International Journal of Electrical and Computer Engineering (IJECE)
Vol. 15, No. 3, June 2025, pp. 3191~3201
ISSN: 2088-8708, DOI: 10.11591/ijece.v15i3.pp3191-3201

Application of satisfiability problem solvers for assessing the strength of hash algorithms

Kunbolat Algazy, Kairat Sakan, Andrey Varennikov, Nursulu Kapalova
Information Security Laboratory, Institute of Information and Computational Technologies, Almaty, Republic of Kazakhstan

Article Info

Article history:

Received Jun 11, 2024
Revised Dec 17, 2024
Accepted Jan 16, 2025

Keywords:

Boolean formulas
Cryptanalysis
Hash function
Propositional encoding
Satisfiability problem

ABSTRACT

This article presents a methodology for assessing the strength of cryptographic algorithms and provides experimental data obtained from studying the cryptographic strength of the developed hash function HBC-256 using modern satisfiability problem (SAT) solvers. Various SAT solvers implementing the conflict-driven clause learning (CDCL) algorithm, based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm, were used to conduct the cryptanalysis of the HBC-256 hash function. The most effective was the parallel SAT solver Parkissat, and thus it was used for more in-depth research. A series of experiments were conducted to determine how resistant the HBC-256 hashing algorithm is to preimage attacks for one, two, three, and four rounds. For this purpose, four sets of files were prepared using special propositional encoding tools, each set including 30 files in the standard of center for discrete mathematics and theoretical computer sciences (DIMACS) format. These files contain Boolean formulas in conjunctive normal form (CNF), used as input for modern SAT solvers. To obtain more accurate time measurements, the same experiment was repeated multiple times, after which the average time was determined. The results of this study show that SAT solvers encounter significant difficulties when attempting to solve the preimage search problem for the full-round version of the HBC-256 hash function, even when only 30 bits of the original message are unknown.

Corresponding Author:

Kairat Sakan
Information Security Laboratory, Institute of Information and Computational Technologies
28 Shevchenko, Almaty, 050010, Republic of Kazakhstan
Email: 19kairat78@gmail.com

1. INTRODUCTION

Ensuring computer security in large open networks such as the internet is one of the most pressing areas of modern informatics. However, providing security is a challenging task as it involves addressing numerous technical issues to guarantee the highest possible level of reliability to meet the needs of regular users. These issues necessitate the use of specific protocols to ensure secure data exchange and modern cryptographic methods. Even when a protocol has a formal proof of security, it can still be compromised if the cryptographic algorithm used for its implementation possesses undesirable algebraic properties. Thus, one of the essential steps in creating a cryptographic scheme, algorithm, or protocol is conducting an initial cryptanalysis to assess the strength of the proposed scheme.

Modern methods successfully applied to the analysis of cryptographic algorithms, such as linear cryptanalysis or differential cryptanalysis, rely solely on statistical tools. Another approach for analyzing cryptographic algorithms to obtain more reliable security guarantees is propositional encoding followed by the application of satisfiability (SAT) solvers [1]. Many practically significant problems related to

information management and processing in discrete systems can be effectively reduced to the satisfiability problems of Boolean formulas. This applies to problems of synthesis and verification in microelectronics, some theoretical programming issues, inversion problems of discrete functions, management of communication protocols, and many others.

A Boolean formula of n variables is an expression constructed according to specific rules over an alphabet that includes Boolean variables $x_1, \ldots, x_n$, parentheses, and special symbols that are known as Boolean (propositional) connectives. Boolean formulas are also referred to as "propositional formulas" or "formulas of logic algebra." There are Boolean formulas presented in a special form known as normal forms. The primary object of further consideration will be Boolean formulas in conjunctive normal form (CNF) [2].

Let $\{0,1\}^n$ be the set of all words of length n over the alphabet $\{0,1\}$. In several sources, the elements of $\{0,1\}^n$ are referred to as Boolean vectors. Any Boolean formula F of n variables define a completely defined Boolean function $f_F: \{0,1\}^n \to \{0,1\}$. A formula F is called satisfiable if there exists an assignment $\alpha \in \{0,1\}^n$ of values to the variables in F such that $f_F(\alpha) = 1$. Such an assignment is called a satisfying assignment for F. If no such assignment exists, F is called unsatisfiable. For any Boolean formula F, a circuit can be constructed from functional elements over an arbitrary complete basis, for example, $\{\wedge,\neg\}$. From this circuit, using Tseitin transformations, one can construct a CNF $C(F)$ that is satisfiable if and only if F is satisfiable. The formula $C(F)$ is generally a formula with more variables than F. However, importantly, the transition from F to $C(F)$ is performed in polynomial time relative to the length of the binary representation of F. Given this, everywhere below, the Boolean satisfiability problem will refer to the problem of satisfiability of any CNF: that is, given an arbitrary CNF C, the question is to determine whether C is satisfiable. SAT is a classical NP-complete problem [3]. Thus, if $P \neq NP$, then SAT cannot be solved in polynomial time (relative to the size of the CNF) in the general case. Nevertheless, the last 20 years have seen significant progress in the development of SAT-solving algorithms, achieving impressive results on extensive classes of so-called industrial benchmarks [4], [5].

One of the most important classes of logical equations is formed by equations in the form of CNF=1, where 1 denotes the true value. The problems of finding solutions to this class of equations belong to the so-called SAT problems. Special software tools called SAT solvers are used to solve SAT problems. For some cases, SAT solvers allow finding a satisfying assignment for the CNF, *i.e.*, a set of variable values that evaluates the CNF to "true." From this assignment, the desired secret key can be efficiently obtained. This approach is called SAT cryptanalysis.

In the field of cryptanalysis, SAT solvers can be applied to a variety of tasks [6]. They are used to search for a key using a ciphertext-only attack, or to search for a key using a known-plaintext attack. Additionally, SAT solvers are employed to prove that a cipher is faithful, meaning the same ciphertext cannot be generated using different keys, which implies the absence of a universal key capable of decrypting ciphertext encrypted with any other key. They also demonstrate that a cipher is not closed, which means that for any two keys, there does not exist a third key such that double encryption with the first two keys is equivalent to encryption with the third key. Furthermore, SAT solvers can prove that a cipher does not have weak keys, specifically showing that no keys exist for which double encryption is equivalent to sequential encryption and decryption operations. Lastly, SAT solvers are used to evaluate the strength of hashing algorithms by searching for collisions, first or second preimages, and other vulnerabilities.

Scientific studies on the application of SAT solvers for analyzing cryptographic hash functions have demonstrated various approaches and their effectiveness. These works explore both the theoretical aspects and the practical efficiency of applying SAT solvers in this field. Researchers have shown how SAT solvers can be utilized to identify vulnerabilities in hash functions, such as finding collisions or pre-images. Moreover, advancements in SAT solver algorithms have significantly improved their performance, making them a powerful tool for evaluating the security of modern cryptographic hash functions.

In the study [7], SAT solvers were used to find collisions in the SHA-256 hash function based on so-called "semi-free-start" collisions. Authors integrated the programmatic SAT+CAS paradigm with differential cryptanalysis methods previously employed in collision attacks on SHA-256. Although these attacks are still far from finding collisions for the full version of SHA-256, they show progress in applying SAT solvers for analyzing the security of hash functions.

Programmable SAT solvers for cryptanalysis allow the customization of the SAT-solving process for specific cryptographic tasks, making them more flexible and effective for analyzing hash functions [8]. They enable the consideration of cryptographic operations' characteristics and apply them during the propagation and conflict analysis processes. The authors enhance the propagation and conflict analysis mechanisms of conflict-driven clause learning (CDCL) solvers by incorporating specialized algorithms tailored to the cryptographic primitives under examination. This method proves to be highly effective, particularly in the differential path analysis and algebraic fault detection of hash functions. Preliminary

results highlight the potential of this approach, confirming it as a substantial advancement compared to traditional Blackbox SAT-based cryptanalysis techniques.

Lingeling is a highly optimized SAT solver that has been used successfully in various SAT competitions. It has been expanded into two parallel variants: Plingeling and Treengeling. In study [9], two solvers, Plingeling and Treengeling, are described, both of which support parallel and distributed computing. Plingeling divides the task among multiple threads on a single processor, while Treengeling distributes the work across several nodes in a distributed system.

While SAT solvers represent a powerful tool for solving certain classes of problems, their application in cryptography is limited due to the complexity, scale, and specificity of cryptographic tasks. Successful use of SAT solvers in cryptography requires a deep understanding of both the cryptographic primitives themselves and the methods for transforming these problems into a solvable form for SAT solvers. The main limitations in applying SAT solvers include aspects such as exponential complexity, excessively large and complex Boolean formulas, and the challenging scalability of simplified versions of cryptographic algorithms to full-round versions. Overcoming these limitations in solving cryptographic tasks with SAT solvers requires a comprehensive approach, including algorithm optimization, the development of new methods, and the application of various strategies. One way to overcome these limitations is through the use of parallel computing. The use of parallel or distributed computing in SAT solvers is especially beneficial when dealing with large formulas, as it allows different parts of the task to be processed simultaneously, significantly improving the efficiency of solving problems. Based on this, the SAT solver experiments in this research were conducted on the HP Enterprise DL380 Gen10/2 Xeon Gold server.

SAT solvers are powerful tools for evaluating the cryptographic strength of hash functions. With the continuous growth of computational capabilities, research is ongoing to enhance the efficiency of solvers and adapt them for analyzing modern cryptographic hash functions. The scientific community explores various approaches and SAT solvers to identify vulnerabilities such as collisions, first-preimage, and second-preimage attacks. The essence of this research lies in assessing the cryptographic strength of the newly developed hash function HBC-256 using state-of-the-art SAT solvers, as the development of any hash function must be accompanied by a thorough security analysis, including modeling potential attacks with SAT solvers.

To address this issue, the study conducted a practical cryptanalysis of the cryptographic hash function HBC-256. A series of experiments were carried out to evaluate the resilience of the HBC-256 hashing algorithm against preimage attacks. This article presents an evaluation of the performance of parallel and sequential SAT solvers in solving preimage search problems for the HBC-256 hashing algorithm. The contributions of this work are as follows:

- A brief overview of existing methods and approaches for checking the satisfiability of a Boolean function represented in CNF and finding its value set is provided;
- A comparative analysis of the effectiveness of the SAT solvers Lingeling, CaDiCaL, Kissat, Plingeling, Treengeling, and Parkissat is conducted using the HBC-256 hash function as an example;
- Using Parkissat, which demonstrated the best results in the comparative analysis, the high cryptographic strength of the HBC-256 hash function is proven.

2. METHOD

The following methods were employed during the research: i) propositional encoding to transform cryptographic algorithms into Boolean formulas and ii) solution search for the satisfiability problem of Boolean formulas using efficient SAT solvers. The use of SAT solvers in algebraic cryptanalysis provides a powerful tool for solving complex systems of equations, making this approach particularly effective for analyzing cryptographic systems. The advantage of algebraic cryptanalysis lies in its versatility, as this approach can be applied to any cipher that can be expressed in algebraic form, including symmetric ciphers, asymmetric schemes, elliptic curve-based schemes, and hash functions, whereas differential and linear cryptanalysis are generally limited to symmetric block ciphers and their effectiveness depends on the specifics of the algorithm. However, the application of SAT solvers to cryptographic problems requires significant computational resources, limiting their practical use in some cases. Therefore, in terms of performance and efficiency, the main focus will be on selecting the specific SAT solver.

2.1. Propositional encoding to transform cryptographic algorithms into Boolean formulas

Since manual encoding is impractical, cryptographic algorithms are typically transformed into Boolean formulas using specialized tools. These tools enable the expression of cryptographic algorithms in high-level programming languages, followed by the conversion of the resulting program into the corresponding Boolean formula in CNF. The resulting formula is then saved in a file with the standard DIMACS format, making it compatible for use as input data by modern SAT solvers.

For the preparation of experimental data in this paper, the Transalg software suite was employed as such a tool. Transalg was developed by the Matrosov Institute for system dynamics and control theory of the Siberian Branch of the Russian Academy of Sciences, located in Irkutsk [10]. This suite facilitates the conversion of arbitrary algorithms into Boolean formulas. These algorithms compute everywhere-defined discrete functions, *i.e.*, functions of the form $f: \{0,1\}^* \to \{0,1\}^*$, and are expressed using the specially designed programming language TA akin to C [11].

2.2. Searching for solutions to Boolean formula satisfiability problems using efficient SAT solvers

Any Boolean satisfiability problem consists of two key subtasks-checking the satisfiability of an arbitrary Boolean function represented in CNF and finding a set of values for which such a CNF is satisfied. The foundation of most SAT solvers is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [12], [13] which was specifically proposed for determining the satisfiability of Boolean formulas written in CNF, *i.e.*, for solving SAT problems. The DPLL algorithm serves as the basis for most efficient SAT solvers.

The main idea of the DPLL algorithm is to apply depth-first search methods and utilize the unit propagation rule. The DPLL algorithm divides the set of variables of the CNF formula into two subsets, A and B, where subset A contains variables with a value of "true" and subset B contains variables with a value of "false." At each step, an arbitrary variable from the CNF formula is chosen, and it is assigned a value of "true" (adding the variable to subset A). Then the original formula is simplified, and the simplified problem is solved. If the simplified CNF formula is satisfiable, then the variable value chosen is correct; otherwise, the chosen variable is assigned a value of "false," and it is moved to subset B. The problem is then solved again for the chosen "false" variable value. Thus, either the correct variable value ("true" or "false") will be found, or it will be proven that the original formula is unsatisfiable [14], [15].

The foundation of the vast majority of modern complete SAT solvers, effective on broad classes of practical tests, is the conflict-driven clause learning (CDCL) algorithm. In turn, this algorithm is based on the DPLL algorithm. The main difference between these two algorithms is that CDCL uses memory to store search history more precisely, and information about dead-end branches of the search tree is recorded in the form of new constraints, called conflict clauses [16]–[20]. This allows for significantly deeper backtracking in some cases than in DPLL. In addition to conventional or sequential SAT solvers, parallel SAT solvers have gained widespread use. These solvers are divided into three categories: portfolio solvers, divide-and-conquer solvers, and solvers based on parallel local search.

Portfolio solvers are built on the use of multiple algorithms or different configurations of the same algorithm. All solvers in the parallel portfolio work on different processors to solve the same problem. If one solver program is completed, the portfolio solver reports whether the problem is satisfiable or unsatisfiable according to that solver. All other solvers stop working. Diversifying portfolios by including different solvers, each of which performs well on different types of problems, enhances solver efficiency. Many solvers use random number generators internally. Diversifying their initial values (seeds) is a simple way to diversify the portfolio. Other diversification strategies include enabling, disabling, or diversifying certain heuristics in the sequential solver [21], [22].

Unlike parallel portfolios, parallel divide-and-conquer solvers attempt to divide the search space among individual instances of SAT solvers running in parallel threads. However, due to the use of methods such as unit propagation, after partitioning the SAT problem into individual subproblems, they may vary significantly in complexity, leading to the challenging problem of load balancing. One strategy for parallel local search to solve SAT involves using opposite truth values for one or more variables (flip variables) in different nodes of the computing system. These variables are selected using heuristics that attempt to determine how this change can expedite the SAT problem-solving process by reducing the number of unsatisfied clauses. Another approach is to apply the aforementioned portfolio approach [23], [24].

Despite the abundance of SAT solvers of various types available today, a universal solution that performs equally well for all tasks has not yet been found. Each solver may excel at tasks that others struggle with while performing significantly worse on some other tasks. It is practically impossible to predict in advance which solver should be used in a given situation, so the choice of solver that best suits solving a particular problem is determined experimentally.

2.3. Hash function HBC-256

The cryptographic algorithm chosen for testing using modern SAT solvers was the HBC-256 data hashing algorithm, developed in the Information Security Laboratory of the Institute of Information and Computational Technologies of the Ministry of Science and Higher Education of the Republic of Kazakhstan. The HBC-256 hash function was built using the Merkle-Damgard construction with the wide-pipe modification, which is one of the most common modifications. To ensure one-wayness, the Davies-Meyer scheme was employed. The algorithm HBC-256 consists of 4 rounds, and the length of the resulting

hash value is 256 bits. The compression function utilized in this algorithm is based on a proprietary block cipher called CF, which was designed with consideration of requirements for encryption algorithms in software and hardware implementations [25]. The input and output block lengths of the compression function CF, as well as the round key length, are all 128 bits.

To enhance the performance, the structure of the HBC-256 hashing algorithm is designed to allow the adjustment of the number of input blocks hashed simultaneously in parallel streams. This number is determined by the parameter *k*, which ranges from 3 to 8. The value of parameter k is selected based on the volume of data being hashed and the available computational resources of the computer used. Thus, the parameter *k* defines the length *L* of the input block for the HBC-256 hashing algorithm, where $L=128\times k$ bits. The structure of the hashing algorithm is shown in Figure 1.

The HBC-256 algorithm under consideration, known for its high level of security, enhanced performance, and suitability for hardware implementation using parallel computations, is extensively described in [26]. This paper provides detailed results of conducted research on avalanche effects, strict avalanche effects, and statistical security. Additionally, works [27], [28] present conclusions on the impracticality of using methods such as differential, linear, and algebraic cryptanalysis to find collisions in the HBC-256 function. In our case, when preparing experimental data, we used the value k=3. Thus, the length *L* of the input block of the algorithm is 384 bits.

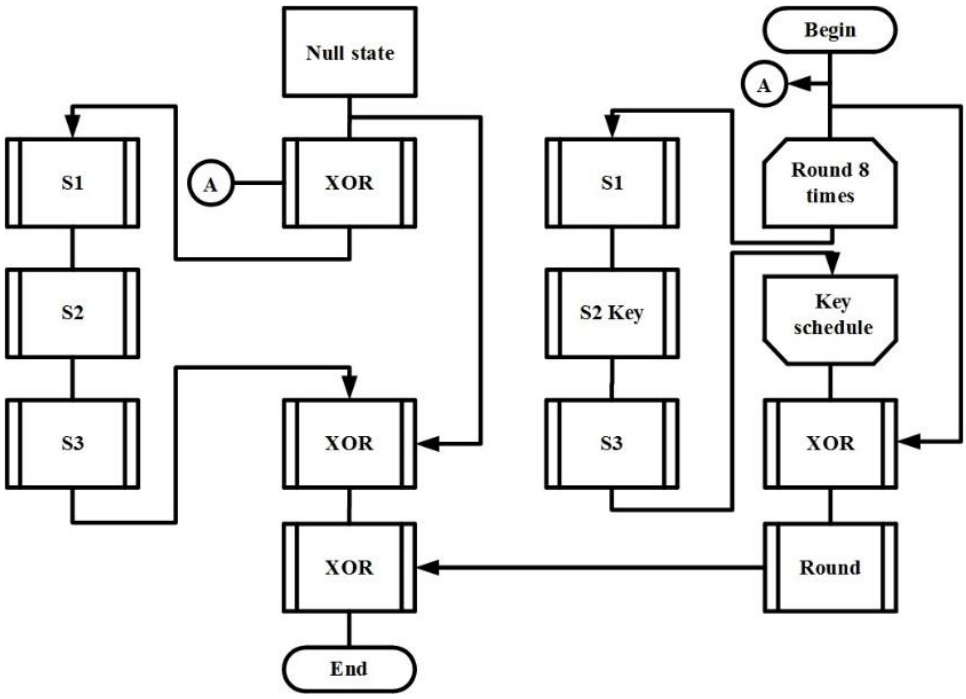

Figure 1. The scheme of the HBC-256 hashing algorithm

3. RESULTS AND DISCUSSION

The Transalg translator takes as input a program that computes a discrete function, written in a specialized procedural programming language (TA language). The result of translating a TA program is a system of Boolean equations that encode the computation process of the considered function. The phases of text analysis of the TA program, constructing a syntax tree, and traversing the resulting tree for interpretation are implemented in a standard manner. A non-trivial aspect of the translation is the procedure for interpreting the language constructs, as this step is responsible for generating the system of Boolean equations that encode the algorithm's execution process. At this stage, numerous local problems arise, the resolution of which can significantly affect both the size and structure of the resulting code.

As mentioned above, the Transalg software suite was used to transform this algorithm into a Boolean formula. Initially, using the specialized programming language TA, which is part of the Transalg software suite, a program code was written to implement all the operations performed by the algorithm. These operations include byte permutation, bitwise cyclic shift, substitution procedure using four 16-byte S-boxes, byte-wise XOR operation, and addition of elements from selected rows and columns of the matrix.

Then, the text of this program was passed to the parsing module of the Transalg software suite, the result of which is a syntax tree describing the internal representation of this program. Traversal of the syntax tree and construction of Boolean equations are performed by the transformation module. When generating the final propositional code, the transition to CNF is carried out using Tseitin transformations [29]. As a result of the propositional encoding of the HBC-256 hashing algorithm, a Boolean formula in CNF was obtained, containing 196,864 Boolean variables and consisting of 986,480 clauses, each of which represents a disjunction of literals. The total number of literals, *i.e.*, Boolean variables or their negations, included in these clauses, is 3,321,880. Table 1 shows the number of Boolean variables, literals, and clauses depending on the number of rounds of the HBC-256 hash function.

Table 1. Number of Boolean variables, literals, and clauses in the Boolean formula describing the HBC-256 hash function

Number of rounds	Number of variables	Number of literals	Number of clauses
1	47,232	791,464	235,024
2	97,024	1,633,912	485,168
3	146,944	2,477,896	735,824
4	196,864	3,321,880	986,480

Once the propositional representation of the HBC-256 hashing algorithm is obtained, depending on the problem being solved, it is necessary to assign values to the input variables, each corresponding to a specific bit of the input block of the hashed message, and the output variables, each corresponding to a specific bit of the hash code. After that, a series of tests are performed using six modern SAT solvers to obtain information about the computation runtime, memory usage, and other experimental data.

Depending on whether input and/or output variable values are specified, SAT solvers can solve the following 4 tasks:

a. If neither input variable values nor output variable values are specified, random input variable values (input message) are generated, and then the corresponding output variable values (hash code) are computed.
b. If input variable values (input message) and output variable values (hash code) are specified, the task of hash code verification is solved.
c. If only input variable values (input message) are specified, the task of computing the values of the corresponding output variables (hash code) is solved.
d. If only output variable values (hash code) are specified, the task of computing the values of the corresponding input variables (input message) is solved.

The first three tasks, from the perspective of SAT solvers, are trivial and can be solved in just a few seconds. However, the fourth task (pre-image search), which is of the greatest interest in terms of cryptanalysis, cannot be solved in a reasonable amount of time. This is natural because any hash function must be resistant to pre-image attacks. One of the commonly used techniques when employing SAT solvers is Guess-and-Determine, which involves fixing the values of certain variables, significantly reducing the overall time required to find a solution to the cryptanalysis task at hand. Therefore, we will simplify our task by specifying, in addition to the output variable values (hash code), the values of certain input variables, and then attempt to find the values of the remaining input variables using a SAT solver.

The computations were performed under the condition that the output variable values, representing a 256-bit hash value, were fully known. Input variable values representing a 384-bit plaintext block were provided, with n initial bits missing, where $1 \leq n \leq 30$. The task is to find the values of the input variables corresponding to the missing bits using SAT solvers and thereby fully recover the original message block. For experiments aimed at testing the resilience of the HBC-256 algorithm against this type of attack, three sequential SAT solvers were selected: CaDiCaL (1.6.0), Lingeling (sc2022), and Kissat (1.0.3), along with three parallel SAT solvers: Plingeling (sc2022), Treengeling (sc2022), and Parkissat (1.0.3) [30], [31]. These solvers have at various times achieved top rankings in the annual SAT competition. Table 2 shows the time taken to find the missing bits of the original message using the aforementioned SAT solvers.

As can be seen from this table, it is preferable to use parallel SAT solvers to solve our problem. Among the parallel SAT solvers (for our problem), the most efficient one was the Parkissat SAT solver, so it will be used in further experiments. Empty cells in the table indicate that the execution time exceeded the established limit. Now, we will conduct a series of experiments to determine the resistance of the HBC-256 hashing algorithm to preimage attacks using one, two, three, and four rounds of the hashing function. Table 3 presents the solution search time using the Parkissat SAT solver, and Figure 2 shows the dynamics of this time depending on the number of unknown variables for four rounds.

Table 2. Time to find the specified number of plaintext bits (in seconds)

Number of unknown bits (n)	SAT Solvers					
		Sequential			Parallel	
	CaDiCaL	Lingeling	Kissat	Parkissat	Plingeling	Treengeling
1	0.37	8.30	0.52	2.51	1.20	8.54
2	0.64	8.56	1.09	4.92	9.10	8.62
3	0.71	8.00	175.30	4.96	1.30	8.81
4	0.75	21.56	859.16	4.81	12.20	69.90
5	1.15	23.18	4409.98	4.8	24.20	72.33
6	1.62	53.18	1499.53	4.74	39.80	92.40
7	2.28	-	1009.55	4.7	16.70	82.01
8	3.19	-	10694.76	4.61	13.50	36.03
9	6.95	-	65077.76	4.99	32.20	39.46
10	10.82	-	1176.17	5.2	12.50	137.15
11	21.03	-	158982.26	6.21	13.50	63.95
12	51.56	-	41866.38	7.15	11.50	89.50
13	37.09	-	21610.30	7.87	15.70	174.97
14	54.38	-	8233.98	15.32	32.40	182.36
15	349.83	-	30135.18	19.49	33.50	108.63
16	53.14	-	3951.08	28.63	22.70	646.78
17	-	-	5936.58	43.51	61.20	-
18	-	-	-	132.54	61.40	-
19	-	-	-	162.57	259.60	-
20	-	-	-	224.31	54.50	-
21	-	-	-	466.26	1309.70	-
22	-	-	-	982.04	1480.70	-
23	-	-	-	2 136.75	448.90	-
24	-	-	-	2 691.54	2043.50	-
25	-	-	-	14 640.47	55822.00	-
26	-	-	-	23 361.06	22479.30	-
27	-	-	-	72 882.68	-	-
28	-	-	-	95 320.19	-	-
29	-	-	-	180 660.11	-	-
30	-	-	-	315 674.76	-	-

Table 3. Solution search time using the Parkissat SAT solver (in seconds)

Number of unknown bits (*n*)	Solution search time			
	1 round	2 rounds	3 rounds	4 rounds
1	2.16	2.18	2.48	2.51
2	1.73	2.82	3.78	4.92
3	1.80	2.84	3.85	4.96
4	1.76	2.72	3.85	4.81
5	1.75	2.81	3.73	4.8
6	1.71	2.71	3.92	4.74
7	1.76	2.74	3.79	4.7
8	1.75	2.69	3.95	4.61
9	1.73	2.80	4.27	4.99
10	1.74	2.70	5.28	5.2
11	1.76	2.88	4.84	6.21
12	1.73	2.65	6.01	7.15
13	1.55	2.93	6.46	7.87
14	1.85	4.18	7.91	15.32
15	1.99	4.58	10.43	19.49
16	2.51	4.92	11.44	28.63
17	4.19	9.25	26.11	43.51
18	6.19	10.55	38.83	132.54
19	11.98	12.85	70.16	162.57
20	36.45	25.18	168.73	224.31
21	42.92	67.79	320.02	466.26
22	47.58	113.76	514.44	982.04
23	77.77	347.71	1,257.46	2 136.75
24	389.85	749.35	1,329.39	2 691.54
25	694.88	1,100.26	3,848.17	14 640.47
26	1,632.66	2,038.35	6,278.91	23 361.06
27	2,489.21	4,269.47	20,488.25	72 882.68
28	4,997.66	9,984.80	76,047.94	95 320.19
29	19,304.95	27,842.22	87,101.22	180 660.11
30	52,062.85	63,398.35	174,276.97	315 674.76

It is necessary to note the following nuance. When running the same SAT solver multiple times with the same input and output data, we obtain different times each time. This is because, at each step of the algorithm for solving the Boolean equation, the selection of the next variable for subsequent assignment of the value “true” or “false” is made using a random number generator. If the variable is “luckily” chosen, meaning the value chosen for it is part of the solution, this positively affects the search time. Otherwise, it will be necessary to return to this variable and assign it the opposite value, and then repeat the solution search procedure (a backtracking mechanism). For example, searching for the values of 25 unknown variables using the Parkissat SAT solver can take from 49 minutes to 15 hours, where they are shown in Table 4.

To obtain a realistic assessment of time, it is necessary to repeat the same experiment multiple times, for example, 10 times, and then find the average time spent. In our case, the average time to find a solution was 23,361.06 seconds, or 6 hours, 29 minutes, and 21.06 seconds. As evident from Table 3 and the graph in Figure 2, with an increase in the number of unknown bits (starting from 25 and above), the time required to solve the problem sharply increases. It took over 87 hours using a computer with two 56-core processors to find the preimage of the message, assuming that 30 unknown bits were present while the values of the remaining 354 bits were fixed. The experiments were conducted on a computer with two 56-core AMD EPYC 7663 processors running at 2.0 GHz each (total of 112 cores) and 128 GB of RAM, operating on the Linux Ubuntu 22.04.3 LTS operating system.

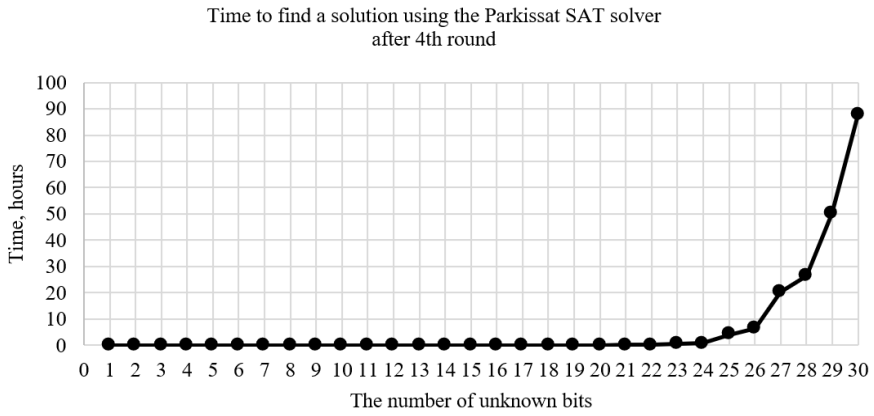

Figure 2. Dynamics of solution search time using the Parkissat SAT solver

Table 4. Search time for values of 25 unknown Boolean variables

Attempt number	Search time	
	(h, m, s)	(seconds)
Attempt #1	2 h 52 m 21.42 s	10,341.42
Attempt #2	5 h 22 m 00.07 s	19,320.07
Attempt #3	2 h 35 m 51.17 s	9,351.17
Attempt #4	7 h 39 m 49.85 s	27,589.85
Attempt #5	49 m 30.07 s	2,970.07
Attempt #6	1 h 56 m 12.93 s	6,972.93
Attempt #7	15 h 08 m 07.98 s	54,487.98
Attempt #8	13 h 52 m 30.57 s	49,950.57
Attempt #9	3 h 25 m 15.64 s	12,315.64
Attempt #10	11 h 11 m 50.55 s	40,310.55

4. CONCLUSION

The study undertook a practical cryptanalysis of the cryptographic hash function HBC-256. The 4-round compression function was described in the form of an equation in CNF=1 format using 986,480 clauses, each of which represents a conjunction of literals and 196,864 Boolean variables. The solution to this equation was obtained using the SAT solver Parkissat.

The results of the experimental studies show that for the full HBC-256 hash function consisting of 4 rounds, the SAT solver struggles to solve the problem even when only 30 bits of the original message are unknown. Since finding the preimage requires determining all 384 bits of the original message block, this task is practically infeasible within a reasonable time frame. Therefore, it can be concluded that the HBC-256 algorithm is resistant to preimage attacks using this method.

A limitation of SAT solvers is their brute-force nature, and the problem of reducing the search space is fundamental to these methods. A solution might not be found within an acceptable time frame, making it practically impossible to provide a theoretical estimate of the possibility of finding a preimage. Therefore, as a future research direction, it should be noted that other types of cryptanalyses and their combinations, including the combination of differential and algebraic cryptanalysis, will be explored to obtain a theoretical estimate of the strength of the considered hash function.

FUNDING INFORMATION

The research work was funded by the Ministry of Science and Higher Education of Kazakhstan and carried out within the framework of the project BR24993052 "Development and study of cryptographic algorithms for information protection in resource-constrained systems and evaluation of their strength" at the Institute of Information and Computational Technologies.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Kunbolat Algazy	✓	✓				✓		✓			✓		✓	
Kairat Sakan		✓				✓		✓	✓		✓			
Andrey Varennikov		✓	✓			✓	✓		✓	✓				
Nursulu Kapalova				✓	✓				✓			✓		✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

- The data that support the findings of this study are available on request from the corresponding author, KS.
- Derived data supporting the findings of this study are available from the corresponding author, KS, on request.
- The data that support the findings of this study are available from the corresponding author, KS, upon reasonable request.

REFERENCES

[1] F. Massacci and L. Marraro, "Logical cryptanalysis as a SAT problem: encoding and analysis of the U.S. data encryption standard," *Journal of Automated Reasoning*, vol. 24, no. 1–2, pp. 165–203, 2000, doi: 10.1023/a:1006326723002.
[2] K. Hu and Z. Chu, "An efficient circuit-based SAT solver and its application in logic equivalence checking," *Microelectronics Journal* , vol. 142, p. 106005, Dec. 2023, doi: 10.1016/j.mejo.2023.106005.
[3] K. Buño and H. Adorna, "Solving 3-SAT in distributed P systems with string objects," *Theoretical Computer Science*, vol. 964, p. 113976, Jul. 2023, doi: 10.1016/j.tcs.2023.113976.
[4] C. M. Li, F. Xiao, M. Luo, F. Manyà, Z. Lü, and Y. Li, "Clause vivification by unit propagation in CDCL SAT solvers," *Artificial Intelligence*, vol. 279, p. 103197, Feb. 2020, doi: 10.1016/j.artint.2019.103197.
[5] L. Cheng and L. Feng, "Model abstraction for discrete-event systems using a SAT solver," *IEEE Access*, vol. 11, pp. 17334–17347, 2023, doi: 10.1109/ACCESS.2023.3246123.
[6] A. E. J. Hyvärinen and N. Manthey, "Designing scalable parallel SAT solvers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7317 LNCS, Springer Berlin Heidelberg, 2012, pp. 214–227, doi: 10.1007/978-3-642-31612-8_17.
[7] N. Alamgir, S. Nejati, and C. Bright, "SHA-256 Collision Attack with Programmatic SAT," *CEUR Workshop Proceedings*, vol. 3717, pp. 91–110, 2024.
[8] S. Nejati and V. Ganesh, "CDCL(Crypto) SAT solvers for cryptanalysis," *CASCON 2019 Proceedings - Conference of the Centre for Advanced Studies on Collaborative Research - Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, pp. 311–316, 2020.

[9] A. Biere, "Lingeling, Plingeling and Treengeling entering the SAT competition 2013," in *Proceedings of SAT Competition 2013*, 2013, pp. 51–52.
[10] I. Otpuschennikov, A. Semenov, I. Gribanova, O. Zaikin, and S. Kochemazov, "Encoding cryptographic functions to SAT using TRANSALG system," *Frontiers in Artificial Intelligence and Applications*, vol. 285, pp. 1594–1595, 2016, doi: 10.3233/978-1-61499-672-9-1594.
[11] S. Cai, X. Zhang, M. Fleury, and A. Biere, "Better decision heuristics in CDCL through local search and target phases," *Journal of Artificial Intelligence Research*, vol. 74, pp. 1515–1563, Aug. 2022, doi: 10.1613/JAIR.1.13666.
[12] C. Bright, I. Kotsireas, and V. Ganesh, "Applying computer algebra systems with SAT solvers to the Williamson conjecture," *Journal of Symbolic Computation*, vol. 100, pp. 187–209, Sep. 2020, doi: 10.1016/j.jsc.2019.07.024.
[13] D. Loveland, A. Sabharwal, and B. Selman, "DPLL: The core of modern satisfiability solvers," in *Outstanding Contributions to Logic*, vol. 10, Springer International Publishing, 2016, pp. 315–335, doi: 10.1007/978-3-319-41842-1_12.
[14] E. Maro, "Modeling of algebraic analysis of PRESENT cipher by SAT solvers," *IOP Conference Series: Materials Science and Engineering*, vol. 734, no. 1, p. 12101, Jan. 2020, doi: 10.1088/1757-899X/734/1/012101.
[15] E. Maro, "Algebraic analysis of SM4 cipher using SageMath," *IOP Conference Series: Materials Science and Engineering*, vol. 1047, no. 1, p. 12086, Feb. 2021, doi: 10.1088/1757-899X/1047/1/012086.
[16] C. Ansótegui, M. L. Bonet, J. Giráldez-Cru, J. Levy, and L. Simon, "Community structure in industrial SAT instances," *Journal of Artificial Intelligence Research*, vol. 66, pp. 443–472, Oct. 2019, doi: 10.1613/jair.1.11741.
[17] P. M. Bittner, T. Thüm, and I. Schaefer, "SAT encodings of the at-most-k constraint: a case study on configuring university courses," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11724 LNCS, Springer International Publishing, 2019, pp. 127–144, doi: 10.1007/978-3-030-30446-1_7.
[18] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for SAT solvers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9710, Springer International Publishing, 2016, pp. 123–140, doi: 10.1007/978-3-319-40970-2_9.
[19] R. G. Biyashev, A. Smolarz, K. T. Algazy, and A. Khompysh, "Encryption algorithm 'QAMAL NPNS' based on a nonpositional polynomial notation," *Journal of Mathematics, Mechanics and Computer Science*, vol. 105, no. 1, Apr. 2020, doi: 10.26577/jmmcs.2020.v105.i1.17.
[20] M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and conquer: Guiding CDCL SAT solvers by lookaheads," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7261 LNCS, Springer Berlin Heidelberg, 2012, pp. 50–65, doi: 10.1007/978-3-642-34188-5_8.
[21] S. E. Kochemazov and O. S. Zaikin, "Towards better SAT encodings for hash function inversion problems," in *2024 47th ICT and Electronics Convention, MIPRO 2024 - Proceedings*, May 2024, pp. 25–30, doi: 10.1109/MIPRO60963.2024.10569193.
[22] R. G. Biyashev, S. E. Nyssanbayeva, and Y. Y. Begimbayeva, "Development of the model of protected cross-border information interaction," *Open Engineering*, vol. 6, no. 1, pp. 199–205, Aug. 2016, doi: 10.1515/eng-2016-0025.
[23] S. Alouneh, S. Abed, M. H. Al Shayeji, and R. Mesleh, "A comprehensive study and analysis on SAT-solvers: advances, usages and achievements," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2575–2601, Mar. 2019, doi: 10.1007/s10462-018-9628-0.
[24] S. Abed, A. Ashkanan, W. Mansoor, and A. Gawanmeh, "Enhanced SAT solvers based hashing method for bitcoin mining," in *Advances in Intelligent Systems and Computing*, vol. 1134, Springer International Publishing, 2020, pp. 191–198, doi: 10.1007/978-3-030-43020-7_26.
[25] N. A. Kapalova, K. S. Sakan, A. Haumen, and Suleimenov, "Requirements for symmetric block encryption algorithms developed for software and hardware implementation," *KazNU Bulletin. Mathematics, Mechanics, Computer Science Series*, vol. 112, no. 4, pp. 134–147, Dec. 2021, doi: 10.26577/JMMCS.2021.v112.i4.12.
[26] K. Sakan, S. Nyssanbayeva, N. Kapalova, K. Algazy, A. Khompysh, 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.
[27] K. Algazy, K. Sakan, N. Kapalova, S. Nyssanbayeva, and D. Dyusenbayev, "Differential analysis of a cryptographic hashing algorithm HBC-256," *Applied Sciences (Switzerland)*, vol. 12, no. 19, p. 10173, Oct. 2022, doi: 10.3390/app121910173.
[28] K. Algazy, K. Sakan, and N. Kapalova, "Evaluation of the strength and performance of a new hashing algorithm based on a block cipher," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 3, pp. 3124–3130, Jun. 2023, doi: 10.11591/ijece.v13i3.pp3124-3130.
[29] E. Kuiter, S. Krieter, C. Sundermann, T. Thüm, and G. Saake, "Tseitin or not Tseitin? The impact of CNF transformations on feature-model analyses," in *ACM International Conference Proceeding Series*, Oct. 2022, pp. 1–13, doi: 10.1145/3551349.3556938.
[30] D. Michaelson, D. Schreiber, M. J. H. Heule, B. Kiesl-Reiter, and M. W. Whalen, "Unsatisfiability proofs for distributed clause-sharing SAT solvers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 13993, Springer Nature Switzerland, 2023, pp. 348–366, doi: 10.1007/978-3-031-30823-9_18.
[31] L. Le Frioux, S. Baarir, J. Sopena, and F. Kordon, "PaInleSS: a framework for parallel SAT solving," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10491 LNCS, Springer International Publishing, 2017, pp. 233–250, doi: 10.1007/978-3-319-66263-3_15.

BIOGRAPHIES OF AUTHORS

Kunbolat Algazy received a master degree in mathematics from Al-Farabi Kazakh National University in 2001 and a Ph.D. degree in information security systems, Almaty, Kazakhstan, in 2021. In 2001-2014 he worked in 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 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 email: kunbolat@mail.ru.

Kairat Sakan Ph.D., graduated from the Faculty of Mechanics and Mathematics of the Al-Farabi Kazakh National University, majoring in "mathematics and applied mathematics" (KazNU, Almaty, Kazakhstan) in 2001. In 2001-2002, he worked as a teacher at the Department of Applied Mathematics and Mathematical Modeling at KazNU. Between 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 system's security in the state structure for several years. Since 2018, he has been working as a researcher in the Information Security Laboratory at the Scientific Institute of Information and Computing Technologies. The field of scientific research is information security system in the public and private sector. He can be contacted at email: 19kairat78@gmail.com.

Andrey Varennikov received his master's degree in mathematics and computer science from Kazakh National University in 1981 and more than 40 years specializing in the development of cryptographic protection software, high-capacity multiuser distributed databases, workflow applications, artificial intelligence, system programming. He currently works in the Institute of Information and Computational Technologies of the Ministry of Science and Higher Education of the Republic of Kazakhstan. He can be contacted at email: avarennikov@ipic.kz.

Nursulu Kapalova received her master's degree in mathematics from Al-Farabi Kazakh National University in 2002 and her degree candidate of technical sciences (Almaty, Kazakhstan) in 2009. Currently she is a leading researcher in the laboratory "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. Area of scientific work: development and research in the field of information protection. She can be contacted at email: nkapalova@mail.ru.