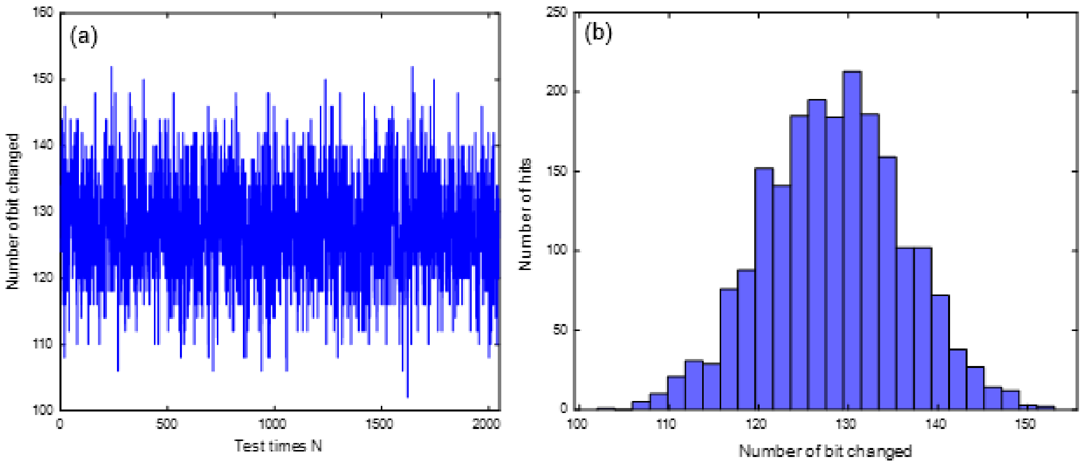

Figure 8: Changed bit number spreading: a) plot of $\boldsymbol{B_i}$ and b) histogram of $\boldsymbol{B_i}$

3.3. Collision resistance analysis

When two distinct input messages are mapped to precisely same hash value a collision occurs, so the aim of the collision attack is to seek to find two distinct messages that result in collision. One of the most important features of efficient encryption algorithms is collision resistance. In order to prove the highly collision resistant of our proposed scheme and its produced hashes, a test is performed as follow: First, produce a message randomly along with its corresponding hash value and saved in ASCII format. Then, choose a bit from the original message and replace it so a new message is generated with a small different. Next, hash the new generated message and store the corresponding digest in ASCII format. Finally, compare the two message hashes and count the number of ASCII characters that located in the same places and have the same values (number of hits). In this paper, the test is performed N = 2048 times, and plotted the distribution number of hits in Fig. 9. As shown in Fig. 9 and Table 3, no hit happens in 1820 tests, one hit occurs in 214 tests, and 12 tests hit twice. As the maximum number of hits is only 2, the collision of the proposed algorithm is very small.

Table 3: Number of hits in Collision test for N = 2048

Number of equal characters	0	1	2	3	4	5
The proposed algorithm	1820	214	12	0	0	0

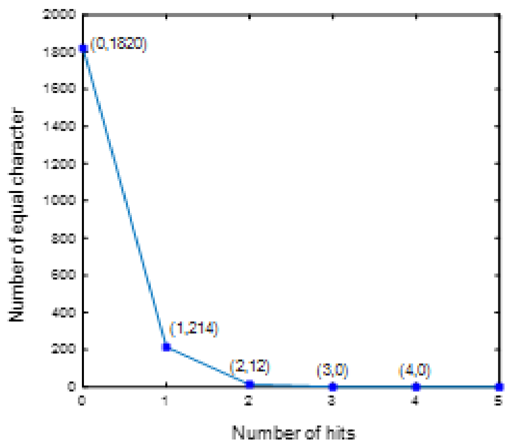

Figure 9: Distribution of number of hits for N = 2048

In addition, the absolute difference between two distinct hashes is also defined as:

$$d = \sum_{i=1}^{N} |t(e_i) - t(e_i')| \quad (16)$$

Where e_i indicate to the i^{th} character of ASCII of the original hash value while e_i' donate the i^{th} ASCII character of the modified hash value, and the function $t(\,)$ maps the results into the corresponding decimal values. The collision test is performed N = 2048 times, and the results is listed in Table 4. The values of maximum, minimum, and mean of the absolute difference d of two different hash values are 4032, 1726, and 2729.9, respectively and the mean/character of absolute difference d of two hash values for our scheme is 85.3094 that is very near to the ideal theoretical mean/character value 85.3333.

Table 4: Absolute differences d between two hash values

Maximum	Minimum	Mean	Mean / Character
4032	1726	2729.9	85.31

4. Comparative Analysis

4.1. Statistical Performance Comparison

Comparison between the proposed algorithm and some relevant and important hash algorithms is done based on security evaluation. The statistical results of all chosen hash algorithms are reported in Table 5 a). As listed in Table 5 a), the values of $\bar{B}$ and P of the proposed scheme are near-ideal, and standard deviation of ΔB is smaller than traditional hash algorithms of SHA256, Keccak-256 and other recent hash function of PLHF [18]. The proposed algorithm also has smaller standard deviation of ΔP than traditional algorithms of SHA1, SHA2, Keccak-256, and other recent algorithm of PLHF [21]. The proposed algorithm also mapped to variable output length of 128-bit and the statistical performance is compared with some recent hash algorithm such as Li-Ge [22], and Je and his colleagues [23]. The results are shown in Table 5 b) and it is very close to the ideal result.

Table 5: Comparison on statistical performance a) Hash-256 b) Hash-128

a) Hash-256 bit

Algorithm	Statistical performance of the algorithms				
	$\boldsymbol{B_c}$	$\boldsymbol{\bar{B}}$	$\boldsymbol{P}$ (%)	$\Delta\boldsymbol{B}$	$\Delta\boldsymbol{P}$(%)
SHA1	49–114	80.16	50.10	6.24	3.90
SHA256	71 - 193	128.61	50.24	10.83	4.23
Keccak-256	80–207	129.34	50.52	10.21	3.99
PLHF [21]	91 - 168	127.91	49.96	8.93	3.49
Proposed	106 - 150	128.05	50.02	7.82	3.04

b) Hash-128 bit

Algorithm	Statistical performance of the algorithms				
	$\boldsymbol{B_c}$	$\boldsymbol{\bar{B}}$	$\boldsymbol{P}$ (%)	$\Delta\boldsymbol{B}$	$\Delta\boldsymbol{P}$(%)
MD5	N/A	64.03	50.02	5.66	4.42
Li-Ge [22]	42 - 85	63.87	49.90	5.58	4.36
Je and his colleagues [23]	45 - 86	63.99	49.99	5.64	4.38
Proposed	46 - 82	63.82	49.88	5.67	4.43

4.2. Collision Resistance Comparison

Collision resistance test of this algorithm and other hash functions is running 2000 times, and the results are listed in Table 6 as a distance between two hash values. Also, the distances between two different hash values are calculated as $D_{hash} = \sum_{i=1}^{80}|hex(C_1) - hex(C_2)|$, where C_1, C_2 are the two-hexadecimal characters that located at the same location in two output hash values. The maximum and the minimum D_{hash} can be calculated using the same random message, and the average D_{hash} can be obtained by:

$$D_{average}(avg) = \frac{\sum_{i=1}^{2000} D_{hash_i}}{2000} \tag{17}$$

Table 6: Distances between two hash values in different schemes

Algorithm	Size of output	N_s(max)	D_{hash}(max)	D_{hash}(min)	D_{avg}(avg)
SHA1	160	2	2556	857	85.4
SHA256	256	2	4098	1365	85.375
Keccak-256	256	2	1405	1368	85.406
PLHF [21]	256	1	4096	1364	85.344
Proposed	256	2	4032	1726	85.309

As shown in Table 6, the fluctuation of PLHF is small, which means it is more stable and has stronger resistance against random collision attack than other algorithms. Its security parameter, $D_{average}(avg)$, is approximately equal to the optimal value 85.33 which is superior to other parallel schemes such SHA1, SHA256, Keccak-256 and PLHF.

4.3. Speed Analysis Comparison

Speed analysis is conducted by comparing the number of operations between this scheme and traditional hash function SHA-256 for one block of 512-bits. The comparison of total number of operations is listed in Table 7.

Table 7: Number of operation comparison

Operation	SHA-256	PLHF [21]	Proposed
Addition (+)	600	560	80
Bitwise Operation ($\oplus, \wedge, \vee$)	1024	0	243
Multiplication	0	320	0
Shift operation (<<, >>, <<< , >>>)	672	280	160
Total	2296	1160	483

As illustrate in Table 7, the number of operation of SHA-256 [19] is approximately five times of the number of operation of the proposed hash also, the proposed hash has less than half number of operation of PLHF[21]. So, These result show how the proposed hash is much fast.

5. Conclusion

This paper propose an efficient 256-bit cryptographic hash function algorithm that based on enhanced generic 3C hash function structure. The algorithm is achieved by adjusting the M-D iterative structure to be more robust against the extension attacks and differential multi-blocks attacks. Further, parallelization is implemented in this paper to reduce the number of operations and hence improve the speed of hashing algorithm. Based on experiments and security analysis, the proposed hash function achieves the security requirements and has other advantages such as high resistance to collision attacks and great statistical diffusion and confusion performance compared with conventional schemes.

References

[1] National Bureau of Standards (1999) Data encryption Li, Yantao, and Guangfu Ge. "Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security." Multimedia Tools and Applications 78.13 (2019), pp 17973-17994.

[2] Deng S., Xiao D., Li, Y., & Peng, W. "A novel combined cryptographic and hash algorithm based on chaotic control character." Communications in Nonlinear Science and Numerical Simulation 14.11 (2009), pp 3889-3900.

[3] Deng, Shaojiang, et al. "Analysis and improvement of a hash-based image encryption algorithm." Communications in Nonlinear Science and Numerical Simulation 16.8 (2011), pp 3269-3278.

[4] Schneider, Marc, and Shih-Fu Chang. "A robust content based digital signature for image authentication." Proceedings of third IEEE International Conference on Image Processing. Vol. 3. IEEE (1996), pp 227–230.

[5] Mıhçak, M. Kıvanç, Ramarathnam Venkatesan, and Tie Liu. "Watermarking via optimization algorithms for quantizing randomized semi-global image statistics." Multimedia Systems 11.2 (2005), pp 185-200.

[6] Li, Yantao. "Collision analysis and improvement of a hash function based on chaotic tent map." Optik 127.10 (2016), pp 4484-4489.

[7] Shi-Hong, Wang, and Shan Peng-Yang. "Security analysis of a one-way hash function based on spatiotemporal chaos." Chinese Physics B 20.9 (2011), pp 090504–090507.

[8] Wang, Shihong, Da Li, and Hu Zhou. "Collision analysis of a chaos-based hash function with both modification detection and localization capability." Communications in Nonlinear Science and Numerical Simulation 17.2 (2012), pp 780-784.

[9] Ahmad, Musheer, et al. "A simple secure hash function scheme using multiple chaotic maps." 3D Research 8.2 (2017), pp 13.

[10] Rivest, Ron. "The md5 message-digest algorithm (rfc 1321)." Internet Activities Board (1992).

[11] Standard, Secure Hash. "FIPS Pub 180-1." National Institute of Standards and Technology 17 (1995): 15.

[12] Mendel, Florian, Tomislav Nad, and Martin Schläffer. "Improving local collisions: new attacks on reduced SHA-256." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg (2013), pp 262-278.

[13] Stevens, Marc. "New collision attacks on SHA-1 based on optimal joint local-collision analysis." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg (2013), pp 245-261.

[14] Wang, Xiaoyun, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1." Annual international cryptology conference. Springer, Berlin, Heidelberg, (2005), pp 17-36.

[15] FIPS, NIST. 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS). publication 180-2. Technical report, Department of Commerce, 2002.

[16] Sanadhya, Somitra Kumar, and Palash Sarkar. "New collision attacks against up to 24-step SHA-2." International conference on cryptology in India. Springer, Berlin, Heidelberg (2008), pp 91-103.

[17] Khovratovich, Dmitry, Christian Rechberger, and Alexandra Savelieva. "Bicliques for preimages: attacks on Skein-512 and the SHA-2 family." International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg (2012), pp 244-263.

[18] Gauravaram, Praveen, Millan, W., Dawson, E., & Viswanathan. "Constructing secure hash functions by enhancing Merkle-Damgård construction." Australasian Conference on Information Security and Privacy. Springer, Berlin, Heidelberg (2006), pp 407-420.

[19] Elkamchouchi, Hassan M., Mohamed E. Nasr, and Roayat Ismail Abdelfatah. "A new secure and fast hashing algorithm (SFHA-256)." 2008 National Radio Science Conference. IEEE (2008), pp 1-8.

[20] Selent, Douglas. "Advanced encryption standard." Rivier Academic Journal 6.2 (2010), pp 1-14.

[21] Yang, Yijun, Chen, F., Sun, Z., Wang, S., Li, J., Chen, J., & Ming, Z. "Secure and efficient parallel hash function construction and its application on cloud audit." Soft Computing 23.18 (2019), pp 8907-8925.

[22] Li, Yantao, and Guangfu Ge. "Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security." Multimedia Tools and Applications 78.13 (2019), pp 17973-17994.

[23] Teh, Je Sen, Azman Samsudin, and Amir Akhavan. "Parallel chaotic hash function based on the shuffle-exchange network." Nonlinear Dynamics 81.3 (2015), pp 1067-1079.