Information security, reliability of data transfer are today an important component of the globalization of information technology. Therefore, the proposed work is devoted to highlighting the results of the design and development of a hacking-resistant algorithm to ensure the integrity of information transfer via digital technology and computer engineering. To solve such problems, cryptographic hashing functions are used. In particular, elements of deterministic Chaos were introduced into the developed cyclic hashing algorithm. The investigation analyzes in detail the strengths and weaknesses of known hashing algorithms. They are shown to have disadvantages. The main ones are a large number of matches (Hamming (x, y) and the presence of a weak avalanche effect, which lead to a significant decrease in the reliability of the algorithm for hacking. The designed hashing algorithm uses an iterative Merkley-Damgard structure, augmented by the input message to a length multiple of 512 bits. Processing in blocks of 128-bit uses cellular automata with mixed rules of 30, 105 and 90, 150 and takes into account the dependence of the generation of the initial vector on the incoming message. This allows half of the 10,000 pairs of arbitrary messages to have an inverse Hamming distance of 0 to 2. The proposed algorithm is four times slower than the well-known family of "secure hash algorithms." However, computation speed is not a critical requirement for a hash function. Decreasing the sensitivity to the avalanche effect allows the generation time to be approximately halved. Optimization of the algorithm, as well as its testing was carried out using new technologies of the Java programming language (version 15). Suggestions and recommendations for improving this approach to data hashing are given also

Keywords: hashing algorithm, chaos theory, cellular automata, compression function, transformation function

UDC 004.312.26
DOI: 10.15587/1729-4061.2021.242849

DEVELOPMENT OF A HASH ALGORITHM BASED ON CELLULAR AUTOMATA AND CHAOS THEORY

Yuriy Dobrovolsky
Corresponding author
Doctor of Technical Sciences*
E-mail: y.dobrovolsky@chnu.edu.ua
Dmytro Hanzhelo*
Mariia Hanzhelo*
Denis Trembach*
Georgy Prokhorov
PhD*
*Department of Computer System Systems
Y. Fedkovich Chernivtsi National University
Kotsiubynskoho str., 2, Chernovtsy, Ukraine, 58012

Received date 16.08.2021
Accepted date 10.10.2021
Published date 29.10.2021

***How to Cite**: Dobrovolsky, Y., Hanzhelo, D., Hanzhelo, M., Trembach, D., Prokhorov, G. (2021). Development of a hash algorithm based on cellular automata and chaos theory. Eastern-European Journal of Enterprise Technologies, 5 (9 (113)), 48-55. doi: https://doi.org/10.15587/1729-4061.2021.242849*

1. Introduction

Information protection is today an important component of the globalization of information technology. The importance of protecting the confidentiality of user information in information systems is not in doubt. Equally important is the task of ensuring the integrity of information, for which cryptographic hashing functions are successfully used. Hashing is the process of converting a binary sequence of arbitrary length into a binary sequence of fixed length. Such transformations are called convolution functions or Toffoli hash functions [1].

Cryptographic hash functions are an indispensable and ubiquitous tool used to perform a variety of tasks, including authentication, data integrity checking, file protection, and even malware detection. There are many hashing algorithms that differ in cryptographic strength, complexity, bit depth, and other properties. It is believed that the idea of hashing, which belongs to an employee of International Business Machines (IBM), appeared about 50 years ago and has not changed fundamentally since then. In the modern world, hashing has acquired a lot of new properties and is used in many areas of information technology.

The main application of hashing in modern cryptography is the construction of associative arrays, the search for duplicates in a series of data sets, the construction of unique identifiers for data sets, storing passwords, creating a digital signature, protecting the integrity of user information [2, 3].

A cryptographic hash function, often referred to simply as a hash, is a mathematical algorithm that converts an arbitrary array of data into a fixed-length string of letters and numbers. Moreover, provided that the same type of hash is used, this length will remain unchanged, regardless of the amount of input data. The hash function can be cryptographically strong only if the main requirements are satisfied. Resistance to recovery of hashed data and resistance to collisions, that is, the formation of two identical hash values from two different data arrays. Interestingly, none of the existing hashing algorithms formally falls under these requirements, since finding the inverse of the hash value is only a matter of computing power. In fact, in the case of some particularly advanced algorithms, this process can take unreasonable time consumption.

Accordingly, the development of new hashing algorithms is an urgent task in software engineering.

2. Analysis of literature data and problem statement

Hashing process based on cryptographic hash functions is widely used in cryptocurrencies. For example, the functioning of many blockchains is based on hashing algorithms of the SHA-2 or SHA-3 families and their variations [4]. The investigation of cryptographic hashing algorithms in this case is considered in detail in [5], which shows the strengths and weaknesses of most of the algorithms used today.

In Ukraine, as in most countries of the world, there are rules for information security in the banking system. They are regulated by the decree of September 28, 2017 No. 95 [6], which requires the use of hashing security algorithms SHA-224, SHA-256, SHA-384, SHA-512, "Kupina", or more crypto-resistant. Similar requirements are formulated in DSTU 7564: 2014 [7].

However, as mentioned in [8, 9], these algorithms have a number of disadvantages. Their improvement occurs in different ways.

In particular, in [8], to overcome the shortcomings of hashing algorithms of the SHA families, a specialized hash function is proposed that accepts 512-bit message blocks and generates a 256-bit hash value. The random signed sequence is added as an additional input to the hash compression function.

The authors of [9] have improved the throughput of the SHA3 hash algorithm by reducing the number of clock cycles required to obtain the hash value. However, a decrease in the maximum frequency was observed.

Chaos theory [10] is used in the creation of new hashing algorithms, which is applied in several stages to encrypt images. This algorithm performs bitwise encryption using the SHA-1 hashing algorithm.

An information block can be divided into several subblocks by means of a fixed division, as suggested in [11]. This uses the SIFT operator to retrieve information about the characteristics of key points in subblocks. Similarly, using the compilation policy by the data distribution method for the IMC implementation of the Keccak hash algorithm (SHA), the algorithm performance is increased by more than 70 % [12], which indicates the effectiveness of such a method.

Analysis of a computational collision problem using a hash algorithm based on a chaotic map using message expansion and aggregation operation, which increases the sensitivity between messages and hash values, which helps to reduce the propability of conflicts [13].

To improve the security of the hash sequence during image processing, a fractional order mapping and chaotic scrambling are constructed to encrypt the eigenvector, and the image information is confirmed by the Hamming distance [14].

In [15], an implementation of a message authentication code using a random initial sequence of a linear congruential generator is presented. The applied hashing algorithm proved to be more reliable due to the increased complexity of the traditional SHA-160. This scheme has proven to be effective and applicable for a variety of environments with high security requirements [5].

Data protection in web applications is implemented with the MD5 hash function, which provides the function and form of password encryption. It, like the hashing algorithms of the SHA families, has problems in the form of a collision (coincidence) attack. It can have the same hash value for two different input messages, which is unacceptable from a data security point of view. Improving the reliability in this case is proposed using the SHA 512 algorithm [16]. Risk mitigation is provided by a new hash function that modifies the code to recover the system and test the implementation. Penetration testing was performed on a User Entry Test (UAT). The UAT result shows an agreement of 86.00 %.

A combined method of data protection is also used, which consists in using the features of the processor design and a set of instructions and rules for a particular application. In particular, in [17], it is proposed to combine the features of the operation of a specialized integrated circuit and digital signal processing – for the hashing algorithm RIPEMD-160, a specialized configuration of registers and an instruction set architecture are formed. It includes 12 special and 35 general instructions. Despite the cumbersomeness of the method, its tests show that its performance exceeds that of analogs.

So, the analysis of literature data [4–17] allows to state the following.

Known hashing algorithms (SHA family, MD, etc.) have disadvantages. The main ones are a large number of coincidences (small Hamming distance) between hash images of similar input data and the presence of weak avalanche effect, which lead to a significant decrease in the reliability of the algorithm for cracking.

Let's formulate the requirements for the hash function [18]:

1. Determinism: the same messages with the same input conditions always lead to the same hash image.

2. The impossibility of reproducing a preimage from a known hash image.

3. The impossibility of finding two arbitrary messages within a reasonable time, giving the same hash-image (Collision of the 1st kind).

4. The impossibility of finding (in a reasonable time) for a given message another arbitrary message giving the same values of the hash image (collision of the second kind).

5. Effective avalanche effect: a change in one bit of an incoming message should lead to a change in at least half of the bits of the hash image [19].

Most traditional hashing systems are based on cyclical algorithms, in which a sequence of operations is repeated, the algorithmic sequence is known and open. This can lead to the fact that, by analyzing a large series of data, some patterns can be generalized, which will lead to an unacceptable decrease in cryptographic strength. The stability of such systems depends only on the computing power of the cryptanalyst.

3. The aim and objectives of research

The aim of research is to find a way to increase the sensitivity to the avalanche effect of the data hashing algorithm based on cellular automata by introducing elements of deterministic Chaos into the cyclic algorithm.

To achieve the aim, the following objectives were set:

– develop an algorithm for splitting the original message into blocks;

– develop an initializing block – an input vector using the theory of cellular automata;

– develop an algorithm for cyclic compression of text by blocks;

– create transformation functions using a one-dimensional cellular automaton;

– check the avalanche effect of the developed hash function.

4. Materials and research methods

The algorithm for achieving this goal is based on the use of the principles of generating unpredictability, which is based on the theory of Chaos.

Cellular automata (CA) are one of the examples of simple chaotic systems [20]. A cellular automaton consists of cells

that have a strictly defined state, and can change it discretely depending on their state and the state of neighboring cells. How their state will change is determined by the rules (transition functions). Depending on them, the state of the cells can vary greatly depending on the initial conditionsй.

To implement the algorithm, the iterative structure of the Merkle-Damgard [21] was chosen as a basis. Its content provides for splitting an incoming message of arbitrary length into blocks of a specific length, and then working with them using the compression function *f*. This function takes 2 arrays of the same size as input, and generates a third array of the same size.

As input parameters, as a rule, the message block itself and the hash image of the previous step are used. The main advantage of this structure is that if the compression function is collision resistant, then the entire hash function will be stable [21].

One-dimensional cellular automata [9], including 30, 90, 105 and 150 rules, were used as a simple chaos generator. With their help, new mixed rules were built, which were subsequently used for the compression function and the generation of the input vector. These rules were chosen because of their investigated and proven chaotic properties. All of them belong to the 3rd class of rules, which are considered chaotic [23].

With the help of CA cellular automata, new mixed rules were developed, which were subsequently used for the compression function and the generation of the input vector. The chaotic properties of the CA made it possible to introduce an element of uncertainty into the cyclic algorithm.

5. Research results of the developed hash function algorithm based on cellular automata and chaos theory

5. 1. Development of an algorithm for splitting the original message into blocks

Processing a message of arbitrary length to the desired length (multiple of 512 bits) is performed similarly to MD5 and SHA-1 algorithms. Then, in accordance with the illustration of the text splitting model using the example of the phrase "Hello, world!" shown in Fig. 1 is appended to the end of the message with such a block of zeros so that the length of the last message is 448. This addition makes it possible to add the size of the original message in binary code to the last 64 bits.

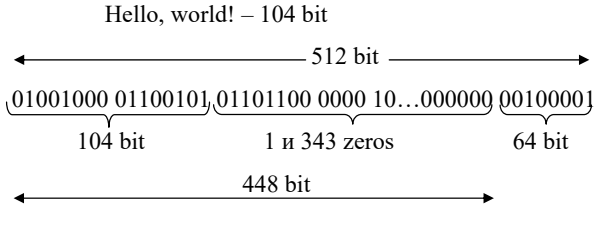

Fig. 1. The illustration of a text splitting using the example of the phrase "Hello, world!"

This padding will occur in any case, even if the original message was 448 bits long.

5. 2. Development of an initialization block - an input vector using the theory of cellular automata

In the well-known, widely used algorithms (SHA MD5), the input vector is formed as a sum of 64 bits at the beginning and at the end of the message, which, from the point of view of chaos theory, gives a weak avalanche effect. Namely, a change in the message by one bit gives a change in the input vector by a maximum of one bit. Therefore, to increase the sensitivity to the avalanche effect, the algorithm for obtaining the input vector should be improved.

The mathematical model describing the quality of sensitivity to the avalanche effect (avalanche criterion) is based on the bit independence criterion, according to which when one input bit changes, any two output bits change independently of each other [24].

The function $f: \{0,1\}^n$ satisfies the bit independence criterion if for any $i, j, k \in \{1,2, \ldots n\}$, where $j \neq k$, inverting the i bit at the input causes the j and k bits at the output to change. These changes are independent to a certain extent.

To measure the degree of independence of the two output bits, the BIC correlation coefficient (a_j, a_k) – (Bayesian information criterion) is introduced between the j-th and k-th components of the output vector for the modified i-th component of the input vector.

$$BIC = \max_{1<<j,k<<n, j\neq k} = BIC\left(a_j, a_k\right).$$

This parameter demonstrates how well the function f satisfies the bit independence criterion.

It takes values in the interval [0, 1], and in the best case is equal to 0, then it is possible to talk about complete independence, in the worst case 1, when there is a complete dependence.

In the proposed algorithm, the dependence of the output byte from the input one has a pseudo-chaotic nature according to the definition of a chaotic group of cellular automata.

Thus, for one iteration, the BIC correlation coefficient is 0.5. For 128 iterations, this coefficient will be equal to

$$BIC(a_i, a_j) = (0.5)^{128} \rightarrow 0.$$

Modern hashing algorithms provide a correlation coefficient of 0.5. That is, when hashing two identical strings that differ by 1 bit, approximately half (0.5) bytes will be the same.

In the case of the proposed algorithm, the matches are practically independent. This means that by controlling the number of iterations, the sensitivity of the algorithm to the avalanche effect can be increased to almost maximum value. At the same time, resource costs increase significantly. On the other hand, by reducing the number of iterations, an optimal level of correlation between the speed and sensitivity of the algorithm to the avalanche effect can be achieved.

Since the compression function requires 2 arrays at the input, and at the first stage, there is still no hash array from the previous step, the input vector is used instead. The block diagram of the formation of the input vector is shown in Fig. 2.

M_0 – incoming message with length n bits. The incoming message is padded with zero bytes so that its length is a multiple of 512, and then used as a data source for the vector. The first 64 and last 64 bits are taken from the padded incoming message and combined into a 128-bit array. Using the resulting key as the initial state of the cellular automaton, mixed rules 30 and 105 are applied to it, so that rule 30 applies to all odd cells, and 105 applies to even cells. As a result of 128 iterations, a 128-bit block is obtained, which will be used together with the first block of text for the compression function.

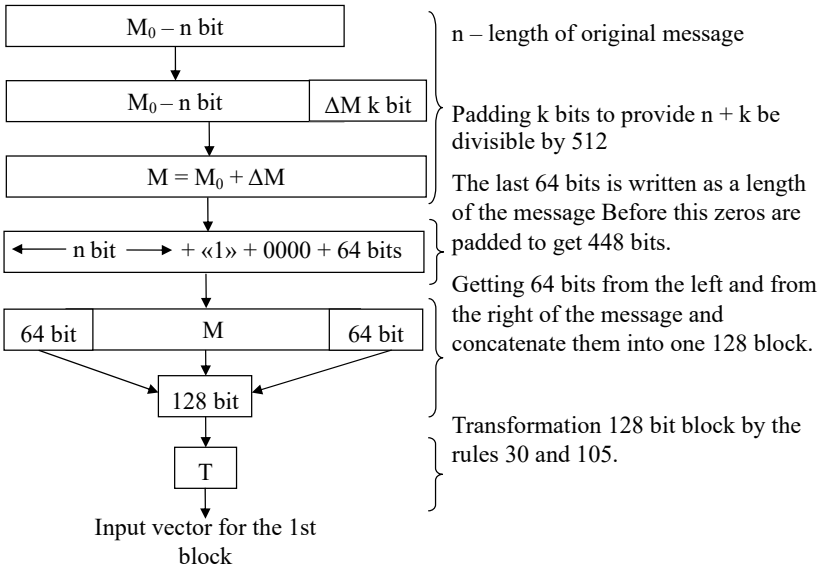

Fig. 2. Block diagram of the formation of the input vector

where C_i – compressed part of input message M_i;

h_{i-1} – hash-image of the previous block;

S_k –cyclic bitwise shift depending on k;

k – number of 128 bit sub-block in 512-bit block;

S={7, 12, 17, 22}.

All text is split into blocks of 512 bits. These blocks are split into 4 more sub-blocks of 128 bits each. For each of these 4 blocks, let's apply a cyclic left shift. That is, for the first block S_1=7, for the second S_2=12, for the third S_3=17 for the fourth S_4=22, according to the fact that S={7, 12, 17, 22}.

In this case, a shift vector from the well-known MD5 hashing algorithm is used. If necessary, it can be replaced with another one optimized for the avalanche effect, and, in principle, generally be an input element of the hashing procedure, if it is used as a library, which often happens.

Of course, 128 iterations when creating an input block slows down the algorithm, but at the same time, the sensitivity to the avalanche effect increases. But this, in turn, complicates the "reversibility" of the process, that is, obtaining a preimage from a hash image.

5. 3. Development of an algorithm for cyclic compression of text by blocks

The text is padded to a multiple of 512 and splitted into blocks of 512 bits. Each such block is divided into sub-blocks of 128 bits.

The compression function, in a specific implementation of the algorithm, "compresses" 128 bits of the M_{ik} message and the hash sum of the previous block h_{i-1} into a new hash of 128 bits. It consists of two stages: direct compression with the hash of the previous (cyclically shifted) block, and the transformation function using a cellular automaton with 90 and 150 rules [25–27].

The result of the compression function will be a 128-bit block, which is the key for the next block of text. Fig. 3 shows the hashing algorithm M_1 (the first 512 bits) of the message block.

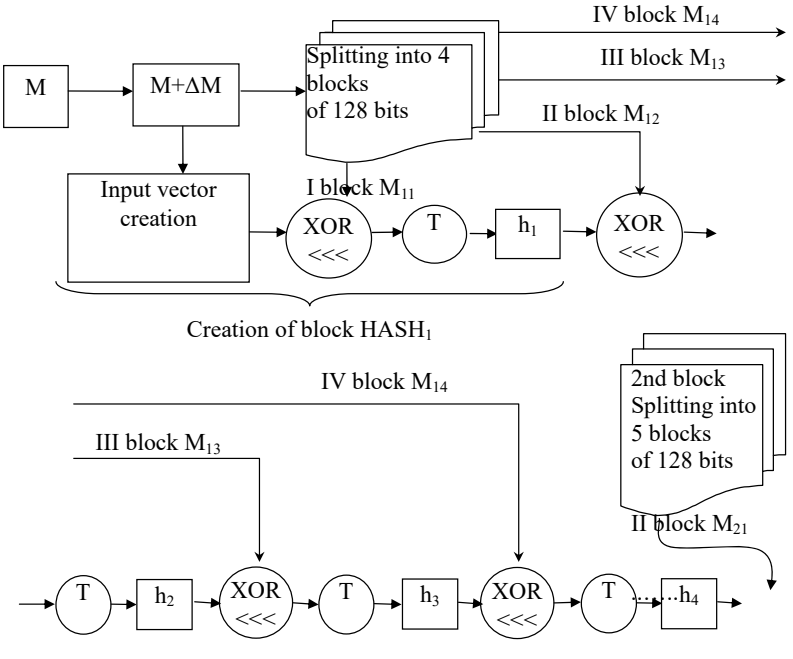

Fig. 3. Hashing algorithm M_1 (first 512 bits) of a message block: T – transformation using cellular automata using rules 90 and 150 (entanglement); XOR <<< – Shift compression

Rule 90 is applied to the entire array, then rule 150 is applied to the result. Let's repeat the operation 64 times. As a result, a 128×128 matrix is obtained, the diagonal of which is the HASH function h1. In this case, large computing power is used. However, this disadvantage is compensated for by the expected high sensitivity to avalanche effect. As mentioned above, the algorithm uses the rules of cellular automata belonging to the 3rd class of rules, which are considered chaotic [9, 23]. Their randomness, in turn, should provide a high sensitivity of the algorithm to the avalanche effect.

Compression of a block with a hash is provided according to formula (1) [20], which is widely used in the theory of cellular automata, but was not used earlier to form the initial vector:

$$C_i=M_i\oplus(h_{i-1}<<<S_k), \quad (1)$$

After executing this function, a 128-bit compressed hash image is obtained. The next step is to apply the transform function.

5. 4. Creating a transformation function using a one-dimensional cellular automaton

For it, a one-dimensional cellular automaton was chosen, which is subject to rules 90 and 150, since this pair has the best chaotic characteristics [21]. The rules are applied step by step. First, rule 90 is applied to the original 128-bit array. In the next step, rule 150 is applied to the result. And so in turn 128 steps, which means that each rule is applied 64 times. In Fig. 4 shows a diagram of the execution of the function of transforming a compressed block in 128 bits.

Let's chose just such a set of rules and their combinations, since this pair showed the least number of collisions with this application. All possible pairs of rules (30, 90, 105, 150) [28] were experimentally tested, and it was confirmed that using (90, 150) mixing "vertically" not a single collision was found on all 8 and 16 bit numbers. This means that each number turned into some other unique number.

As a result, a 128×128 matrix is obtained, where the vertical representations of the transformation results for each of the 128 steps. Even lines are the result of rule 90, odd lines are the result of rule 150.

The resulting row (128-bit block) is not the last 128th row, but the main diagonal of the matrix. This block will be transferred to the next 128-bit block for compression. It is converted from binary to hexadecimal and the result is a 32 byte hash sum.

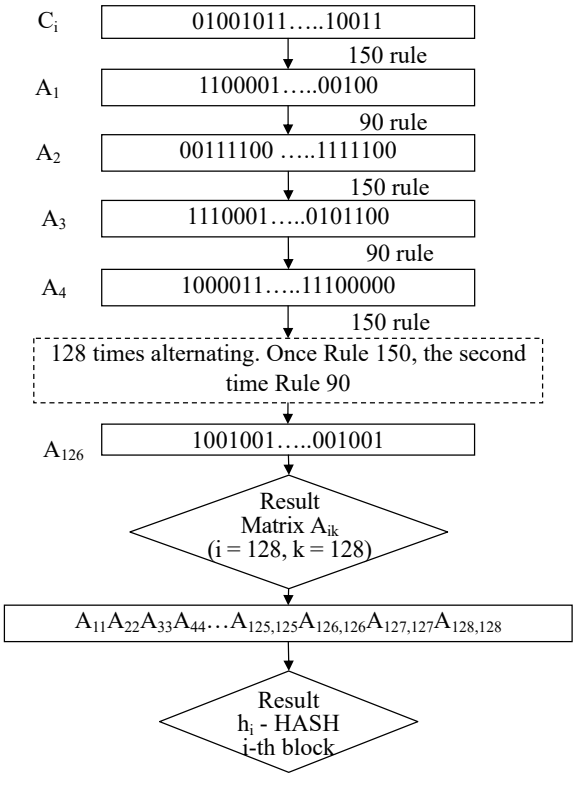

Fig. 4. Diagram of the transformation function of a compressed 128-bit block

By itself, the use of cellular automata of the "chaotic group" greatly complicates the derivation of the preimage from the result. The use of cellular automata of a chaotic group for 128 iterations significantly complicates the derivation of the preimage from the final hash-image [28].

Thus, the use of cellular automata of a chaotic group makes it difficult to quickly recover the original text from the hash image, since it is difficult to catch the pattern of creating a hash function from the original. However, this increases the computing power, which can be reduced by decreasing the number of iterations and possibly reducing the sensitivity to the avalanche effect. In each specific case of using this algorithm, the user can smoothly optimize the ratio between resource consumption and sensitivity to the avalanche effect by varying the number of iterations.

5. 5.Testing the avalanche effect of the developed hash function

The avalanche effect is a mandatory property of cryptographic hash functions, reflecting the idea of a high degree of nonlinearity: a slight change in the input (flipping one bit) produces a significant change in the output (about half of the bytes have changed). Usually, if function f has an avalanche effect, then the Hamming distance between two strings that differ by one bit (byte) is no more than half the length of the output string.

Mathematically, hash function F: $\{0, 1\}^m \rightarrow \{0, 1\}^n$ has an avalanche effect, if:

$$\forall x, y \in \{0, 1\},$$

$$m\text{: Hamming } (x, y)=1= \Rightarrow \text{average (Hamming } (\mathbf{F}(x), \mathbf{F}(y)))=n/2, \quad (2)$$

where Hamming (x, y) denotes the Hamming distance between two n-bit blocks x and y, that is, the number of matches.

The results were obtained from an experiment according to the following algorithm.

In the famous phrase "Hello, world!" in each iteration of the processing cycle, one of the 13 composite bytes was randomly selected and increased by 1 (incremented). The hash image of the resulting string was calculated and compared with the hash image of the previous string. The length of the output hash image was 32 bits, that is, the required average Hamming value for these pairs is 16. This means that about half of the bytes should change their value when one byte in the input message changes. In this case, Hamming Distance is equal to the inverse Hamming distance.

The experiment was of an evaluative nature. For a more complete experiment, in the future, strings of 512 bytes and the number of pairs of 1 million will be used.

In Fig. 5 schematically shows the results of checking the avalanche effect of the considered hash function. Distance is inverse to Hamming distance – Hamming^{-1} denotes the number of matches in two blocks of the same length.

Hammimg^{-1} (x, y)=length(x)–Hamming(x, y).

That is, if the strings do not match in any of the characters, then Hamming^{-1}=0. And if they have matched 1 byte, then Hamming^{-1}=1.

For verification, a set of 10,000 pairs (M_i, M_i') of arbitrary messages was created, such that Hamming $(M_i, M_i')=1$, that is, the lines differed by one byte.

The test was carried out using a bench developed by the authors based on the new Java 11 methods.

The test results are shown below in Table 1.

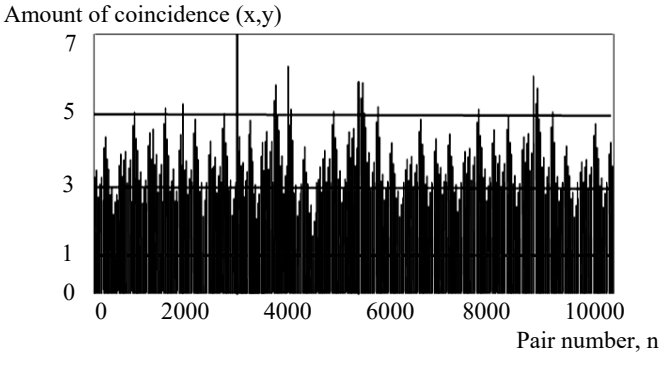

Fig. 5. The obtained distribution of coincidences between hashes of random messages and hashes of their one-bit changes

Table 1

Results of testing the avalanche effect of the considered hash function

Hamming^{-1}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Coincidence frequency	1654	3268	898	18	2472	829	830	0	5	0	6	0	0	8	0	12	0

For greater clarity of the result obtained, in Fig. 6 shows the dependence of the logarithm of the number of coincidences on the frequency of coincidences.

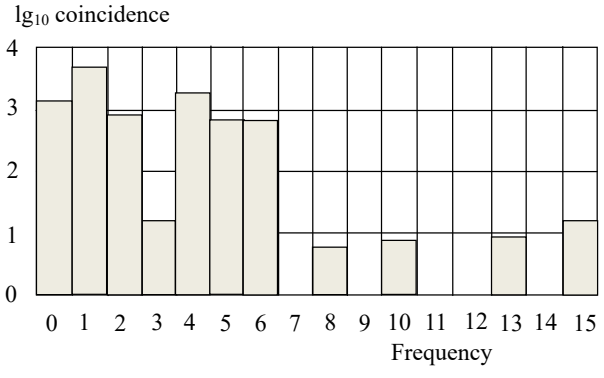

Fig. 6. Dependence of the logarithm of the number of coincidences on the frequency of coincidences

The test illustrated in Fig. 5, 6, shows that there are 1654 pairs with 0 bit coincidences. A one-bit match showed 3268 pairs, 2 coincidences showed 898 pairs, etc. It should be noted that more than 10 coincidences showed only a few pairs. The coincidences between the hash values of the selected pairs are centered around the values 0–2. This means that about half of 10 thousand pairs have 0–2 coincidences, instead of the required 16.

The explanation of the obtained result is that, in contrast to the majority of similar studies (for example [29], where the formation of the initial vector is "in general"), the developed algorithm is based, among other things, on the correct formation of the input vector.

Another standard way to investigate the avalanche effect is to hash the well-known phrase "about the fox" and compare the hash patterns when one byte of the phrase changes. The results of this experiment are shown in Table 2.

Table 2

Results of an avalanche effect study based on hashing the phrase "the quick brown fox jumps over the lazy dog" and comparing hash images when one byte of the phrase changes

String	Hash
the quick brown fox jumps over the lazy dog	b8f1006d52624ca6ad6f1d4271271519
the quick brown fox jumps over the lazy dog.	a6df80ef786045948051bd48f92d4859
the quick brown foS jumps over the lazy dog	6cf2846d5f0cddebcc7e752c4952c041

The given result is presented for clarity, although it does not represent sufficient verification statistics. The second line differs from the first by the presence of a period at the end of the phrase. This is not entirely correct, since the lengths of the two original strings must be the same.

It can be seen that the hash image changes almost completely when one bit at the input changes, which also indicates a good avalanche effect of the designed hash function.

Testing the proposed algorithm for susceptibility to the avalanche effect showed that about half of the hash patterns out of 10 thousand 32-bit pairs have a coincidence from 0 to 2, instead of the required 16.

Thus, when checking the sensitivity of the algorithm to the avalanche effect, it turned out that out of 10,000 pairs, only 15 showed coincidences up to a quarter of the signs, instead of the required 50 % [19]. More than 80 % of the pairs tested yielded zero to two coincidences per 32-bit hash. This result turned out to be four times more sensitive than that of the existing algorithms (SHA1, SHA256, and MD5) [31].

6. Discussion of the results of the development of a hash function algorithm based on cellular automata and chaos theory

The results obtained during the development of the hash function algorithm based on cellular automata and chaos theory, namely, the high sensitivity of the algorithm to the avalanche effect, illustrated in Fig. 5, 6 and in Table 2 is explained, first of all, by an increase in the computation time. The proposed algorithm works four times slower than the well-known SHA family. Nevertheless, the hashing process in the general information processing algorithm (receiving, primary processing, packing, sending, unpacking, verifying, etc.) takes relatively little time resources, then it is possible to assume that the performance requirements should not be critical. Security depends on time, that is, the safer option will take longer to compute the hash value [5]. Decreasing the sensitivity to the avalanche effect allows the generation time to be approximately halved.

In the proposed algorithm, the original message is divided into blocks, just as it was done in [8]. Similarly to [10, 13], the algorithm uses chaos theory. New in the proposed algorithm, as shown in Fig. 2, is the compression of a block with a hash and the formation of an input vector using the theory of cellular automata. In particular - in blocks of 128-bit with mixed rules CA 30, 105 and 90, 150, taking into account the dependence of the generation of the initial vector on the incoming message.

In algorithms like SHA, MD5, the input vector is formed as the sum of 64 bits at the beginning and at the end of the message. In this case, a change in the message by one bit gives a change in the input vector by a maximum of one bit. In the case of the proposed algorithm, increasing the sensitivity to the avalanche effect is achieved by improving the algorithm for obtaining the input vector.

The algorithm for cyclical compression of text by blocks consists of compression with a hash of the previous (cyclically shifted) block, and a transformation function using a cellular automaton with 90 and 150 rules.

The creation of a transformation function using a one-dimensional cellular automaton of a chaotic group for 128 iterations makes it much more difficult to obtain a preimage from the final hash-image.

The proposed algorithm has limitations in application when the time provided for processing the input message is limited. This is because it is four times slower than the

well-known SHA family. At the same time, decreasing the sensitivity to the avalanche effect makes it possible to approximately halve the generation time.

Of the five requirements put forward to the hash function (determinism, impossibility of obtaining a preimage, collisions of the 1st and 2nd kind, avalanche effect) [19], three of them were carefully checked. The collision requirement has been checked on an assessment basis. This is an obvious flaw in the study. For correct checking of collisions of the 1st and 2nd kind, it is necessary to carefully define the requirements for the purity of the experiment. In particular, how to ensure the generation of several hundred million different lines. In a preliminary experiment, 4 million lines were generated, which took 1 gigabyte of RAM. Thus, testing the sensitivity of the algorithm to the avalanche effect showed matches up to a quarter of the characters, instead of the required half characters [19]. More than 80 % of tested pairs showed from zero to two matches per 32-bit hash, which is almost four times more sensitive than existing algorithms (SHA1, SHA256 and MD5).

The development of research on improving the hash function algorithm based on cellular automata and chaos theory is planned in the direction of correct checking of collisions of the 1st and 2nd kind. In addition, the use of new technologies of the Java programming language (version 15) will help optimize the output parameters of the proposed function.

7. Conclusions

1. The algorithm for splitting the original message of arbitrary length into blocks to the required one (512 bits) has been added by adding a single bit to the end of the message, which is padded with zeros until the message length is 448. This addition allows adding the size of the original message in binary code to the last 64 bits.

2. It is proposed to compress a block with a hash and form an input vector using the theory of cellular automata, in particular, in blocks of 128-bit with mixed rules CA 30, 105 and 90, 150, taking into account the dependence of the generation of the initial vector on the incoming message.

3. A function of text compression by blocks is proposed, which consists of compression with a hash of the previous (cyclically shifted) block, and a transformation function using a cellular automaton with rules 90 and 150. With its help, a 128-bit block is obtained, which is the key for the next block of text.

4. For the transformation function, a one-dimensional cellular automaton is selected, which is subject to rules 90 and 150, which are applied alternately in steps of 64 times. This pair of rules with this application showed the least number of collisions. In particular, no collisions were found on all 8 and 16 bit numbers.

5. Testing the proposed algorithm for susceptibility to the avalanche effect showed that about half of the hash patterns of 10 thousand 32-bit pairs match from 0 to 2, instead of the required 16. The proposed algorithm works four times slower than the well-known SHA family. At the same time, a decrease in the sensitivity to the avalanche effect allows the generation time to be approximately halved..

Acknowledgements

The authors would like to express their sincere gratitude to Dyachuk Rostislav Lyubomirovich - Assistant of the Department of Software for Computer Systems, Yu Fedkovich Chernivtsi National University for the refactoring of the algorithm code and advice on the application of new methods of the Java 11 programming language.

The work was carried out within the framework of the theme “Research, modeling and development of software complex dynamic systems” at the Department of Computer Systems Software, Yu. Fedkovich Chernivtsi National University. Registration number: 0121U109232.

References

1. Toffoli, T., Margolis, N. (1987). Cellular Automata Machines. Cambridge: MIT Press. doi: http://doi.org/10.7551/mitpress/1763.001.0001
2. Jeon, J.-Ch. (2013). Analysis of hash functions and cellular automata based schemes. International Journal of Security and Applications, 7 (3), 303–316. Available at: http://article.nadiapub.com/IJSIA/vol7_no3/28.pdf
3. Paar, C., Pelzl, J. (2010). Understanding cryptography. Berlin-Heidelberg: Springer-Verlag. doi: https://doi.org/10.1007/978-3-642-04101-3
4. Pasyeka, M., Pasieka, N., Bestylnyy, M., Sheketa, V. (2019). Analysis of the use of the highly effective implementation of the sha-512 hash functions for the development of software systems. Cybersecurity: Education, Science, Technique, 3 (3), 112–121. doi: http://doi.org/10.28925/2663-4023.2019.3.112121
5. Kuznetsov, O. O., Horbenko, Yu. I., Onopriienko, V. V., Stelnyk, I. V., Mialkovskyi, D. V. (2019). The study of cryptographic hashing algorithms used in modern blockchain systems. Radiotekhnika, 3 (198), 54–74. doi: http://doi.org/10.30837/rt.2019.3.198.05
6. Pro zatverdzhennia Polozhennia pro orhanizatsiiu zakhodiv iz zabezpechennia informatsiinoi bezpeky v bankivskii systemi Ukrainy (2017). Postanova Pravlinnia Natsionalnoho banku Ukrainy No. 95. 28.09.2017. Available at: https://zakon.rada.gov.ua/laws/show/v0095500-17#Text
7. DSTU 7564: 2014 "Informatsionnye tekhnologii. Kriptograficheskaia zaschita informatsii. Funktsiia kheshirovaniia" (2014). Priniatii prikazom Ministerstva ekonomicheskogo razvitiia i torgovli Ukrainy No. 1431. 02.12.2014. Available at: https://usts.kiev.ua/wp-content/uploads/2020/07/dstu-7564-2014.pdf
8. Tiwari, H., Asawa, K. (2012). A secure and efficient cryptographic hash function based on NewFORK-256. Egyptian Informatics Journal, 13(3), 199–208. doi: http://doi.org/10.1016/j.eij.2012.08.003
9. El Moumni, S., Fettach, M., & Tragha, A. (2019). High throughput implementation of SHA3 hash algorithm on field programmable gate array (FPGA). Microelectronics Journal, 93, 104615. doi: http://doi.org/10.1016/j.mejo.2019.104615

10. Hasheminejad, A., Rostami, M. J. (2019). A novel bit level multiphase algorithm for image encryption based on PWLCM chaotic map. Optik, 184, 205–213. doi: http://doi.org/10.1016/j.ijleo.2019.03.065
11. Hao, W., Liming, Z., Haowei, M., Xingang, Z., Jinping, C. (2020). Perceptual Hash algorithm for GF-2 image using SIFT and SVD[J]. Bulletin of Surveying and Mapping, 8, 44–49. doi: https://doi.org/10.13474/j.cnki.11-2246.2020.0246
12. Xue, Wang, Liu, Lv, Wang, Zeng. (2019). An RISC-V Processor with Area-Efficient Memristor-Based In-Memory Computing for Hash Algorithm in Blockchain Applications. Micromachines, 10 (8), 541. doi: http://doi.org/10.3390/mi10080541
13. Li, Y. (2016). Collision analysis and improvement of a hash function based on chaotic tent map. Optik, 127 (10), 4484–4489. doi: http://doi.org/10.1016/j.ijleo.2016.01.176
14. Tao, F., Qian, W. (2019). Image hash authentication algorithm for orthogonal moments of fractional order chaotic scrambling coupling hyper-complex number. Measurement, 134, 866–873. doi: http://doi.org/10.1016/j.measurement.2018.11.079
15. Sodhi, G. K., Gaba, G. S., Kansal, L., Bakkali, M. E., Tubbal, F. E. (2019). Implementation of message authentication code using DNA-LCG key and a novel hash algorithm. International Journal of Electrical and Computer Engineering (IJECE), 9 (1), 352–358. doi: http://doi.org/10.11591/ijece.v9i1.pp352-358
16. Sumagita, M., Riadi, I. (2018). Analysis of Secure Hash Algorithm (SHA) 512 for Encryption Process on Web Based Application. International Journal of Cyber-Security and Digital Forensics, 7 (4), 373. Available at: https://link.gale.com/apps/doc/A603050342/AONE?u=anon~26dfe3b7&sid=bookmark-AONE&xid=80bc955a
17. Safaei Mehrabani, Y. (2018). Synthesis of an Application Specific Instruction Set Processor (ASIP) for RIPEMD-160 Hash Algorithm. International Journal of Electronics Letters, 7 (2), 154–165. doi: http://doi.org/10.1080/21681724.2018.1477182
18. Mittelbach, A. Fischlin, M. (2021). The Theory of Hash Functions and Random Oracles. Springer International Publishing. doi: http://doi.org/10.1007/978-3-030-63287-8
19. Georgacopoulou, C. (1986). An investigation of hashing algorithms and their performance. Bradford.
20. Liu, Y. (2020). Modelling Urban Development with Geographical Information Systems and Cellular Automata. CRC Press. doi: http://doi.org/10.1201/9781420059908
21. Ch, J. (2013). Analysis of hash functions and cellular automata based schemes. International Journal of Security and Applications, 7 (3), 303–316. Available at: http://article.nadiapub.com/IJSIA/vol7_no3/28.pdf
22. Belfedhal, A. E., Faraoun, K. M. (2015). Building Secure and Fast Cryptographic Hash Functions Using Programmable Cellular Automata. Journal of Computing and Information Technology, 23 (4), 317–328. doi: http://doi.org/10.2498/cit.1002639
23. Martinez, G. (2013), A Note on Elementary Cellular Automata Classification. Journal of Cellular Automata, 8 (3-4), 233–259. Available at: https://arxiv.org/pdf/1306.5577.pdf
24. Vergili, I., Yucel, M. D. (2001). Avalanche and Bit Independence Properties for the Ensembles of Randomly Chosen n x n S-Boxes. Turkish Journal of Electrical Engineering and Computer Science, 9, 137–145. Available at: https://journals.tubitak.gov.tr/elektrik/issues/elk-01-9-2/elk-9-2-3-0008-1.pdf
25. Mironov, I. (2005). Hash functions: Theory, attacks, and applications. Available at: https://www.microsoft.com/en-us/research/publication/hash-functions-theory-attacks-and-applications/
26. Li, W., Packard, N. (1990). The Structure of the Elementary Cellular Automata Rule Space. Complex Systems, 4, 281–297.
27. Wolfram, S. (2002). A New Kind of Science. Champaign: Wolfram Media, 1192.
28. Wolfram, S. (2002). Cellular Automata and Complexity. Westview Press.
29. Pieprzyk, J. (1993). Design of hashing algorithms. Springer-Verlag.
30. Belfedhal, A. E., Faraoun, K. M. (2015). Building Secure and Fast Cryptographic Hash Functions Using Programmable Cellular Automata. Journal of Computing and Information Technology, 23 (4), 317–328. doi: http://doi.org/10.2498/cit.1002639
31. Ostapov, S. E. Yevseiev, S. P., Korol, O. H. (2013). Tekhnolohii zakhystu informatsii. Kharkiv: Vyd. KhNEU, 476. Available at: http://kist.ntu.edu.ua/textPhD/tzi.pdf