Research Article

Constructing Keyed Hash Algorithm Using Enhanced Chaotic Map with Varying Parameter

Hongjun Liu,1,2 Abdurahman Kadir,3 Chao Ma,1 and Chengbo Xu1

1School of Mathematical Sciences, University of Jinan, Jinan, Shandong 250022, China
2Shandong Famous Teacher’s Workshop, Weifang Vocational College, Weifang, Shandong 262737, China
3School of Information Management, Xinjiang University of Finance and Economics, Urumqi 830012, China

Correspondence should be addressed to Hongjun Liu; wfliuhongjun@126.com

Received 9 December 2019; Revised 26 April 2020; Accepted 14 May 2020; Published 20 July 2020

Academic Editor: Alessio Gizzi

Copyright © 2020 Hongjun Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A keyed hash algorithm is proposed based on 1-D enhanced quadratic map (EQM) with varying parameter. Three measures, including assigning unique one-time keys, key expansion, and hash length extension, are taken to enhance its security. First, the message is transformed into a parameter sequence for the EQM to be absorbed, and then the extended keys are generated as the initial values of the EQM. Finally, the EQM is iterated with redundant loops to transform the variable values into a hash value. The algorithm is so flexible that it can generate hash value with different lengths of 256, 512, 1024, or more bits through a parameter switcher, and redundant loops can eliminate the transient effect of chaos and mitigate the increasing threat of the side-channel attack. Security evaluations and comparison demonstrated its practicability and reliability.

1. Introduction

Hash algorithm is widely used for assuring data integrity in cryptography [1]; it can map a message with arbitrary length to a hash value with fixed length. If the input message is unknown, it is extremely difficult to reconstruct via its hash value. In theoretical cryptography, the security level of a hash algorithm could be defined by three properties [2]: preimage resistance, second preimage resistance, and collision resistance. None of the existing hash algorithms is secure absolutely. Even if a hash algorithm has not been broken up to now, a successful attack against a weakened variant may result in its abandonment, such as the theoretical weaknesses of SHA-1 were found in 2005 [3], a successful attack on MD5 in 2008 [4], and Google announced a collision in SHA-1 in 2017 [5]. Although some recognized hash algorithms, such as SHA-2, SHA-3, and SM3, are still secure up to now, however, all kinds of attacks on them are going on and on [6].

Many hash algorithms based on chaotic maps have been proposed [7]; however, some 1-D chaotic maps, such as logistic map and tent map, are typically insecure or slow, and most of these hash algorithms have been broken successfully.

Xiao et al. [8] constructed a hash algorithm based on the piecewise linear chaotic map with changeable parameter; however, Guo et al. [9] analyzed its weakness and utilized weak keys to construct a collision successfully. Kwok and Tang [10] designed a hash algorithm based on a high-dimension chaotic map, and a compression function was developed according to the diffusion and confusion properties of the chaotic map; however, Deng et al. [11] analyzed the potential flaws in this hash algorithm and took corresponding measures to enhance the influence of a single-bit change in the message on the changes in the final hash value. Liu et al. [12] proposed a keyed hash function using a hyperchaotic system with time-varying parameter perturbation, which is flexible and has a larger key space. Teh et al. [13] designed a keyed hash function based on the logistic map with fixed point representation. Li et al. designed four 128-bit parallel hash functions based on cross-coupled map lattices [14], tent map [15], circular shifts [16], and dynamic S-box [17] with varying parameters.

An attacker can crack the hash value of a short password using a precomputed rainbow table [18]. Petr et al. [19] designed a secure and efficient hash function with extra
Mathematical Problems in Engineering

2. The EQM

The classical quadratic map can be expressed using equation (1) [20]:

$$x_{i+1} = y - x_i^2,$$  \hspace{1cm} (1)

where the control parameter $y \in [0, 2]$ and state variable $x \in [-2, 2]$. The bifurcation diagram, phase diagram, and Lyapunov exponent shown in Figure 1 demonstrate that equation (1) has abundant bifurcations and dense windows; hence, the distribution of state points is not uniform, and its randomness is not good.

Based on equation (1), we constructed a 1-D EQM using equation (2):

$$x_{i+1} = \sin\left(y - 2^k(y - x_i^2)^2\right),$$  \hspace{1cm} (2)

where the range of control parameter $y$ is extended to $[0, 18]$ and the exponent $k \in [7, 1000]$. The bifurcation diagram and the phase diagram shown in Figures 2(a) and 2(b) demonstrate that the EQM has ergodicity and better randomness, and it is surjective within the interval $x \in [-1, 1]$. Figure 2(c) demonstrates that the Lyapunov exponent increases gradually with the increase of $y \in [0, 18]$ [21]; hence, the map achieves chaotic state. The state variable $x$ and exponent $k$ can serve as keys.

### 3. Hash Algorithm

Input: message $M$ with $n$ characters, which can be single-byte or multibyte, and theoretically, the length of the message can be infinite. A unique one-time 256-bit key is assigned according to each user’s identification.

Output: hash value $H$ with len-bit.

A hash algorithm $H(M, \text{len}, \text{key})$ can be described as follows:

Step 1 (message pre-encoding): for each character $m(i) \in M, i = 1, 2, \ldots, n$, transform it into a corresponding Unicode value using equation (3) to obtain $r(i)$ and serve as varying parameter $y$ of equation (2). It should be noted that even if $M$ is a null string, we can pad four specific characters of “====” to it.

$$r(i) = \begin{cases} \text{unicode}(m(i))^2 \times 10^{-4}, & \text{if unicode}(m(i)) < 1000, \\ \text{unicode}(m(i)) \times 2 \times 10^{-4}, & \text{if unicode}(m(i)) \geq 1000. \end{cases}$$  \hspace{1cm} (3)

Step 2 (key derivation): transform a 256-bit key into its hexadecimal number $K = (k(1), k(2), \ldots, k(64))$, and then generate four initial values $x_0(1, 1), x_0(2, 1), x_0(3, 1)$, and $x_0(4, 1)$ using equation (4) and exponent sequence $S = \{s_1, s_2, \ldots, s_{16}\}$ as salt using equation (5):

$$\begin{align*}
x_0(1, 1) &= \sum_{i=1}^{16} k(i) \times 10^{-4}, \\
x_0(2, 1) &= \sum_{i=17}^{32} k(i) \times 10^{-4}, \\
x_0(3, 1) &= \sum_{i=33}^{48} k(i) \times 10^{-4}, \\
x_0(4, 1) &= \sum_{i=49}^{64} k(i) \times 10^{-4},
\end{align*}$$  \hspace{1cm} (4)

$$\%s_1 = \sum_{i=1}^{4} k(i), s_2 = \sum_{i=5}^{8} k(i), \ldots, s_{16} = \sum_{i=61}^{64} k(i).$$  \hspace{1cm} (5)

Step 3 (message absorption): iterate equation (2) 16 rounds with initial parameter sequence $r(i)$, exponent $k = s_1$, and initial value $x_0(1, 1)$, from the second round, set $r(i) = x_1(1, i)$, and so on. Similarly, iterate equation (2) with $r(i)$ and initial values $x_0(2, 1), x_0(3, 1)$, and $x_0(4, 1)$, respectively. Finally, we can obtain four variable values $x_1(1, n), x_1(2, n), x_1(3, n)$, and $x_1(4, n)$ to serve as new initial values of equation (2) with exponent $k = s_2$.

Step 4 (hash value generation): after iterating equation (2) 300 times to eliminate the transient process, continue to iterate it len/16 times using four initial values $x_1(1, n), x_1(2, n), x_1(3, n)$, and $x_1(4, n)$ with the salt sequence $k \in \{s_1, s_2, \ldots, s_{16}\}$ as salt in turn to obtain four variable sequences $x_{TS}(j), y_{TS}(j), z_{TS}(j)$, and $w_{TS}(j), j \in [1, \text{len}/16]$. Transform them into unsigned integers within the interval $[0, 255]$ using equation (6) to generate two groups of hash value $H_1$ and $H_2$ in hexadecimal form using equation (7), and concatenate them to obtain the final hash value $H = H_1 || H_2$:
Figure 1: Bifurcation diagram, phase diagram, and Lyapunov exponent of equation (1).

Figure 2: Bifurcation diagram, phase diagram, and Lyapunov exponent of equation (2).
Xₜ₁ = uint8(mod(xₜ₁(1:len/16) × 10¹⁴, 256))
Yₜ₁ = uint8(mod(yₜ₁(1:len/16) × 10¹⁴, 256))
Zₜ₁ = uint8(mod(zₜ₁(1:len/16) × 10¹⁴, 256))
Uₜ₁ = uint8(mod(uₜ₁(1:len/16) × 10¹⁴, 256))

\[
H₁ = \text{dec2hex(bitxor}(Xₜ₁, Yₜ₁))
\]
\[
H₂ = \text{dec2hex(bitxor}(Zₜ₁, Uₜ₁))
\]

The flowchart of the proposed hash algorithm is shown in Figure 3.

4. Experimental Evaluation

4.1. Key Space. The proposed hash algorithm has a one-time 256-bit external key \(K\); hence, the key space \(S = 2^{256}\) [22], which is large enough to resist the brute-force attack [23].

4.2. Hash Sensitivity to Message and Keys. A good hash algorithm based on the chaotic map, should be very sensitive to any small change of the input message and initial conditions [12]. In the following tests, \(M₁\) represents the original input message, \(M₂, M₃,\) and \(M₄\) represent minor modifications to \(M₁\), and \(M₅\) represents a minor change to \(K\).

The original message \(M₁\): “as of 2018, the development of actual quantum computers is still in its infancy, but experiments have been carried out in which quantum computational operations were executed on a very small number of quantum bits. Both practical and theoretical research continue, and many national governments and military agencies are funding quantum computing research in additional effort to develop quantum computers for civilian, business, trade, and environmental and national security purposes, such as cryptanalysis. A small 16-qubit quantum computer exists and is available for experiments via the IBM quantum experience project.”

\(M₂\): replace the first character “A” of \(M₁\) with “a.”
\(M₃\): replace the last character “.” of \(M₁\) with “.”.
\(M₄\): add a blank space to the end of \(M₁\).
\(M₅\): change one bit to \(K\).

The 256-, 512-, and 1024-bit hash values in hexadecimal form are given in Table 1, and the results of Hamming distance demonstrate that any slight modifications on messages or key will lead to about 50% difference in the hash value.

4.3. Statistical Distribution of Hash Value. The hash value generated by a good hash algorithm should be evenly distributed. Here, we use Figure 4 to show the distributions of the message \(M₁\) and hash values of \(H₁_{256}\) and \(H₁_{512}\); from Figure 4(a), we can find that the ASCII values of \(M₁\) are localized within some specified intervals, while the hash
| Message | Hash value in hexadecimal format | Hamming distance |
|---------|---------------------------------|-----------------|
| H1256  | 3EE44E87DCE79D8BB2A43D968A6E3EAEB78AD6CDCCD2ACG9060DAAA246C2 | 0               |
| H3256  | 0FD775424006281DA91ACBB236146D382A06CCD15522D9B876B7A515C66E60D | 133              |
| H4256  | FA4B95913D9C978E860A6B78065F922FEDF6EF6EA65EAD690114B726E56 | 142              |
| H5256  | FCD83E20323D6DDB35896278D664474E8ED9F3CB292D457DBC899E115 | 137              |
| H512   | 28ER2B7A3633DB07A253R2C1F1ABD1F1930DEA29F66B2A39D2344AE69DBD642651509C62D567D98281F1C2B02CF079CFF01A1CD2D8C11CB8D1A0BB | 0               |
| H5112  | EF634EAB0B1CD07A139EEA9E0F3F621A628A60A3943C76A6CC891CEB1A65924A2E8D32B053C0E44F42D274780865E3CF131DC5699F2EA9BBAD | 236              |
| H5112  | 0A381CC165B44A9F21DD6763FA1D4E1ACAB1BE3CE92A01C17E73A9E603D86C90768A6D0A9DA9AA6CC61612A3EC6B517A6F3E3DE3B4F4EB23F858421C0ED8 | 259              |
| H612   | CE146D559E4705C31CD59C8480F8D085599C38C6D06C1D0F10A6D9F937E79633355A4A35A32A60B77739778B6445D27DE43CF7EF859F9DEA81BDA13 | 264              |
| H712   | 13D4A9E5D7938F82726206380F621AA8432F4602D222032BFD103E481F8252D60B6871ACCF99C2D4C530ED5D801A5BE27E9611A282F0301D8D71B | 260              |
| H1024  | D9D5557D693DD2D4341CD2B1408BCEC203265A30B3E14CE348FBDB3S1C51A1468EF137EB84A9796C9E89B38DBA651BE4D78CE10E8DF5B35AE13B | 0               |
| H2024  | IB363D853BE6F68050C64488DE783EF1B6997C3A3816E639459A9F58372E7FD3662CD9B8990F17FEC2AA68884C89DF9EAF3E79FA7D8868D68FF77 | 477              |
| H3024  | ED5F3041B9A0013E98191930DE7122E4D1E8AC0F1E801A4702D2B6783CC6934CD4B478BE2AD9464AD5A9B1440A5048FF93C507D8E4D | 503              |
| H4024  | 5CD87CE49BFF44C85D637F326EA116EF5584129D9CFDBDEC1D18930C39607F26F21452D8410R5730F6CEB14B84766C83836A8B82D36D2F15C46F75B41C | 488              |
| H5024  | 58D1CE4B25E262A02E4C63520CBAE517956096F3432B68EA78F3C8B51DE5CA25E7F6AD9249157636B434206D3B8C27197 | 515              |
| H6024  | 8A0AC152E932DD2AF9728408BBA463375084C875933B1IDC86E5ED043C3D4A8CE2B4E85A5C731D383EE93C472904A132A44A4CD9CD6C81B9A715D6F4AE5727191A86D83BFA | 515              |

Table 1: The corresponding hash value under different conditions and Hamming distance.
values shown in Figures 4(b) and 4(c) distribute uniformly. In addition, we utilize the hash algorithm to calculate the 256-, 512-, and 1024-bit hash values of a null string; from Figure 5, we can infer that the distributions of hash values are also uniform.

4.4. Statistical Analysis of Confusion and Diffusion. The hash value of a good hash algorithm should be confused and diffused completely [12], and the ideal result is that one-bit change to the input bits would lead to 50% change in the output bits. Here, we conducted a large number of experiments to analyze its performance. First, a random message \( M \) with the size of \( L = \text{len} \times 50 \) is generated, and len-bit hash value is calculated. Second, a single bit in \( M \) is changed, and a new len-bit hash value is calculated. Two hash values are compared bit by bit to obtain the total number of changed bits. The experiment is repeated \( N = 5000 \) times with len = 256-bit, 512-bit, and 1024-bit, respectively.

The corresponding histogram distribution of the total number of different bits is plotted in Figure 6, which demonstrates that the total numbers of changed bits concentrate around the ideal number 128-bit, 256-bit, and 512-bit, i.e., about 50% bits are changed; hence, the results of diffusion and confusion are ideal.

The following statistics are used to test the performance of the hash algorithm. Here, len is the length of the hash value, \( N \) is the number of tests, \( B_i \) denotes the number of different bits between the hash values obtained in the \( i \)-th test, \( B_{\text{min}} \) denotes the minimum number of different bits, \( B_{\text{max}} \) denotes the maximum number of different bits, \( \overline{B} \) denotes the mean changed bit number, \( P \) denotes the mean changed probability, \( \Delta B \) denotes the standard deviation of numbers of changed bits, and \( \Delta P \) denotes the standard deviation [13].

Tables 2–4 are statistical results obtained by changing one bit to \( M_1 \) randomly and executing the hash algorithm \( N \) times to obtain hash values with different hash lengths of 256-, 512-, and 1024-bit. Every time, the total number of changed bits between the new and the original hash values is calculated.

Tables 5–7 are the comparison results with other hash algorithms, and the results demonstrate that, for all the values belonging to \( N \), the mean changed bit number \( \overline{B} \) is very close to the ideal number of changed bits \( \text{len}/2 \), from which we can infer that the hash algorithm has strong capability of confusion and diffusion. Meanwhile, the mean changed probability \( P \) is very close to the ideal value of 50%, which is one of the desired features of confusion. Another good feature of the hash algorithm is that both \( \Delta P \) and \( \Delta B \)
are very small for all the tests, which means that the confusion and diffusion capability is very stable.

4.5. Collision Analysis

4.5.1. Meet-in-the-Middle Attack. To seek a collision, the meet-in-the-middle attack is conducted on intermediate variables, and a collision could be found if two intermediate variables match [22, 23]. This type of attack is invalid for the proposed hash algorithm, due to the initial values of EQM serving as keys, which can make the inverse computation extremely difficult. Hence, the proposed hash algorithm can resist the meet-in-the-middle attack.

4.5.2. Collision Analysis. To perform a collision analysis, message $M_1$ with the length of $L=50$ len is randomly generated, and its len-bit hash values are calculated and stored in ASCII form (8-bit per character). Then, we randomly change one bit to $M_1$, calculate its hash value, and compare two hash values to obtain the absolute difference $d$ between two hash values using the following equation [12]:

$$d = \sum_{i=1}^{\text{len}/8} |\text{dec}(e_i) - \text{dec}(e'_i)|,$$

where $e_i$ and $e'_i$ denote the $i$-th ASCII character of two hash values and the function $\text{dec}$ maps an ASCII character to its decimal value. The theoretical value of average absolute distance per character is 85.3333.

In Table 8, we present the minimum, maximum, and mean values of the absolute difference between two hash values, from which we can infer that when we set $h=256$ and 512, the results of the proposed hash algorithm are as good as some existing hash algorithms, such as SHA-2, SHA-3, and other chaos-based hash algorithms.

4.6. Rainbow Table Resistance Analysis. Rainbow table is a practical example of space/time tradeoff; it uses more computer processing time at the cost of less storage when calculating a hash value on every attempt or less processing time and more storage when comparing to a simple lookup table with one entry per hash. Use of a key derivation function that employs a salt makes this attack ineffective [19]. In the proposed hash algorithm, we took two measures to make the rainbow table attack ineffective. (1) One-time keys: we assign different one-time keys by the key sequence sampled from noise to different users according to their identifications. (2) Random salt: as for equation (2), we add salt derived from the key in each iteration through perturbing the exponent $k$ to make the rainbow table attack ineffective.

Figure 5: Distribution of 256-, 512-, and 1024-bit hash values of a null string.
4.7. Speed Analysis. In order to analyze the computation speed, we implemented the proposed hash algorithm on a PC with 2.50 GHz Intel Core i7-6500U, 16G Memory and Windows 10 operation system, and the tested message consists of 20,000 ASCII characters; the speed is about 131.2 Mbps with $N = 2048$. Experiments showed that the running speed is unaffected by the hash value length.

4.8. Computational Complexity. The computational complexity [16] of the proposed hash algorithm depends on the number of different bits.

![Figure 6: Distribution of the total number of different bits. (a) 256-bit hash value. (b) 512-bit hash value. (c) 1024-bit hash value.](image)

**Table 2: Statistical result for a 256-bit hash value.**

| $N$   | $B_{\text{min}}$ | $B_{\text{max}}$ | $\bar{B}$  | $P$ (%) | $\Delta B$ | $\Delta P$ (%) |
|-------|------------------|------------------|-----------|---------|-----------|-------------|
| 256   | 107              | 145              | 128.87    | 50.34   | 7.59      | 2.96        |
| 512   | 99               | 150              | 128.32    | 50.12   | 7.95      | 3.11        |
| 1024  | 99               | 150              | 128.29    | 50.11   | 7.81      | 3.05        |
| 2048  | 99               | 152              | 128.15    | 50.06   | 7.80      | 3.05        |
| 4096  | 99               | 153              | 128.05    | 50.02   | 7.86      | 3.07        |

**Table 3: Statistical results for a 512-bit hash value.**

| $N$   | $B_{\text{min}}$ | $B_{\text{max}}$ | $\bar{B}$  | $P$ (%) | $\Delta B$ | $\Delta P$ (%) |
|-------|------------------|------------------|-----------|---------|-----------|-------------|
| 256   | 229              | 289              | 257.21    | 50.23   | 11.33     | 2.21        |
| 512   | 222              | 293              | 257.26    | 50.24   | 11.56     | 2.26        |
| 1024  | 220              | 293              | 256.47    | 50.09   | 11.48     | 2.24        |
| 2048  | 220              | 299              | 256.08    | 50.01   | 11.42     | 2.23        |
| 4096  | 220              | 299              | 255.99    | 49.99   | 11.30     | 2.21        |
message length and iterations of the EQM. For any message \( M \) with the character length \( n \), there are \( n \) times to transform it into a parameter sequence, and the time complexity is \( O(n) \). For the EQM, there are \( n + 300 + \text{len}/16 \) iterations with varying parameter; hence, the time complexity is \( O(n) \). Therefore, the total computational complexity of the proposed hash algorithm is \( O(n) \).

Table 4: Statistical results for a 1024-bit hash value.

| \( N \) | \( B_{\text{min}} \) | \( B_{\text{max}} \) | \( \bar{B} \) | \( P(\%) \) | \( \Delta B \) | \( \Delta P(\%) \) |
|-------|----------------|----------------|----------|-----------|--------|-----------|
| 256   | 470            | 549            | 511.33   | 49.93     | 15.84  | 1.55      |
| 512   | 469            | 555            | 511.03   | 49.90     | 15.52  | 1.51      |
| 1024  | 462            | 562            | 511.41   | 49.94     | 15.94  | 1.56      |
| 2048  | 456            | 562            | 511.70   | 49.97     | 16.16  | 1.58      |
| 4096  | 451            | 567            | 512.20   | 50.02     |        |           |

Table 5: Comparison with other algorithms for 256-bit hash value and \( N = 2048 \).

| Hash algorithm      | \( B_{\text{min}} \) | \( B_{\text{max}} \) | \( \bar{B} \) | \( P(\%) \) | \( \Delta B \) | \( \Delta P(\%) \) |
|---------------------|----------------|----------------|----------|-----------|--------|-----------|
| SHA2-256            | 100            | 155            | 127.98   | 50.00     | 8.10   | 3.16      |
| SHA3-256 [24]       | 100            | 154            | 127.96   | 49.98     | 8.04   | 3.14      |
| [6]                 | 101            | 168            | 128.08   | 50.03     | 8.12   | 3.21      |
| [22]                | 103            | 155            | 128.81   | 49.93     | 8.13   | 3.18      |
| [23]                | 102            | 161            | 128.00   | 50.00     | 8.02   | 3.13      |
| Our hash algorithm (256-bit) | 99        | 152            | 128.15   | 50.06     | 7.80   | 3.05      |

Table 6: Comparison with other algorithms for 512-bit hash value and \( N = 2048 \).

| Hash algorithm      | \( B_{\text{min}} \) | \( B_{\text{max}} \) | \( \bar{B} \) | \( P(\%) \) | \( \Delta B \) | \( \Delta P(\%) \) |
|---------------------|----------------|----------------|----------|-----------|--------|-----------|
| SHA2-512            | 215            | 292            | 256.27   | 50.05     | 11.51  | 2.25      |
| SHA3-512            | 216            | 287            | 254.75   | 49.76     | 11.32  | 2.21      |
| [22]                | 219            | 296            | 255.89   | 49.98     | 11.27  | 2.20      |
| [23]                | 223            | 296            | 256.10   | 50.02     | 11.12  | 2.17      |
| Our hash algorithm (512-bit) | 220        | 294            | 255.69   | 49.94     | 11.45  | 2.24      |

Table 7: Comparison with other algorithms for 1024-bit hash value and \( N = 2048 \).

| Hash algorithm      | \( B_{\text{min}} \) | \( B_{\text{max}} \) | \( \bar{B} \) | \( P(\%) \) | \( \Delta B \) | \( \Delta P(\%) \) |
|---------------------|----------------|----------------|----------|-----------|--------|-----------|
| [22]                | 450            | 565            | 511.89   | 49.99     | 16.08  | 1.57      |
| [23]                | 461            | 572            | 512.06   | 50.00     | 15.96  | 1.55      |
| Our hash algorithm (1024-bit) | 450        | 562            | 511.72   | 49.97     | 15.90  | 1.55      |

Table 8: Absolute difference for hash values for \( N = 10,000 \).

| Absolute difference | Maximum \( d \) | Minimum \( d \) | Mean \( d \) | Mean \( d \) per character |
|---------------------|---------------|---------------|-------------|--------------------------|
| 256-bit [22]        | 4,118         | 1,758         | 2,878       | 89.938                   |
| 256-bit [23]        | 4,138         | 1,686         | 2,732       | 85.375                   |
| SHA2-256            | 3,757         | 1,786         | 2,776       | 86.746                   |
| SHA3-256            | 3,525         | 1,825         | 2,741       | 85.675                   |
| Our hash algorithm (256-bit) | 3,931     | 1,573         | 2,754       | 86.081                   |
| 512-bit [22]        | 7,369         | 3,958         | 5,476       | 85.563                   |
| 512-bit [23]        | 7,210         | 3,971         | 5,463       | 85.359                   |
| SHA2-512            | 7,132         | 3,840         | 5,430       | 84.852                   |
| SHA3-512            | 6,481         | 5,678         | 5,612       | 87.684                   |
| Our hash algorithm (512-bit) | 7,036        | 4,532         | 5,642       | 87.293                   |
| Our hash algorithm (1024-bit) | 13,041    | 8,439         | 10,781      | 84.226                   |
5. Conclusion

A novel hash algorithm is constructed based on the EQM; three measures, including assigning unique one-time keys adaptively, key expansion, and hash length extension, are taken to resist against the rainbow table attack. Three steps of message pre-encoding, message absorption, and generation of hash value are implemented. The hash algorithm is so flexible that it can be keyed or unkeyed and can generate 256-bit, 512-bit, 1024-bit, or longer hash value through a parameter switcher. Any characters, including single-byte and double-byte characters, can be transformed into a parameter sequence for EQM to absorb. Simulation results and performance analysis demonstrated the effectiveness and flexibility of the proposed hash algorithm. In the future, we intend to research chaos-based parallel hash algorithm that can resist attacks from the quantum computing.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors’ Contributions

Hongjun Liu was the major contributor and contributed to algorithm design; Abdurahman Kadir contributed to algorithm optimization; Chao Ma was responsible for algorithm design; Abdurahman Kadir contributed to algorithm design; Chao Ma was responsible for algorithm design; and Chengbo Xu contributed to diagram design.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (no. 61662073).

References

[1] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, “Blockchain,” Business & Information Systems Engineering, vol. 59, no. 3, pp. 183–187, 2017.
[2] M. J. Schiereck, Sha-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Federal Inf. Process. Stds. (NIST FIPS)-202, Gaithersburg, MD, USA, 2015.
[3] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full SHA-1,” Advances in Cryptology-CRYPTO 2005, vol. 3621, pp. 17–36, 2005.
[4] A. Sotirov, M. Stevens, J. Appelbaum et al., “MD5 considered harmful today, creating a rogue CA certificate,” in Proceedings of the 25th Annual Chaos Communication Congress, Leipzig, Germany, January 2008.
[5] T. Fox-Brewer, Google just “Shattered” an Old Crypto Algorithm-Here’s Why That’s Big for Web Security, Forbes, Waltham, MA, USA, 2017.
[6] J. Guo, G. Liao, G. Liu, M. Liu, K. Qiao, and L. Song, “Practical collision attacks against round-reduced SHA-3,” Journal of Cryptology, vol. 33, no. 1, pp. 228–270, 2019.
[7] M. A. Liu, S. Jamali, and N. Khasmakh, “A novel keyed parallel hashing scheme based on a new chaotic system,” Chaos, Solitons & Fractals, vol. 87, pp. 216–225, 2016.
[8] D. Xiao, X. Liao, and S. Deng, “One-way Hash function construction based on the chaotic map with changeable-parameter,” Chaos, Solitons & Fractals, vol. 24, no. 1, pp. 65–71, 2005.
[9] W. Guo, X. Wang, D. He, and Y. Cao, “Cryptanalysis on a parallel keyed hash function based on chaotic maps,” Physics Letters A, vol. 373, no. 36, pp. 3201–3206, 2009.
[10] H. S. Kwok and W. K. S. Tang, “A chaos-based cryptographic hash function for message authentication,” International Journal of Bifurcation and Chaos, vol. 15, no. 12, pp. 4043–4050, 2005.
[11] S. Deng, Y. Li, and D. Xiao, “Analysis and improvement of a chaos-based Hash function construction,” Communications in Nonlinear Science and Numerical Simulation, vol. 15, no. 5, pp. 1338–1347, 2010.
[12] H. Liu, A. Kadir, and J. Liu, “Keyed hash function using hyper chaotic system with time-varying parameters perturbation,” IEEE Access, vol. 7, no. 1, pp. 37211–37219, 2019.
[13] J. S. Teh, K. Tan, and M. Alawida, “A chaos-based keyed hash function based on fixed point representation,” Cluster Computing, vol. 22, no. 2, pp. 649–660, 2019.
[14] Y. Li and G. Ge, “Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security,” Multimedia Tools and Applications, vol. 78, no. 13, pp. 17973–17994, 2019.
[15] Y. Li, “Collision analysis and improvement of a hash function based on chaotic tent map,” Optik, vol. 127, no. 10, pp. 4484–4489, 2016.
[16] Y. Li and X. Li, “Chaotic hash function based on circular shifts with variable parameters,” Chaos, Solitons & Fractals, vol. 91, pp. 639–648, 2016.
[17] Y. Li, G. Ge, and D. Xia, “Chaotic hash function based on the dynamic S-Box with variable parameters,” Nonlinear Dynamics, vol. 84, no. 4, pp. 2387–2402, 2016.
[18] J. Horálek, F. Holík, O. Horák, L. Petr, and V. Sobeslav, “Analysis of the use of rainbow tables to break hash,” Journal of Intelligent & Fuzzy Systems, vol. 32, no. 2, pp. 1523–1534, 2017.
[19] H.-J. Petr, S. Hong, and J. Shin, “A novel secure and efficient hash function with extra padding against rainbow table attacks,” Cluster Computing, vol. 21, no. 1, pp. 1161–1173, 2018.
[20] F. J. S. Moreira, Chaotic Dynamics of Quadratic Maps, IMPA, Colchester, UK, 1993.
[21] H. Liu, Y. Zhang, A. Kadir, and Y. Xu, “Image encryption using complex hyper chaotic system by injecting impulse into parameters,” Applied Mathematics and Computation, vol. 360, pp. 83–93, 2019.
[22] A. Xu, H. Yahyaoui, and M. Almulia, “Keyed hash function based on a chaotic map,” Information Sciences, vol. 186, no. 1, pp. 249–264, 2012.
[23] A. Kanso and M. Ghebleh, “A fast and efficient chaos-based keyed hash function,” Communications in Nonlinear Science and Numerical Simulation, vol. 18, no. 1, pp. 109–123, 2013.
[24] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “The keccak sponge function family, 2011, submission to NIST’s SHA-3 competition,” 2011.