ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/352377187

Enhancement the ChaCha20 Encryption Algorithm Based on Chaotic Maps

Chapter · January 2021

DOI: 10.1007/978-981-16-0666-3_10

CITATIONS
9

READS
2,922

2 authors:

Hussain H. Alyas
University of Babylon
2 PUBLICATIONS **14** CITATIONS

SEE PROFILE

Alharith A. Abdullah
University of Babylon
59 PUBLICATIONS **298** CITATIONS

SEE PROFILE

All content following this page was uploaded by Alharith A. Abdullah on 09 October 2021.

The user has requested enhancement of the downloaded file.

Enhancement the ChaCha20 Encryption Algorithm Based on Chaotic Maps

Hussain H. Alyas and Alharith A. Abdullah

Abstract As a result of the increasing demand for lightweight applications, the researchers set out to implement appropriate security mechanisms, one of which is lightweight cryptographic hash functions. Light hash functions are a common topic in cryptography, and its strength depends primarily on cryptographic analysis of hash functions. Therefore, it always requires an improvement in the designs of light hash functions in addition to paying attention to their performance in terms of implementing software and hardware. In this paper, we propose a lightweight hash functions scheme that consolidates the method of the ChaCha20 data encryption with chaotic maps to keys generation. It has made the existing ChaCha20 encryption method stronger when chaos-based random number generation method has been utilized. Based on these random numbers, the ChaCha20 would generate keys with high randomness and perform cryptographic operations. The analysis results show that improving the ChaCha20 algorithm by chaotic maps generated keys with high randomness and lightweight hash functions efficiency and thus increased security.

Keywords Lightweight hash function · ChaCha20 stream cipher algorithm · Chaotic maps

1 Introduction

Along with the use of traditional cryptographic transformations that are already standardized internationally, the technology "lightweight cryptography" is being developed. "Lightweight cryptography" refers to cryptographic protocols or algorithms designed to be implemented in restricted environments such as sensors, RFID tags, contactless smart cards, and so on and has lightweight properties described on the basis of target platforms. In hardware implementations, the size of chips and (or) power consumption are important measurements of lightweight properties. In software implementation, smaller code and (or) memory size have the advantage of

H. H. Alyas (✉) · A. A. Abdullah
College of Information Technology, University of Babylon, Babil, Iraq

© The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2021

R. Kumar et al. (eds.), *Next Generation of Internet of Things*,
Lecture Notes in Networks and Systems 201,
https://doi.org/10.1007/978-981-16-0666-3_10

lightweight applications [1]. Lightweight is interpreted as a simple and small-sized algorithm with low energy consumption, and low computational power requirement. The need for lightweight cryptographic hash functions has been repeatedly expressed by several authors. A conventional hash function is large in its internal state size. It consumes high power, which are two disadvantages that make it less preferred for devices that are resource-constrained.

The ChaCha20 stream cipher authenticator is cryptographic algorithms that were designed by Daniel J. Bernstein to ensure high-security margins [2]. The principle advantage of ChaCha20 is that it is designed based on ARX (Addition, Rotation and XOR) cryptography technique which provides faster performance, less complexity and eliminates timing attack [3]. In this paper, we improve the security of a lightweight algorithm (ChaCha20) and convert it into a hash function and characterized by high speed and low implementation resources. The proposed method consists of three layers: chaotic layer, ChaCha20 layer and hash layer. First layer is considered to be the first setup of random keys by using chaotic maps (Logistic 3D and Chebyshev) to generate random numbers for ChaCha20 encryption algorithm. As chaotic maps are nonlinear, chaos-based key generation increases security level and ensures good cryptographic properties, and it is difficult to crack any cipher generated through chaos-based secret key [4]. These random numbers are considered as initial parameters of ChaCha20 algorithm in second layer. Finally, the encryption process is done by the algorithm to be the blocks cipher and transformed it to the hash function. This paper is organized in the following way: Sect. 2 explores works related to the lightweight functions, while the ChaCha20 stream cipher is presented in the third section. Section 4 describes the Chaotic Maps. Details about our proposed algorithm are presented in Sect. 5. Section 6 shows experimental results and performance analysis. Finally, Sect. 7 is the conclusion.

2 Related Work

The topic of lightweight hash functions is very popular in cryptography. Khushbooet al. proposed a lightweight hash function, like Neeva-hash that satisfies the basic idea of lightweight cryptography. The Neeva-hash function is based on sponge mode of reiteration with software-friendly permutation that provided great security and efficiency in the technology of RFID [5]. However, the hash Charifaet al. proposed can be implemented in many application-based purposes. The function of the lightweight hash is based on the (L-CAHASH) cellular automata. This approach results in higher randomness quality and faster software hashing in comparison to other known lightweight hash functions. The new proposed hash function is proven to be robust and efficient. Our analysis and the results we obtained prove that this proposed hash function matches the security requirement of the RFID tags [6].

In [7], an extension of LCAHASH system is represented. LCAHASH is a lightweight hash algorithm previously developed in [6] to improve its robustness and efficiency. There is a description of the proposed system and a security analysis. The results for a statistical battery of tests (Dieharder) are included too. The results illustrate that the proposed system exhibits adequate statistical and cryptographic characteristics. In [8], the authors have discussed various features of the lightweight cryptography. They have proposed a hash function for the spinal codes to effectively reduce the time and the decoding complexity of the sponge structure with chaotic mapping (SSCM) algorithm changing slowly with increasing length of the message. Chaotic mapping which includes two-dimensional logistic mapping and two-dimensional Chebyshev mapping is mainly used in the internal-permutation function of the proposed hash function. Several authors have directed their research on lightweight encryption in various technical fields. Salim Reza et al. [9], in a lightweight security scheme is proposed. This proposed scheme consolidates ChaCha20 data encryption method, chaos-based key generation and public key-based scheme of authentication. The given mathematical analysis of the proposed scheme shows that it is suitable for SGs in terms of low power consumption, less time for processing and higher throughput that makes it lighter in weight and faster. This lightweight security scheme prevents replay attacks or any kind of other timing attacks.

3 ChaCha20 Stream Cipher

The ChaCha family of stream ciphers is high-throughput stream ciphers developed for software platforms. The original specification of the ChaCha family allows for different instantiations of the cipher in order to support various "security versus performance," e.g., it allows for different key, nonce, and counter lengths as well as for a different number of rounds. In this section, we provide a description of ChaCha20, as specified by the RFC7539, which represents a conservative instantiation with respect to security [10]. Let (pi, ci) denote 64-byte blocks of the plaintext and ciphertext, respectively. ChaCha20 functions on words of 32 bits each time with a 256-bit key of $K = (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7)$. The outputs block with length of 512-bit stream of key. Then, the obtained key with plaintext is XORed together [11]. The encryption state is stored within (16 × 32)-bit word values and arranged as a (4 × 4) matrix:

$$\begin{matrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{matrix}$$

The ChaCha20 stream cipher ChaCha20-SC: $\{0, 1\}^{128} \times \{0, 1\}^{256} \times \{0, 1\}^{32} \times \{0, 1\}^{96} \rightarrow \{0, 1\}^{512}$, $(K, N, P) \rightarrow C$ encrypts the plaintext P into a ciphertext C using a secret 32-byte key K and a 12-byte nonce N and a 16-byte constant and a 4-byte counter C. The plaintext is simply encrypted by cutting P into 64-byte blocks

$(pi)0 \le i < 2^{32}$ and XOR-ing them with the 64-byte output blocks k_i^N of the ChaCha20 block function, i.e. $ci = pi \oplus k_i^N$. Conversely, the decryption is obtained in a simple backward manner by $pi = ci \oplus k_i^N$. The initial state of X is obtained 16 × 32-bit values as follows: The first row is filled with the constants (0 × b9c5e6be, 0 × 640035bf, 0 × 27d098ba, 0 × cdf87453), the second and third rows are filled with the 32-byte secret key K in little-endian order, ($k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7$), the word element at position 12 is filled with the block counter (c_0) and the remaining three word elements are filled with the nonce N (n_0, n_1, n_2,):

$$\begin{array}{cccc} 0 \times \mathrm{b9c5e6be} & 0 \times 640035\mathrm{bf} & 0 \times 27\mathrm{d}098\mathrm{ba} & 0 \times \mathrm{cdf}87453 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ c_0 & n_1 & n_2 & n_3 \end{array}$$

The counter thus has 32-bit (1 × 32 b), and the nonce has 96-bit (3 × 32 b). The ChaCha20 block function works internally by iterating the quarter round function QR: $\left(\{0, 1\}^{32}\right)^4 \to \left(\{0, 1\}^{32}\right)^4$ over a 64-byte internal state X. The ChaCha20 block function using a single combination of the key, nonce and increase or decrease in the counter (c_0) in location [X_{12}]. While in our paper we replaced the location of the counter with one of the location ChaCha20 states [X_0, X_1,…, X_{15}] based on the chaotic map instead of increasing or decrease the block counter, which is a suitable for the most applications because it gives more randomness to the key that extracted from the algorithm. The state [X_{12}] is represented as a (4 × 4) matrix, where each element represents a 32-bit word, and the element at position (i, j) is denoted by [X_{12}].

The core function is defined by iterating several rounds on the input block, where each round consists of four parallel quarter round (QR) operations. A QR updates four words (i.e., a block quarter) as defined in Fig. 1 where ⊞ means addition modulo 2^{32}, ⊕ means XOR and ⋘ means left bitwise rotation. The ChaCha20 block function consists of 10 "double rounds" which alternately execute a "column round" (four quarter rounds operating on the columns of X) and a "diagonal round" (four quarter rounds operating on the diagonals of X), thus resulting in a total of 20 round iterations.

Fig. 1 ChaCha quarter round (a, b, c, d)

```
a ⊞= b;    d ⊕= a;    d ⋘= 16;
c ⊞= d;    b ⊕= c;    b ⋘= 12;
a ⊞= b;    d ⊕= a;    d ⋘= 8;
c ⊞= d;    b ⊕= c;    b ⋘= 7;
```

4 Chaotic Maps

The theory of chaos describes a phenomenon which has inevitability latency rules that determine irregular appearances. It can be considered one of the hardest nonlinear problems. The origin of chaos has started in mathematics and physics and expanded into engineering. Mathematics has described that theory as "random," which means it results from the simple systems inevitability affected by the system's initial conditions. Currently, there is a significant interest in studying and applying chaotic systems and the importance of this theory in multidisciplinary areas of study such as cryptography, physics, engineering, neurophysiology, and many other fields. Chaos has a set of important properties including the sensitive, the irregular, the long-term prediction, the deterministic, and the property of nonlinearity. Studies in the last century focused on the use of chaos in cryptography so as to get features from which to achieve the system' security design based on the phenomenon of chaos [12]. The following subsection briefly introduces the logistic map and Chebyshev chaotic map that are used for the proposed system.

4.1 Logistic Map (LM)

This map is 3D chaotic map [13]. It can be described mathematically as follows:

$$X_{i+1} = \lambda X_i(1-X_i) + \beta Y_i^2 X_i + \alpha Z_i^3 \tag{1}$$

$$Y_{i+1} = \lambda Y_i(1-Y_i) + \beta Z_i^2 Y_i + \alpha X_i^3 \tag{2}$$

$$Z_{i+1} = \lambda Z_i(1-Z_i) + \beta X_i^2 Z_i + \alpha Y_i^3 \tag{3}$$

The security and difficulty of 3D logistic map is strengthened by three quadratic coupling constant factors. In this map, X, Y, Z are the trajectory of the system, λ, β, α are the system's parameters. When $3.53\backslash\lambda\backslash3.81$, $0\backslash\beta\backslash0.022$, $0\backslash\alpha\backslash0.015$ and the initials value's range is [0,1] using the initial conditions: $\lambda = 3.61$, $\beta = 0.003$, $\alpha = 0.05$, $X_0 = 0.55$, $Y_0 = 0.8$, $Z_0 = 0.3$, the system is in a state of chaos and three chaotic sequences can be generated in the region [0,1]. Therefore, the parameters of the system and its initial values can be considered as secret keys.

4.2 Chebyshev (CH)

It can be described mathematically as follows [14]:

$$X_{n+1} = \cos(k \ \arccos X_n),\ X_n \in [-1,\ 1] \tag{4}$$

where k is the Chebyshev map's order, using the initial conditions: $K = 5, X_0 = 0.5$. The Lyapunov index of the map is positive when $k \geq 2$, the map is in the chaotic region then. An infinite length of non-periodic, chaotic real-valued sequence can be produced by the map under the infinite precision condition.

5 Proposed Algorithm

We divided our system to three different layers: chaotic layer, ChaCha20 layer and hash layer as illustrated in Fig. 2. We will describe the three layers in details.

1. **1st Layer (chaotic layer)**:

In the chaotic layer, we used two kinds of chaotic maps (Chebyshev and 3D logistic map). We start with 3D logistic map, in which the initial parameter to the function is (λ, β, α, X, Y and Z) where these parameters applied to the Eqs. 1, 2 and 3, the output of these equations is converted to unsigned integer unit to be 32-bit integer (0-4294967295), the output of logistic 3D is 100 values; these values reverse it and added to be 200 values and each value has 32 bits.

In Chebyshev, the initial parameters to the function are ($X_{\circ}$, K), The output of these function is converted to unsigned integer unit to be 32-bit integer (0-4294967295), the function generates 100 values and reverses these values to get another 100 values to get the final 200 values and each value has 32 bits.

The next step generates unit matrix (16 × 16) the value of matrix is content from first values of the logistic function row by row and after that continue with the values of Chebyshev row by row as well.

Now, we generate the period table which is including the interval of values between $\left(0 \text{ and } \frac{1}{16} \approx 0.0625\right)$ as shown in Table 5.

We used this table to compare it with the values of logistic 3D and Chebyshev to get the old value which represents the values of (i, j) in unit matrix. Also there is another value named a new value. This value comes from the reset values of Chebyshev that generates and not used in the unit matrix (16 × 16).

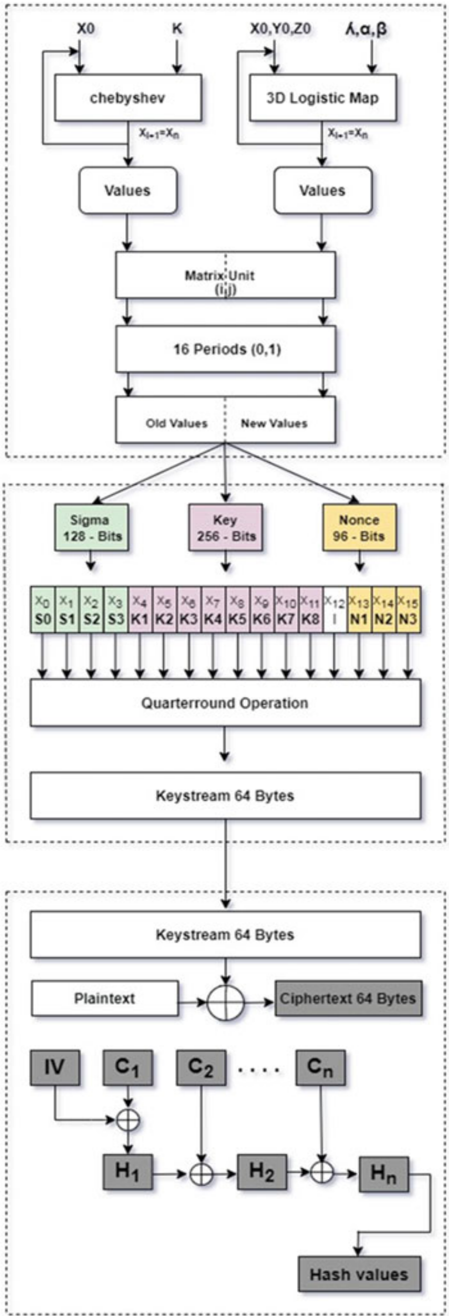
The old value represents the initial enters of the ChaCha20, and in case the old value repeats it, then we use the new values by replacing it with the old value.

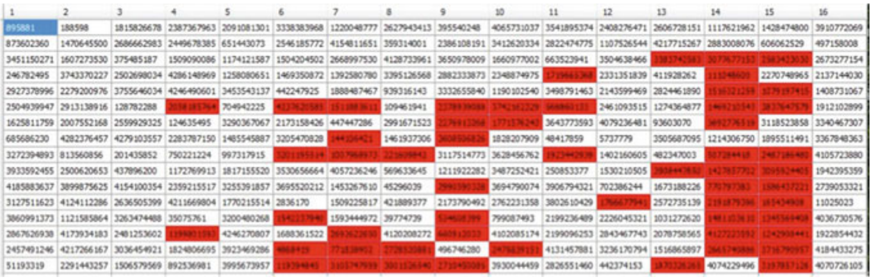
Let's see an example of steps to the operation of filling in a matrix (16 × 16) by chaotic values (logistic 3D map and Chebyshev) as shown in Tables 1 and 2.

Step 1 100 values are generated by logistic 3D map, and then we convert the values into normal values (0.895881–895881), e.g., convert it to unsigned integers (unit) with modulo 32^2 and then each value is reversed to become 200 values. We repeat the same process over 100 values generated by Chebyshev to become 200 as well, shown in Tables 3 and 4.

Step 2: Create matrix 16 × 16(Column × row).

Fig. 2 Proposed system





1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
895881	188598	1815826678	2387367963	2091081301	3338383968	1220048777	2627943413	395540248	4065731037	3541895374	2408276471	2606728151	1117621962	1428474800	3910772069
873602360	1470645500	2686662983	2449678385	651443073	2546185772	4154811651	359314001	2386108191	3412620334	2822474775	1107526544	4217715267	2883008076	606062529	497158008
3451150271	1607273530	375485187	1509090086	1174121587	1504204502	2668997530	4128733961	3650978009	1660977002	663523941	3504638466	[illegible]	[illegible]	[illegible]	2673277154
246782495	3743370227	2502698034	4286148969	1258080651	1469350872	1392580780	3395126568	2882333873	2348874975	[illegible]	2331351839	411928262	[illegible]	2270748965	2137144030
2927378996	2279200976	3755646034	4246490601	3453543137	442247925	1888487467	939316143	3332655840	1190102540	3498791463	2143599469	2824461890	[illegible]	[illegible]	1408731067
2504939947	2913138916	128782288	[illegible]	704942225	[illegible]	[illegible]	109461941	[illegible]	[illegible]	[illegible]	2461093515	1274364877	[illegible]	[illegible]	1912102899
1625811759	2007552168	2559929325	124635495	3290367067	2173158426	447447286	2991671523	[illegible]	[illegible]	3643773593	4079236481	93603070	[illegible]	3118523858	3340467307
685686230	4282376457	4279103557	2283787150	1485545887	3205470828	[illegible]	1461937306	[illegible]	1828207909	48417859	5737779	3505687095	1214306750	1895511491	3367848363
3272394893	813560856	201435852	750221224	997317915	[illegible]	[illegible]	[illegible]	3117514773	3628456762	[illegible]	1402160605	482347003	[illegible]	[illegible]	4105723880
3933592455	2500620653	437896200	1172769913	1817155520	3530656664	4057236246	569633645	1211922282	3487252421	250853377	1530210505	[illegible]	[illegible]	[illegible]	1942395359
4185883637	3899875625	4154100354	2359215517	3255391857	3695520212	1453267610	45296039	[illegible]	3694790074	3906794321	702386244	1673188226	[illegible]	[illegible]	2739053321
3127511623	4124112286	2636505399	4211669804	1770215514	2836170	1509225817	421889377	2173790492	2762231358	3802610429	[illegible]	2572735139	[illegible]	[illegible]	11025023
3860991373	1121585864	3263474488	35075761	3200480268	[illegible]	1593444972	39774739	[illegible]	799087493	2199236489	2226045321	1031272620	[illegible]	[illegible]	4036730576
2867626938	4173934183	2481253602	[illegible]	4246270807	1688361522	[illegible]	4120208272	[illegible]	4102085174	2199096253	2843467743	2078758565	[illegible]	[illegible]	1922854432
2457491246	4217266167	3036454921	1824806695	3923469286	[illegible]	[illegible]	[illegible]	496746280	[illegible]	4131457881	3236170794	1516865897	[illegible]	[illegible]	4184433275
51193319	2291443257	1506579569	892536981	3995673957	[illegible]	[illegible]	[illegible]	[illegible]	3930044459	2826551460	442374153	[illegible]	4074229496	[illegible]	4070726105

Fig. 3 Matrix (16 × 16)

Table 1 Logistic 3D map (100 values)

#	Logistic 3D map (v)	Cut point (v)
1	0.895881	895881
2	0.361749731299574	361749731299574
3	0.84651601518165	84651601518165
…	…	…
100	0.38299816823404	38299816823404

Table 2 Chebyshev (100 values)

#	Chebyshev (v)	Cut point (v)
1	0.799066410533737	799066410533737
2	0.92112067866505	92112067866505
3	0.546801907660972	546801907660972
…	…	…
100	0.877320199797859	877320199797859

Table 3 Logistic 3D map (200values)

#	Chebyshev (v)	Unsigned integers (v)
1	x_1	(x_1)
2	Reverse (x_1)	(Reverse (x_1))
3	x_2	(x_2)
4	Reverse (x_2)	(Reverse (x_2))
…	…	…
199	X_{199}	(X_{199})
200	Reverse (x_{199})	(Reverse (x_{199}))

Step 3: We enter values of logistic 3D map and then Chebyshev to the (16 × 16) matrix in a row as shown in Fig. 3.

Table 4 Chebyshev (200 values)

#	Logistic 3D map (v)	Unsigned integers (v)
1	x_1	(x_1)
2	Reverse (x_1)	(Reverse (x_1))
3	x_2	(x_2)
4	Reverse (x_2)	(Reverse (x_2))
…	…	…
199	X_{199}	(X_{199})
200	Reverse (x_{199})	(Reverse (x_{199}))

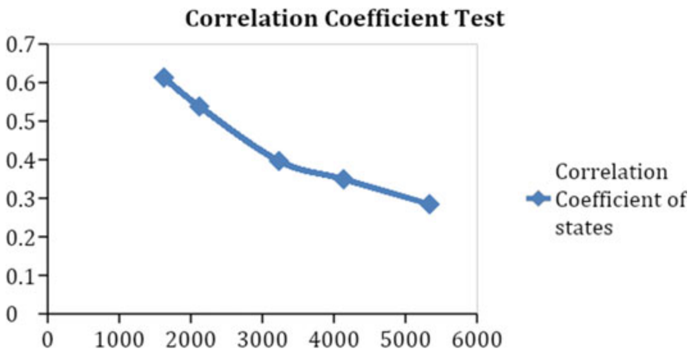


Fig. 4 Correlation coefficients test on states of original and modified algorithm

Step 4: Create a periods table (0, 1) starting from (0.000 to 0.0625) and ending at (0.9375–1.0000) as in Table 5.

Step 5: We extract (i, j) through periods that include the values of logistic 3D map and Chebyshev. For example, if we take the first value from logistic maps (0.895881), we notice that it is within the period (0.8750–0.9375) at location (14) which represents (i) and the first value from Chebyshev (0.799066410533737) we note that it is within a period of (0.7500–0.8125) at location (12) which represents (j).

The above applies on the 100 values of logistic 3D map and Chebyshev.

Step 6: We take (i, j) on the matrix to extract the value that is used as initial values in ChaCha20 state. We called this the old value.

The new value for the first old value is derived from the remaining (400) values used in the matrix that consists of (256) values, where the value with a sequence (257) is the first new value and the value with a sequence (258) is the second new value, etc. The purpose is not to repeat the value (i,j), as in Table 6.

Finally enter the values into the initial state of ChaCha20 as (constant, key, nonce).

2. **2nd Layer (ChaCha20 layer)**:

In this layer, we generate the key in each round where the length of the key is 512 bits, and we divided the plaintext into blocks each block 512 bits, now X-OR with

Table 5 Periods

#	From	To
0	0.0000	0.0625
1	0.0625	0.1250
2	0.1250	0.1875
3	0.1875	0.2500
4	0.2500	0.3125
5	0.3125	0.3750
6	0.3750	0.4375
7	0.4375	0.5000
8	0.5000	0.5625
9	0.5625	0.6250
10	0.6250	0.6875
11	0.6875	0.7500
12	0.7500	0.8125
13	0.8125	0.8750
14	0.8750	0.9375
15	0.9375	1.0000

Table 6 Deriving old and new values from matrix

#	Logistic [x]	Location [i]	Chebyshev [x]	Location [j]	Old values	New values
1	0.895881	14	0.799066410533737	12	1269421360	3803453717
2	0.361749731299574	5	0.92112067866505	14	1704499801	3441014062
…	…	…	…	…	…	…
100	…	…	…	…	…	…

the key to produce the ciphertext. Also, we implement ChCh20 algorithm consisting 16 initial states as in Sect. 3.

3. **3rd Layer (hash layer)**:

We divided the ciphertext into blocks with 64 bytes each block X-OR with the hashing except the first block X-OR with the initial vector (IV) and produce first hashing this operation continuo accumulator hashing until we get the final hash value, and this means we convert the ChaCha20 from lightweight encryption algorithm to the lightweight hashing algorithm. By doing this, we have achieved a hash function by converting an input value that is numerical into a compressed numerical value. The hash function's input is of arbitrary length, while the output's length is always fixed 64 bytes. All processes generating the hash value are explained in Algorithm 1.

Algorithm 1: Lightweight Hash value generation

Input: data

Output: hash[data]

Step1

Generation set of value using Chebyshev based on Length of data

Generation set of value using logistic 3D map based on length of data

Step2

Generation matrix call Select matrix using check value in this matrix and return index from Select matrix by to steps

Step2-1 generation matrix

```
xFrom = 0.0
xTo = 1 / 16
    foreach(i=0; i< 16 ; i++ )
Selec_matrix[i].From=xFrom;
Selec_matrix[i].To=xTo;
xFrom=xTo;
xTo=xTo + 1/16;
    End for
```

Step2-2 return index from Select matrix

```
    Get xvalue
    foreach(index=0; index< 16 ; index++ )
      if (xvalue> Select matrix [index].From &&
xvalue<= Select matrix [index].To  then
 Get index;
    End for
```

Step3

Step3-1 Full value key , nonce and Sigma from step1

Step3-2 set initial hash=00000000000000000 [512 bit]

Step3-3 (foreach $block_i$ in data) do

Get value1 from step1

Get index_State from step2-2

Ganartionkey_i by paramaters[key , nonce and Sigma, ndex_State]

$ci \leftarrow block_i i \oplus k^N{}_i$

$hash \leftarrow hash \oplus ci$

End for

A hash is a "one-way" cryptographic function. It is not "encryption" that can be reversely decrypted to the original version. A hash is fixed in size for any size of source text). This one-way function makes hashed texts suitable, when appropriate, to be compared, as opposed to decrypting the text to get the original version of the text. For example, if you add two binary digits together and take only the one binary digit of the result, we have started with two bits of data and ended up with one bit. We destroyed, or lost, one bit of data in the process. Thus, the original text cannot be returned as shown below:

$$0 + 0 = 0$$
$$1 + 1 = 0$$

There are two different inputs to the one output of 0 in which these two inputs did the forward code take to get the output of 0.

6 Experimental Results and Performance Analysis

We have implemented the proposed enhancement ChaCha20 in this section using C# programming with an environment Windows 7 Ultimate, Home Basic (64-bit) Operating System, Intel Core i5- M520, Processor 2.40 GHz clock rate, and memory of 4 GB RAM, 320 GB hard disk.

6.1 Correlation Coefficient

The “correlation coefficient” assumes a range of values between-1 and +1. As indicated by [15, 16].

The equation that describes the use of the correlation coefficients functions is the following:

$$E(c) = \frac{1}{s}\sum_{i=1}^{s} P_i \tag{5}$$

In this equation: s is the total bits’ number, are the series of s measurements for P and c, P is the value of original bits or plaintext, c is the value of change bits or ciphertext, $E(c)$ is a mathematical expectation of c.

The following equation represents the variance of p:

$$D(c) = \frac{1}{s}\sum_{i=1}^{s}[P_i - E(P)]^2 \tag{6}$$

Finally, the related coefficients Γ_{PC} can be described in this Eq. 7:

$$\Gamma_{\mathrm{PC}} = \frac{E\{[P - E(c)][c - E(c)]\}}{\sqrt{D(P)}\sqrt{D(c)}} \tag{7}$$

The experiment examined all functions of the ChaCha20. The analyses of the data, in which correlation value for shown in Table 7, show the results of the experiments conducted out according to levels of ChaCha20 states. Table 7 represents the respective performances of the correlation coefficient measured in recall. The correlation coefficient factor (Security Analysis) is used to measure the relationship between two states original & modified algorithms after the encryption process; the original ChaCha20 algorithm encryption and its modified version for states. As the difference between the two states appears, this difference increases as the size of data increases as shown in Fig. 4. Therefore, modified algorithm values must be different from the original. Therefore, using two algorithms, the success of the process. It means values smaller than 1 and near-zero values of the correlation coefficient. The correlation coefficient of the proposed algorithm and its original. It starts from (0.61282) and gradually decreases toward near-zero in the tests shown in tables which showed several results of correlation coefficients, but when we entered text with size (5336

Table 7 Correlation coefficient results

Statistical test	Text size (in bits)				
	1624	2120	3232	4136	5336
Correlation coefficient	0.61282	0.53790	0.39657	0.34923	0.28447

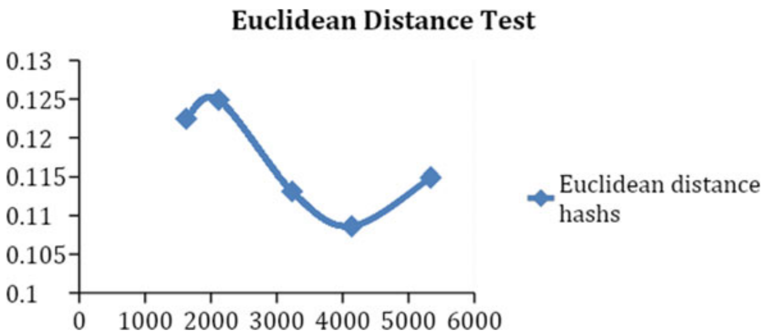


Fig. 5 Euclidean distance test for randomly hashing of original and modified algorithms

bits) then it was encryption and converted to the hash function, and the best result was the correlation coefficient near zero, which was given upon implementation (0.28447).

6.2 Calculation of Scores

For the comparison of two samples A and B, we have to calculate the scores between the two samples. Because of less number of computations in comparison to others, we are using Euclidean distance to measure the similarity.

Squared Euclidean Distance: This is commonly used when the data is dense or in continuous form. Euclidean distance is a measure of the absolute distance between two hash sequences. The Euclidean distance of original and modified hashing is as follows in Eq. 8.

$$D(oh, mh) = \sqrt{\sum_{i=1}^{N} (oh(i) - mh(i))^2} \tag{8}$$

where (D) is Euclidean distance to measurements the similarity of two randomly hashing, the (oh) is original hash and (mh) modified hash.

The tests of the calculations of scores shown in Table 8 illustrate the results of the experiments made according to hashing similarity levels. The table represents the

Table 8 Calculations of score results

Statistical test	Text size (in bits)				
	1624	2120	3232	4136	5336
Euclidean	0.12247	0.12489	0.11313	0.10862	0.11489

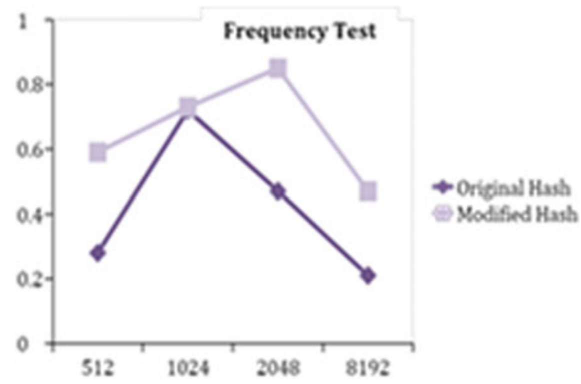


Fig. 6 Frequency test on original and modified hash code

measures of similarity in their respective performances which are measured in recall. A set of text sizes was randomly entered to measure the percentage of similarity in the hash function resulting from the ChaCha20 algorithm, which was improved by the chaotic maps. Through observing the tests, several results appear to measure the Euclidean distance for two results randomly hashing. We showed distinct results for Euclidean distance estimated at (%10.862) when entering text size (4136 bits). These are considered the best results compared to other results as shown in Fig. 5.

6.3 NIST Suite Randomness Test

The NIST randomness test package is made of 15 tests designed to examine the binary sequences randomness by calculating the value of the *P*-value.

In this work, five tests were used (frequency test, runs test, the longest run of ones in a block, entropy test and cumulative sums) which represented the 15 tests. That means if the sequence of key bits passes these five tests, then it can pass all the other ten tests [17]. Tables 9 shows the results of implementing the original and modified hash with different text sizes (512, 1024, 2048 and 8192) in bits.

In order to evaluate the work, we will compare the results of each of the five tests between the original and modified hash by calculating a value *P*-value.

One of the most important tests is the frequency test, since all subsequent exams depend mainly on passing this test [17]. So in the first test, we depended on frequency and measuring the extent of randomness of ones and zeros in a hash code. The results showed that the frequency randomness in the modified algorithm is higher than the original algorithm, which makes the hash code solution difficult to frequency analysis attack as shown in Fig. 6.

In general, the results show that the modified algorithm achieved a relatively higher randomness than the original algorithm where obtained 0.963 in runs test and 0.818 in cumulative sums test. While a high level of randomness was obtained about 1000 in longest block run and entropy tests.

Table 9 NIST (randomness tests) results

Statistical Test	Text size (in bits)							
	512		1024		2048		8192	
	Original hash	Modified hash	Original hash	Modified hash	Original hash	Modified hash	Original hash	Modified hash
Frequency test	0.288	0.595	0.723	0.737	0.479	0.859	0.215	0.479
Runs	0.780	0.389	0.074	0.963	0.505	0.156	0.393	0.564
Longest block run	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Approximate entropy	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Cumulative sums	0.574	0.654	0.737	0.054	0.223	0.818	0.265	0.818

Our results showed the high randomness enjoyed by the hash code after modifying the algorithm, which is an important factor to increase the security of the hash code as it makes it difficult for an attacker to analyze the code by guessing or breaking the code in addition to the complexity and time needed to reach the correct hash code used, which is a very important factor in hash algorithms.

7 Conclusion

In this paper, we have proposed a hash function that is lightweight, and that matches the requirements of many constrained devices such as wireless sensors, radio frequency identification tags, and other devices of embedded systems. The lightweight algorithm (ChaCha20) that we propose has been improved and strengthened by relying on chaotic maps. This improvement resulted in the generation of keys with high randomness used for encryption and decryption operations converting the ChaCha20 algorithm to a hash function that is light in weight and that has minimal computational complexity and low cost. The results that have emerged indicate our contribution to improving the security system.

References

1. Kuznetsov A, Gorbenko Y, Andrushkevych A, Belozersev I (2017) Analysis of block symmetric algorithms from international standard of lightweight cryptography ISO/IEC 29192-2. In: 4th international science conference problem infocommunications science technology. PIC S&T, pp 203–206. https://doi.org/10.1109/infocommst.2017.8246380
2. Bernstein DJ (2008) ChaChaa variant of Salsa 20. Work Rec SASC 1–6. http://cr.yp.to/chacha/chacha-20080120.pdf

3. Nir Y, Langely A (2015) RFC 7539—ChaCha20 and Poly1305 for IETF Protocols. http://www.rfc-editor.org/info/rfc7539
4. Hamdi M, Rhouma R, Belghith S (2015) A very efficient pseudo-random number generator based on chaotic maps and S-box tables. Int J Comput Electr Autom Control Inf Eng 9(2):481–485
5. Bussi K, Dey D, Kumar M, Dass BK (2016) Neeva : a lightweight hash function, pp 1–14. https://eprint.iacr.org/2016/042
6. Hanin C, Echandouri B, Omary F (2017) L-CAHASH : a novel lightweight hash function based on cellular automata for RFID. In: Third international symposium, UNet, Casablanca, Morocco, pp 287–298. https://doi.org/10.1007/978-3-319-68179-5
7. Sadak A, Echandouri B, Ziani FE, Hanin C, Omary F (2019) LCAHASH-1. 1 : a new design of the LCAHASH system for IoT. https://doi.org/10.14569/ijacsa.2019.0101134
8. Wang L (2019) A hash function based on sponge structure with chaotic map for spinal codes. In: International conference on computer, information and telecommunication systems, pp 1–5
9. Salim Reza SM, Ayob A, Arifeen MM, Amin N, Hanif Md Saad M, Hussain A (2020) A lightweight security scheme for advanced metering infrastructures in smart grid. Bull Electr Eng Inform 9(2). https://doi.org/10.11591/eei.v9i2.2086
10. De SF, Schauer A, Sigl G (2017) ChaCha20-Poly1305 authenticated encryption for high-speed embedded IoT applications. Autom Test Eur 692–697. https://doi.org/10.23919/date.2017.7927078
11. McLaren P, Buchanan WJ, Russell G, Tan Z (2019) Deriving ChaCha20 key streams from targeted memory analysis. J Inf Secur Appl 48:102372. https://doi.org/10.1016/j.jisa.2019.102372
12. AbodZ A (2018) A hybrid approach to steganography system based on quantum encryption and chaos algorithm. J Univ Babylon Pure Appl Sci 26:280–294
13. Ye Jiao G, Pan C, Huang X (2018) An effective framework for chaotic image encryption based on 3D logistic map. Commun Netw, Secur. https://doi.org/10.1155/2018/8402578
14. Feng Y, Liu Y, Zhao G, Xia M (2016) An improved AES encryption algorithm based on the Hénon and Chebyshev Chaotic Map. Int J Simul Syst Sci Technol 17(48):25.1–25.8. https://doi.org/10.5013/ijssst.a.17.48.25
15. Taylor R (1990) Interpretation of the correlation coefficient: a basic review. J Diagn Med Sonogr 6:35–39
16. Wong K (2010) Interpretation of correlation coefficients. Hong Kong Med J 16:237
17. Rukhin A, Soto J, Nechvatal J, Smid M, Barker E (2001) A statistical test suite for random and pseudorandom number generators for cryptographic applications. Booz-Allen and Hamilton Inc Mclean Va

View publication stats