UDC 51-74, 519.688, 004.056
DOI: 10.15587/1729-4061.2022.252060

DEVELOPMENT AND ANALYSIS OF THE NEW HASHING ALGORITHM BASED ON BLOCK CIPHER

Kairat Sakan*
PhD Student**
Saule Nyssanbayeva
Doctor of Technical Sciences, Professor**
Nursulu Kapalova
Candidate of Technical Sciences, Associate Professor**
Kunbolat Algazy
Corresponding author
PhD**
Ardabek Khompysh*
PhD**
Dilmukhanbet Dyusenbayev
Software Engineer**
*Faculty of Information Technology
Al-Farabi Kazakh National University
Al-Farabi ave., 71, Almaty, Republic of Kazakhstan, 050040
**Information Security Laboratory
Institute of Information and Computational Technologies
Shevchenko str., 28, Almaty,
Republic of Kazakhstan, 050010

This paper proposes the new hash algorithm HBC-256 (Hash based on Block Cipher) based on the symmetric block cipher of the CF (Compression Function). The algorithm is based on the wipe-pipe construct, a modified version of the Merkle-Damgard construct. To transform the block cipher CF into a one-way compression function, the Davis-Meyer scheme is used, which, according to the results of research, is recognized as a strong and secure scheme for constructing hash functions based on block ciphers. The symmetric CF block cipher algorithm used consists of three transformations (Stage-1, Stage-2, and Stage-3), which include modulo two addition, circular shift, and substitution box (four-bit S-boxes). The four substitution boxes are selected from the "golden" set of S-boxes, which have ideal cryptographic properties.

The HBC-256 scheme is designed to strike an effective balance between computational speed and protection against a preimage attack. The CF algorithm uses an AES-like primitive as an internal transformation.

The hash image was tested for randomness using the NIST (National Institute of Standards and Technology) statistical test suite, the results were examined for the presence of an avalanche effect in the CF encryption algorithm and the HBC-256 hash algorithm itself. The resistance of HBC-256 to near collisions has been practically tested.

Since the classical block cipher key expansion algorithms slow down the hash function, the proposed algorithm is adapted for hardware and software implementation by applying parallel computing. A hashing algorithm was developed that has a sufficiently large freedom to select the sizes of the input blocks and the output hash digest. This will make it possible to create an almost universal hashing algorithm and use it in any cryptographic protocols and electronic digital signature algorithms

Keywords: hash function, hash digest, block cipher, hash function security, collision

Received date 27.01.2022
Accepted date 22.03.2022
Published date 29.04.2022

How to Cite: *Sakan, K., Nyssanbayeva, S., Kapalova, N., Algazy, K., Khompysh, A., Dyusenbayev, D. (2022). Development and analysis of the new hashing algorithm based on block cipher. Eastern-European Journal of Enterprise Technologies, 2 (9 (116)), 60–73. doi: https://doi.org/10.15587/1729-4061.2022.252060*

1. Introduction

The rapid development of electronic devices, communications, and Internet technologies in recent decades has provided the possibility of almost instantaneous exchange of personal and collective data. Adversaries can relatively easily obtain huge amounts of confidential data using access to electronic sensors, computers, mobile terminals, and various social networks. This raises security issues in the use and transmission of data. Of particular importance among the most important components of information security are encryption and hashing, which are the most widely used cryptographic methods for ensuring the confidentiality, integrity, and availability of data.

Hashing was originally used to check the integrity of messages but has now become widespread in computer science and programming to optimize critical data operations. The field of application of the hashing mechanism is extremely wide.

Modern secure hash algorithms are crucial for the integrity of data and confirmation of the authorship of information during its transmission and storage in infocommunication systems and general-purpose networks. Hash functions are used to perform authentication, verify the integrity of information, protect data and files, including, in some cases, the detection of malicious software and much more. Hash functions solve the problem in terms of the volume of incoming data, which is why algorithms that can operate with concise values are very popular in the modern world of digital technologies. The hash mechanism is also used to reduce the time required to generate and verify a signature, as well as to reduce its length.

Hashing is also a fundamental transformation for blockchain technology, applied in areas such as financial transactions, user identification, or the creation of cybersecurity technologies. A blockchain is a connected chain of records called blocks. Each block contains its own hash value, the hash value of the previous block, and a timestamp, which prevent an attacker from making changes to the data [1, 2].

The very first hash function was built around the DES (Data Encryption Standard) block cipher. Since then, a lot of new hash functions have been developed using new constructions and ways of constructing them. Conventionally, hash function constructs can be divided into three categories: hash functions based on block ciphers, hash functions based on arithmetic functions, and special hash functions.

Designed hash functions must be subject to rigorous security checks. When designing an efficient hash function based on block ciphers, it is recommended to use well-studied cryptographic transformations and constructions that allow their subsequent software, firmware, and hardware implementations. The intensive development of information technology capabilities, including computing power, contributes to the emergence of new and modification of existing attacks, which requires constant development and updating of protection systems.

Thus, the area of research under consideration is relevant. A comprehensive study of the block cipher components used in the development of hash functions, as well as their relevance to modern technologies, is necessary and requires continuous and breakthrough scientific research.

2. Literature review and problem statement

As it is known, standards for the IT industry should be harmonized with international technical regulations, as our country is integrating into the global economy. In the area of information security, each state strives to develop its own national standards in the field of cryptography.

In 2015, the US state standard FIPS 202, SHA-3 (Keccak hash function), a variable bit length hashing algorithm, was approved and published. Keccak is based on the Sponge (cryptographic sponge) construction [3]. SHA-3 is one of the most widely used hash functions. At the moment, it is known that the scientific community is conducting a variety of studies on the strength of its latest version since previous versions of SHA-3 were broken or had vulnerabilities. The SHA-3 hashing process consists of two steps: absorption and compression. At the first stage, each message block of a fixed length of r bits is added to the current state of the matrix and 24 rounds of the compression function f are performed. At the second stage, the state matrix is truncated to the desired hash digest length by iteratively executing the compression function f.

Japan has the JIS X 5057-2: 2003 (ISO/IEC 10118-2: 2000) standard. "Information Technology. Security methods. Hash functions. Part 2. Hash functions using n-bit block cipher" [4]. The hash function of this standard is suitable for environments where the n-bit block cipher algorithm is already implemented. Since 2018, SHA-1 has been used as a standard JIS hash function.

In 2016, China approved the standard "GB/T 32905-2016 Information security technology SM3 cryptographic hash algorithm". In 2017, SM3 was standardized by the International Organization for Standardization (ISO IEC.10118-3) [5]. SM3 is a 256-bit hash algorithm, for a message M of length l (l<264) generates a 256-bit hash value and uses the Merkle-Damgard structure. It is mainly used in electronic signatures, cryptographic checksums, and pseudo-random number generators.

Since 2015, Ukraine has been operating the National Standard "DSTU 7564:2014 Information Technologies. Cryptographic information protection. Hashing function" [6]. It was developed for the gradual replacement of the interstate standard GOST 28147:2009. The Kupyna hash function uses the Davies-Meyer scheme and its permutations are built on the Kalyna block cipher.

South Korea uses its own LSH hashing standard, developed in 2014. LSH is one of the cryptographic algorithms approved by the Korean Cryptographic Module Verification Program. The advantage of this algorithm is that it more than doubles the performance of international standards (SHA2/3) in various software environments. LSH is still protected from known hash attacks. LSH is collision-resistant for $q<2^{n/2}$ and has preimage resistance and second preimage resistance for $q<2^n$ in an ideal cipher model, where q is the number of requests for the LSH construction [7].

The interstate standard GOST 34.11-2018 has been put into effect in the Russian Federation. "Information Technology. Cryptographic information protection. Hashing function", which is prepared on the basis of the application of the standard GOST R 34.11-2012 ("Streebog"). The algorithm calculates a hash function with an input data block size of 512 bits and a hash code size of 256 or 512 bits. It uses a compression function based on three transformations: nonlinear bijective transformation, byte permutation, linear transformation (SPL) [8]. This standard has been adopted in Armenia, Kyrgyzstan, Republic of Kazakhstan, and Tajikistan.

A new standard "STB 34.101.77-2020 Information Technologies and Security" has been put into effect in Belarus since 2020. "Cryptographic algorithms based on the sponge function" [9]. The cryptographic hashing algorithm used in this standard is based on the cryptographic sponge function.

In Republic of Kazakhstan, foreign cryptographic algorithms and standards are currently used in the existing electronic data protection systems. Since this poses a security risk, the creation of a domestic hashing algorithm to control the integrity of confidential information is an urgent task for our country. This work is targeted at the development of domestic information security systems and the creation of software and hardware packages for their practical use.

To date, standards for hash functions and cryptographic hashing algorithms have been adopted in many foreign countries, including the United States, Japan, China, Ukraine, South Korea, etc.

Republic of Kazakhstan uses international standards and mainly foreign hardware and software. The creation of domestic algorithms for cryptographic information protection, including hashing algorithms, is an urgent and necessary task.

The development of cryptographic primitives makes progress, and hash functions are used in many applications and on various platforms, which forces us to place high demands on their strength. In this regard, a lot of research is being carried out in the field of developing new and modifying existing hash algorithms.

The work [10] shows several attacks of finding the second preimage (pseudo-preimage) and collisions on the cryptographic hash function Kupyna-256 and Kupyna-512. Since Kupyna uses the wide-pipe construction, it is difficult to build a pseudo-preimage attack on it. The authors of the paper argue that there were not so many cryptanalytic studies of Kupina. They demonstrated all known attacks on it and their qualitative and quantitative indicators. In addition, the paper emphasizes that the modular constant addition operation provides additional resistance to the "meet-in-the-middle" attack.

In [11], a lightweight one-way cryptographic hash algorithm LOCHA was developed to create a hash digest of a fixed and relatively small length for a power-intensive wireless network. The focus is on lightening the algorithm so that when used in networks such as WSNs (Wireless Sensor Networks), nodes can successfully run the algorithm with low power consumption. The use of simple mathematical operations such as residue of division (mod), arithmetic modulo addition, and two substitution tables of 97 and 67 primes ensures high performance in obtaining a 96-bit hash digest. Despite the simplicity of implementation, this algorithm is not limited in scope. This is because LOCHA has proven to be more secure than other strong hashing algorithms such as MD5, SHA1. But, over time, the reliability of such hash functions can decrease due to the small and static size of their hash digest.

The work [12] proposes a hash function model with scalable output. The model is based on an artificial neural network (ANN) trained to mimic the chaotic behavior of the Mackey-Glass time series. This hashing method can be used to check data integrity and generate a digital signature. This makes it possible to create cryptographic services according to user requirements and time constraints due to output scalability. The authors confirm that changing the ANN architecture, that is, adding neurons to the output layer or removing them, makes it possible to obtain hash digests of the desired length. The results of three independent tests confirm that the hashing algorithm on ANN satisfies all the requirements for a hash function that creates short-term hash digests.

The paper [13] considers a hashing algorithm, determined by a timestamp, for the secure distribution of data between vehicles. The proposed algorithm fulfills all the basic properties such as preimage resistance, collision resistance of a one-way hash function without a key.

One method to make cryptographic hash functions more resistant to future attacks is through combinations of hash functions. The work [14] analyzes hash combinators, such as XOR combiner, concatenation combiner, and Hash-Twice, which combine two or more hash functions. The paper presents some approaches for combining two or more hash functions that do not provide n-bit security of preimage stability. Several attacks are defined by which second preimage resistance does not provide n-bit security of combined hash functions using concatenation and cascade methods of two n-bit hash functions. The upper security bound for the indicated hash combinators is also determined, based on the most well-known general attacks on preimages and attacks on finding the second preimage. In tabular form, the updated security status of the above hash combinators after the authors received new research results is presented. This shows that the security of most combinators is not as high as expected. As a result, given the basic security requirements, these hash combinators of two or more n-bit hash functions do not provide greater, sometimes even n-bit security. Therefore, the development of one n-bit ideal hash function is considered to be still relevant.

An extended overview of the current state of security of hash functions is presented in the paper [15]. The work highlights the existing models and security aspects in the development of a compression function through a modular approach, which refers to the creation of a hash function based on a block cipher or permutation. This paper, which presents modern scientific views and the process of modular design, substantiates its relevance and demonstrates the key points in the development. The authors pose open problems of modular design and present ways to solve them.

The paper [16] describes a hash function developed on the basis of a block cipher. The authors, using the Davis-Meier mode, built a new hash function, which was investigated for safety against collisions, and also presented approaches for using k-fold hash input lengths. The developed hash function inherits all the properties of a random oracle with a high degree. The developed double-length hash functions (DLHF) can be used on devices with a limited size since the block cipher used provides $O(2^{128})$ security.

The work [17] describes hashing modes (schemes) used as a transformation of block ciphers into a compression function. The AES (Advanced Encryption Standard) block cipher algorithm is considered as a compression function, various modes for hashing are investigated and several preimage attacks are carried out, it is also described in detail how to reduce the complexity of attacks by applying key neutral bits.

One of the significant problems in cryptography is ensuring resistance to multicollisions. This problem arose from the birthday attack, the answer to which was to double the length of the resulting hash value. This solution turned out to be inadequate to the available computing resources of the society and the time constraints for hashing. Increasing the bit depth in the Wide Pipe design obviously negatively affects the performance of computing resources. In 2010, a modification of Fast Wide Pipe [18] was proposed, which made it possible to double the computational speed compared to Wide Pipe. Each internal state value is divided into two halves. The first half is fed to the input of the compression function, and the second is added to the result of the same iteration. However, this scheme requires additional computer memory, so research work in this direction continues.

The hashing algorithms considered in [3–9] are the state standards of the countries of the world developed in the field of IT technologies. Each state seeks to create its own reliable cryptographic standards, including those for hash functions. Kazakhstan does not have its own cryptographic standard, as well as its own standard for hashing data. Therefore, for Kazakhstan, the issue of creating its own hashing standard, which determines the algorithm and procedure for calculating the hash function of the transmitted information, is relevant. In this regard, comprehensive studies of existing hashing algorithms are being carried out, and work is underway to develop a domestic reliable hashing algorithm. The new hashing algorithm proposed in this paper can become a candidate for the state standard.

The papers [10–18] present the effective methods and structures of hash functions developed to date. Hashing algorithms for various purposes have been studied in detail, including lightweight hashing algorithms and hashing algorithms for blockchain technology. The results of various cryptographic attacks of finding the first preimage, as well as the search for collisions of the first and second kind for hash functions are analyzed. In these papers, various techniques have been applied to improve the performance of hashing. The difference of the algorithm proposed by us lies in the fact that to increase the speed in one round we apply a non-linear cryptographic primitive with a special principle twice. In addition, when calculating each new byte, linear and non-linear functions are performed alternately. This approach to building a hash function is not considered in other works and is characterized by increased computational performance without compromising the security of a hash function built based on block ciphers.

3. The aim and objectives of the study

The aim of this work is to develop a fast and reliable hash function based on a symmetric block cipher algorithm, as well as to study and evaluate its reliability using cryptanalysis methods.

To achieve this aim, the following objectives were set:

– to develop a symmetric block encryption algorithm;

– to develop a hash algorithm that meets the basic requirements for cryptographic hash functions, and provides high performance and flexibility in hardware-software implementation;

– to conduct a study and evaluate the reliability of the developed hash algorithm by methods of statistical and cryptographic analysis;

– to implement hardware-software implementation of the developed hash algorithm.

4. Materials and methods

It is worth noting that designing a good hash algorithm is more difficult than designing a symmetric encryption algorithm. A cryptographic hash function is a mathematical algorithm that converts an arbitrary array of data into a fixed-length string [19]. The main requirement for cryptographic hash functions is that for any message represented in binary form, the value of the hash digest must be quickly and efficiently calculated. Besides, a high-quality hash function should have a number of properties [20, 21]. The most convenient and popular hashing method involves dividing a message into blocks of a fixed length, after which these blocks are iteratively processed.

Currently, the most popular and security-oriented approach is to build hash functions based on block ciphers. In this approach, a block cipher is taken as the compression function, with two inputs representing a message block and a key [22]. The work [23] presents 64 possible PGV schemes (Preneel, Govaerts, and Vandewalle) for constructing hash functions based on the block cipher $E:\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, where n is the block length in bits. Of the 20 collision-resistant PGV schemes, the most commonly used is the Davies and Meyer scheme: $y_i = f(h_{i-1}, M_i) \oplus y_{i-1}$, where y_i and M_i are the input of the compression function f, and y_i is its output [24]. To develop our HBC-256 hash algorithm, we use the Davis and Meyer scheme.

Currently, the most widespread is the assessment of the cryptographic strength of hashing and encryption algorithms based on the methods of linear and differential cryptanalysis. Differential cryptanalysis technique is to track the change in the difference between the output bits depending on the change in the input bits at each round of transformation. It should be noted that the presence of the “avalanche effect” in the algorithm is a necessary condition for ensuring cryptographic resistance to differential cryptanalysis [25, 26].

The following two criteria are usually used to analyze the avalanche effect:

– avalanche criterion;

– strict avalanche criterion.

If the avalanche criterion requires an average change of 50 % of the bits in the output sequence when each bit in the input sequence changes, then the strict avalanche criterion requires a change with a probability of 1/2 of each particular bit in the output sequence when each particular bit in the input sequence change; and they are estimated by the following relations, respectively:

– $\varepsilon_a = |2k_i - 1|$, here i is the number of the modified bit in the input sequence, k_i is the probability of changing half of the bits in the output sequence when changing the ith bit at the input, ε_α is the avalanche parameter;

– $\varepsilon_s = |2k_{si,j} - 1|$, where i is the number of the modified bit in the input sequence, j is the number of the analyzed bit in the output sequence, $k_{s\,i,j}$ is the probability of changing the jth bit in the output sequence when the ith bit at the input changes as compared to the output value with the unchanged input value.

The hash digest $h(M)$ for any message of arbitrary length M must satisfy the properties of pseudo-randomness. This is one of the main requirements for hashing algorithms, i.e. it should be difficult to distinguish a hash-based pseudo-random number generator from a random number generator. For a hash digest to be considered random and unpredictable, at least it is necessary that there is no period, and that various combinations of bits of a certain length are distributed evenly over its entire length. This requirement can be statistically interpreted as the complexity of the law of generating a pseudo-random sequence of the hashing algorithm [27–29].

A hash function h is said to be collision resistant if it is computationally undecidable to find any two inputs that map to the same hash pattern for the given hash function. Collision attacks are carried out to establish two different messages M_1 and M_2 with the same hash digests $h(M_1)=h(M_2)$. In the classical attack, unlike a preimage attack, the cryptanalyst does not deliberately select the hash value.

A hash function is said to be near-collision resistant if it is computationally difficult to find any two messages M_1 and M_2 such that their hash digests. $h(M_1)$ and $h(M_2)$ differ by only a few bits for a given hash function [30]. A pair of messages M_1 and M_2, with $M_1 \neq M_2$, is called an ϵ-near collision for, if $d(h(M_1), h(M_2)) \le \epsilon$ holds, where d is the Hamming distance [31].

To study the reliability and performance of the developed algorithm, we used its software and hardware-software implementation, written in C++ in the Qt Creator 4.15.2 integrated development environment using the Qt library version 5.15, as well as a software package for statistical analysis, developed at the Institute of Information and Computational Technologies of the Committee of Science of the Ministry of Education and Science of the Republic of Kazakhstan.

5. Results of development of the new hash algorithm and its security study

5. 1. Development of the new encryption algorithm

5. 1. 1. Encryption algorithm scheme

The CF encryption algorithm belongs to the class of symmetric block cipher with a block and key length of 128 bits. The algorithm uses both linear (modulo 2 addition, cyclic left shifts) and nonlinear (four substitution S-boxes) transformations. The cipher structure is a variant of a substitution-permutation network (SP-network) with four rounds (R_1=4) [32]. One round of encryption consists of three transformations called Stage-1, Stage-2, and Stage-3 and is shown in Fig. 1.

The values of the input text $A(a_0, a_1, a_2, \ldots, a_{15})$ are written as the 4×4 square matrix A:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

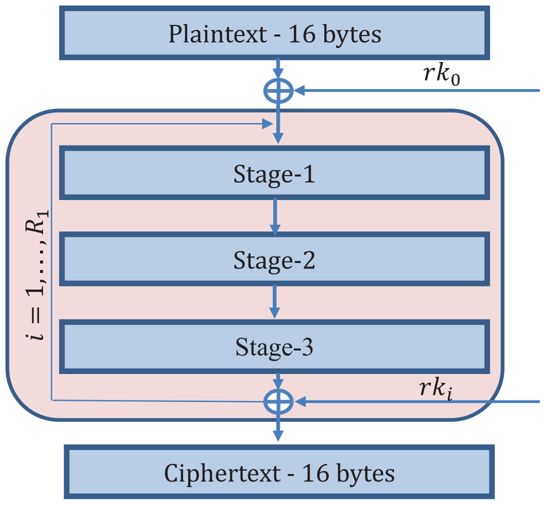


Fig. 1. General scheme of the encryption algorithm

Stage-1 transformation. This transformation, which consists of two steps, is used to obtain from a given matrix A a new matrix of the same size.

Step 1. The intermediate values c_{ij} of the matrix A are calculated by adding the element of the matrix a_{ij} modulo 2 with the remaining three elements of the ith row and three elements of the jth column.

Step 2. At this step, the new value c_{ij} passes through the substitution S-box (*SBOX* procedure) to be stored in the same place as the new value of the matrix A.

The Stage-1 transformation, consisting of the 1st and 2nd Steps, can be written as:

$$\left.\begin{array}{l} c_{ij}=\left(\oplus\sum\limits_{k=0}^{3}a_{ik}\right)\oplus\left(\oplus\sum\limits_{k=0,k\neq i}^{3}a_{kj}\right),\\ \qquad a_{ij}=SBOX\left(c_{ij}\right), \end{array}\right\}$$

$$i=0,1,2,3;\quad j=0,1,2,3, \tag{1}$$

where c_{ij} is the intermediate value of the matrix A, *SBOX* is the substitution S-box, $\oplus\sum$ denotes the sum of terms modulo 2.

SBOX procedure. The nonlinear bijective transformation S is defined through the SBOX procedure. The four substitutions S_0, S_1, S_2, S_3 are specified, where $S_i : Z_{(2^4)} \to Z_{(2^4)}$, $i=0,\ldots,3$. Four "golden" S-boxes as per Table 1 were selected for the transformation [33].

Table 1

Four "golden" S-boxes

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	S-box
$S_0(x)$	0	F	B	8	C	9	6	3	D	1	2	4	A	7	5	E	Serpent, S_3
$S_1(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D	HB-1, S_2
$S_2(x)$	7	C	E	9	2	1	5	F	B	6	D	0	4	8	A	3	HB-2, S_0
$S_3(x)$	4	A	1	6	8	F	7	C	3	0	E	D	5	9	B	2	HB-2, S_1

Serpent is the lightweight cipher Serpent, HB-1 is the lightweight cipher Hummingbird-1, HB-2 is the lightweight cipher Hummingbird-2.

The principle of SBOX operation is shown in Fig. 2. The input is one byte $\bar{a}_{ij}$ of the matrix A, which has a binary representation $\bar{a}_{ij}=\left(b_7b_6b_5b_4b_3b_2b_1b_0\right)_2$. S-boxes perform the replacement procedure at the nybble or quadbit level, called the left nibble $t_1=b_7b_6b_5b_4$ and the right nibble $t_0=b_3b_2b_1b_0$ (written in binary). Further, according to the table, $p_1=S_i(t_1)$ and $p_0=S_j(t_0)$ are determined. The indices i and j of the matrix element correspond to the numbering of the S-boxes. Further, the resulting nibbles through the ith and jth S-boxes are combined into a byte. Here, the nibbles are swapped, i.e. p_1 is stored in the right nibble, and p_0 is stored on the left. The byte thus obtained is sent to the output $a_{ij}=(q_7q_6q_5q_4q_3q_2q_1q_0)_2$. Therefore, $a_{ij}=SBOX\left(\bar{a}_{ij}\right)$.

Stage-2 transformation. This transformation consists of two operations: cyclic shift and *XOR*. The elements of the matrix A obtained in Stage-1 are stored in the form of a one-dimensional array (a_{00}, a_{01}, a_{02}, a_{03}, a_{10}, a_{11}, a_{12}, a_{13}, a_{20}, a_{21}, a_{22}, a_{23}, a_{30}, a_{31}, a_{32}, a_{33}). Then all the elements of the array are perceived as bytes, and their bit representations are combined using the concatenation operator:

$W=a_{00}\|a_{01}\|a_{02}\|a_{03}\|a_{10}\|a_{11}\|a_{12}\|a_{13}\|a_{20}\|a_{21}\|a_{22}\|a_{23}\|a_{30}\|a_{31}\|a_{32}\|a_{33}$. Next, a cyclic left shift is performed in 1-bit increments: $V=W\lll 1$ until a 16-byte result is obtained $V=b_{00}\|b_{01}\|b_{02}\|b_{03}\|b_{10}\|b_{11}\|b_{12}\|b_{13}\|b_{20}\|b_{21}\|b_{22}\|b_{23}\|b_{30}\|b_{31}\|b_{32}\|b_{33}$. After that, the *XOR* operation is performed byte by byte: $A=W\oplus V$, and the obtained bytes are accepted as new values of the 4×4 matrix A from left to right, from top to bottom.

Stage-3 transformation. This transformation is similar to the Stage-1 transformation. Here, too, the transformation consisting of two steps is performed with the matrix A. The difference is that the elements of the matrix are calculated from bottom to top, from right to left.

Let's write this transformation, consisting of the 1st and 2nd steps, similar to the previous one:

$$\left.\begin{array}{l} c_{ij}=\left(\oplus\sum\limits_{k=0}^{3}a_{ik}\right)\oplus\left(\oplus\sum\limits_{k=0,k\neq i}^{3}a_{kj}\right),\\ a_{ij}=SBOX\left(c_{ij}\right), \end{array}\right\}$$

$$i=3,2,1,0;\quad j=3,2,1,0. \tag{2}$$

At the end of each round, the values obtained after the Stage-3 transformation are summed modulo 2 with the round key values.

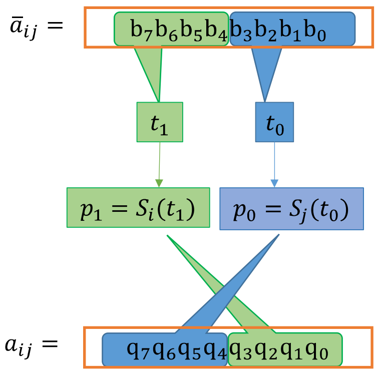


Fig. 2. General scheme of the encryption algorithm

5. 1. 2. Round key schedule algorithm

This section discusses an algorithm CFKey for deploying round keys based on a master key $K(k_0,k_1,k_2,\ldots,k_{15})$ with a length of 16 bytes. We assume that the master key K is the round key K_0. The total number of round keys is the same as the number of rounds R_1 of the encryption algorithm. The values of the round key $K_0(k_0,k_1,k_2,\ldots,k_{15})$ are stored in a 4×4 matrix A in the following form:

$$A = \begin{pmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}. \quad (3)$$

The CFKey key schedule algorithm consists of the StageKey-1, StageKey-2, and StageKey-3 transformations. The presented round key schedule algorithm is schematically shown in Fig. 3.

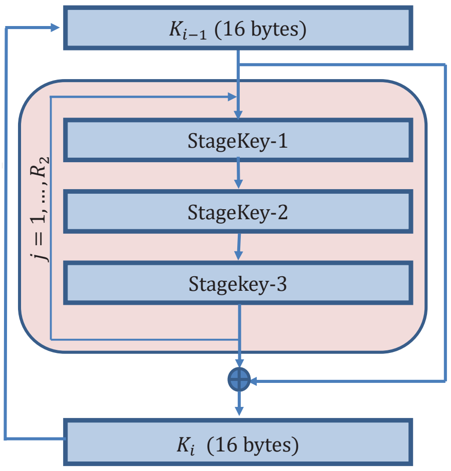


Fig. 3. Scheme for generating round keys

Note that the CFKey algorithm is functionally very similar to the CF algorithm: the StageKey-1 and StageKey-3 transformations are completely identical to the Stage-1 and Stage-3 transformations, respectively. The difference lies in StageKey-2. This transformation consists of only one operation, which is a cyclic shift. It also performs a one-bit cyclic left shift. There is no *XOR* operation in StageKey-2.

The CFKey algorithm is repeated R_2=8 times, and then the resulting 16-byte result is added with the round key K_{i-1} modulo 2 (*XOR*) and finally the next round key K_i, where i=1,...,R_1 is formed.

5. 2. Development of the hash algorithm based on the encryption algorithm

5. 2. 1. General information on the hash algorithm

The HBC-256 (Hash-based on Block Cipher) data hash algorithm is based on the proposed Compression Function (CF) block cipher. The compression function takes two inputs – a 128-bit message block m^j and a 128-bit round encryption key – and outputs an intermediate 128-bit hash code. The design uses a well-established approach to building a hash function – the Merkle-Damgard construct with the most common wide-pipe modification capable of withstanding a length extension attack. To create the final n-bit hash digest, the message block size and the size of the intermediate hash code should have the same length of w bits, where $n<w$. To meet the requirements of wide-pipe modification, in one hashing cycle we simultaneously execute CF 3 times for different m^j, j=0, 1, 2. This is why the length of the intermediate hash code w is equal to 128*3 bits [34].

The general scheme of hashing the message $K(M_0,M_1,M_2,...,M_{t-1})$ is shown in Fig. 4, where $M_r(m^0,m^1,...,m^{k-1})$, r=0,1,..., t–1 (for k=3). The proposed algorithm suggests taking the message M itself as the master key, and the previous intermediate hash code h^j_{i-1} for the encrypted text. Based on the blocks of the message M_r, the required number R_1 of round keys is generated.

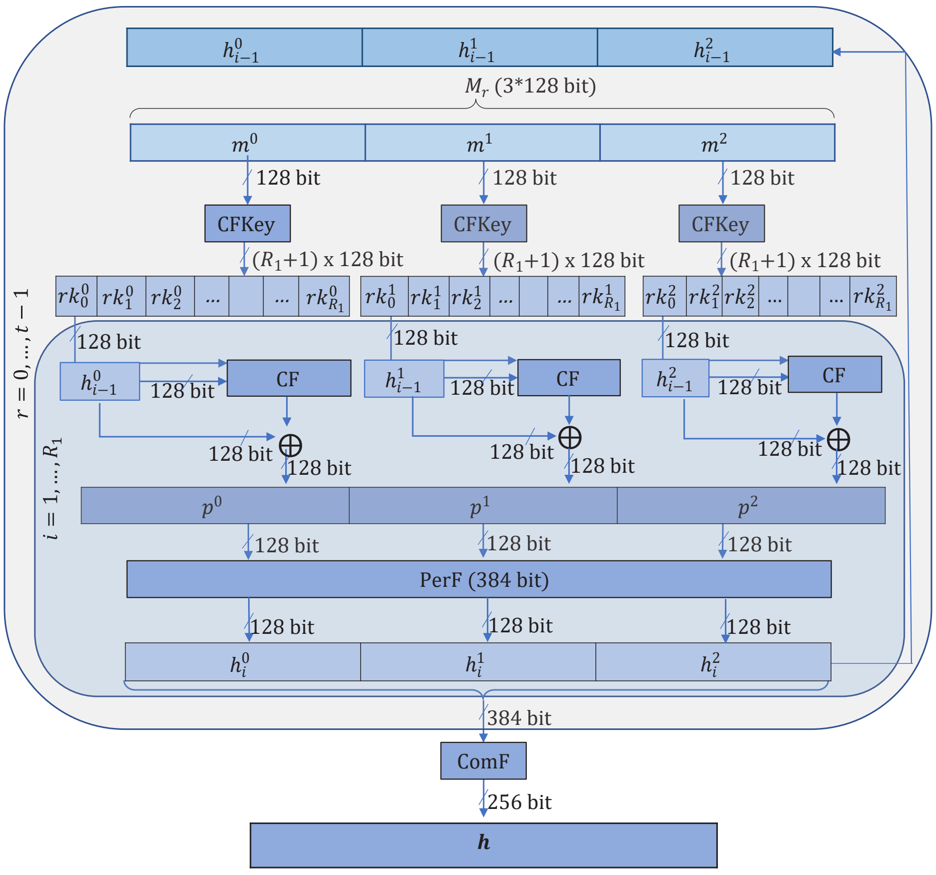


Fig. 4. General scheme of the hash algorithm

To enhance collision resistance, we use the Davies-Meyer scheme, where the CF output is summed (XOR operation) with the result of the previous hashing iteration h_{i-1}^j. The p^j value is the result of the ith iteration of the Davies-Meyer hash function. This scheme is used in hash algorithms based on block ciphers and acts as a one-way compression function.

5. 2. 2. Hash algorithm execution order

First, the block M_0 consisting of the first 384 bits of the message M is taken and divided into three 128-bit parts m^0, m^1, m^2. Based on each m^j, separate round keys rk_i^j are generated using the CFKey round key schedule algorithm, where i=1,2,...,R_1 and j=0,1,2. At the very beginning, the initialization vector or hash code takes on the value 0, i.e. $h_0^j = 0^{128}$. To obtain h_1^j, for all three parts, the CF encryption algorithm is simultaneously executed, taking as input the round key rk_0^j and h_0^j. Further, according to the Davies-Meyer scheme, we obtain p^j as the result of summing h_1^j and h_0^j modulo 2. After that, using the *PerF* (Permutation Function) procedure, the values of all three p^j are permuted, which are then divided into three parts, each 128 bits long. The *PerF* byte permutation procedure is carried out as per the formula:

$$\left.\begin{aligned} h_{3i} &= p_i, \\ h_{3i+1} &= p_{i+16}, \\ h_{3i+2} &= p_{i+32}, \end{aligned}\right\} \quad i = 0,\ldots,15, \tag{4}$$

where $p = p^0 \parallel p^1 \parallel p^2$. The 16-byte intermediate hash digests $h^j (j=0, 1, 2)$ are determined by $h = h_1^0 \parallel h_1^1 \parallel h_1^2$.

Formula (4) can be represented in Table 2.

Table 2

Byte permutation (x are byte positions, starting from 0)

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PerF(x)	0	16	32	1	17	33	2	18	34	3	19	35	4	20	36	5
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PerF(x)	21	37	6	22	38	7	23	39	8	24	40	9	25	41	10	26
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
PerF(x)	42	11	27	43	12	28	44	13	29	45	14	30	46	15	31	47

In subsequent iterations, the resulting 3 parts are accepted as h_0^j. Then again, the CF encryption algorithm is executed with the updated values h_0^j and the following round key. This procedure for each M_r (m^0, m^1, m^2) is repeated R_1 times, r=0, 1, ..., t–1. When calculating the intermediate hash digest of the block M_{r+1} (m^0, m^1, m^2), the values of the initialization vector h_0^j will take the values of the intermediate hash digest of the block M_r (m^0, m^1, m^2).

After processing the last block M_{t-1}, from the obtained hash digest of length 384 bits using the *ComF* (Compression Function), we determine the final hash digest h of length 256 bits:

$$h = ComF\left(h_{R_1}^0, h_{R_1}^1, h_{R_1}^2\right). \tag{5}$$

The order of padding. The HBC-256 algorithm iteratively processes 384-bit blocks of the input message M. If the length of M is a multiple of 384, then at the end of M, one more 384-bit block is added, consisting of zero bits, except for the first and last bits, which are equal to one. If the length of M is not a multiple of 384, then M is padded with so many bits that it is a multiple of 384. Suppose that the length of the input message M is not a multiple of 384 and is equal to l bits. We add a bit "1" at the end of M, after that, we add s zero bits, where $(l-2) \equiv s \bmod 384$ and add the last bit "1".

The order of division into parts. For hashing, the padded message M is divided into t blocks of 384 bits each as follows: $M = M \| Pad(M) = M_0 \| M_1 \| M_2 \| \ldots \| M_{t-1}$. *Pad* is abbreviated from *"padding"*.

The hashing process is iteratively performed according to the scheme in Fig. 1 with the input message M_r with a length of 384 bit, r=0, 1, ..., t–1.

Obtaining a hash digest. The final hash digest is determined through the *ComF* procedures. In our case, the values of the first and second block are taken as the final hash digest, the length of which is 256 bits: $h = h_1^0 \parallel h_1^1$.

5. 3. Proposed hash algorithm security study

Table 3 presents data on the complexity of attacks for every three problems when probability p=0.5.

Table 3

Data on the complexity of finding preimages and collisions

Specifications	Problems		
	Finding a preimage	Finding the second preimage	Finding a collision
Value of k at p=0.5 and $N=2^{256}$	$k=0.69*2^{256}$	$k=0.69*2^{256}+1$	$k=0.83*2^{128}$
Method applied	Brute force	Brute force	Birthday paradox

Here, k is the minimum number of different (different from each other) hashed data required for the attack, N is the number of possible hash digests relative to the length of the hash digest.

5. 3. 1. Assessment of the "avalanche effect" of the hash algorithm

The analysis of the propagation of the avalanche effect and the implementation of the avalanche effect after the 1st, 2nd, and 4th rounds were carried out according to the CF encryption algorithm scheme. The results after the 4th round are presented in Table 3. As an example, a 128-bit message M_0 in the form 0xcc156c4ce024d5113d680d7cce6d8b2 was selected for the analysis. For $1 \le i \le 128$, 128 plaintexts M_i were generated with the difference of one bit from M_0 as follows: $M_{129-i} = M_0 \oplus (i << 1)$.

After applying CF to these 129 messages M_i (i=0, 1, ..., 128), the corresponding 128-bit ciphertexts C_i were obtained. Then the probabilities k_i (i=1, 2, ..., 128) between the ciphertext C_0 and the remaining 128 ciphertexts were calculated. Table 4 gives the calculated probability k_i.

Next, we consider the avalanche effect of the HBC-256 hashing algorithm. The value $M_0 = 0^{384}$ is taken as an example of a 384-bit original message M_0. To analyze the avalanche effect of the HBC-256 algorithm, 384-bit messages were generated as follows:

$$M_i : M_{385-i} = M_0 \oplus (i << 1), \quad i = 1, 2, \ldots, 384. \tag{6}$$

Table 5 shows the dynamics of statistical indicators of the avalanche parameter ε_α depending on the number of hashing rounds.

Table 4

Analysis of the avalanche effect of the CF algorithm after the 4th round

i	k_i	i	k_i	i	k_i	i	k_i	i	k_i	i	k_i	i	k_i	i	k_i
1	0.48	17	0.44	33	0.52	49	0.52	65	0.44	81	0.42	97	0.54	113	0.45
2	0.56	18	0.49	34	0.50	50	0.57	66	0.48	82	0.51	98	0.52	114	0.43
3	0.52	19	0.55	35	0.59	51	0.55	67	0.48	83	0.48	99	0.46	115	0.58
4	0.53	20	0.54	36	0.50	52	0.53	68	0.47	84	0.52	100	0.38	116	0.48
5	0.53	21	0.48	37	0.50	53	0.51	69	0.47	85	0.52	101	0.49	117	0.53
6	0.52	22	0.46	38	0.52	54	0.52	70	0.49	86	0.53	102	0.46	118	0.45
7	0.41	23	0.42	39	0.49	55	0.52	71	0.50	87	0.48	103	0.49	119	0.50
8	0.43	24	0.45	40	0.46	56	0.56	72	0.61	88	0.51	104	0.51	120	0.46
9	0.54	25	0.50	41	0.48	57	0.55	73	0.51	89	0.59	105	0.45	121	0.48
10	0.45	26	0.55	42	0.52	58	0.41	74	0.54	90	0.45	106	0.46	122	0.39
11	0.51	27	0.55	43	0.48	59	0.51	75	0.51	91	0.38	107	0.47	123	0.52
12	0.43	28	0.42	44	0.45	60	0.58	76	0.45	92	0.53	108	0.55	124	0.51
13	0.49	29	0.52	45	0.53	61	0.52	77	0.54	93	0.45	109	0.53	125	0.45
14	0.55	30	0.46	46	0.50	62	0.50	78	0.52	94	0.51	110	0.58	126	0.55
15	0.50	31	0.45	47	0.55	63	0.65	79	0.51	95	0.52	111	0.46	127	0.52
16	0.52	32	0.53	48	0.47	64	0.52	80	0.52	96	0.48	112	0.51	128	0.47

Table 5

Statistical indicators of the avalanche parameter ε_α of the hash algorithm

Statistical indicators ε_α	Round 1	Round 2	Round 4	Round 8	Round 12
Largest value	0.7240	0.1870	0.1770	0.1720	0.1720
Least value	0.5940	0	0	0	0
Arithmetic mean	0.6645	0.0399	0.0407	0.0405	0.0398
Geometric mean	0.0007	0.0009	0.0009	0.0009	0.0009
Variance	0.6560	0.0310	0.0160	0.0150	0.0260
Mode	0.6670	0.0310	0.0360	0.0360	0.0310

Fig. 5 shows the probabilities k_i of change after the 4th round, showing an almost uniform distribution.

Fig. 5 above vertically represents the probability of a change in the avalanche effect from a change in the bits, which is represented horizontally.

5. 3. 2. Statistical analysis of the algorithm

To assess the randomness, we checked the hash digests using the software package "Automated system for the selection of statistical tests by D. Knuth and graphic tests", which implements a set of statistical tests [35]. For this, 60 files of different formats were selected, each of which contained from 20 to 1,000 KB of information. Data on files for analysis are presented in Table 6.

After processing each file with the HBC-256 algorithm, a corresponding 256-bit hash digest was obtained. Graphical and evaluation statistical tests were applied to the new 60 files with hash digest sequences. In graphical tests, the statistical properties of hash patterns are displayed as graphical dependencies, and in evaluation tests, the statistical properties are determined by numerical characteristics. As a result, according to the relevant data, a conclusion is made about the success of the passed test. Fig. 6, 7 show data on the number of files that successfully passed the graphics and evaluation tests.

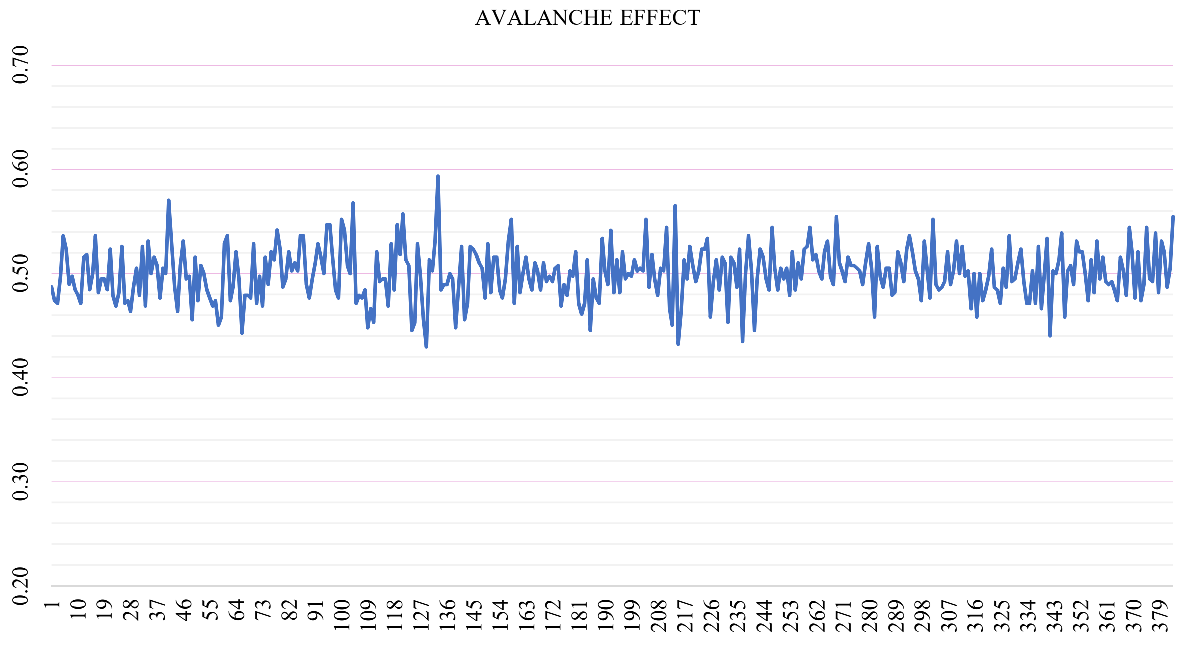


Fig. 5. Probabilities of bit change

Table 6

Plaintext files used in testing the hashing algorithm

File number	File type	Description
1,2,3	*.docx	Microsoft Word document
4,5,6	*.xls	Microsoft Excel document
7,8,9	*.pptx	Microsoft PowerPoint document
10,11,12	*.pdf	Cross platform open format
13,14,15	*.rar	Archived RAR document
16,17,18	*.zip	Archived ZIP document
19,20,21	*.jpg	Graphic document in raster format
22,23,24	*.png	Graphic document in raster format
25,26,27	*.gif	Graphic document in raster format
28,29,30	*.txt	Text file
31,32,33	*.lex	Adobe Linguistic Library Data file
34,35,36	*. djvu	Graphic and text format document
37,38,39	*.html	Web document
40,41,42	*.xml	Web document
43,44,45	*.wmz	Vector image media file
46,47,48	*.mp3	Sound information document
49,50,51	*.mp4	Sound/video information document
52,53,54	*.cat	System file for merging files
55,56,57	*.dll	Dynamic Link Library file
58,59,60	*.log	Event log file

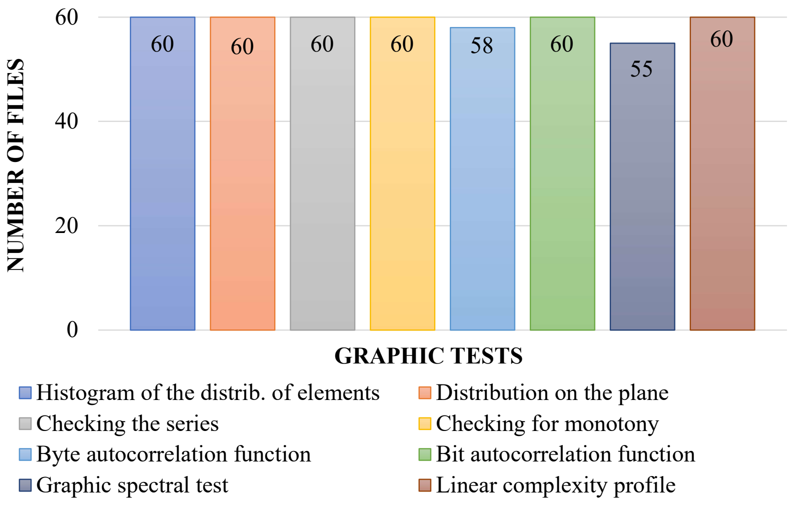


Fig. 6. Results of graphic tests

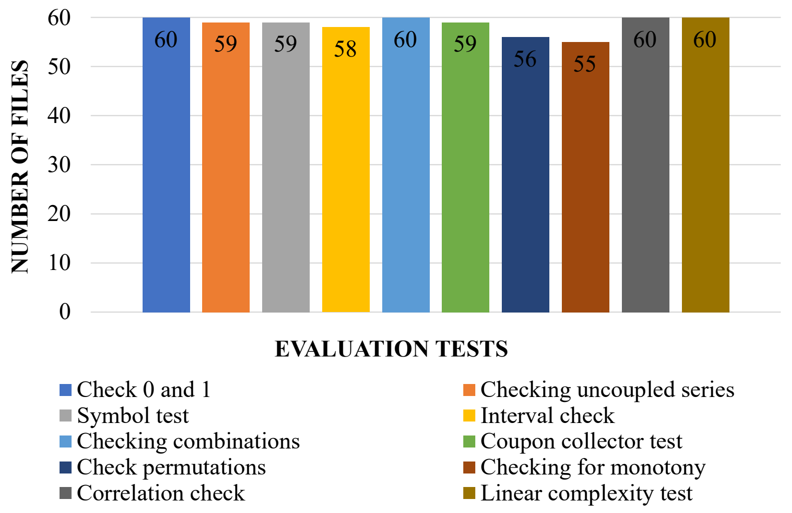


Fig. 7. Results of evaluation tests

The number of files that have passed graphical and evaluation tests are presented in Fig. 6, 7.

5. 3. 3. Near-collision resistance

To check the degree of resistance to ϵ-near collision, it is enough for us to conduct experimental studies using a large number of hash patterns. 25 thousand messages were randomly generated and their hash digests were calculated. The number of all possible combinations from the resulting set of all hash digests of 2 hash digests is equal to $C^2_{25000} = 312{,}487{,}500$. Using the software, Hamming distances were calculated for all pairs of hash digests, the values of which are in the range [0, 256], as well as the minimum and maximum Hamming distances, which are 81 and 175, respectively. By setting $\epsilon = 20$, we determined the number of pairs of messages with reasonably good Hamming distances:

$$108 \le d\left(M_i, M_j\right) \le 148 = 309{,}283{,}762,$$

$$i, j = 1, \ldots, 25{,}000, \quad i \ne j (\text{e.g.}, 98.975\,\%).$$

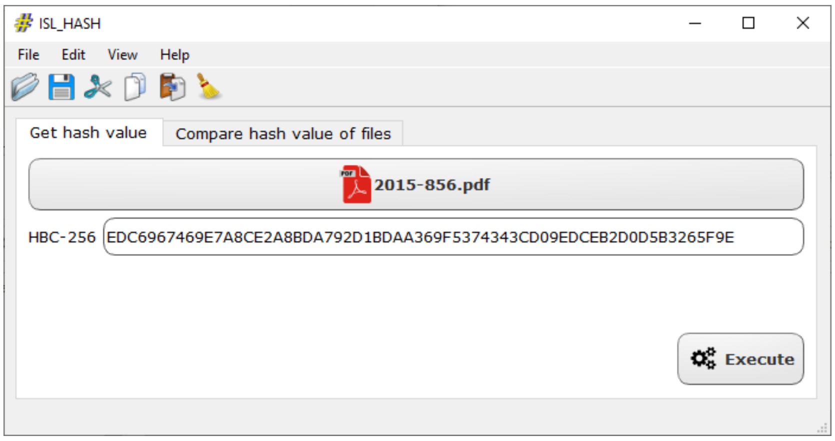
Using this formula, we can determine the number of all cases in the given example from 108 to 148.

5. 4. Software and hardware-software implementation of the developed hashing algorithm

5. 4. 1. Software implementation of the HBC-256 algorithm

The software implementation of the HBC-256 hashing algorithm is made in the form of a data hashing program ISL_HASH 1.0. The program is designed to obtain a hash image of data of arbitrary length. The input data is the content of any file stored on an external storage medium or text entered through the on-screen form. The output is displayed on the screen and can be saved as a "*.hash" file. The program is implemented in C++. No pre-installation is required to run the program. The result of the hash image is displayed as hexadecimal numbers.

Fig. 8 shows the working window of the ISL_HASH 1.0 data hashing program, where the "2015-856.pdf" file is hashed using the HBC-256 algorithm.



HBC-256 EDC6967469E7A8CE2A8BDA792D1BDAA369F5374343CD09EDCEB2D0D5B3265F9E

Fig. 8. ISL_HASH 1.0 data hashing program window

The following are the main technical characteristics of the ISL_HASH 1.0 data hashing program:

– Program type: 32-bit GUI application.
– Product version: 1.0.
– Executable file name: ISL_HASH.EXE.
– Executable file size: 13.5 MB.

5. 4. 2. 1. Hardware-software platform and implementation technology choice

The development board MYIR Z-turn was chosen for implementation. This board is equipped with a single-chip system (hereinafter SoC) Xilinx Zynq XC7Z020, a high-speed USB OTG interface chip, 1GB RAM, and a 16MB NAND Flash memory chip.

The SoC includes:

– Artix-7 field-programmable gate array (hereinafter FPGA);

– microprocessor with Cortex A9 core.

The program code of hashing algorithm HBC-256 for the Cortex processor was written in the C programming language using assembler inserts.

The FPGA design was made using the VHDL technological markup language.

5. 4. 2. 2. Working principle of the Product

The Cortex processor is designed to implement the functions of interacting with a PC, supporting the USB interface, and controlling the FPGA, on which the hardware-software implementation of the HBC-256 algorithm is performed.

Power supply and data exchange with the PC are carried out via the USB interface. Upon initiation, a connection to the PC is established in the Mass Storage Device (MSD) mode. As a drive for storing data, fast RAM is used, in which a 512 MB area is allocated for this purpose. The Cortex processor continually scans this area of memory against the FAT file system for new files. After detecting a new file copied by means of the operating system, the processor sends data blocks to the FPGA via the internal AXI bus using Direct Memory Access (DMA) technology. The FPGA, having received the next data block, performs the transformation in accordance with the description of the HBC-256 algorithm. At the end of the transfer of blocks, the central processor reads the result of the hash algorithm from the FPGA and creates a new file in the area allocated for storing data with a name corresponding to the name of the source file, but with the additional extension "hash". Also, additional debugging information is written to this file – the size of the source file, the number of blocks, the time of the hashing operation, and the speed of the transformation.

5. 4. 2. 3. Debug board resource statistics

The Cortex processor operates at 667 MHz, the FPGA at 150 MHz. The final consumption of the board is about 0.3W. FPGA resources involve 2,370 logical cells, 32 clock cycles for transforming one 384-bit block (Table 7).

In Table 7 we can see the hashing speed of 5 files of different sizes.

Table 7

Examples of performance research results

Item	File size, bytes	Execution speed, Mb/s
1	384,000	179.832
2	1,024,000	179.885
3	64,000,000	179.917
4	128,000,000	179.916
5	246,000,000	179.918

6. Discussion of the hash algorithm

6. 1. Discussion of the developed symmetric block encryption algorithm

The peculiarity of using block ciphers in hashing algorithms is that the security of a hashing algorithm directly depends on the cryptographic strength of the cipher used in it. That is, when using strong block encryption algorithms, the security of the hashing algorithm is guaranteed. But one of the disadvantages of this approach is the reduction in the speed of the algorithm. This is because, during the hashing process, the round keys are updated each time iteratively, depending on the amount of data, i.e. round encryption keys are continuously generated. The developed CF encryption algorithm uses byte data processing, which improves its performance. Alternate execution in one round of linear and non-linear transformations, which are the operation of adding matrix elements modulo two and the replacement table (S-box), provides the diffusion property of the cipher. The proposed algorithm is characterized by the following features:

1. The number of rounds of the used block cipher is reduced, without prejudice to its cryptographic strength, in order to increase the performance of the developed hashing algorithm. The non-linearity of the transformations at

Stage-1 and Stage-3 is ensured by the fact that the S-box is executed twice in one round.

2. The structure of the algorithm provides for the simultaneous execution of several CF compression functions of 128-bit length, which, using parallel computing also speeds up the hashing process. In this paper, we consider the case when k=3, i.e. three 128-bit blocks of hashable data are processed simultaneously. With the increase in the technical characteristics of the processor, the number of simultaneously processed blocks can be increased.

3. Instead of traditional 8-bit S-boxes, four 4-bit S-boxes are used. Such an original approach gives the algorithm another advantage, which lies in the fact that, depending on the arrangement of the matrix elements, the same values of the input data take on different values at the output.

6. 2. Discussion of the developed hash algorithm

Typically, the design of a hash function uses the Wide Pipe construction to deal with multiple collisions. The essence of this construction is to increase the size of the internal state, which makes the search for multiple collisions resource-intensive. However, this scheme requires additional memory. This shortcoming in the proposed algorithm is eliminated by the fact that in one hashing cycle, the CF algorithm is executed k=3 times for different m^j, j=0, 1, 2. The general scheme of the developed hash algorithm HBC-256 is illustrated in Fig. 4. Therefore, the length of the intermediate hash image w is 128*3 bits. By adjusting the parameter k, performance can be improved. The flexibility in optimization, the possibility of parallel computing in hardware implementation and the achievement of an optimal balance of resources/performance should also be noted.

6. 3. Discussion of HBC-256 hash algorithm security study

When evaluating the security of any hash function, three problems are examined [36]:

1. Preimage search, i.e. the search for the message M itself with a known hash digest $h(M)$.

2. The search for a second preimage, i.e. search for a message M_2 with known M_1, with $M_1 \neq M_2$ and $h(M_1)=H(M_2)$.

3. Collision search, i.e. search for any two messages M_1 and M_2 such that $M_1 \neq M_2$ and $h(M_1)=H(M_2)$.

The listed problems in relation to the HBC-256 algorithm are specified by the following parameters. The length of the HBC-256 hash digest is n=256 bits, so the number of all possible hash digests is $N=2^{256}$. For each of the three problems, we define k as the minimum number of implementations with a probability p=0.5. Table 3 presents data on the complexity of attacks for each task.

First, we are discussing the analysis of the avalanche effect of the CF encryption algorithm. As is known, the range of variation of the avalanche parameter lies in the range from 0 to 1, inclusive. The closer the value of the avalanche parameter is to zero, the more the avalanche effect appears in the encryption algorithm. The experiment (Table 4) showed that 98.5 % of the k_i values (probabilities) of the considered rounds lie in the interval (0.41; 0.59). The average of all changes is equal to 49.93 %. Therefore, changing a bit in the input yields about 50 % changes in the output. The analysis showed that the average values of the avalanche parameter ε_α for rounds 1, 2, and 4 are 0.074, 0.071, and 0.073, respectively. The algorithm's avalanche effect is high even after the first round of encryption. For the purity of the experiment, the avalanche criterion was used for the analysis after the 8th, 16th, and 24th rounds of encryption and confirmed the necessary degree of propagation of the avalanche effect of the CF algorithm.

Next, we are considering the analysis of the avalanche effect of the HBC-256 hashing algorithm. We examined the hashing results after the 1st, 2nd, 4th, 8th, and 12th rounds. After the first round, the avalanche parameter average of 0.66 was found to be the worst. However, due to the high spread of the avalanche effect of the CF algorithm, starting from the 2nd round of hashing, an acceptable level of bit diffusion is observed. In Table 5, it could be seen that the hash function after the 1st round does not provide the required degree of the avalanche effect. Its values obtained, depending on the location of the changed bit, are in the interval (0.594, 0.724), which is far from 0. But after the 2nd and subsequent rounds, the statistical indicators take almost the same values, i.e. the range of their deviation from each other is very narrow.

The statistical indicators of the avalanche parameter ε_α of the HBC-256 algorithm given in Table 5 give positive results in evaluating the effectiveness of the algorithm. From Fig. 5, we can conclude that a change in one bit of the input data leads to a 50 % change in the 328-bit hash code. The χ^2 (Chi-square) value of the k_i probabilities is 189.49. Further, with a confidence value α=0.05 and a degree of freedom df=383, the permissible level of agreement with the null hypothesis H_0 is $\chi^2_{\alpha=0.05,df=383}=429.63$. In our case, $\chi^2=189.49>\chi^2_{\alpha=0.05,df=383}=429.63$, therefore the obtained results k_i are positive and, accordingly, the hash algorithm HBC-256 meets the requirements of the avalanche criterion.

Next, we consider the results of graphical and evaluation statistical tests in Fig. 6, 7. During the study, depending on the type of file, different results were obtained for different tests. From the evaluation of the results, it can be argued that the resulting hash digests are statistically secure. Thus, the HBC-256 hashing algorithm under consideration has good statistical properties.

Here we are discussing the results of the analysis for near-collision resistance. As a result, we were able to establish that the number of pairs of hash digests that have a Hamming distance between 108 and 148 is almost 99 % of all possible pairs. This means that hash digests are protected from attack by near collisions. For a near collision, the Hamming distance between two messages should be small, namely up to 16 bits [37]. According to the results of the analysis, the HBC-256 algorithm is resistant to the attack associated with near collisions.

6. 4. Discussion of software and hardware-software implementation of the developed hashing algorithm

6. 4. 1. Software implementation

To conduct a comparative analysis of the results of the developed HBC-256 hashing algorithm, we considered the following two hashing algorithms based on block ciphers:

1) The GOST-R 34.11-2012 Streebog cryptographic algorithm for calculating the hash function, which in 2013 was adopted as a state standard in the Russian Federation. For the analysis, a variant of the algorithm with a hash image size of 256 bits was chosen.

2) The MGR cryptographic algorithm for calculating the hash function proposed by Indian scientists Khushboo Bussi, Dhananjoy Dey, and others [38]. According to the authors, this hash function is a modification of the Streetbog algorithm, where an AES-like block cipher is used as a compression function.

A comparative analysis of the Streebog, MGR, and HBC-256 algorithms was carried out in terms of their efficiency. When testing, all-time measurements were performed on a PC with an Intel(R) Core i7-8700 processor with a frequency of 2.90 GHz and 4 GB RAM.

Table 8

Efficiency results of hash-functions

File size	GOST-R	MGR	HBC-256 (software implementation)
1 MB	3.34 sec.	1.2 sec.	0.58 sec.
5 MB	16.52 sec.	5.84 sec.	2.98 sec.
10 MB	33.01 sec.	11.50 sec.	5.6 sec.
20 MB	66.13 sec.	22.95 sec.	11.94 sec.

From Table 8, it can be seen that the software implementation of the HBC-256 algorithm showed the best results in terms of performance compared to the Streebog and MGR algorithms.

6. 4. 2. Hardware-software implementation (Product)

From Tables 7, 8 we can see that our hardware-software implementation of the HBC-256 algorithm showed very good results of performance compared to our software implementation.

The hardware-software implementation performed can compete with analogs performing hash transformation under existing algorithms. The Product in terms of execution speed and the number of FPGA resources is commensurate with or surpasses the existing analogs. At the same time, since a commercially available, not a tailor-made, debug board was chosen as the hardware platform, it is possible to improve some parameters of the Product, namely:

– dimensional overall features;

– replacing the SoC with a less functional one (less than 10 % of FPGA resources are used, one Cortex processor core is disabled), which will reduce power consumption;

– optimization and parallelization of hardware implementation to achieve an optimal balance of FPGA resources/performance;

– increase the amount of RAM to be able to process larger files.

6. 5. Limitations and further theoretical and practical studies of the hash algorithm

In practical use, the developed hashing algorithm does not require significant restrictions. The results of the study, obtained during the assessment of reliability and speed, showed that the developed hash function fully complies with the main requirements. It was noted that, taking into account modern technological capabilities, the length of the hash digest can be increased. With regard to the amount of hashed information, the parameter k indicated in Fig. 4 should be taken into account. The amount of information to be hashed should be more than $16(k–1)$ bytes. In our case, for k=3, the amount of hashed information must be at least 32 bytes. Otherwise, the round keys of the last part are not used. Research work in this direction will be continued.

A theoretical study of the four 4-bit S-boxes presented in the paper is required. We have considered the first four S-boxes with good cryptographic properties indicated in [33]. Future studies will analyze the influence of the selected four S-boxes on each other since the question of the independence of the choice of S-boxes remains open. Since the S-box is the only cryptographic primitive that provides non-linearity in the algorithm, it should not have any weaknesses.

In the future, in research work, the security of the developed hashing algorithm should be analyzed at a deeper level. It is supposed to carry out a number of cryptographic attacks, as well as differential and linear cryptanalysis. The results of these studies will be used to improve the proposed hashing algorithm.

7. Conclusions

1. As is commonly known, hash functions are built according to an iterative scheme with a number of transformations performed at each step. The transformations include a compressing function, the role of which can be performed by a block cipher. To implement such a scheme, the authors developed a new CF algorithm. Theoretical and experimental tests have shown that the algorithm fully complies with the basic cryptographic requirements. It is assumed that the study of the cryptographic strength of the CF encryption algorithm will be continued in subsequent works.

2. In this paper, we propose a security-oriented hash algorithm HBC-256 based on the CF block cipher. The compression function is based on a Merkle-Damgard construct using a wide-pipe modification that is not susceptible to length expansion attacks. In order to turn the block cipher CF into a one-way compression function, the Davies-Meyer scheme is applied. The scheme of the algorithm is built in such a way as to increase performance through parallel computing by manipulating the parameter k, the number of parts from 3 to 8, depending on the amount of hashed data. The next stage of work will be a further study of the reliability of the proposed HBC-256 algorithm using other cryptanalysis methods and collision search attacks.

3. The hash digest was tested for randomness using the NIST and statistical test suite. From the results obtained, it was found that the binary sequence generated by the proposed algorithm is close to random. The results were examined for the presence of an avalanche effect in the CF encryption algorithm and the HBC-256 hash algorithm itself. Based on the tests and studies carried out, it has been found that the CF encryption algorithm, and therefore the hashing algorithm itself, is efficient to provide a good avalanche effect. The paper presents the statistical indicators of the avalanche parameter ε_α, which shows acceptable results. The resistance of HBC-256 to near collisions has been practically tested. Thus, the HBC-256 algorithm is resistant to attacks related to near collisions. The reliability of the algorithm against various attacks is currently being studied.

4. The structure of the algorithm makes it possible to increase its performance in hardware-software implementation. In addition, the algorithm can be efficiently implemented in hardware.

Acknowledgments

The research work was carried out within the framework of the project OR11465439 – "Development and research of hashing algorithms of arbitrary length for digital signatures and evaluation of their strength" at the Institute of Information and Computational Technologies.

References

1. Teeluck, R., Durjan, S., Bassoo, V. (2020). Blockchain Technology and Emerging Communications Applications. Security and Privacy Applications for Smart City Development, 207–256. doi: https://doi.org/10.1007/978-3-030-53149-2_11
2. Chen, J., Gan, W., Hu, M., Chen, C.-M. (2021). On the construction of a post-quantum blockchain for smart city. Journal of Information Security and Applications, 58, 102780. doi: https://doi.org/10.1016/j.jisa.2021.102780
3. Dworkin, M. J. (2015). SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. NIST. doi: https://doi.org/10.6028/nist.fips.202
4. X 5057-2:2003 (ISO/IEC 10118-2:2000). Available at: http://kikakurui.com/x5/X5057-2-2003-01.html
5. The SM3 Cryptographic Hash Function. Available at: https://tools.ietf.org/id/draft-oscca-cfrg-sm3-02.html
6. DSTU 7564:2014. Information Technologies. Cryptographic Data Security. Hash function. Available at: http://online.budstandart.com/ru/catalog/doc-page?id_doc=66229
7. Kim, D.-C., Hong, D., Lee, J.-K., Kim, W.-H., Kwon, D. (2015). LSH: A New Fast Secure Hash Function Family. Lecture Notes in Computer Science, 286–313. doi: https://doi.org/10.1007/978-3-319-15943-0_18
8. GOST 34.11-2018. Information technology. Cryptographic data security. Hash-function. Available at: https://docs.cntd.ru/document/1200161707
9. STB 34.101.77-2020. Informatsionnye tekhnologii i bezopasnost'. Kriptograficheskie algoritmy na osnove sponge-funktsii. Vzamen STB 34.101.77-2016. Available at: http://www.apmi.bsu.by/assets/files/std/bash-spec24.pdf
10. Zou, J., Dong, L. (2018). Cryptanalysis of the Round-Reduced Kupyna. Journal of Information Science and Engineering, 34 (3), 733–748. doi: https://do.org/10.6688/JISE.201805_34(3).0010
11. Chowdhury, A. R., Chatterjee, T., DasBit, S. (2014). LOCHA: A Light-weight One-way Cryptographic Hash Algorithm for Wireless Sensor Network. Procedia Computer Science, 32, 497–504. doi: https://doi.org/10.1016/j.procs.2014.05.453
12. Tchórzewski, J., Jakóbik, A., Iacono, M. (2021). An ANN-based scalable hashing algorithm for computational clouds with schedulers. International Journal of Applied Mathematics and Computer Science, 31 (4), 697–712. doi: https://doi.org/10.34768/amcs-2021-0048
13. Mondal, A., Mitra, S. (2016). TDHA: A Timestamp Defined Hash Algorithm for Secure Data Dissemination in VANET. Procedia Computer Science, 85, 190–197. doi: https://doi.org/10.1016/j.procs.2016.05.210
14. Bao, Z., Dinur, I., Guo, J., Leurent, G., Wang, L. (2020). Generic Attacks on Hash Combiners. Journal of Cryptology, 33 (3), 742–823. doi: https://doi.org/10.1007/s00145-019-09328-w
15. Andreeva, E., Mennink, B., Preneel, B. (2015). Open problems in hash function security. Designs, Codes and Cryptography, 77 (2-3), 611–631. doi: https://doi.org/10.1007/s10623-015-0096-0
16. Naito, Y. (2012). Blockcipher-Based Double-Length Hash Functions for Pseudorandom Oracles. Lecture Notes in Computer Science, 338–355. doi: https://doi.org/10.1007/978-3-642-28496-0_20
17. Bao, Z., Ding, L., Guo, J., Wang, H., Zhang, W. (2020). Improved Meet-in-the-Middle Preimage Attacks against AES Hashing Modes. IACR Transactions on Symmetric Cryptology, 318–347. doi: https://doi.org/10.46586/tosc.v2019.i4.318-347
18. Nandi, M., Paul, S. (2010). Speeding Up the Wide-Pipe: Secure and Fast Hashing. Lecture Notes in Computer Science, 144–162. doi: https://doi.org/10.1007/978-3-642-17401-8_12
19. A study on hash functions for cryptography (2002). SANS Institute. Available at: https://www.giac.org/paper/gsec/3294/study-hash-functions-cryptography/105433
20. Al-Kuwari, S., Davenport, J., Bradford, R. (2011). Cryptographic Hash Functions: Recent Design Trends and Security Notions. IACR. Available at: https://eprint.iacr.org/2011/565.pdf
21. Denton, B., Adhami, R. (2012). Modern Hash Function Construction. Available at: https://www.researchgate.net/publication/267298547_Modern_Hash_Function_Construction
22. Hosoyamada, A., Yasuda, K. (2018). Building Quantum-One-Way Functions from Block Ciphers: Davies-Meyer and Merkle-Damgård Constructions. Advances in Cryptology – ASIACRYPT 2018, 275–304. doi: https://doi.org/10.1007/978-3-030-03326-2_10
23. Preneel, B., Govaerts, R., Vandewalle, J. (1993). Hash functions based on block ciphers: a synthetic approach. Lecture Notes in Computer Science, 368–378. doi: https://doi.org/10.1007/3-540-48329-2_31
24. Manuel, S., Sendrier, N. (2007). XOR-Hash: A Hash Function Based on XOR. In WEWRC '07.
25. Vergili, I., Yucel, M. D. (2001). Avalanche and Bit Independence Properties for the Ensembles of Randomly Chosen n×x S-Boxes. Turkish Journal of Electrical Engineering & Computer Sciences, 9 (2), 137–145. Available at: https://journals.tubitak.gov.tr/elektrik/issues/elk-01-9-2/elk-9-2-3-0008-1.pdf
26. Mulyarchik, K. S. (2013). Lavinnyy effekt v algoritmakh shifrovaniya na osnove diskretnykh khaoticheskikh otobrazheniy. Doklady BGUIR, 6 (76), 86–91. Available at: https://libeldoc.bsuir.by/bitstream/123456789/1592/1/Mulyarchik_Lavinniy.PDF
27. Dobrovolsky, Y., Prokhorov, G., Hanzhelo, M., Hanzhelo, D., Trembach, D. (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

28. Kapalova, N., Khompysh, A., Arici, M., Algazy, K. (2020). A block encryption algorithm based on exponentiation transform. Cogent Engineering, 7 (1), 1788292. doi: https://doi.org/10.1080/23311916.2020.1788292
29. Algazy, K. T., Babenko, L. K., Biyashev, R. G., Ishchukova, E. A., Kapalova, N. A., Nysynbaeva, S. E., Smolarz, A. (2020). Differential Cryptanalysis of New Qamal Encryption Algorithm. International Journal of Electronics and Telecommunications, 4, 647–653. doi: https://doi.org/10.24425/ijet.2020.134023
30. Lamberger, M., Mendel, F., Rijmen, V., Simoens, K. (2011). Memoryless near-collisions via coding theory. Designs, Codes and Cryptography, 62 (1), 1–18. doi: https://doi.org/10.1007/s10623-011-9484-2
31. Maram, B., Gnanasekar, J. M. (2016). Evaluation of Key Dependent S-Box Based Data Security Algorithm using Hamming Distance and Balanced Output. TEM Journal, 5 (1), 67–75. doi: https://dx.doi.org/10.18421/TEM51-11
32. Biyashev, R. G., Kalimoldayev, M. N., Nyssanbayeva, S. E., Kapalova, N. A., Dyusenbayev, D. S., Algazy, K. T. (2018). Development and analysis of the encryption algorithm in nonpositional polynomial notations. Eurasian Journal of Mathematical and Computer Applications, 6 (2), 19–33. doi: https://doi.org/10.32523/2306-6172-2018-6-2-19-33
33. Saarinen, M.-J. O. (2012). Cryptographic Analysis of All 4 4-Bit S-Boxes. Lecture Notes in Computer Science, 118–133. doi: https://doi.org/10.1007/978-3-642-28496-0_7
34. Kosta, B. P., Sanyasi, P. (2021). Design and Implementation of a Strong and Secure Lightweight Cryptographic Hash Algorithm using Elliptic Curve Concept: SSLHA-160. International Journal of Advanced Computer Science and Applications, 12 (2). doi: https://doi.org/10.14569/ijacsa.2021.0120279
35. Kapalova, N. A., Nysanbaeva, S. E. (2008). Analiz statisticheskikh svoystv algoritma generatsii psevdosluchaynykh posledovatel'nostey. Mater. X Mezhdunar. nauch.-prakt. konf. Informatsionnaya bezopasnost'. Ch. 2. Taganrog: Izd-vo TTI YuFU, 169–172.
36. Ivanov, M. A. Khesh-funktsii. Teoriya, primenenie i novye standarty (chast' 1). Available at: https://docplayer.com/28902735-Hesh-funkcii-teoriya-primenenie-i-novye-standarty-chast-1.html
37. Kumar, M., Dey, D., Pal, S. K., Panigrahi, A. (2017). HeW: AHash Function based on Lightweight Block Cipher FeW. Defence Science Journal, 67 (6), 636. doi: https://doi.org/10.14429/dsj.67.10791
38. Bussi, K., Dey, D., Mishra, P. R., Dass, B. K. (2019). MGR Hash Functions. Cryptologia, 43 (5), 372–390. doi: https://doi.org/10.1080/01611194.2019.1596995