A Highly Parallelizable Hash Algorithm Based on Latin Cubes

Ming Xu
Department of Mathematics and Physics,
Shijiazhuang Tiedao University, China
13400115751@126.com

Abstract: *Latin cubes are the high-dimensional form of Latin squares. Latin cubes have discreteness, uniformity and 3D attribute. There have been some applications of Latin squares in hash algorithms, but few applications of Latin cubes in this field. In this paper, a highly parallelizable hash algorithm based on four Latin cubes of order 4 is proposed. The parallelism is reflected in two aspects: on the one hand, the whole message is divided into several blocks, and all the blocks are processed in parallel; on the other hand, each block is further divided into several channels, and these channels are also processed in parallel. The whole hash procedure is based on four fixed Latin cubes. By the aid of uniformity and 3D attribute of Latin cubes, the algorithm has good statistical performances and strong collision resistance. Furthermore, the parallel structure makes the algorithm have satisfactory computation speed. Therefore the algorithm is quite suitable for the current applications of communication security.*

Keywords: *Hash algorithm, latin cubes, 3D attribute, parallelism.*

Received February 28, 2023; accepted May 24, 2023
https://doi.org/10.34028/iajit/20/6/10

1. Introduction

Hash algorithm is a special kind of cryptographic algorithm. It transforms a message with arbitrary length into a hash value with fixed length. Hash algorithms have important applications in many fields, such as file checksum, authentication protocol and digital signature. In recent years, accompanied with the rapid development of blockchain technique, the importance of hash algorithms has been realized by more and more people. In many links of a blockchain, such as the computation of node addresses and the proof of work in Bitcoin mining, hash algorithms always play very important roles.

Hash algorithms should satisfy compression, irreversibility, collision resistance, etc. Among these properties, collision resistance is crucial for the security of hash algorithms. Many hash algorithms have been attacked due to the weak ability of collision resistance [18, 21, 24]. Chaotic systems have high sensitivity to tiny changes in initial values and system parameters, which can provide strong ability of collision resistance for hash algorithms. Moreover, random-like behavior of chaotic systems can provide good statistical performance for hash algorithms. Then lots of hash algorithms based on chaos have been proposed [5, 9, 10, 13]. Apart from chaos, there are also some other instruments widely used in hash algorithms, such as lattice theory, neural network and Latin squares.

In [19], Snášel designs a hash algorithm based on quasigroups, i.e., Latin squares. To ensure security, the algorithm uses Latin squares of large order. To reduce complexity, the algorithm uses modular subtraction Latin squares to replace ordinary Latin squares. However, the special structure of modular subtraction Latin squares brings security vulnerabilities to the algorithm, consequently the algorithm is attacked by Slaminková in [18]. The main reason for being attacked is that the algorithm in [19] uses 2D mappings generated from Latin squares to construct the compression function. Latin cubes can overcome this defect effectively.

Latin cubes are the high-dimensional form of Latin squares. There have been some applications of Latin squares in hash algorithms [7, 19], but few applications of Latin cubes in this field. Analogous to Latin squares, Latin cubes also have discreteness and uniformity, which can make the hash algorithms have good statistical performance. Unlike Latin squares, Latin cubes have 3D attribute, which can make the hash algorithms have strong diffusibility.

The structures of traditional hash algorithms, such as Merkle-Damgard structure [16], HAIFA structure [4] and Sponge structure [3], always belong to the sequential model, that is, the current message unit cannot be processed until the previous message units have been completed. Sequential model is not very suitable for parallel computing environment. In reality, the CPUs in common use generally have a few to several dozen cores, some special processors even have hundreds of cores. To fully utilize the processing power of multi-core processors, research and development of the parallel hash algorithms become imperative [8, 12, 20, 22, 25].

In this paper, we use Latin cubes to construct a highly parallel hash algorithm. The main contributions are stated as follows:

1. For the first time, Latin cubes are utilized to construct a hash algorithm. Latin cubes have discreteness and uniformity, which are the basic requirements for the cryptographic applications. In particular, Latin cubes have 3D attribute, which can bring strong diffusivity for the hash algorithm. Furthermore, Latin cubes have close relations with some more complex configurations in combinatorial design theory. The attempts in this work can stimulate the research of other configurations in hash algorithms, and even other fields of cryptography.
2. In the hash algorithm, Latin cubes are used as 3D state tables. The hash process is essentially the shifting and selecting process of the state tables. By the 3D attribute of Latin cubes, there are totally three states to determine each iterative value in the hash process, which contributes to the strong collision resistance of the algorithm.
3. The structure of the hash algorithm is highly parallel, then it can work efficiently in the parallel computing environment. The parallelism of the algorithm is reflected in two aspects: the whole message is divided into several blocks, the processing of each block is parallel; each block is divided into several channels, the processing of each channel is parallel too. The parallelism can improve the efficiency of the algorithm greatly. Furthermore, it can make the hash value have even sensitivity to the message units at different positions.
4. The paper is organized as follows: In section 1, the background is stated. In section 2, some basic definitions and conclusions are introduced. In section 3, concrete process of the algorithm is described. In section 4, performance of the algorithm is evaluated. Finally, we conclude the work in section 5.

2. Preliminaries

The algorithm uses four Latin cubes of order 4 as the state tables. The definitions of Latin square and Latin cube are stated as follows:

- *Definition* 1: A Latin square of order n is an $n \times n$ array (n rows and n columns) defined on n-set $S=\{0, 1, ..., n-1\}$, satisfying each number appears exactly once in each row and each column.
- *Definition* 2: A Latin cube of order n is an $n \times n \times n$ cube (n rows, n columns and n files) defined on n-set $S=\{0, 1, ..., n-1\}$, satisfying each number appears exactly once in each row, each column and each file.

In a Latin cube $A=(a_{ijk})_{n\times n\times n}$, if any two subscripts are fixed, then the element a_{ijk} will take all the numbers in S when the other subscript varies from 0 to n-1.

By the two definitions, each plane of a Latin cube is a Latin square. Therefore, Latin cubes have discreteness and uniformity, as Latin squares do. In particular, Latin cubes have 3D attribute.

There are totally 55296 Latin cubes of order 4 [14], but not each of them is suitable for hash algorithms. After a large number of experiments, we take the four Latin cubes in Figure 1 to construct the proposed hash algorithm. Each Latin cube is shown in terms of 4 Latin squares.

$$A_0: \begin{array}{cccc} 2&0&3&1\\ 1&3&0&2\\ 3&1&2&0\\ 0&2&1&3 \end{array} \quad \begin{array}{cccc} 3&1&2&0\\ 0&2&1&3\\ 1&0&3&2\\ 2&3&0&1 \end{array} \quad \begin{array}{cccc} 0&2&1&3\\ 3&1&2&0\\ 2&3&0&1\\ 1&0&3&2 \end{array} \quad \begin{array}{cccc} 1&3&0&2\\ 2&0&3&1\\ 0&2&1&3\\ 3&1&2&0 \end{array}$$

$$A_1: \begin{array}{cccc} 3&1&2&0\\ 0&2&1&3\\ 2&0&3&1\\ 1&3&0&2 \end{array} \quad \begin{array}{cccc} 2&0&3&1\\ 1&3&0&2\\ 0&1&2&3\\ 3&2&1&0 \end{array} \quad \begin{array}{cccc} 1&3&0&2\\ 2&0&3&1\\ 3&2&1&0\\ 0&1&2&3 \end{array} \quad \begin{array}{cccc} 0&2&1&3\\ 3&1&2&0\\ 1&3&0&2\\ 2&0&3&1 \end{array}$$

$$A_2: \begin{array}{cccc} 1&3&2&0\\ 0&2&3&1\\ 2&0&1&3\\ 3&1&0&2 \end{array} \quad \begin{array}{cccc} 2&0&1&3\\ 3&1&0&2\\ 0&3&2&1\\ 1&2&3&0 \end{array} \quad \begin{array}{cccc} 3&1&0&2\\ 2&0&1&3\\ 1&2&3&0\\ 0&3&2&1 \end{array} \quad \begin{array}{cccc} 0&2&3&1\\ 1&3&2&0\\ 3&1&0&2\\ 2&0&1&3 \end{array}$$

$$A_3: \begin{array}{cccc} 0&2&1&3\\ 3&1&2&0\\ 1&3&0&2\\ 2&0&3&1 \end{array} \quad \begin{array}{cccc} 1&3&0&2\\ 2&0&3&1\\ 3&2&1&0\\ 0&1&2&3 \end{array} \quad \begin{array}{cccc} 2&0&3&1\\ 1&3&0&2\\ 0&1&2&3\\ 3&2&1&0 \end{array} \quad \begin{array}{cccc} 3&1&2&0\\ 0&2&1&3\\ 2&0&3&1\\ 1&3&0&2 \end{array}$$

Figure 1. The 4 Latin cubes in the algorithm.

Apart from Latin cubes, another technique used in the proposed algorithm is Logistic map, which is defined by Definition 3.

- *Definition* 3: The Logistic map is defined as:

$$x_n = ux_{n-1}(1-x_{n-1}) \; n = 1, 2, 3, ... \qquad (1)$$

x_n is a floating-point number in (0, 1). u is a system parameter, $0 \le u \le 4$. when $u > 3.573815$, this system exhibits chaotic characteristics.

Although Logistic map has some defects [9], it does not influence security of the hash algorithm, because the Logistic map is only used to generate some initial value sequences. In the proposed algorithm, the core components are the four Latin cubes of order 4, and the Logistic map is only taken as a seed generator of the hash algorithm.

3. The Hash Algorithm

3.1. Secret Key of the Algorithm

The secret key K consists of: system parameter $u_0 \in (3.574, 4]$ and initial value $x_0 \in (0,1]$ of the Logistic map; vector V of 80 bits which is expressed as a concatenation of 40 2-bit variables V_i, that is, $V=V_0V_1...V_{39}$. The counts in the algorithm all start at 0.

3.2. Description of the Hash Process

The inputs of the algorithm are secret key K and a message. The length of messages can be arbitrary. The output of the algorithm is an N-bit hash value. To resist birthday attacks, N should satisfy $N\geq128$. Actually, N can be set to arbitrary values greater than or equal to 128 in the proposed algorithm, as long as we adjust the length of message block accordingly. For convenience, we set N=128 in this paper. The proposed hash process consists of: message extension, initialization, message block processing, and hash value generation.

3.2.1. Message Extension

Given a message, we pad it to obtain a message M satisfying that its length is a multiple of 2048 bits (256 bytes). Specifically, if the length of original message is n bits, then we pad it with m bits $(010101...01)_2$ satisfying $(n+m)$ mod 2048=2048-64. The left 64 bits are reserved to denote the length of original message. The padded message M is then divided into several blocks with length of 2048 bits, i.e., $M=(M_0, M_1, ..., M_{l-1})$. For each M_i, we perform hash operations in parallel. The intermediate hash value of M_i is denoted as H_i. As can be seen, the block length is a large number, it is because there are totally four parallel channels to process each block in the proposed algorithm.

3.2.2. Initialization

We totally use 40 working Latin cubes to perform 40 transformations on message M. The 40 working Latin cubes $L_i(i=0, ..., 39)$ are generated by vector $V=V_0V_1...V_{39}$ and the four Latin cubes in Figure 1. The assignment of the 40 working Latin cubes is done following Equation (2):

$$Li \leftarrow Ayi\ (0 \leq i \leq 39) \qquad (2)$$

Apart from the assignment of 40 working Latin cubes, the initialization also includes the generation of initial value sequence. Assume there are l message blocks, then we need to generate an initial value sequence with length $40l$. The concrete process is described as follows:

- *Step* 1: Firstly, iterate Logistic map n times. n is the length of original message. The parameter and initial value of Logistic map are u_0 and x_0 respectively. Discard the n values, then continue iterating Logistic map $40l$ times to generate a chaotic sequence $x=(x_0, x_1, ... , x_{40l-1})$.
- *Step* 2: Divide x into l sub-sequences sequentially, each sub-sequence $x_i(0 \leq i< l)$ has length of 40.
- *Step* 3: Process each sub-sequence x_i by Equation (3):

$$[lx_i, fx_i] = sort(x_i) \qquad (3)$$

where [,]=*sort*() is the sequencing index function. After ascending to x_i, a new sequence fx_i is obtained. lx_i are the index values of fx_i. Denote lx_i as $lx_i=(c^i_0, c^i_1, ..., c^i_{39})$.

- *Step* 4: For each element c^i_j in lx_i ($0\leq j\leq39$, $0\leq i<l$), take its first 8 bits, then divide the 8 bits into 4 2-bit units, denoted as $c^i_j=(c^i_{j,0}, c^i_{j,1}, c^i_{j,2}, c^i_{j,3})$.

In the algorithm, each message block is divided into several channels. Considering that the Central Processing Unit (CPU) of our computer has 4 cores, we set the number of channels to 4 in the algorithm description. Actually, the number of channels can be adjusted flexibly according to the running platform. The sequences c^i_j ($0\leq j\leq39$, $0\leq i<l$) in Step 4 are used as the initial value sequences on each channel. Specifically, the first 2-bit units $c^i_{j,0}$ are used as the initial value sequence on channel 0, the next 2-bit units $c^i_{j,1}$ are used as the initial value sequence on channel 1, and so on.

3.2.3. Message Block Processing

In the proposed algorithm, the l message blocks (M_0, M_1, ..., M_{l-1}) are processed in parallel. Each message block consists of 2048 bits (256 bytes). Without loss of generality, we take M_i as an instance to demonstrate the message block processing. M_i is firstly divided into 256 characters (each character consists of 8 bits), i. e. $M_i=(m^i_0, m^i_1, ..., m^i_{255})$. For each character m^i_j, we re-divide it into 4 2-bit units, i.e. $m^i_j=(m^i_{j,0}, m^i_{j,1}, m^i_{j,2}, m^i_{j,3})$. These units range from 0 to 3. Then M_i is divided into 4 channels:

- *Channel* 0: It consists of the first unit of each character in M_i. We denote channel 0 as $(m^i_{0,0}, m^i_{1,0}, m^i_{2,0}, ... , m^i_{255,0})$;
- *Channel* 1: It consists of the second unit of each character in M_i. We denote channel 1 as $(m^i_{0,1}, m^i_{1,1}, m^i_{2,1}, ... , m^i_{255,1})$;
- *Channel* 2: It consists of the third unit of each character in M_i. We denote channel 2 as $(m^i_{0,2}, m^i_{1,2}, m^i_{2,2}, ... , m^i_{255,2})$;
- *Channel* 3: It consists of the last unit of each character in M_i. We denote channel 3 as $(m^i_{0,3}, m^i_{1,3}, m^i_{2,3}, ... , m^i_{255,3})$.

Next, we process the 4 channels in parallel. To be specific, we perform 40 transformations on each channel as described in Tables 1, 2, 3, and 4.

The transformations on channel 0 are described by Equations (4) and (5).

- *Cycle* 0:

$$\begin{cases} t_{0,0} = L_0\,(c^i{}_{0,0}, m^i{}_{0,0}, m^i{}_{1,0}) \\ t_{0,j} = L_0\,(t_{0,j-1}, m^i{}_{j,0}, m^i{}_{j+1,0})(1 \leq j \leq 254) \\ t_{0,255} = L_0\,(t_{0,254}, m^i{}_{255,0}, t_{0,0}) \end{cases} \qquad (4)$$

- Cycle k: ($1\leq k\leq39$):

$$\begin{cases} t_{k,0} = L_k\,(c^i{}_{k,0}, t_{k-1,0}, t_{k-1,1}) \\ t_{k,j} = L_k\,(t_{k,j-1}, t_{k-1,j}, t_{k-1,j+1})(1 \leq j \leq 254) \\ t_{k,255} = L_k\,(t_{k,254}, t_{k-1,255}, t_{k,0}) \end{cases} \qquad (5)$$

In Equations (4) and (5), L_i ($0\leq i\leq39$) are the 40 working Latin cubes designated in Section 3.2.2, $c^i_{j,k}$ ($0 \leq i\leq l-1$, $0\leq j \leq39$, $0\leq k \leq3$) are the initial values generated in Section 3.2.2. $t_{k,j}$ ($0 \leq k\leq 39$, $0 \leq j\leq 255$) are the iterative values.

The transformation formulas on the other channels are similar to the formulas on channel 0.

Table 1. The 40 transformations on channel 0.

		$m^i_{0,0}$	$m^i_{1,0}$	$m^i_{2,0}$	…	$m^i_{255,0}$
L_0	$c^i_{0,0}$	$t_{0,0}$	$t_{0,1}$	$t_{0,2}$	…	$t_{0,255}$
L_1	$c^i_{1,0}$	$t_{1,0}$	$t_{1,1}$	$t_{1,2}$	…	$t_{1,255}$
⋮	⋮	⋮	⋮	⋮	…	⋮
L_{39}	$c^i_{39,0}$	$\boxed{t_{39,0}}$	$t_{39,1}$	$\boxed{t_{39,2}}$	…	$t_{39,255}$

Table 2. The 40 transformations on channel 1.

		$m^i_{0,1}$	$m^i_{1,1}$	$m^i_{2,1}$	…	$m^i_{255,1}$
L_0	$c^i_{0,1}$	$t'_{0,0}$	$t'_{0,1}$	$t'_{0,2}$	…	$t'_{0,255}$
L_1	$c^i_{1,1}$	$t'_{1,0}$	$t'_{1,1}$	$t'_{1,2}$	…	$t'_{1,255}$
⋮	⋮	⋮	⋮	⋮	…	⋮
L_{39}	$c^i_{39,1}$	$\boxed{t'_{39,0}}$	$t'_{39,1}$	$\boxed{t'_{39,2}}$	…	$t'_{39,255}$

Table 3. The 40 transformations on channel 2.

		$m^i_{0,2}$	$m^i_{1,2}$	$m^i_{2,2}$	…	$m^i_{255,2}$
L_0	$c^i_{0,2}$	$t''_{0,0}$	$t''_{0,1}$	$t''_{0,2}$	…	$t''_{0,255}$
L_1	$c^i_{1,2}$	$t''_{1,0}$	$t''_{1,1}$	$t''_{1,2}$	…	$t''_{1,255}$
⋮	⋮	⋮	⋮	⋮	…	⋮
L_{39}	$c^i_{39,2}$	$\boxed{t''_{39,0}}$	$t''_{39,1}$	$\boxed{t''_{39,2}}$	…	$t''_{39,255}$

Table 4. The 40 transformations on channel 3.

		$m^i_{0,3}$	$m^i_{1,3}$	$m^i_{2,3}$	…	$m^i_{255,3}$
L_0	$c^i_{0,3}$	$t'''_{0,0}$	$t'''_{0,1}$	$t'''_{0,2}$	…	$t'''_{0,255}$
L_1	$c^i_{1,3}$	$t'''_{1,0}$	$t'''_{1,1}$	$t'''_{1,2}$	…	$t'''_{1,255}$
⋮	⋮	⋮	⋮	⋮	…	⋮
L_{39}	$c^i_{39,3}$	$\boxed{t'''_{39,0}}$	$t'''_{39,1}$	$\boxed{t'''_{39,2}}$	…	$t'''_{39,255}$

The output of each channel is the concatenation of every other value in the last round of the iteration (In Tables 1, 2, 3, and 4, we make marks on the outputs by the boxes).

From Equation (5), it can be seen that each iterative value $t_{k,j}$ is determined by three values. They are $t_{k,j-1}$ (the value on the left), $t_{k-1,j}$ (the value above), and $t_{k-1,j+1}$ (the value on the top right). After 40 iterations, each bit of the outputs is related to all the bits of the message. Then a tiny change in the message can result in a big difference in the outputs of the four channels. It makes the algorithm have strong collision resistance.

3.2.4. Hash Value Generation

After obtaining the outputs of the four channels, we further compute the intermediate hash value H_i of M_i ($0 \le i \le l-1$) as follows:

1. Conduct XOR operations on outputs of the four channels, results are denoted as (a^i_0, a^i_1, ..., a^i_{127}), i.e.,

$$(a^i_0, \dots, a^i_{127}) = (t_{39,0} \oplus t'_{39,0} \oplus t''_{39,0} \oplus t'''_{39,0}, \dots, t_{39,254} \oplus t'_{39,254} \oplus t''_{39,254} \oplus t'''_{39,254}) \tag{6}$$

2. Choose one bit from each a^i_j ($0 \le j \le 127$) following Equation (7), then connect them sequentially to generate the 128-bit hash value H_i.

$$h_j = \begin{cases} a^i_j \,\&1 & j \bmod 2 == 1 \\ \left(a^i_j >> 1\right) \&1 & j \bmod 2 == 0 \end{cases} \quad (0 \le j \le 127) \tag{7}$$

In Equation (7), "&" denotes Bitwise AND. After all the intermediate hash values are obtained, we use Equation (8) to Equation (9) to compute the final hash value H.

$$H = H_0 \otimes H_1 \otimes \cdots \otimes H_{l-1} \tag{8}$$

where "$\otimes$" is defined as:

$$\otimes = \begin{cases} (H \oplus H_i) << 1 & i \bmod 2 == 0 \\ (H \oplus H_i) >> 1 & i \bmod 2 == 1 \end{cases} \tag{9}$$

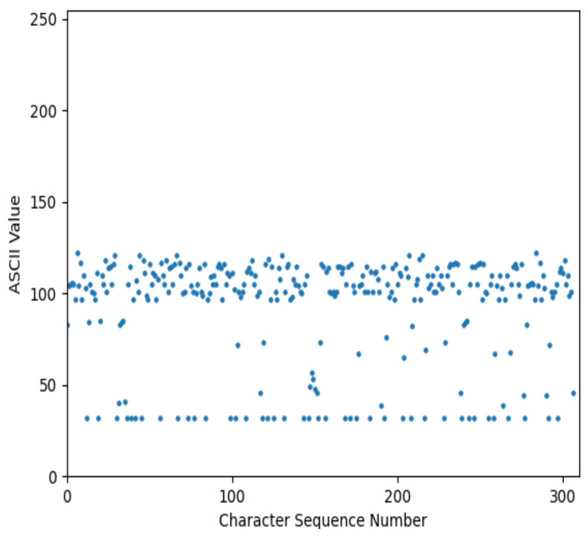
From the algorithm description, it can be seen the algorithm is highly parallelizable. The parallelism is reflected in two aspects: the parallelism of different message blocks and the parallelism of different channels in each message block. If the positions of any two message blocks are exchanged, the final hash value will be totally different, because the hash process of each message block is related to its initial value sequence, and message blocks at different positions correspond to different initial value sequences. For the same reason, if the positions of any two channels in one message block are exchanged, the final hash value will be different too. Furthermore, parallelism makes each message block and each channel have equivalent effects on the final hash value.

4. Performance Evaluation

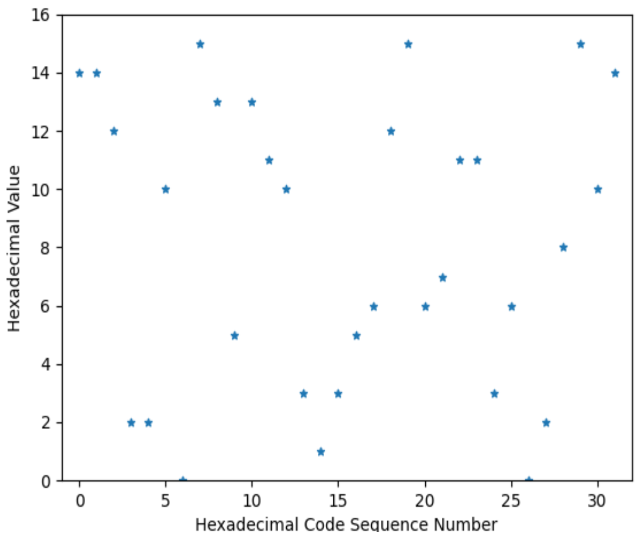
The algorithm is evaluated from the following aspects: uniform distribution of hash values, sensitivity of hash values to messages and secret keys, confusion and diffusion properties, collision resistance and efficiency. Meanwhile, we make comparisons between the algorithm and several representative hash algorithms [1, 2, 6, 12, 23]. For ease of demonstrations, the secret key K is set as: u_0=3.99999, x_0=0.123456789, V=00000000000000000000000000000000000000.
A message M is chosen randomly as:
Shijiazhuang Tiedao University (STU) is a key vocational university under the direct administration of Hebei province. It was early established in 1950. Its predecessor is the Chinese people's Liberation Army Railway Engineering Institute. STU is situated in Chang'an District, Shijiazhuang, Hebei province.

4.1. Uniform Distribution of Hash Values

The uniformity is an important index for the security of hash algorithms [15]. To evaluate the uniformity of hash values, the algorithm is firstly implemented on message *M*. The distributions of message *M* and its hash value are plotted in Figure 2. As shown in Figure 2, the ASCII code of message *M* spreads in a small range [32, 126], while its hexadecimal hash value spreads uniformly. Moreover, we also evaluate the uniformity in an extreme situation, that is, we implement the algorithm on a "blank space" message. The distributions of the special message and its hash value are plotted in Figure 3. In the extreme situation, the hash value still has uniform distribution. All the experimental results in this section demonstrate that there isn't any leak of statistical information in the proposed hash algorithm.

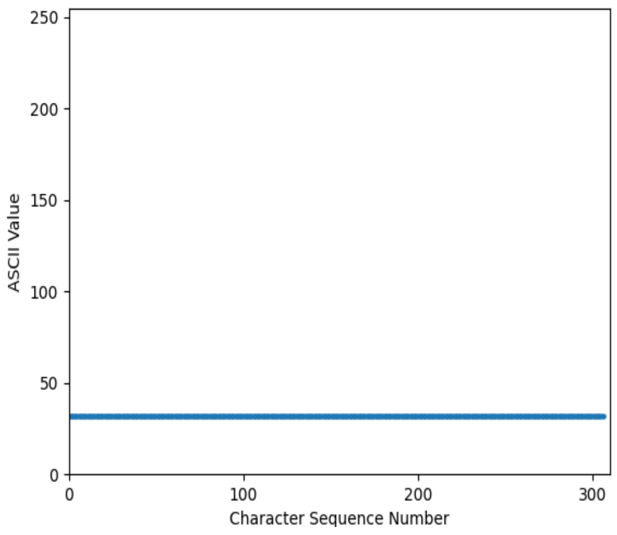


a) Distribution of the message *M* in ASCII code.

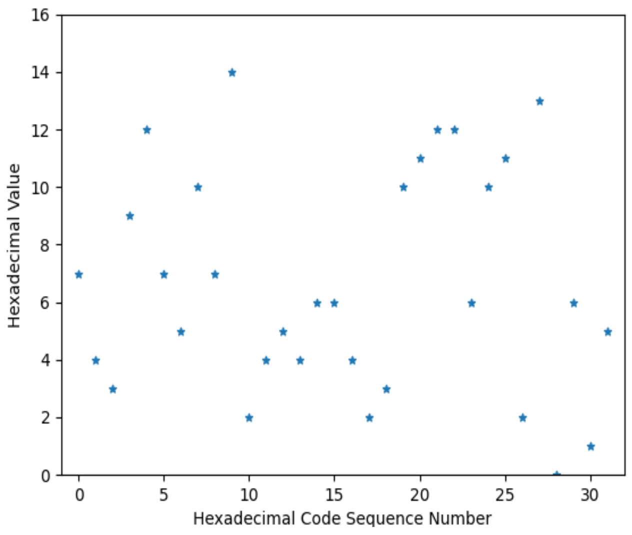


b) Distribution of the hexadecimal hash value (eec22a0fd5dba31356cf67bb36028fae).

Figure 2. Distributions of the message *M* and hexadecimal hash value.



a) Distribution of the all "blank-space" message.



b) Distribution of the hexadecimal hash value (7439c75a7e245466423abcc6ab2d0615).

Figure 3. Distributions of the all "blank-space" message and hexadecimal hash value.

4.2. Hash Sensitivity

The sensitivity of hash values to messages and secret keys is another important index for the security of hash algorithms [24]. To evaluate sensitivity, we implement the algorithm under the following different situations:

- *Situation* 1: The original message *M*.
- *Situation* 2: Change the first character "*S*" of *M* into "*T*".
- *Situation* 3: Change the full stop "." at the end of *M* into question mark "?".
- *Situation* 4: Change the initial value x_0 in secret key K from "0.123456789" to "0.1234567890001".
- *Situation* 5: Change the parameter u_0 in secret key *K* from "3.99999" to "3.99999000001".
- *Situation* 6: Change the vector *V* in secret key *K* from "0000000000000000000000000000000000000000" to "1000000000000000000000000000000000000000".

Table 5 lists the hexadecimal hash values under the six situations and corresponding hamming distances. The binary hash values are depicted in Figure 4. It can be seen that subtle changes of messages or secret keys can bring large differences in hash values, then the proposed algorithm has high sensitivity to both messages and secret keys.

Table 5. The hexadecimal hash values and corresponding hamming distances under the six different situations.

Situation	**Hash value**	**Hamming distance**
1	eec22a0fd5dba31356cf67bb36028fae	0
2	fbb490411d51710abfd8c7347495b80e	60
3	eec76910071486a00f4a480a556bb120	61
4	92ac92b07430a37d5a6b070c50ce9b6d	62
5	1109084680dafb16ba7aabc6252d141c	65
6	d056d4c793e73187e75a75285ddc77cd	65

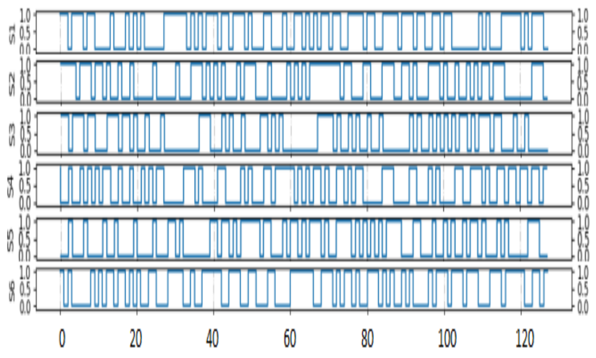


Figure 4. Binary hash values under six different situations.

4.3. Confusion and Diffusion

Confusion and diffusion are two necessary properties of general cryptographic algorithms [17] and not limited to hash algorithms. The confusion and diffusion tests for the proposed algorithm are conducted as follows: choose a message randomly and compute its hash value; modify one bit of the message randomly and compute its hash value; make comparisons between the two hash values, count the number of different bits at the same position, then compute six metrics for confusion and diffusion, which are $B_{\min}$, $B_{\max}$, B_{ave}, ΔB, P and ΔP. The six metrics are specifically defined by Equation (10) to Equation (15).

$$B_{min} = \min\{B_1, B_2, \dots, B_T\} \tag{10}$$

$$B_{max} = \max\{B_1, B_2, \dots, B_T\} \tag{11}$$

$$B_{ave} = \frac{1}{T}\sum_{i=1}^{T} B_i \tag{12}$$

$$\Delta B = \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}(B_i - B_{ave})^2} \tag{13}$$

$$P = \frac{B_{ave}}{N} \times 100\% \tag{14}$$

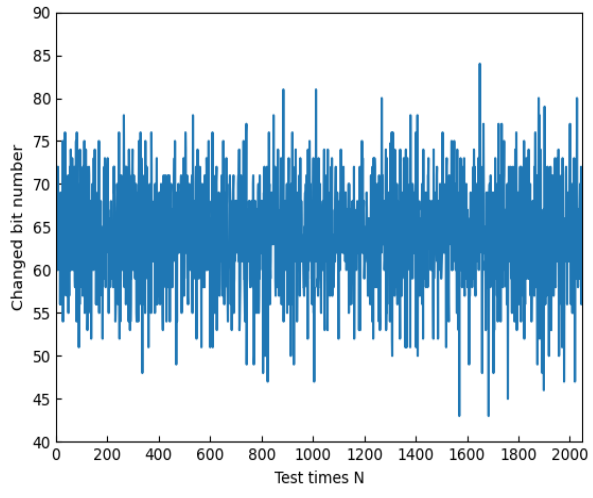
$$\Delta P = \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}(B_i\,/\,N - P)^2 \times 100\%} \tag{15}$$

In Equation (10) to Equation (15), B_i ($i=1, \dots, T$) is the number of changed bits, T is the testing times, and N is the length of hash values. In the simulation experiments, we set N=128, T=256, 512, 1024, 2048, respectively. Table 6 lists experimental results of the six metrics. The corresponding distributions of B_i are shown in Figure 5.

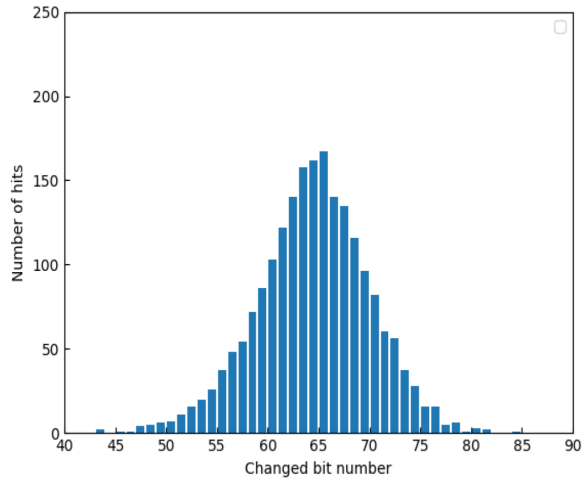
As shown in Table 6, the mean value of B_{ave} is 63.9978, and the mean value of P is 49.9982%. The two experimental results are extremely close to the ideal values of B_{ave} and P, which are 64 bits and 50% respectively. The small values of ΔB and ΔP demonstrate the stable capability of confusion and diffusion. In Figure 5, the histogram of B_i is very close to normal distribution centering on ideal value 64. All the simulation results demonstrate that the algorithm has satisfactory confusion and diffusion properties, then it can resist linear or differential attacks effectively.

Table 6. Statistical results of B_i.

T	**256**	**512**	**1024**	**2048**	**Mean**
B_{min}	51	51	48	47	43
B_{max}	76	63.99609	64.02734	63.96680	64.00067
B_{ave}	63.99609	5.398255	5.485012	5.458532	5.528058
ΔB	5.398255	5.398255	5.485012	5.458532	5.528058
P(%)	49.9969	49.9969	50.0214	49.9741	50.0005
ΔP(%)	4.2174	50.0214	49.9741	50.0005	49.9982



a) Plot of B_i.



b) Histogram of B_i.

Figure 5. Plot and histogram of B_i.

4.4. Collision Resistance

Collision resistance of a hash algorithm means that it is very hard to find two different messages with the same hash value [24]. To realize strong collision resistance, the algorithm uses Latin cubes to conduct 40 rounds of iteration on each channel. By the 3D attribute of Latin cubes, each internal state in the iterative process is related to three internal states on the left, on the top and on the upper-right. Take channel 0 for example, each $t_{k,j}$ in the iterative process is related to $t_{k,j-1}$, $t_{k-1,j}$, and $t_{k-1,j+1}$. The 3D attribute of Latin cubes strengthens the avalanche effect greatly.

We evaluate the collision resistance of the algorithm through 2048 repeated experiments: choose a message randomly, compute its hash value in ASCII code format; change one bit of the chosen message randomly, generate its hash value in ASCII code format as well; make comparisons between the two hash values and count the number of hits, i.e., number of same ASCII characters at same position. The simulation results are shown in Figure 6.

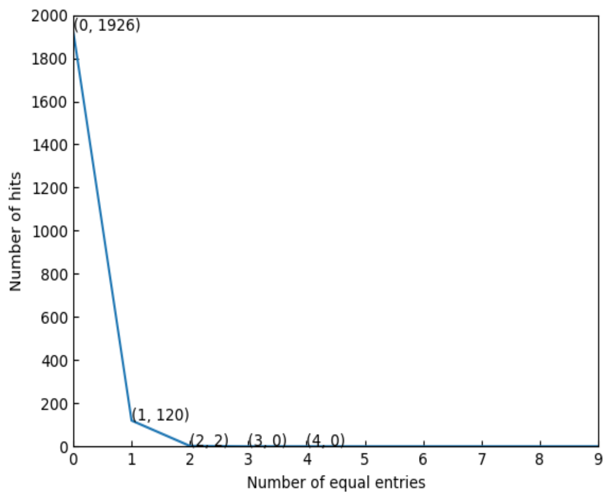


Figure 6. Distributions of the number of hits.

From Figure 6, there are 2 tests with 2 hits, 120 tests with 1 hits, while in 1926 tests, there is not any hit. The maximum of hits is 2, then the collision in the proposed algorithm is quite low.

Furthermore, the absolute difference of the two hash values is computed by Equation (16).

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

In Equation (16), *t*() is a function which converts its inputs into equivalent decimal values. a_i is the *i*th ASCII character in the original hash value, and a_i' is the *i*th ASCII character in the new hash value. Table 7 lists the results of absolute difference *d* in 2048 tests. All the experimental results demonstrate strong collision resistance of the algorithm.

Table 7. Absolute differences of two hash values.

Max	Min	Mean	Mean/character
2297	653	1366.6176	85.4136

4.5. Efficiency

All the simulation experiments are performed under C99, running on a Personal Computer (PC) with Intel Core i7-7500U, four-core, 2.70GHz, 8 GB RAM and Microsoft Windows 10 operation system. To compare the actual running speed, we use 6 different algorithms to process 100KB message. The algorithms [1, 2, 6, 23] are the candidates of SHA-3 in the final round, and the algorithm [2] is the ultimate winner; the algorithm [12] is a representative parallel hash algorithm, and the degree of parallelism in [12] is 2. Partial source code of hash algorithms [1, 2, 6, 23] are from [11]. All these algorithms are implemented on the same platform for 2000 times, and the average running time of each algorithm is listed in Table 8.

Table 8. Average running time.

Algorithm	Ours	Aumasson *et al.* [1]	Bertoni *et al.* [2]
Time(S)	0.009	0.003	0.006
Algorithm	Gauravaram *et al.* [6]	Li and Ge [12]	Wu [23]
Time(S)	0.016	0.01	0.112

From Table 8, it can be seen that the proposed algorithm is faster than algorithms in [6, 12, 23], and slower than algorithms in [1, 2]. The proposed algorithm has satisfactory running speed. The high efficiency is obtained mainly by the parallelism of the proposed algorithm. The parallelism of the proposed algorithm can take full advantage of the multicore computers. The degree of parallelism depends on the number of cores in a computer. Our computer has 4 cores, then in the algorithm description, the number of channels in each message block is set to 4. For a computer with more cores, the number of channels can be set to a larger value. Moreover, limited by the practical running environment, the parallelism of different message blocks cannot be displayed simultaneously. However, we can compute the running time under 4*n*-core environment in theory, because the cost of message separation is very low, which can be ignored. To be specific, if the computer has 4*n* cores, we can process *n* message blocks simultaneously. Compared with the running time under 4-core environment, the time will reduce to 1/*n* approximately. The degree of parallelism can be adjusted flexibly according to the number of cores in the future, considering that the number of cores in computers will be on the increase.

5. Conclusions and Future Work

In this paper, we propose a highly parallelizable hash algorithm based on Latin cubes. This work has three main advantages:

1. Introduce Latin cubes into hash algorithm design for the first time.
2. The algorithm is highly parallel, and the degree of parallelism can be adjusted flexibly according to the number of cores in the computers. This feature is quite adaptive to development trends of computers and internet.
3. Latin cube is a typical configuration in combinatorial design theory. It has close relations to some other configurations, such as orthogonal array. The work in this paper will stimulate the application research of other combinatorial configurations, and these research results will provide more possibilities for the hash algorithm design.

In the algorithm, we use four specific Latin cubes which are chosen by simulation experiments. In the sequential studies, we will discuss which Latin cubes are suitable for hash algorithms from the perspective of combinatorial design theory. Moreover, Latin cubes have close relations with some other combinatorial configurations, such as orthogonal arrays. We will further discuss the application research of other configurations in hash algorithms.

References

[1] Aumasson J., Henzen L., Meier W., and Phan R., *SHA-3 Proposal BLAKE*, NIST, 2008. https://perso.uclouvain.be/fstandae/source_codes/hash_atmel/specs/blake.pdf

[2] Bertoni G., Daemen J., Peeters M., and Assche G., Keer R., *Keccak Implementation Overview*, NIST, 2012. https://keccak.team/files/Keccak implementation-3.2.pdf,

[3] Bertoni G., Daemen J., Peeters M., and Assche G., *Cryptographic Sponge Functions*, NIST, 2011. https://keccak.team/files/CSF-0.1.pdf

[4] Biham E. and Dunkelman O., "A Framework for Iterative Hash Functions-HAIFA," The 2nd NIST Hash Workshop, 2006. https://csrc.nist.rip/groups/ST/hash/documents/DUNKELMAN_talk.pdf

[5] Chenaghlu M., Jamali S., and Khasmakhi N., "A Novel Keyed Parallel Hashing Scheme Based on a New Chaotic System," *Chaos, Solitons and Fractals*, vol. 87, pp. 216-225, 2016. DOI:10.1016/j.chaos.2016.04.007

[6] Gauravaram P., Knudsen L.., Matusiewicz K., Mendel F., and Rechberger C., *Groestl-a SHA-3 Candidate*, NIST, 2011. https://ehash.iaik.tugraz.at/wiki/Groestl

[7] Ghosh R., Verma S., Kumar R., Kumar S., and Ram S., "Design of Hash Algorithm Using Latin Square," *Procedia Computer Science*, vol. 46, pp. 759-765, 2015. https://doi.org/10.1016/j.procs.2015.02.144

[8] Huang Z., "A More Secure Parallel Keyed Hash Function Based on Chaotic Neural Network," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 8, pp. 3245-3256, 2011. https://doi.org/10.1016/j.cnsns.2010.12.009

[9] Kocarev L. and Lian S., *Chaos-Based Cryptography: Theory, Algorithm and Applications*, Springer, 2011. https://link.springer.com/book/10.1007/978-3-642-20542-2

[10] Li Y., Deng S., and Xiao D., "A Novel Hash Algorithm Construction Based on Chaotic Neural Network," *Neural Computing and Applications*, vol. 20, pp. 133-141, 2011. DOI: https://doi.org/10.1007/s00521-010-0432-2

[11] Li Z. and Yang Y., *Programming Implement for Typical Cryptographic Algorithms*, National Defense Industry Press, 2013.

[12] Li Y. and Ge G., "Cryptographic and Parallel Hash Function Based on cross Coupled Map Lattices Suitable for Multimedia Communication Security," *Multimedia Tools and Applications*, vol. 78, pp. 17973-17994, 2019. https://doi.org/10.1007/s11042-018-7122-y

[13] Li Y., Ge G., and Xia D., "Chaotic Hash Function Based on the Dynamic S-Box with Variable Parameters," *Nonlinear Dynamics*, vol. 84, no. 4, pp. 2387-2402, 2016. DOI: 10.1007/s11071-016-2652-1

[14] Mullen G. and Weber R., "Latin Cubes of Order ≤ 5," *Discrete Mathematics*, vol. 32, no. 3, pp. 291-297, 1980. https://doi.org/10.1016/0012-365X(80)90267-8

[15] NIST, "Secure Hash Standard," http://csrc.nist.gov/CryptoToolkit/tkhash.html, Last Visited, 2023.

[16] Rivest R., "The MD4 Message-Digest Algorithm," *LNCS*, vol. 537, pp. 303-311, 1991. https://doi.org/10.1007/3-540-38424-3_22

[17] Shannon C., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28, pp. 656-715, 1949.

[18] Slaminková I. and Vojvoda M., "Cryptanalysis of a Hash Function Based on Isotopy of Quasigroups," *Tatra Mountains Mathematical Publications*, vol. 45, no. 1, pp. 137-149, 2010. DOI:https://doi.org/10.2478/v10127-010-0010-0

[19] Snasel V., Abraham A., Dvorský J., Kromer P., and Platoš J., "Hash Functions Based on Large Quasigroups," *in Proceedings of the Computational Science: 9th International Conference Baton Rouge*, Los Angeles, pp. 521-529, 2009. DOI:https://doi.org/10.1007/978-3-642-01970-8_51

[20] Teh J., Samsudin A., and Masoumi A., "Parallel Chaotic Hash Function Based on the Shuffle-Exchange Network," *Nonlinear Dynamics*, vol. 81, no. 3, pp. 1067-1079, 2015. DOI:10.1007/s11071-015-2049-6

[21] Wang S., Li D., and Zhou H., "Collision Analysis of a Chaos-Based Hash Function with both Modification Detection and Localization Capability," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 2, pp. 780-784, 2012. https://doi.org/10.1016/j.cnsns.2011.06.017

[22] Wang Y., Wong K., and Xiao D., "Parallel Hash Function Construction Based on Coupled Map Lattices," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 7, pp. 2810-2821, 2011. https://doi.org/10.1016/j.cnsns.2010.10.001

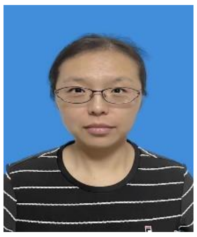
[23] Wu H., *The Hash Function JH*[1], NIST, 2011.http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf. 2011.

[24] Xiao D., Peng W., Liao X., and Xiang T., "Collision Analysis of One Kind of Chaos-Based Hash Function," *Physics Letters A*, vol. 374, no.

10, pp. 1228-1231, 2010. https://doi.org/10.1016/j.physleta.2010.01.006

[25] Xu M., "A New Chaos-Based Image Encryption Algorithm," *The International Arab Journal of Information Technology*, vol. 15, no. 3, pp. 493-498, 2018. https://www.iajit.org/PDF/May%202018%2C%20No.%203/11035.pdf

[26] Zhang P., Zhang X., and Yu J., "A Parallel Hash Function with Variable Initial Values," *Wireless Personal Communications: An International Journal*, vol. 96, no. 2, pp. 2289-2303, 2017. https://doi.org/10.1007/s11277-017-4298-9



Ming Xu female, Born in China, 1981; PhD in Applied Mathematics, Hebei Normal University, Shijiazhuang, China, the degree was earned in 2019; The main research field: Cryptography.