

# Evaluation of the strength and performance of a new hashing algorithm based on a block cipher

Kunbolat Algazy<sup>1,2</sup>, Kairat Sakan<sup>1,2</sup>, Nursulu Kapalova<sup>1,2</sup>

<sup>1</sup>Laboratory of Information Security, Institute of Information and Computational Technologies, Almaty, Kazakhstan

<sup>2</sup>Faculty of Information Technology, Al-Farabi Kazakh National University, Almaty, Kazakhstan

## Article Info

### Article history:

Received Jul 28, 2022

Revised Sep 7, 2022

Accepted Oct 1, 2022

### Keywords:

Algebraic cryptanalysis

Collision

Cryptanalysis

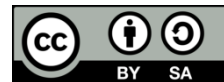
Cryptography

Hash function

## ABSTRACT

The article evaluates the reliability of the new HBC-256 hashing algorithm. To study the cryptographic properties, the algorithm was implemented in software using Python and C programming languages. Also, for the algebraic analysis of the HBC-256 algorithm, a system of Boolean equations was built for one round using the Transalg tool. The program code that implements the hashing algorithm was converted into a software program for generating equations. As a result, one round of the compression function was described as conjunctive normal form (CNF) using 82,533 equations and 16,609 variables. To search for a collision, the satisfiability (SAT) problem solver Lingeling was used, including a version with the possibility of parallel computing. It is shown that each new round doubles the number of equations and variables, and the time to find the solution will grow exponentially. Therefore, it is not possible to find solutions for the full HBC256 hash function.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



## Corresponding Author:

Kairat Sakan

Faculty of Information Technology, Al-Farabi Kazakh National University

71 al-Farabi Ave., Almaty, 050040, Kazakhstan

Email: 19kairat78@gmail.com

## 1. INTRODUCTION

In today's information world one of the key values is to ensure the reliability and security of information. Many information systems, including low-resource internet of things (IoT) devices, use various cryptographic transformations to ensure information security during data storage and transmission. One of the basic cryptographic transformations involved in various security issues is hash functions; one-way mathematical transformations that convert an arbitrary input data array into a unique sequence of fixed length. Modern hash functions are used to implement various information security procedures, such as user authentication [1]–[5], data integrity control [6], [7], electronic signature [8], formation of cryptocurrency transactions [9]–[12], search for malicious software [13]–[17], creation of stego-containers (hash-based approach) [18], and optimization of biometric identification algorithms [19]. The requirements for cryptographic hash functions are as follows.

- High performance: For any message  $M$  it is possible to efficiently calculate the hash value  $h$  in real time.
- Irreversibility (one-way function): Given a known hash value  $h$ , it is computationally difficult to find a message  $M$  with  $h = \text{hash}(M)$ .
- Weak resistance: Given a known message  $M$ , it is computationally difficult to generate (compute) a message  $M'$  such that  $h = \text{hash}(M) = \text{hash}(M')$ .
- Strong resistance: It is computationally difficult to find random messages  $M$  and  $M'$  such that  $\text{hash}(M) = \text{hash}(M')$ .

Currently, hashing functions are built using three constructions. The first one is various non-linear bit functions. The second one is the Merkle-Damgard construction that uses compression functions in the form of block symmetric encryption algorithms. The third one is specialized structures.

Given that the strength of the algorithms used to calculate hash values underlies the security of many information systems and services, an important task is a comprehensive study of the strength of the developed, modernized, and initially applied cryptographic algorithms for calculating hash values to various cryptanalysis methods and methods for detecting collisions. Baseline security recommendations for IoT [20] of the European Union Agency for Network and Information Security (ENISA) describe that to ensure information security in critical information infrastructures, it is required to provide the following:

- GP-TM-24: Authentication credentials shall be salted, hashed, and/or encrypted.
- GP-TM-34: To ensure proper and effective use of cryptography to protect the confidentiality, authenticity, and/or integrity of data and information (including control messages), in transit and at rest. To ensure the proper selection of standard and strong encryption algorithms and strong keys and disable insecure protocols. To verify the robustness of the implementation.
- GP-TM-36: Build devices to be compatible with lightweight encryption and security techniques.
- GP-OP-04: To use proven solutions, i.e., well-known communications protocols and cryptographic algorithms, and recognized by the scientific community. Certain proprietary solutions, such as custom cryptographic algorithms, should be avoided.

Based on recommendations, standards, and international experience in the field of information security, it can be argued that research devoted to the analysis of the strength of cryptographic algorithms of hash functions is relevant and requires continuous work to assess the current state in this field of knowledge for each algorithm used or its modification.

**2. METHOD**

**2.1. HBC-256 hash function**

The hash-based on block cipher (HBC-256) hashing algorithm is new and belongs to the class of new hash functions. A detailed description of the algorithm and some approaches to its analysis are presented in [21]. The HBC-256 hash function is built on the Merkle-Damgard construction. The essence of the design is an iterative process of sequential transformations when the input of each iteration receives a block of the source text and the output of the previous iteration. At each iteration, the transformation occurs by a special compression function (CF).

The general structure of the compression function can be represented as shown in Figure 1. The input of the compression function receives a 128-bit message. This message is also the master key from which the round keys are generated, and the same message is the input message of the compression function. The main difference is that multiple processing (8 rounds) takes place to generate the next key. Also, different Stage-2 operations are applied during key generation and the compression function.

The general hashing scheme processes the message M, which consists of three 128-bit blocks. After all, three blocks have been processed, they are shuffled and the first 256 bits form the desired hash value. If the length of the original message is less than 384 bits, then the message is padded as described in [21].

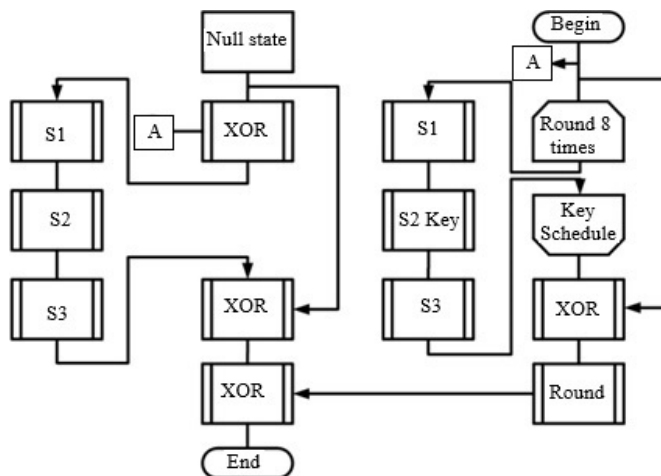


Figure 1. General structure of the compression function

Each round for the compression function consists of three transformations called Stage-1, Stage-2, and Stage-3. For Stage-1 and Stage-3 operations, data is represented as a 4×4 matrix, where each element of the matrix is one byte. For the Stage-1 operation, the transformation is performed from left to right and from top to bottom. To form each element, the addition modulo two is implemented to all elements of the row and column at the intersection of which the element is located, and then it is replaced using S-boxes. The Stage-3 transformation is similar to the Stage-1 transformation but the transformation is applied from right to left and from bottom to top. To replace each byte, two S-boxes are used from those presented in Table 1. The byte that needs to be replaced by S-boxes is in a 4×4 matrix at the intersection of the  $i^{\text{th}}$  column and the  $j^{\text{th}}$  row. Therefore, to convert each byte, it is necessary to divide it into two nibbles. The high nibble is replaced by  $S_i$ -box and the low nibble by  $S_j$ -box. After that, the result of the replacement is reversed: the output of  $S_i$  forms the low nibble of the new state, and the output of  $S_j$  is the high nibble as can be seen in Figure 2.

The Stage-2 transformation consists of two operations: a circular shift and a modulo 2 addition (XOR) operation. The elements of a 4×4 matrix are written as a single block of data by concatenating all bytes. Further, a cyclic shift to the left by one bit is performed. The result of the Stage-2 operation is the result of the modulo two addition of the original state and the state shifted to the left by one bit. When generating a key in the Stage-2 operation, there is no modulo two addition operation, and the result of the function is a shift to the left by one bit.

A hash function or compression function is a function that converts an array of input data of arbitrary length into an output bit string of a fixed length, performed by a certain algorithm. The transformation performed by the hash function is called hashing. The input data is called the input array, key, or message. The result of the transformation is called a hash, hash code, hash sum, message summary, or digest.

In the general case, there is no one-to-one relationship between the hash code and the original data. The values returned by the hash function are less diverse than the values of the input array. The case in which a hash function converts more than one array of input data into the same summaries is called a collision. Collision probability is used to evaluate the quality of hash functions.

For cryptographic hash functions, special requirements apply, i.e., resistance to pre-image and second pre-image attacks and irreversibility. Irreversibility of a hash function is such a property that, for a given value of the hash function  $h(M)$ , it is computationally impossible to find the original block of data  $M$ . Pre-image resistance means that for a given message  $M$  it is computationally difficult to select another message  $M_1$  so that the hash values of these messages match, i.e.,  $hash(M) = hash(M_1)$ . Second pre-image resistance means that it is computationally difficult for any  $M$  to find two different messages  $M$  and  $M_1$  that have the same hash.

Table 1. Four "golden" S-boxes

$x$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0(x)$	0	F	B	8	C	9	6	3	D	1	2	4	A	7	5	E
$S_1(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_2(x)$	7	C	E	9	2	1	5	F	B	6	D	0	4	8	A	3
$S_3(x)$	4	A	1	6	8	F	7	C	3	0	E	D	5	9	B	2

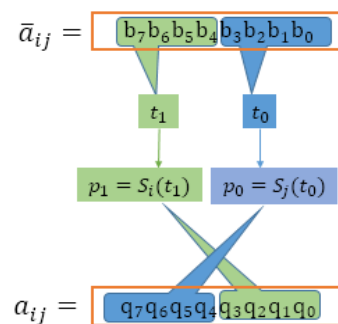


Figure 2. S-box byte transformation

**2.2. Method of algebraic analysis**

Methods of algebraic analysis [22] are universal methods applicable to many varieties of cryptographic algorithms: symmetric block and stream ciphers, and algorithms for computing hash values. Algebraic attacks are based on solving systems of non-linear equations to recover a secret key or message. For the cryptanalysis of hash functions, algebraic attacks can be used to detect collisions and pre-image if a potentially weak compression function is used. The basic idea of algebraic attacks is to recover the secret key

by solving non-linear equations involving the message, the ciphertext, and the key bits. An algebraic attack consists of two stages. Stage 1 is the generation of a sufficient number of non-linear equations of low degree or structured (multidimensional) non-linear equations. Stage 2 is the calculation of key bits by solving a system of equations.

The first step needs to be performed only once for the cryptographic algorithm under consideration. The most commonly used methods for solving equations include linearization algorithms [23]–[25], Gröbner basis [26], and reduction to the SAT problem [27]. Linearization solves the resulting system of non-linear equations by replacing non-linear terms with new variables, so each non-linear monomial is replaced by a new variable. The resulting new system will be linear and can be solved by the Gaussian elimination method. Another class of general algorithms for solving systems of algebraic equations is based on Gröbner bases. In practice, there are some automated tools, such as SAT solvers: CryptoMiniSat, Lingeling, and Cadical, if the number of equations describing the analyzed hash algorithm is not too large.

The algebraic analysis assumes that any encryption process can be represented in the form of algebraic transformations and mathematically describe the explicit dependence of output bits on input bits. The process of compiling such a system (most often just Boolean equations) is quite difficult and takes up most of the time. This type of analysis is not statistical, which means that only a few pairs of plaintext-ciphertext are needed to solve this system. The variables of a Boolean set of equations can take only two values 0 and 1, therefore, the system can be written with several logical bases – ( $\vee, \&, \neg$ ), ( $\&, \neg$ ), ( $\vee, \neg$ ), ( $\oplus, \&$ ). The last three options, respectively, allow us to write the expression in disjunctive normal form (DNF), conjunctive normal form (CNF), and the form of the Zhegalkin polynomial. After creating such an algebraic description, it is necessary to solve the constructed system of equations, which can be done using one of the SAT solvers, the result of which will show whether the system has a solution under given conditions or not.

### 3. RESULTS AND DISCUSSION

#### 3.1. Features of HBC-256 hashing function implementations and experimental data obtained

For hashing functions, we have obtained program implementations using Python and C programming languages. Below is a fragment of the C implementation of the HBC-256 function, which describes the operation of the Stage1 function.

```
void Stage1(struct CompressFunction*HashObject) {
    for (int i = 0; i < NUMBER_OF_ELEMENTS_IN_STATE; i++) {
        for (int j = 0; j < NUMBER_OF_ELEMENTS_IN_STATE; j++) {
            unsigned char tmp = HashObject->state[i][j];
            for (int k = 0; k < NUMBER_OF_ELEMENTS_IN_STATE; k++) {
                tmp ^= HashObject->state[i][k];
            }
            for (int m = 0; m < NUMBER_OF_ELEMENTS_IN_STATE; m++) {
                tmp ^= HashObject->state[m][j];
            }
            HashObject->state[i][j]= SBOX(i, j, tmp);
        }
    }
}
```

Using the obtained implementations, we carried out experiments and time measurements of the processing speed of one message using different personal computer (PC) configurations. During an experiment, the same block of data was hashed 1,000 times, after which the average processing time per data block was calculated. It is important to consider that in multicore systems the experiment was performed using a single core. The results of the experimental measurements are shown in Table 2.

Table 2. Experimental results of software implementations

Algorithm	PC parameters	Language	Max t	Min t	Avg t
HBC-256	Intel(R) Core (TM) i5-11400H	C	0.000728	0.000614	0.000650
HBC-256	Intel Core i5, 8GB RAM	Python	0.052314	0.0276546	0.0379676

#### 3.2. Finding a collision by algebraic analysis

For algebraic analysis, it is necessary to construct a system of Boolean equations. For this purpose, the tool Transalg is used [28], [29]. This software tool converts a cryptographic algorithm into a system of equations and supports writing in the CNF format, in the basis of  $\&$ ,  $\neg$  and in the form of dependencies on the input bits in the symbolic postfix representation.

The program code implementing the hashing algorithm was converted into the program code for generating equations. As a result, one round of the compression function was described as a CNF using 82,533 equations and 16,609 variables. Some of the equations are presented below.

$$\begin{aligned}
 X_{176} &= X_{168} X_{102} \wedge \\
 X_{177} &= X_{169} X_{103} \wedge \\
 X_{178} &= X_{170} X_{104} \wedge \\
 X_{179} &= X_{178} X_{177} \wedge X_{177} X_{178} \wedge X_{176} X_{176} \wedge X_{176} X_{178} \wedge X_{176} X_{177} \wedge \\
 X_{180} &= X_{178} X_{177} X_{178} \wedge X_{176} X_{176} X_{177} \wedge X_{178} \wedge X_{175} X_{175} \\
 X_{181} &= X_{178} X_{177} \wedge X_{176} X_{178} \wedge X_{175} X_{178} \wedge X_{175} X_{177} \wedge X_{178} \wedge \\
 X_{182} &= X_{178} X_{177} \wedge X_{176} X_{177} \wedge X_{175} X_{178} \wedge X_{175} X_{176} \wedge \\
 X_{183} &= X_{174} X_{173} \wedge X_{173} X_{174} \wedge X_{172} X_{174} \wedge X_{172} X_{173} \wedge \\
 X_{184} &= X_{174} X_{173} X_{174} \wedge X_{172} X_{173} \wedge X_{174} \wedge X_{171} X_{171} \\
 X_{185} &= X_{174} X_{173} \wedge X_{172} X_{174} \wedge X_{171} X_{174} \wedge X_{171} X_{173} \wedge X_{174} \wedge \\
 X_{186} &= X_{174} X_{173} \wedge X_{172} X_{173} \wedge X_{171} X_{174} \wedge X_{171} X_{172} \wedge \\
 X_{187} &= X_9 X_{179} \wedge \\
 X_{188} &= X_{10} X_{180} \wedge \\
 X_{189} &= X_{11} X_{181} \wedge \\
 X_{190} &= X_{12} X_{182} \wedge \\
 X_{191} &= X_{13} X_{183} \wedge \\
 X_{192} &= X_{14} X_{184} \wedge \\
 X_{193} &= X_{14} X_{184} \wedge
 \end{aligned}$$

The correctness of the constructed equations was checked using control values for the input and output of the compression function using special code in Java. To partially generate the system and solve it, the use of an SAT solver is necessary. We chose a series of SAT solvers Lingeling, including a version with the ability to parallelize the calculation of Plingeling, as well as cubic and competitive versions of Treengeling and Lingeling. Value checking is performed on one state of one round of the compression function [30].

To test the operation, Lingeling was run with the original system of equations and the constraint on the values of output variables, that is, finding the values of input variables with known output variables (restoring the prototype). These calculations were run with the condition that the outputs are equal to the test condition. The solution was known, that is, there was a check for side solutions (collisions), as well as an estimate of the computation speed of the known input value (test restoration of the prototype). Without additional options using a single-processor kernel, this problem took 241,000 sec = 67 hours and did not find an existing solution or an additional one Figure 3. To speed up solution finding, some of the input variables were marked and the speed of finding the solution was checked on a test case. Data with the calculation time for partially marked values are shown in Tables 3 and 4. Thus, the constructed system of equations with some probability allows obtaining a prototype for one round of compression function. Further work should be aimed at constructing a system of equations describing a full-round hashing function. Also, the solution search algorithm can be reconfigured to find a first-order collision.

c	s	seconds	irredundant	redundant clauses	glue iterations"	MB stability
c	s	variables	conflicts	large ternary binary	jlevel jlevel'	agility tlevel
c 1	s 1321200.1	15832	81756	60400771	2432040	1940 87 58 131 43 0 524 49 987 5010
c 2	s 1321200.1	10088	78408	26500005	2907539	1735 0 86 123 97 -1 458 48 990 5901
c 0	s 1321200.4	10076	78435	55252397	2875160	1795 0 59 143 47 -15 488 50 984 3245
c 5	s 1321200.6	10060	79816	57215567	2911408	1735 0 61 131 45 -2 475 49 988 3020
c 3	s 1321200.7	10080	78413	55188498	2876906	1805 0 63 134 47 -6 466 50 989 5203
c 4	s 1321201.0	10071	78512	90044163	3174600	1827 0 46 177 29 -11 508 49 974 2135
c 1	s 1324800.4	10032	81756	60509350	2430984	1940 87 59 127 43 -1 524 49 990 4429
c 0	s 1324800.2	10076	78435	55326336	2972343	1795 0 64 154 47 -23 545 49 987 5562
c 4	s 1324800.1	10071	78512	90214095	3169680	1830 0 46 168 29 3 548 49 976 2750
c 5	s 1324800.4	10060	79816	57281178	2852573	1735 0 59 164 45 -7 457 50 987 5510
c 2	s 1324800.4	10088	78408	26544507	2938074	1735 0 85 121 97 0 481 50 990 5989
c 3	s 1324800.8	10080	78413	55349526	2937044	1805 0 48 145 47 -14 503 52 987 4699
c 1	s 1328400.1	10032	81756	60596516	2534618	1945 89 62 139 43 -12 590 50 988 8871
c 5	s 1328400.2	10060	79816	57322403	2916374	1735 0 53 137 45 -8 501 48 984 5397
c 4	s 1328400.4	10071	78512	90354947	3146121	1830 0 46 148 29 -9 526 49 973 3105
c 2	s 1328400.6	10088	78408	26587538	2924946	1735 0 85 123 97 -0 469 48 990 6023
c 0	s 1328400.9	10080	78435	55490715	2867430	1795 0 50 190 47 26 467 50 981 2151
c 3	s 1328400.6	10076	78413	55419123	2904285	1805 0 57 138 47 -9 490 50 988 4795
c 0	s 1322000.2	10076	78435	55579671	2975017	1795 0 54 203 46 24 541 49 981 2403
c 1	s 1322000.4	10032	81756	60689363	2523433	1947 90 65 133 43 -1 577 50 989 5703
c 3	s 1322000.1	10080	78413	55508296	2896525	1805 0 57 151 46 -17 473 51 984 5655
c 5	s 1322000.4	10060	79816	57407588	2890312	1735 0 61 134 45 -3 464 49 985 5061
c 4	s 1322000.5	10071	78512	90475820	3113113	1830 0 46 159 29 18 511 49 969 2067
c 2	s 1322000.8	10088	78403	26630505	2913884	1735 0 84 121 97 0 456 50 990 5753

Figure 3. Calculation of the prototype in Lingeling

Table 3. Computation speed in seconds for calculations on a single-processor core

Number of unknown bits	0	8	16	24
Plingeling	-	15.8	1851	unknown
Treengeling	0.11	19.54	105.41	207987.68

Table 4. Computation speed in seconds for calculations on a six-processor core

Number of unknown bits	0	8	16	24	128
Plingeling	-	0.1	2.9	27807.5	39331.7

#### 4. CONCLUSION

The results obtained correspond to the chosen methods of analysis. For the hashing algorithm, we obtained implementations using Python and C programming languages, which were tested using different computer configurations (Table 1). Algebraic analysis of one round of the hashing function HBC-256 yielded a system of 82,533 equations and 16,609 variables. It took about 11 hours to solve the system and allow us to determine the prototype for one round of encryption (Tables 3 and 4). It should be noted that Plingeling uses randomization in its algorithms to find a solution. Therefore, only one experiment out of five ended with a successful finding of the full prototype. That said, it is naturally clear that with each new round the number of equations and variables will double, and the time to find a solution will grow exponentially. Thus, at the moment it is not possible to find solutions for the full HBC-256 hashing function.

The HBC-256 hashing algorithm under consideration is new and was first presented. Currently, there are no publications on the study of the properties of this hashing algorithm. The reliability of new cryptographic algorithms is confirmed by thorough multiple studies of different aspects of robustness. For hashing functions, it is research in the field of irreversibility and searches for collisions.

The biggest limitation when conducting research is the difficulty of using full-size inputs and outputs for the developed hashing algorithms because the analysis becomes time-consuming and demands computational resources and time. One solution to this problem is to use reduced models or functions to model and approximate the result. The study has not identified any vulnerabilities in the full-round hashing algorithm. Not all possible analysis methods were considered. The methods used were not always applied to full-run versions of the algorithm. All of this will be improved upon in the future. This study is only the first step in investigating the properties of the new hashing algorithm. The proposed approaches can be improved.

#### ACKNOWLEDGEMENTS

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 assessment of their strength" at the Institute of Information and Computational Technologies.

#### REFERENCES




- [1] S. L. Nita and M. I. Mihailescu, "Hash functions," in *Cryptography and Cryptanalysis in Java*, Berkeley, CA: Apress, 2022, pp. 101–112, doi: 10.1007/978-1-4842-8105-5\_8.
- [2] N. Kheshaifaty and A. Gutub, "Engineering graphical captcha and AES crypto hash functions for secure online authentication," *Journal of Engineering Research*, Nov. 2021, doi: 10.36909/jer.13761.
- [3] P. Farshim and S. Tessaro, "Password hashing and preprocessing," in *EUROCRYPT 2021: Advances in Cryptology – EUROCRYPT 2021*, 2021, pp. 64–91, doi: 10.1007/978-3-030-77886-6\_3.
- [4] J. Herrera and M. L. Ali, "Concerns and security for hashing passwords," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, Nov. 2018, pp. 861–865, doi: 10.1109/UEMCON.2018.8796720.
- [5] M. A. D. Brogada, A. M. Sison, and R. P. Medina, "Head and tail technique for hashing passwords," in *2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN)*, Jun. 2019, pp. 805–810, doi: 10.1109/ICCSN.2019.8905384.
- [6] V. Fomichev, D. Bobrovskiy, A. Koreneva, T. Nabiev, and D. Zadorozhny, "Data integrity algorithm based on additive generators and hash function," *Journal of Computer Virology and Hacking Techniques*, vol. 18, no. 1, pp. 31–41, Mar. 2022, doi: 10.1007/s11416-021-00405-y.
- [7] J. Wang, W. Luo, Y. Hu, and H. Jiang, "PN-HASH: An immune-inspired scheme for data integrity check," in *2020 12th International Conference on Advanced Computational Intelligence (ICACI)*, Aug. 2020, pp. 340–348, doi: 10.1109/ICACI49185.2020.9177796.
- [8] T. Espitau, "Mitaka: Faster, simpler, parallelizable and maskable hash-and-sign signatures on NTRU lattices," in *Proceedings of the 8th ACM on ASIA Public-Key Cryptography Workshop*, May 2021, pp. 1–1, doi: 10.1145/3457338.3458293.
- [9] O. Belej, K. Staniec, and T. Więkowski, "The need to use a hash function to build a crypto algorithm for blockchain," in *Theory and Applications of Dependable Computer Systems*, 2020, pp. 51–60, doi: 10.1007/978-3-030-48256-5\_6.
- [10] N. Yuvaraj and P. Mohanraj, "Radial kernelized regressive Merkle–Damgård cryptographic hash blockchain for secure data transmission with IoT sensor node," *Peer-to-Peer Networking and Applications*, vol. 14, no. 4, pp. 1998–2010, Jul. 2021, doi: 10.1007/s12083-021-01135-0.
- [11] N. R. Chilambarasan and A. Kangaiammal, "Matyas–Meyer–Oseas skein cryptographic hash blockchain-based secure access control for E-learning in cloud," in *Inventive Systems and Control*, 2021, pp. 895–909, doi: 10.1007/978-981-16-1395-1\_65.
- [12] H. K. Patil, "Blockchain technology-security booster," in *IGI Global*, 2021, pp. 128–139, doi: 10.4018/978-1-7998-2414-5.ch008.
- [13] R. Punithavathi, K. Venkatachalam, M. Masud, M. A. AlZain, and M. Abouhawwash, "Crypto hash based malware detection in IoMT framework," *Intelligent Automation & Soft Computing*, vol. 34, no. 1, pp. 559–574, 2022, doi: 10.32604/iasc.2022.024715.
- [14] N. Naik, P. Jenkins, N. Savage, L. Yang, T. Boongoen, and N. Lam-On, "Fuzzy-import hashing: A malware analysis approach," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Jul. 2020, pp. 1–8, doi: 10.1109/FUZZ48607.2020.9177636.
- [15] S. C. Peiser, L. Friborg, and R. Scandariato, "JavaScript malware detection using locality sensitive hashing," in *ICT Systems Security*

*Evaluation of the strength and performance of a new hashing algorithm based ... (Kunbolat Algazy)*




- and Privacy Protection. *SEC 2020*, 2020, pp. 143–154, doi: 10.1007/978-3-030-58201-2\_10.
- [16] D. Moon, J. Lee, and M. Yoon, “Compact feature hashing for machine learning based malware detection,” *ICT Express*, vol. 8, no. 1, pp. 124–129, Mar. 2022, doi: 10.1016/j.icte.2021.08.005.
- [17] T. Baba, K. Baba, and T. Yamauchi, “Malware classification by deep learning using characteristics of hash functions,” in *Advanced Information Networking and Applications. AINA 2022*, 2022, pp. 480–491, doi: 10.1007/978-3-030-99587-4\_40.
- [18] R. Riasat, I. S. Bajwa, and M. Z. Ali, “A hash-based approach for colour image steganography,” in *International Conference on Computer Networks and Information Technology*, Jul. 2011, pp. 303–307, doi: 10.1109/ICCNET.2011.6020886.
- [19] S. Karaman and S.-F. Chang, “Hashing for face search,” in *Computer Vision*, Cham: Springer International Publishing, 2021, pp. 553–558, doi: 10.1007/978-3-030-63416-2\_817.
- [20] ENISA, “Baseline security recommendations for IoT,” European Union Agency for Network and Information Security <https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot/@/download/fullReport> (accessed Jun. 01, 2022).
- [21] K. Sakan, S. Nyssanbayeva, N. Kapalova, K. Algazy, A. Khompys, and D. Dyusenbayev, “Development and analysis of the new hashing algorithm based on block cipher,” *Eastern-European Journal of Enterprise Technologies*, vol. 2, no. 9 (116), pp. 60–73, Apr. 2022, doi: 10.15587/1729-4061.2022.252060.
- [22] G. V. Bard, *Algebraic cryptanalysis*. Boston, MA: Springer US, 2009, doi: 10.1007/978-0-387-88757-9.
- [23] N. T. Courtois and J. Pieprzyk, “Cryptanalysis of block ciphers with overdefined systems of equations,” in *ASIACRYPT 2002: Advances in Cryptology — ASIACRYPT 2002*, 2002, pp. 267–287, doi: 10.1007/3-540-36178-2\_17.
- [24] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, “Efficient algorithms for solving overdefined systems of multivariate polynomial equations,” in *Eurocrypt’2000, LNCS 1807*, 2000, pp. 392–407, doi: 10.1007/3-540-45539-6\_27.
- [25] R. Biyashev, D. Dyusenbayev, K. Algazy, and N. Kapalova, “Algebraic cryptanalysis of block ciphers,” in *Proceedings of the 2019 International Conference on Wireless Communication, Network and Multimedia Engineering (WCNME 2019)*, 2019, pp. 129–132, doi: 10.2991/wcnme-19.2019.30.
- [26] M. Bardet, J.-C. Faugère, and B. Salvy, “On the complexity of the F5 Gröbner basis algorithm,” *Journal of Symbolic Computation*, vol. 70, pp. 49–70, Sep. 2015, doi: 10.1016/j.jsc.2014.09.025.
- [27] E. Ishchukova, E. Maro, and P. Pristalov, “Algebraic analysis of a simplified encryption algorithm GOST R 34.12-2015,” *Computation*, vol. 8, no. 2, May 2020, doi: 10.3390/computation8020051.
- [28] A. Biere, “Lingeling, Plingeling, and Treengeling entering the SAT competition 2013,” *Proceedings of SAT Competition*, 2013, pp. 51–52.
- [29] I. Otpuschennikov, A. Semenov, I. Gribanova, and O. Zaikin, “Encoding cryptographic functions to SAT using TRANSALG system,” in *ECAI’16: Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 2016, pp. 1594–1595, doi: 10.3233/978-1-61499-672-9-1594.
- [30] A. Biere, “lingeling,” Github. <https://github.com/arminbiere/lingeling> (accessed Jun. 03, 2022).

## BIOGRAPHIES OF AUTHORS






**Kunbolat Algazy**    received a master's degree in mathematics from Al-Farabi Kazakh National University in 2001 and a Ph.D. degree in information security systems, in Almaty, Kazakhstan, in 2021. From 2001 to 2014 he worked in the field of information protection in the state structure. Between 2014 and 2016, he worked as a teacher at the Department of Mathematics at Satbayev University. Currently, he is a researcher in the laboratory "Information Security" at the Institute of Information and Computing Technology. His research interests include cryptography, cryptanalysis, development, and research in the field of information protection. He can be contacted at [kunbolat@mail.ru](mailto:kunbolat@mail.ru).



**Kairat Sakan**    graduated from the Faculty of Mechanics and Mathematics of Al-Farabi Kazakh National University, majoring in mathematics and applied mathematics (KazNU, Almaty, Kazakhstan) in 2001. From 2001 to 2002, he worked as a teacher at the Department of Applied Mathematics and Mathematical Modeling at KazNU. From 2003 and 2005, he worked as a junior researcher at the Research Institute of Mathematics and Mechanics (IMM) of KazNU. After that, he worked in the field of information protection in the state structure for several years. Since 2018, he has been working as a mathematician in the information protection laboratory at the Scientific Institute of Information and Computing Technologies. Currently, he is a doctoral student at KazNU, majoring in information security systems. His field of scientific research is information protection in the public and private sectors. He can be contacted at [19kairat78@gmail.com](mailto:19kairat78@gmail.com).



**Nursulu Kapalova**    received her master's degree in mathematics from Al-Farabi Kazakh National University in 2002 and her degree candidate in technical sciences (Almaty, Kazakhstan) in 2009. Currently, she is a leading researcher in the laboratory of "Information Security" at the Institute of Information and Computing Technology and an associate professor at the Department of "Information Systems" at Al-Farabi Kazakh National University. Her areas of scientific works are development and research in the field of information protection. She can be contacted at [nkapalova@mail.ru](mailto:nkapalova@mail.ru).