



Article

Differential Analysis of a Cryptographic Hashing Algorithm HBC-256

Kunbolat Algazy ¹, Kairat Sakan ^{1,2,*}, Nursulu Kapalova ¹, Saule Nyssanbayeva ¹
and Dilmukhanbet Dyusenbayev ¹

¹ Institute of Information and Computational Technologies, Almaty 050010, Kazakhstan

² Faculty of Information Technology, Al-Farabi Kazakh National University, Almaty 050010, Kazakhstan

* Correspondence: 19kairat78@gmail.com

Abstract: The article observes the new hashing algorithm HBC-256. The HBC-256 algorithm is based on the block cipher of the compression function CF (Compression Function) and produces a 256-bits hash value. Like any new cryptographic structure, the HBC-256 algorithm requires careful research process in order to confirm its cryptographic properties, namely: pre-image resistance and resistance to collisions of the first and second order. As a result of the research, for the HBC-256 hashing algorithm differential properties of nonlinear elements (S-boxes) and various options for constructing round characteristics are considered. A hypothesis has been advanced about the existence of paired differences, which will make it possible to construct round characteristics for hashing and for the function of round keys generating. It is shown that even for the most optimal way of constructing chains of differences, the probability of finding correct pairs of texts is less than the probability of a complete enumeration of one 128-bit block of input data, which makes the method of differential cryptanalysis unsuitable for finding collisions.

Keywords: hash function; cryptography; cryptanalysis; collision; algorithm; differential cryptanalysis



Citation: Algazy, K.; Sakan, K.; Kapalova, N.; Nyssanbayeva, S.; Dyusenbayev, D. Differential Analysis of a Cryptographic Hashing Algorithm HBC-256. *Appl. Sci.* **2022**, *12*, 10173. <https://doi.org/10.3390/app121910173>

Academic Editor: Arcangelo Castiglione

Received: 7 September 2022

Accepted: 6 October 2022

Published: 10 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Currently, the collection of information and communication technologies (ICTs) and the social system that supports them are often perceived as an ecosystem, or more precisely, a cyber-ecosystem. The components of the cyber ecosystem are constantly maintained and form a dense network of dependencies and information processing-storage, exchange, authentication and assured destruction. One of the biggest challenges facing the cryptography community around the world in this ecosystem is to ensure the confidentiality, authentication, and integrity of the messages we want to send over an insecure data transmission channel such as the Internet. Online banking transactions would not be secure enough and password-protected systems would not exist without the use of hash functions. Cell phone systems would lose their advantages without this additional locking feature, which is the application of hash functions (SHA-1). The concept of a hash function was introduced in 1976, with the invention of public key cryptography by Diffie and Hellman [1].

In the modern information world, one of the key values is to ensure the reliability and security of information. Many information systems, including low-resource 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 are hash functions, one-way mathematical transformations that convert an arbitrary input data array into a unique sequence of fixed length. Hash functions are very widespread. Hash functions are used to store passwords during authentication, protect data in file verification systems, encode information in the blockchain, etc. There are many cryptographic algorithms these days. They are different and differ in complexity, bit depth, cryptographic reliability, and features of work. Hashing algorithms appeared

more than half a century ago. Moreover for many years, from a fundamental point of view, little has changed. However, as a result of their development, data has acquired many new properties, so their application in the field of information technology has already become ubiquitous [2].

The cryptographic hash function must meet the requirements of the “one-way” principle and collision resistance. One way means that the method of calculating the hash value from a given message is simple, but computationally impossible to generate any message that gives an initial hash value. Collision robustness means that it is extremely difficult to find two messages with the same hash function value [3,4].

In blockchain technology, the hash is also used to verify the integrity of data. The hash acts as a guarantee of the integrity of the chain of transactions (payments). Each new transaction block references the hash of the previous block in the registry. The hash of the block itself depends on all the transactions in the block, but instead of sequentially passing transactions to the hash function, they assembled into a single hash value using a binary hash tree. Thus, hashes are used as a replacement for pointers in common data structures: linked lists and binary trees. By using hashes, the overall state of the blockchain, and all transactions ever performed, and their sequence, can be expressed by a single number: a hash of the newest block. Therefore, the hash immutability property of a single block guarantees the immutability of the entire blockchain [5].

Algorithms for calculating hash values like other types of cryptographic algorithms are a fundamental element of many information security systems and protocols. Given the importance of the strength of the applied algorithms and trust in them, many countries (commonwealths of countries) have developed or are developing national cryptographic standards. A widely used algorithm for calculating hash values is the United States standard called Keccak (SHA-3) [6]. Let us consider the approaches applied to the analysis of the strength of the Keccak hash calculation algorithm.

Error injection attacks [7,8] consist in generating a message containing an error using the Keccak algorithm to extract secure hash data. Paper [8] presents a new error detection scheme based on a modification of the Keccak architecture, where the Keccak round was subdivided into two blocks. The attack simulation results showed that the proposed attack scheme achieves 99.995% of the expected error generation. Also, the proposed attack scheme was evaluated from the point of view of hardware implementation on the FPGA. In the paper [8], an efficient error detection scheme based on the architecture of the Keccak algorithm was proposed. The round of the Keccak algorithm is divided into two sub-rounds and a pipeline register is implemented between them. The proposed scheme does not depend on the Keccak implementation method, so the method can be applied to both pipelined and iterative architectures.

The article [9] describes the use of power analysis attacks that use the correlation power analysis (CPA) technique to extract the MAC-Keccak secret key. The proposed attack methodology uses simulated power signal traces and is applied to the design of the high-speed Keccak core given in the SHA-3 competition. A 1-bit, 2-bit, 4-bit, 8-bit, and 16-bit CPA selection function with key size guessing was investigated to analyze the computational complexity of successful MAC-Keccak key extraction. The experiments performed confirmed that the larger the size of the proposed key of the selection function, the fewer traces are required to extract the key, but with a subsequent increase in computation time.

The work [10] is devoted to the study of differential error analysis of the family of SHA-3 algorithms, namely the SHA3-224 and SHA3-256 algorithms. The authors have developed an error injection model and increased the realism of the attack for various SHA-3 implementation architectures. The propagation of errors in the SHA-3 algorithm within the framework of one-byte error models was analyzed and the use of the error signature on the output value to extract secret information was proposed. The results of the study showed that the proposed method can effectively detect introduced single-byte errors and then restore all internal states of the last round operation for both SHA3-224 and SHA3-256. Later authors extended the application of the developed approach to an attack based on

error injection and in the article [11] conducted a study of algebraic analysis of errors, which made it possible to detect injected failures with a much greater speed and restore the full internal state of the penultimate round with a smaller number of injected errors.

The paper [12] explored practical attacks based on collision detection for the Keccak algorithm with a small number of rounds. The following main results were obtained in the work: a collision search attack against a 5-round Keccak-224 was described and a description of the collision search problem for a 6-round Keccak was presented. The attacks were implemented by carefully studying the algebraic properties of the nonlinear layer in the basic Keccak permutation and applying the linearization method. The authors proposed using methods of partial linearization of the output bits of the nonlinear layer, which significantly reduced the computational complexity of the attack. Cubic attacks on the Keccak algorithm and its variants (Keccak-MAC, KMAC, Keyak, and Ketje) were presented in [13]. In paper [14], when attacking Keccak-384/512 with a reduced number of rounds, the authors found that the revealed linear dependencies are not fully used. In order to maximize the use of all 448 and 320 linear dependencies identified for the Keccak-512 and Keccak-384 algorithms, respectively, the authors proposed a special algebraic attack based on the expression of output values in the form of a system of quadratic logical equations. The resulting system of quadratic Boolean equations can be effectively solved using linearization methods. As a result, the authors managed to improve attacks on 2–4 rounds of Keccak-384 and 2–3 rounds of Keccak-512.

Paper [15] includes an extensive review of articles published in 2018 on image encryption schemes and their cryptanalysis using various research methods. The authors of this work classified, on a scientific basis, the methods of cryptographic protection of images into different categories. Also, an important point of this work is the identification of critical problems in the design and evaluation of the security of image encryption schemes. The methods of data protection indicated in the work are also applicable for assessing the security of hashing schemes.

2. Related Works

The HBC-256 (Hash based on Block Cipher) hashing algorithm is a new algorithm for computing a hash value based on the Merkle–Damgård structure. A description of the HBC-256 algorithm and some approaches to its analysis were presented in [16]. Research [16] presents analysis of the avalanche effect of CF encryption algorithm and statistical analysis of the HBC-256. As results CF encryption algorithm (HBC-256 hashing algorithm itself) is efficient to provide a good avalanche effect and binary sequence generated by the HBC-256 algorithm is close to random (using the NIST and statistical test suite).

We remark, however, that in the classical setting the basic approach to implementing collision attacks that can exploit vulnerabilities in a particular hashing algorithm is differential cryptanalysis [17]. A general scheme for applying the differential analysis method to a hash algorithm based on a Merkle–Damgård structure is presented in the next three papers.

The authors of the article [18] in their work describe in more detail the design models of a hash function and discuss various types of attacks and possible ways to overcome them. According to experts, the most successful attack on the Merkle–Damgård hash function is the differential attack. It is used to create a successful collision attack in hash functions.

The Merkle–Damgård construction proves that the security of a hash function depends on the security of the compression function. Hash functions designed without taking into account the weakness of the Merkle–Damgård construction are vulnerable to various attacks. Thus, to build a collision-resistant hash function, it suffices to develop a collision-resistant compression function. Therefore, the use of a block cipher as a compression function imposes higher requirements on its security level.

The problem of building and studying S-boxes with given properties is well known. In [19], the authors present a new framework for efficient analysis of a nonlinear node, the S-box, and analyze the differential probability when the difference is expressed using

modulo 2 addition (xor). This probability was obtained using graph theory and calculated using matrix multiplications.

The aim of the research is to identify strong and weak properties for the new HBC-256 hash function to differential analysis.

3. Materials and Methods of Research

3.1. HBC-256 Hash Function

More details about the HBC-256 hashing algorithm described in [15]. The general scheme of message hashing $M(M_0, M_1, \dots, M_{t-1})$ at $k = 3$ is shown in Figure 1, where k is the number of parts, $M_r(m^0, m^1, \dots, m^{k-1})$ is a message block M consisting of $128 \times k$ bits, $r = 0, 1, \dots, t-1$. The proposed algorithm HBC-256 takes M_r from the message M as a master key rk_0^j , and the encrypted text—the previous intermediate hash value h_{i-1}^j , $j = 0, \dots, k-1$. At the very beginning of the process of hashing the message $M(M_0, M_1, \dots, M_{t-1})$ the initial hash value is taken as $h_0^j = 0^{128}$, i.e., null value. Based on message blocks M_r , round keys rk_i^j , $i = 1, 2, \dots, R_1$, are generated, where R is the number of rounds for hashing message block M_r .

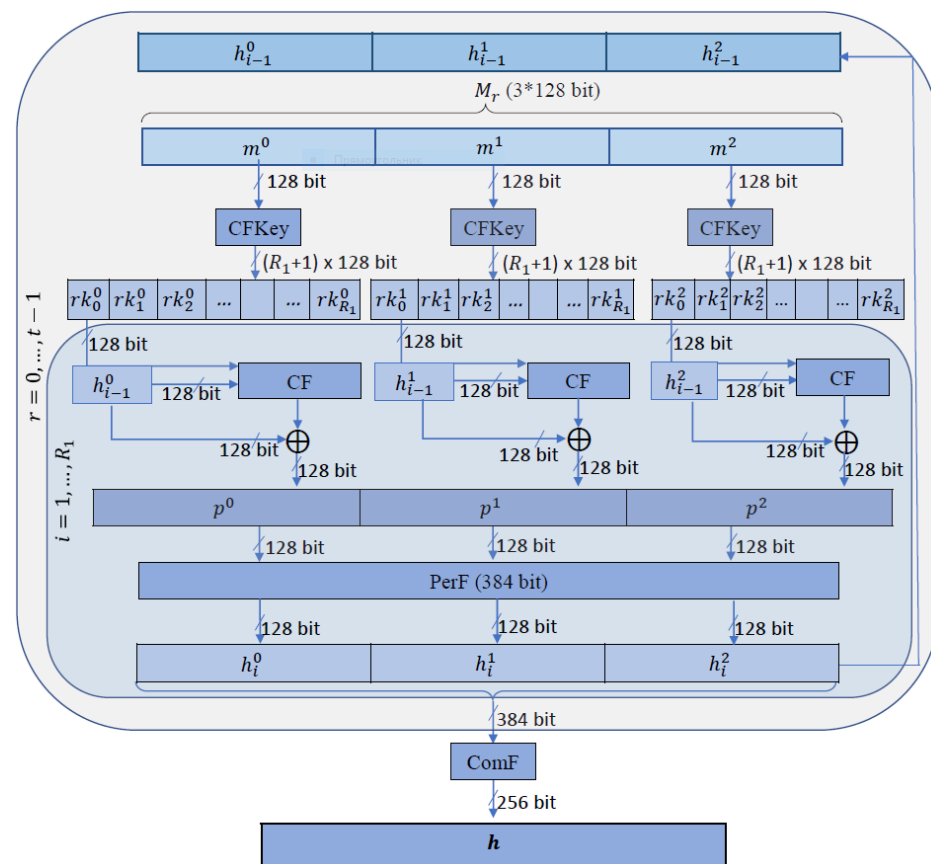


Figure 1. The general hashing scheme of the HBC-256 algorithm.

The HBC-256 data hashing algorithm is based on the proposed CF block cipher (Figure 2). CF has two inputs—a 128-bit h_{i-1}^j and a 128-bit round key rk_i^j and one output—an intermediate 128-bit hash value h_i^j . The nonlinear bijective transformation S is determined using the SBOX procedure. Four “golden” S -blocks were selected according to Table 1.

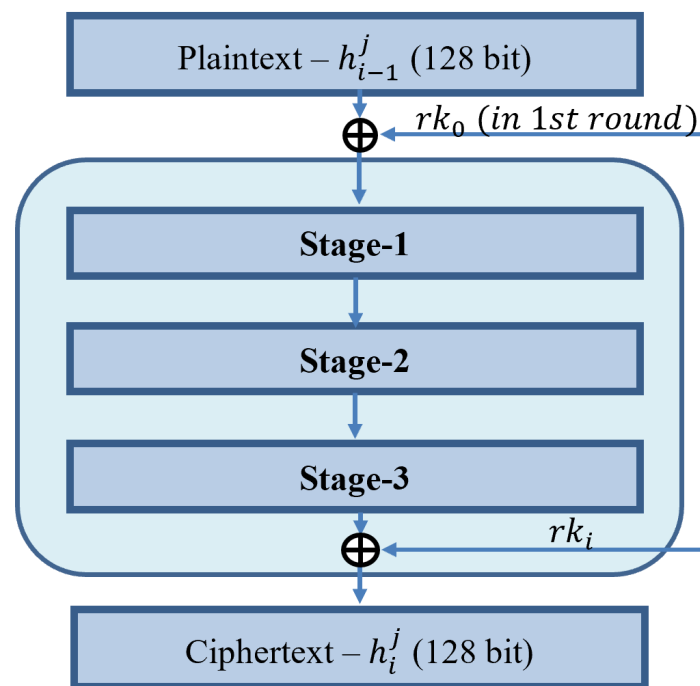


Figure 2. Scheme for the CF algorithm.

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

Based on the Wide-pipe scheme, in a single hash loop, the CF algorithm is simultaneously executed k times for different m^j , $j = 0, \dots, k - 1$. The length of the intermediate hash value is $128k$ bits. The Davis-Meyer CF is computed by p^j as the result of summing h_{i-1}^j and h_i^j modulo 2. The PerF procedure then rearranges the values of all three p^j , which are further taken as h_{i-1}^j . The final hash value is determined through ComF procedures.

3.2. Method of Differential Cryptanalysis

The differential cryptanalysis method was first proposed by E. Biham and A. Shamir to analyze the DES encryption standard [20,21].

Differential cryptanalysis is based on the analysis of differences between encrypted values at various stages of encryption. As a difference, the operation of bitwise addition modulo 2 is used, including the analysis of the difference modulo 2^n . Differential analysis is a method of cryptanalysis of symmetric block ciphers and other cryptographic primitives, in particular, hash functions and stream ciphers [22]. To apply this type of analysis, it is necessary to build tables of differential properties for all non-linear elements of the algorithm—S substitution boxes, addition modulo 2^n , and others. A detailed algorithm for differential analysis of nonlinear elements can be found in [23,24].

In the general case the application of the method of differential cryptanalysis is reduced to the following steps:

1. Analysis of non-linear elements and determination of the most probable differences for them;
2. Construction of a multi-round characteristic (type input difference—output difference) from simple to complex, that is, from one round to n rounds. Determining the

probability of the appearance of the constructed characteristic. Search for correct pairs of texts, i.e., such texts for which the sum of the input values is the same as the value of the input difference, and the sum of the output values is the same as the output difference.

The task of differential cryptanalysis is reduced to constructing such pairs of input–output difference, the probability of which is less than the exhaustive search of all possible values. To construct pairs of texts, the analysis of non-linear elements of a cryptographic primitive (for example, S-boxes) is carried out. After that, the transformations from round to round are compared and the possibility of transition from one state to another is determined.

A pair of texts that corresponds to a given difference is called a correct pair of texts. In the case of symmetric encryption, the correct pair of texts allows to determine the secret encryption key or its fragments. In the case of hash function analysis, the correct pair of texts determines the collision.

The purpose of this research is to observe the differential properties of the new HBC hashing algorithm. To apply differential cryptanalysis, it is necessary to check whether there is a way to transform differences from one operation to another and what is the probability of such transformation. The probability will determine the complexity of the analysis and the value of the difference will set the initial search parameters.

4. Results and Discussion

The first stage of the analysis is the analysis of nonlinear elements and the construction of a table with the results of differential properties. For the HBC-256 algorithm, such nonlinear elements are the substitution S-boxes presented in Table 2. As a result of the analysis of the differential properties of these S-boxes according to [23,25], Tables 2–5 were constructed. From these tables, it can be seen that the maximum possible probability of non-zero differences is 1/4.

Table 2. S_0 box analysis.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	4	2	0	0	0	0	2	2	2	0	2
2	0	0	0	0	0	2	0	2	0	2	4	2	0	0	0	4
3	0	0	2	2	4	0	0	0	2	2	0	0	0	0	0	4
4	0	0	0	0	0	0	4	4	0	0	2	2	2	2	0	0
5	0	2	0	2	0	0	0	0	2	2	2	2	2	0	2	0
6	0	2	0	2	0	0	2	2	4	0	0	0	2	0	0	2
7	0	0	2	4	4	2	0	0	0	2	0	0	0	0	2	0
8	0	0	0	2	0	0	2	0	0	2	0	0	2	4	4	0
9	0	2	2	2	0	0	2	0	2	0	2	2	0	0	0	2
A	0	0	2	0	2	0	2	2	0	4	0	2	2	0	0	0
B	0	2	2	0	2	2	0	0	0	2	2	0	2	2	0	0
C	0	2	0	0	2	0	2	2	4	0	2	0	0	0	2	0
D	0	2	2	0	2	4	0	2	0	0	0	0	0	4	0	0
E	0	4	2	0	0	2	0	0	0	0	0	2	0	2	2	2
F	0	0	2	0	0	0	0	2	2	0	2	2	2	0	4	0

Table 3. S_1 box analysis.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	0	2	0	2	2	4	2
2	0	0	0	2	0	2	0	0	0	0	2	4	2	4	0	0
3	0	2	2	2	2	0	2	2	2	0	0	0	0	2	0	0
4	0	0	0	2	0	4	2	0	0	0	0	2	0	0	2	4
5	0	0	2	2	2	2	0	0	0	0	0	4	4	0	0	0
6	0	0	0	2	4	0	2	0	2	2	0	2	0	0	0	2
7	0	2	0	0	0	0	2	4	4	2	0	0	0	0	2	0
8	0	0	0	0	0	0	2	2	0	4	4	0	2	2	0	0
9	0	2	0	2	2	2	2	2	0	2	0	2	0	0	0	0
A	0	2	0	0	4	0	2	0	0	2	0	0	2	2	0	2
B	0	2	2	0	0	0	0	0	2	0	4	2	0	0	4	0
C	0	0	4	0	0	2	0	2	2	2	0	0	2	0	0	2
D	0	2	2	0	0	2	2	0	2	0	2	0	2	0	2	0
E	0	2	4	2	0	0	0	0	0	2	2	0	0	0	2	2
F	0	2	0	0	2	2	0	2	2	0	0	0	0	4	0	2

Table 4. S_2 box analysis.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	2	2	2	2	4	0	0
2	0	0	0	0	0	2	4	2	0	2	0	2	0	0	4	0
3	0	0	4	0	2	0	0	2	0	0	0	4	0	2	2	0
4	0	0	0	2	0	2	2	2	0	0	0	2	0	2	2	2
5	0	2	2	2	0	0	2	0	0	0	2	0	2	0	4	0
6	0	2	2	2	0	2	0	0	4	2	0	0	2	0	0	0
7	0	0	0	0	2	2	0	0	4	2	0	2	2	0	0	2
8	0	0	0	2	0	0	2	0	0	4	2	0	4	0	0	2
9	0	2	0	0	2	4	2	2	0	0	2	0	0	0	2	0
A	0	2	2	0	0	2	0	2	2	0	2	0	2	0	0	2
B	0	4	2	0	0	0	0	2	2	0	0	4	0	2	0	0
C	0	0	0	2	4	0	0	2	2	2	2	0	0	0	0	2
D	0	0	2	2	2	2	0	0	2	0	2	0	0	2	0	2
E	0	4	0	0	0	0	0	0	0	2	2	0	0	2	2	4
F	0	0	2	2	4	0	4	0	0	0	0	0	2	2	0	0

The analysis of S-boxes is as follows. All possible combinations of two inputs are considered to form the input difference. Input difference values are specified horizontally. For each combination of input difference, the value of the output difference is determined. These values are presented vertically from 0 to F. At the intersection of rows and columns, the number of occurrences of each output difference for each input difference is indicated. To determine the probability, it is necessary to divide the values of Tables 2–5 by 16, since the S-box takes 4 bits as input, which means it has 16 possible combinations. Thus, for an input difference of 0, the output difference will always be 0 and there will never be any other output differences. This follows from the fact that at difference 0 the texts forming the difference are identical. This means that the output is identical texts, for which add modulo 2 is 0 (at the intersection of input difference 0 and output difference 0, the number of repetitions is 16, which corresponds to a probability of $16/16 = 1$).

After a careful study of the data conversion scheme, it was noticed that, according to Figure 2, the S-boxes from Table 1 actually form 16 S-boxes of 8×8 bits. Depending on the byte to which they are applied, combinations of small S-box from Table 1 are used to form a large S-box. Thus, 16 S-blocks were formed and analyzed for each of the state bytes.

Table 5. S_3 box analysis.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	2	0	2	2	0	2	0
2	0	0	0	2	0	2	0	0	0	0	0	2	2	4	2	2
3	0	0	4	0	2	0	0	2	2	0	0	2	0	0	4	0
4	0	0	0	0	0	4	4	0	0	2	2	0	2	0	0	2
5	0	2	2	0	0	2	2	0	0	0	2	2	2	2	0	0
6	0	0	2	2	2	0	2	0	2	4	0	2	0	0	0	0
7	0	2	0	0	0	0	0	2	4	0	0	2	0	2	4	0
8	0	0	0	0	0	0	2	2	0	0	2	2	2	2	2	2
9	0	2	0	0	2	2	0	2	2	2	2	0	2	0	0	0
A	0	0	4	2	0	2	2	2	0	0	2	0	0	2	0	0
B	0	2	0	0	4	2	0	0	0	4	2	0	0	0	2	0
C	0	4	0	2	2	0	0	0	0	2	2	2	0	0	0	2
D	0	0	2	2	0	0	0	0	2	0	2	0	2	4	0	2
E	0	0	2	0	4	0	2	0	2	0	0	0	2	0	0	4
F	0	4	0	2	0	2	2	2	2	0	0	0	0	0	0	2

So, for example, for byte a00 the S-box S_{00} , formed from the double application of the S_0 box (Table 1), will be applied. As a result, S-box S_{00} will take the following form:

$S_{00} = [0, 240, 176, 128, 192, 144, 96, 48, 208, 16, 32, 64, 160, 112, 80, 224, 15, 255, 191, 143, 207, 159, 111, 63, 223, 31, 47, 79, 175, 127, 95, 239, 11, 251, 187, 139, 203, 155, 107, 59, 219, 27, 43, 75, 171, 123, 91, 235, 8, 248, 184, 136, 200, 152, 104, 56, 216, 24, 40, 72, 168, 120, 88, 232, 12, 252, 188, 140, 204, 156, 108, 60, 220, 28, 44, 76, 172, 124, 92, 236, 9, 249, 185, 137, 201, 153, 105, 57, 217, 25, 41, 73, 169, 121, 89, 233, 6, 246, 182, 134, 198, 150, 102, 54, 214, 22, 38, 70, 166, 118, 86, 230, 3, 243, 179, 131, 195, 147, 99, 51, 211, 19, 35, 67, 163, 115, 83, 227, 13, 253, 189, 141, 205, 157, 109, 61, 221, 29, 45, 77, 173, 125, 93, 237, 1, 241, 177, 129, 193, 145, 97, 49, 209, 17, 33, 65, 161, 113, 81, 225, 2, 242, 178, 130, 194, 146, 98, 50, 210, 18, 34, 66, 162, 114, 82, 226, 4, 244, 180, 132, 196, 148, 100, 52, 212, 20, 36, 68, 164, 116, 84, 228, 10, 250, 186, 138, 202, 154, 106, 58, 218, 26, 42, 74, 170, 122, 90, 234, 7, 247, 183, 135, 199, 151, 103, 55, 215, 23, 39, 71, 167, 119, 87, 231, 5, 245, 181, 133, 197, 149, 101, 53, 213, 21, 37, 69, 165, 117, 85, 229, 14, 254, 190, 142, 206, 158, 110, 62, 222, 30, 46, 78, 174, 126, 94, 238].$

Boxes $S_{01} \dots S_{33}$ were formed in the same way. All boxes were analyzed for differential properties. It was shown that for each box there is a range of differences transforming with a probability of $1/4$. This was expected from the results of the analysis of boxes $S_0 \dots S_3$ (Tables 4 and 5). This situation occurs when one S-box receives a difference 0000 at its input (which is converted into an output difference 0000 with probability 1), while the second S-box has input a difference with a probability of occurrence $\frac{1}{4}$. However, such precalculations significantly reduce the time to analyze the hash function structure and allow a better understanding of byte's difference changes. An example of a fragment of the table with the results of the analysis for box S_{00} is shown in Figure 3. The vertical line represents some input differences and the horizontal line the output differences. At the intersection, the cells for which the probability of forming the corresponding output difference for a certain input difference is $\frac{1}{4}$ are colored.

The problem of differential cryptanalysis for finding collisions of the hashing function is posed as follows: it is necessary to construct a transformation of text differences such that the input difference has a non-zero value, and the output difference has a zero value. It is important to note that the probability of finding such a pair of texts should be less than the probability of finding a collision using the brute force method.

dA/dC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1																														
2																														
3																														
4																														
6																														
7																														
8																														
10																														
12																														
13																														
14																														
15																														
16																														
32																														
48																														
64																														
96																														
112																														
128																														
160																														
192																														
208																														
224																														
240																														

Figure 3. A fragment of the table with the results of the analysis for box S_{00} .

An important feature of the HBC-256 hashing algorithm is the fact that the same value is received at the input of the production of the round keys and at the input of the function itself. At the same time, the processing of these values is insignificantly different, which adds additional complexity to the analysis.

We start with the analysis of the function for generating subkeys. The function contains Stage-1, Stage-2, Stage-3 conversions. Note that in Stage-2 only a shift to the left by 1 bit is performed, without the XOR operation, which is present in the Stage-2 operation in the direct hashing function. All three conversions occur 8 times one after another. After that, the original (input) matrix is added to the resulting (output) matrix.

The input matrix goes to the Stage-1 operation. The transformation starts with the element with the index 00. All elements in the same row and the same column with the selected element are added, and the selected element is also taken into account. After that, the obtained new element is changed by the S-box of replacement to a new one, according to the index, and is overwritten in the matrix. Then we move on to the element with index 01 and so on up to element 33. The complexity lies in the element overwriting and in the fact that each element, according to the index, is replaced by its own S-box.

The idea for finding the right pairs of texts was to find such differences in texts that, having gone through the transformation, would involve the least number of array elements. The work began with the search for a suitable matrix fill. Many different combinations were tried. An example of one of the options is shown in Figure 4. In Figure 4 and further, white cells mean zero difference in the state, while black cells mean nonzero value of the difference. Further analysis revealed an even more appropriate variant of difference transformation (Figure 5).

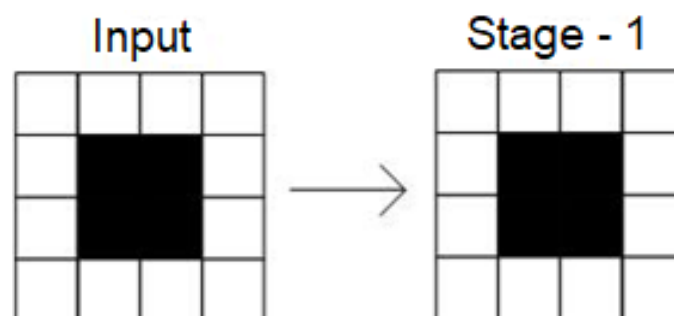


Figure 4. Variant of the initial filling of the data set.

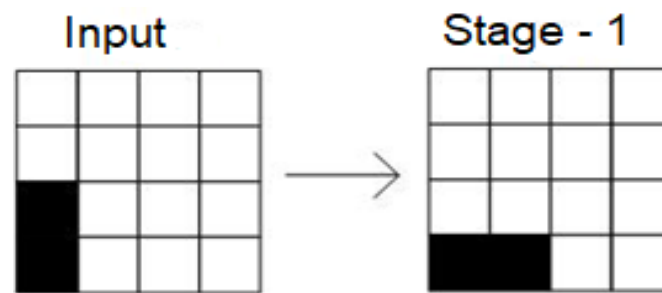


Figure 5. Improved version of the initial filling of the data set.

Further examination of the data transformations from Figure 5, it was found that even after the Stage-2 and Stage-3 transformations, only three different replacement boxes (S_{20} , S_{30} , S_{31}) remain affected each time. These transformations are depicted in Figure 6.

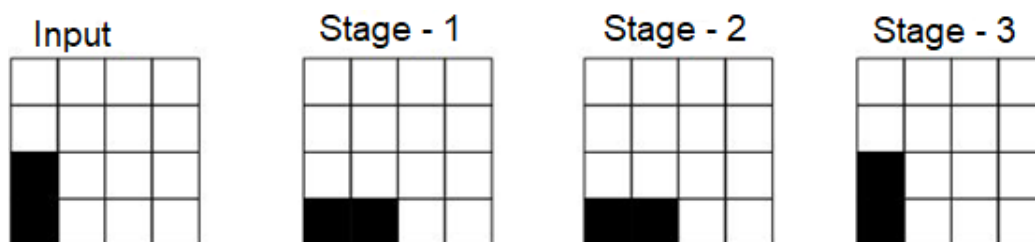


Figure 6. Conversion of the difference through three basic operations.

As a result of the found transformation (Figure 6), a hypothesis appeared: if after the first round of transformations the difference state coincides with the difference of the first subkey, and also if further, the second-round key acquires a zero difference, then further transformations will go the same way, and hence a collision will be obtained (Figure 7).

Hypothesis. There is a difference a which, when passing through one round of hashing, forms a difference $(a + b)$ in bytes 20 and 30. At the same time, the same difference a forms a difference $(a + b)$ in the same bytes (20 and 30) when passing through one round of round keys generation. In addition, the difference $(a + b)$ in bytes 20 and 30 can be converted into itself when passing through one round of the generation of round keys.

Tables S_{20} , S_{30} , and S_{31} must convert the same input difference values into the same output difference values according to Figure 6. It means that if two differences equal to the value a are received at the input, then two differences equal to the value b or $(a + b)$ must be formed at the output, but they must coincide in bytes 20 and 30 (taking into account that different substitution boxes are used there). In Stage2, the difference values may be different. It is important to remember that the Stage2 operation is different for the keying operation and the hashing function itself.

The search is performed for chains, in which the same values at the input of tables S_{30} , and S_{31} will give the same values. This must be done for the Stage-1 conversion. Then these values should be multiplied by two (equal to a shift to the left by 1), which will give us a Stage-2 transformation. Then these values should go back to the original value but by tables S_{20} , and S_{30} . If we transfer all this to symbolic form, we get the input value A . It is rearranged by the replacement tables S_{30} , and S_{31} to the value C . Then this element is multiplied by two, and we get $(C \times 2)$. After that this value is rearranged by substitution tables S_{20} , and S_{30} back to value B .

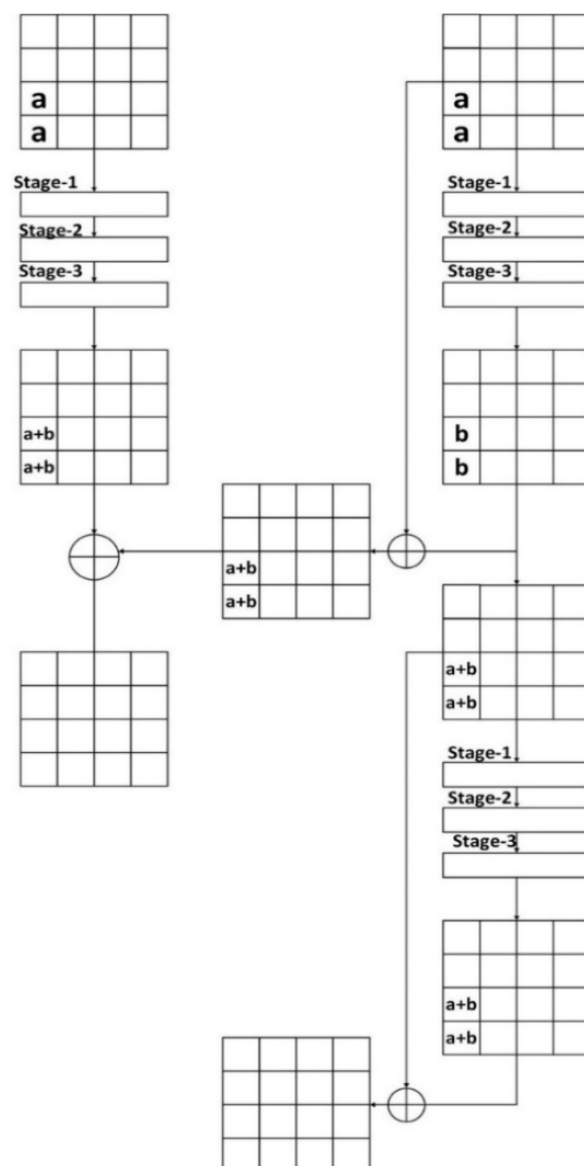


Figure 7. Difference conversion scheme for the HBC-256 hashing function.

Thus, we want to achieve the following indicators for the transformation of differences (Table 6).

Table 6. Difference conversion scheme for the HBC-256 hashing function.

	Hashing Function	Keys Generation
Stage 1	$a \rightarrow a + b$	$a \rightarrow b$
Stage 2	$0 \rightarrow 0$	$a + b \rightarrow a + b$

It should be taken into account that first, the value with index 20 goes to Stage-1, which is converted to zero. Then the value with index 30, which by S-box of permutation is changed to something. You should not use output differences greater than 128 so that after the conversion of Stage-2 neighboring boxes are not changed.

As a result of the analysis, we obtained chains in accordance with the hypothesis for the key conversion (Table 7). All values represent one byte of difference, written in decimal form, the record corresponds to the scheme: Input difference \rightarrow Difference after Stage1 \rightarrow Difference after Stage2 \rightarrow Difference after Stage3, with the values in parentheses for the two S-boxes

involved (number of appearances out of 256). The first column contains the transformations of the first round, the second column contains the transformations of the second round):

Table 7. Found chains for transforming differences in key generation.

Input difference → Difference after Stage1 (a) → Difference after StageKey2 → Difference after Stage3 (b) → Input Difference XOR Difference after Stage3 (a + b)	Input difference (a + b) → Difference after Stage1 → Difference after StageKey2 → Difference after Stage3 (a + b)
27 (8 8) → 19 → 38 → 245 (4 4) → 238	238 (4 4) → 92 → 184 (8 4) → 238
27 (8 8) → 19 → 38 → 53 (4 4) → 46	46 (8 4) → 93 → 186 (4 8) → 46
27 (8 8) → 23 → 46 → 245 (4 4) → 238	238 (4 4) → 92 → 184 (8 4) → 238
27 (4 4) → 43 → 86 → 140 (8 8) → 151	151 (8 8) → 69 → 138 (8 4) → 151
	151 (8 4) → 73 → 146 (4 4) → 151
	151 (4 4) → 85 → 170 (8 4) → 151
27 (4 4) → 43 → 86 → 60 (4 4) → 39	39 (8 8) → 69 → 138 (4 4) → 39
	39 (8 4) → 79 → 158 (4 4) → 39
	39 (4 4) → 85 → 170 (4 4) → 39
27 (4 8) → 46 → 92 → 140 (8 8) → 151	151 (8 8) → 69 → 138 (8 4) → 151
	151 (8 4) → 73 → 146 (4 4) → 151
	151 (4 4) → 85 → 170 (8 4) → 151
28 (8 8) → 19 → 38 → 59 (4 4) → 39	39 (8 8) → 69 → 138 (4 4) → 39
	39 (8 4) → 79 → 158 (4 4) → 39
	39 (4 4) → 85 → 170 (4 4) → 39
28 (8 8) → 23 → 46 → 190 (4 8) → 162	162 (8 4) → 82 → 164 (8 8) → 162
29 (8 8) → 19 → 38 → 197 (4 4) → 216	216 (4 8) → 50 → 100 (4 8) → 216
29 (8 8) → 23 → 46 → 213 (4 4) → 200	200 (4 4) → 52 → 104 (4 4) → 200
29 (4 4) → 43 → 86 → 54 (4 4) → 43	43 (8 8) → 45 → 90 (4 4) → 43
	43 (4 8) → 69 → 138 (4 4) → 43
39 (8 8) → 69 → 138 → 108 (4 8) → 75	75 (8 8) → 69 → 138 (4 4) → 75
	75 (8 16) → 70 → 140 (4 4) → 75
75 (8 16) → 70 → 140 → 143 (8 8) → 196	196 (8 4) → 99 → 198 (4 8) → 196
	196 (8 4) → 100 → 200 (4 8) → 196
77 (8 8) → 69 → 138 → 102 (4 4) → 43	43 (8 8) → 45 → 90 (4 4) → 43
	43 (4 8) → 69 → 138 (4 4) → 43
	43 (8 8) → 45 → 90 (4 4) → 43
	43 (4 8) → 69 → 138 (4 4) → 43
77 (8 16) → 70 → 140 → 239 (4 4) → 162	162 (8 4) → 82 → 164 (8 8) → 162
77 (8 16) → 70 → 140 → 31 (4 4) → 82	82 (4 4) → 113 → 226 (4 4) → 82
77 (16 4) → 85 → 170 → 149 (8 8) → 216	216 (4 8) → 50 → 100 (4 8) → 216
97 (4 4) → 99 → 198 → 195 (4 4) → 162	162 (8 4) → 82 → 164 (8 8) → 162
97 (4 4) → 100 → 200 → 147 (4 4) → 242	242 (8 4) → 113 → 226 (8 8) → 242
	242 (4 8) → 119 → 238 (4 8) → 242
108 (4 16) → 70 → 140 → 166 (4 4) → 202	202 (4 4) → 35 → 70 (4 4) → 202
	202 (4 4) → 43 → 86 (4 4) → 202
	202 (4 8) → 74 → 148 (4 4) → 202
	202 (4 4) → 99 → 198 (4 4) → 202
	202 (4 4) → 100 → 200 (4 4) → 202
154 (4 4) → 73 → 146 → 85 (4 8) → 207	207 (4 4) → 35 → 70 (4 4) → 207
154 (4 8) → 105 → 210 → 82 (4 4) → 200	200 (4 4) → 52 → 104 (4 4) → 200
154 (4 8) → 105 → 210 → 189 (8 4) → 39	39 (8 8) → 69 → 138 (4 4) → 39
154 (4 8) → 105 → 210 → 189 (8 4) → 39	39 (8 4) → 79 → 158 (4 4) → 39
	39 (4 4) → 85 → 170 (4 4) → 39
202 (4 4) → 43 → 86 → 18 (4 4) → 216	216 (4 8) → 50 → 100 (4 8) → 216
202 (4 8) → 74 → 148 → 215 (4 4) → 29	29 (8 8) → 19 → 38 (8 8) → 29
	29 (8 8) → 23 → 46 (16 8) → 29
	29 (4 4) → 43 → 86 (4 8) → 29
	29 (4 8) → 46 → 92 (4 8) → 29
221 (8 4) → 82 → 164 → 181 (4 4) → 104	104 (4 8) → 50 → 100 (8 8) → 104
	104 (4 4) → 52 → 104 (4 8) → 104

Similarly, we find chains for the hashing function (Table 8). We use as input differences only the input differences from Table 7 and look for such chains that at the output the difference corresponding to the difference $(a + b)$ is formed. In Table 8, all values are one byte of difference, written in decimal form, the notation corresponds to the scheme: Input difference \rightarrow Difference after Stage 1 \rightarrow Difference after Stage 2 \rightarrow Difference after Stage 3.

Table 8. Found chains for difference transformation in hashing.

Input difference \rightarrow Difference after Stage1 \rightarrow Difference after Stage2 \rightarrow Difference after Stage3	Input difference \rightarrow Difference after Stage1 \rightarrow Difference after Stage2 \rightarrow Difference after Stage3
27 (8 8) \rightarrow 19 \rightarrow 53 (8 4) \rightarrow 238	77 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 162
27 (4 4) \rightarrow 25 \rightarrow 43 (4 8) \rightarrow 46	77 (4 4) \rightarrow 25 \rightarrow 43 (4 4) \rightarrow 43
27 (4 4) \rightarrow 27 \rightarrow 45 (4 8) \rightarrow 46	77 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 82
27 (8 8) \rightarrow 19 \rightarrow 53 (8 4) \rightarrow 238	77 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 216
27 (8 8) \rightarrow 19 \rightarrow 53 (4 4) \rightarrow 151	97 (4 4) \rightarrow 52 \rightarrow 92 (4 4) \rightarrow 162
27 (8 8) \rightarrow 23 \rightarrow 57 (4 4) \rightarrow 39	97 (4 8) \rightarrow 50 \rightarrow 86 (4 4) \rightarrow 241
27 (8 8) \rightarrow 19 \rightarrow 53 (4 4) \rightarrow 151	108 (4 8) \rightarrow 120 \rightarrow 136 (4 4) \rightarrow 202
28 (8 8) \rightarrow 119 \rightarrow 153 (4 4) \rightarrow 39	154 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 207
28 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 162	154 (4 8) \rightarrow 120 \rightarrow 136 (4 4) \rightarrow 207
29 (4 4) \rightarrow 43 \rightarrow 125 (16 16) \rightarrow 216	154 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 200
29 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 200	154 (4 4) \rightarrow 113 \rightarrow 147 (4 4) \rightarrow 39
29 (4 4) \rightarrow 25 \rightarrow 43 (4 4) \rightarrow 43	202 (4 4) \rightarrow 73 \rightarrow 219 (4 4) \rightarrow 216
29 (4 4) \rightarrow 27 \rightarrow 45 (4 4) \rightarrow 43	202 (4 4) \rightarrow 79 \rightarrow 209 (4 4) \rightarrow 216
29 (4 4) \rightarrow 43 \rightarrow 125 (4 4) \rightarrow 43	202 (4 4) \rightarrow 73 \rightarrow 219 (8 4) \rightarrow 29
39 (4 4) \rightarrow 43 \rightarrow 125 (4 4) \rightarrow 75	221 (4 8) \rightarrow 44 \rightarrow 116 (16 16) \rightarrow 104
39 (8 8) \rightarrow 45 \rightarrow 119 (8 8) \rightarrow 75	221 (4 4) \rightarrow 79 \rightarrow 209 (4 4) \rightarrow 104
75 (8 16) \rightarrow 70 \rightarrow 202 (4 8) \rightarrow 196	

Tables 7 and 8 show that the probability of converting one difference byte ranges from $\frac{1}{2^6}$ to $\frac{1}{2^4}$. In order to determine the final probability of conversion of one round, it is necessary to calculate the number of non-zero difference conversions through S-boxes of replacement. There will be 4 non-zero S-boxes involved for one round of key conversion. Since the generation of one key occurs for 8 rounds, the generation of the first subkey will require the conversion of non-zero bytes through 32 non-zero S-boxes. As a result, the probability of difference conversion when generating the first key can be from $\frac{1}{2^{192}}$ to $\frac{1}{2^{128}}$. The generation of the second-round key will have the same probability. The one-round conversion for the squeeze function will also involve 4 non-zero S-boxes and will have a probability in the range from $\frac{1}{2^6}$ to $\frac{1}{2^4}$. Thus, combining the obtained probabilities according to Figure 7, we obtain that the conversion probability as a result of the hypothesis will have a probability of $\frac{1}{2^{390}}$ in the worst case, and $\frac{1}{2^{250}}$ in the best case.

The results of the differential analysis of the HBC-256 algorithm are based on the S-box analysis performed in Tables 2–5, scheme of analysis is presented at Figures 3 and 7. As a result of the analysis of the obtained data, the possibility of building paired chains of difference transformation (for text conversion and key generation) was hypothesized, which is theoretically confirmed by the data in Tables 7 and 8.

The biggest limitation of the experimental proof of research hypothesis is the difficulty of using full-size inputs and outputs for the developed hashing algorithms, because in this case the analysis becomes time-consuming, uses huge computational resources, and has a high time complexity.

5. Conclusions

The research results correspond to the robustness of full-round new HBC-256 hash algorithm to differential analysis method. As a result of the research, it is shown that for the proposed scheme for constructing differences, the probability of finding a collision is $\frac{1}{2^{390}}$ in the worst case, and $\frac{1}{2^{250}}$ in the best case. The collision itself was not found directly due to

the limited computing resource since such probabilities require the use of a supercomputer. One solution to such problems is to use reduced models or reduced functions, which allow for modeling and approximating the result.

This research is the first observation of differential properties of the new hashing algorithm HBC-256. For the method of differential cryptanalysis of the HBC-256 algorithm, it is necessary to experimentally confirm the hypothesis by computing texts that lead to a collision. Thus, with the current hashing function design, where the key is generated in eight rounds, the practical use of differential cryptanalysis is unjustified.

Author Contributions: Conceptualization, S.N. and N.K.; methodology, N.K. and K.S.; software, D.D.; validation, K.S. and K.A.; formal analysis, K.A.; investigation, K.S., K.A. and D.D.; resources, D.D. and K.S.; data curation, S.N.; writing—original draft preparation, K.S. and K.A.; writing—review and editing, N.K.; visualization, N.K.; supervision, S.N.; project administration, S.N. All authors have read and agreed to the published version of the manuscript.

Funding: The research work was funded by the Ministry of Science and Higher Education of Kazakhstan and 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.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors are grateful to all lab members of “Information security laboratory” (Institute of Information and Computational Technologies) for their useful suggestions and support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Katz, J.; Lindell, Y. *Introduction to Modern Cryptography*, 2nd ed.; Chapman & Hall/CRC: New York, NY, USA, 2014; pp. 231–235.
2. Bogdanov, A.; Knezevic, M.; Leander, G.; Toz, D.; Varici, K.; Verbauwhede, I. SPONGENT: The Design Space of Lightweight Cryptographic Hashing. *IEEE Trans. Comput.* **2013**, *62*, 2041–2053. [\[CrossRef\]](#)
3. Harshvardhan, T. Merkle-Damgård Construction Method and Alternatives: A Review. *J. Inf. Organ. Sci.* **2017**, *41*, 283–304. [\[CrossRef\]](#)
4. Boneh, D.; Shoup, V. *A Graduate Course in Applied Cryptography*; Version 0.5; Stanford University: Stanford, CA, USA, 2020; pp. 248–258.
5. Yano, M.; Dai, C.; Masuda, K.; Kishimoto, Y. *Correction to: Blockchain and Crypto Currency*; Springer Open: Berlin/Heidelberg, Germany, 2020; pp. 1–20. [\[CrossRef\]](#)
6. Morris, J.D. Sha-3 standard: Permutation-based-hash-and-extendable-output-functions. In *Federal Information Processing Standards—(FIPS-202)*; U.S. Department of Commerce: Washington, DC, USA, 2015. [\[CrossRef\]](#)
7. Mestiri, H.; Barra, I.; Machhout, M. A High-Speed KECCAK Architecture Resistant to Fault Attacks. In Proceedings of the 32nd International Conference on Microelectronics (ICM), Aqaba, Jordan, 14–17 December 2020; pp. 1–4. [\[CrossRef\]](#)
8. Mestiri, H.; Barra, I.; Machhout, M. Analysis and Detection of Errors in KECCAK Hardware Implementation. In Proceedings of the IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS), Sfax, Tunisia, 7–10 June 2021; pp. 1–6. [\[CrossRef\]](#)
9. Tran, X.D.; Lukowiak, M.; Radziszowski, S.P. Effectiveness of variable bit-length power analysis attacks on SHA-3 based MAC. In Proceedings of the 2016 IEEE Military Communications Conference, Baltimore, MD, USA, 1–3 November 2016; pp. 794–799. [\[CrossRef\]](#)
10. Luo, P.; Fei, Y.; Zhang, L.; Ding, A.A. Differential Fault Analysis of SHA3-224 and SHA3-256. In Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Santa Barbara, CA, USA, 16 August 2016; pp. 4–15. [\[CrossRef\]](#)
11. Luo, P.; Athanasiou, K.; Fei, Y.; Wahl, T. Algebraic fault analysis of SHA-3. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 151–156. [\[CrossRef\]](#)
12. Song, L.; Liao, G.; Guo, J. Non-full Sbox linearization: Applications to collision attacks on round-reduced Keccak. In Proceedings of the 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, 20–24 August 2017; Volume 10402, pp. 428–451. [\[CrossRef\]](#)
13. Song, L.; Guo, J.; Shi, D.; Ling, S. New MILP Modeling: Improved Conditional Cube Attacks on Keccak-Based Constructions. *Advances in Cryptology—ASIACRYPT 2018*. In Proceedings of the 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, Australia, 2–6 December 2018; Volume 11273, pp. 65–95. [\[CrossRef\]](#)

14. Liu, F.; Isobe, T.; Meier, W.; Yang, Z. Algebraic Attacks on Round-Reduced Keccak. In Proceedings of the 26th Australasian Conference, ACISP 2021, Virtual, 1–3 December 2021; Volume 13083, pp. 91–110. [\[CrossRef\]](#)
15. Li, C.; Zhang, Y.; Xie, E.Y. When an attacker meets a cipher-image in 2019: A year in review. *J. Inf. Secur. Appl.* **2019**, *48*, 102361. [\[CrossRef\]](#)
16. Sakan, K.; Nyssanbayeva, S.; Kapalova, N.; Algazy, K.; Khompysh, A.; Dyusenbayev, D. Development and analysis of the new hashing algorithm based on block cipher. *Eastern-Eur. J. Enterp. Technol.* **2022**, *2*, 60–73. [\[CrossRef\]](#)
17. Cherkesova, L.V.; Safaryan, O.A.; Lyashenko, N.G.; Korochentsev, D.A. Developing a New Collision-Resistant Hashing Algorithm. *Mathematics* **2022**, *10*, 2769. [\[CrossRef\]](#)
18. Al-Odat, Z.; Khan, S. Constructions and Attacks on Hash Functions. In Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 5–7 December 2019; pp. 139–144. [\[CrossRef\]](#)
19. Mouha, N.; Velichkov, V.; De Cannière, C.; Preneel, B. The Differential Analysis of S-Functions. In *Selected Areas in Cryptography*; Biryukov, A., Gong, G., Stinson, D.R., Eds.; SAC 2010. Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6544. [\[CrossRef\]](#)
20. Biham, E.; Shamir, A. Differential cryptanalysis of the full 16-round DES. In Proceedings of the 12th Annual International Cryptology Conference, Santa Barbara, CA, USA, 16–20 August 1992; pp. 487–496. [\[CrossRef\]](#)
21. Biham, E.; Shamir, A. *Differential Cryptanalysis of the Data Encryption Standard*; Springer: Berlin/Heidelberg, Germany, 1993; pp. 133–148. ISBN 978-1-4613-9314-6.
22. Biham, E.; Dunkelman, O. *Differential Cryptanalysis in Stream Ciphers*; No. CS Technion Report CS-2007-10; Computer Science Department, Technion: Haifa, Israel, 2007; pp. 1–12. Available online: <http://eprint.iacr.org/> (accessed on 1 July 2022).
23. Algazy, K.T.; Babenko, L.K.; Biyashev, R.G.; Ishchukova, E.A.; Kapalova, N.A.; Nysynbaeva, S.E.; Smolarz, A. Differential Cryptanalysis of New Qamal Encryption Algorithm. *Int. J. Electron. Telecommun.* **2020**, *66*, 647–653. [\[CrossRef\]](#)
24. Ishchukova, E.; Tolomanenko, E.; Babenko, L. Differential analysis of 3 round Kuznyechik. In Proceedings of the 10th International Conference on Security of Information and Networks, Jaipur, India, 13 October 2017; pp. 29–36. [\[CrossRef\]](#)
25. Khompysh, A.; Kapalova, N.; Algazy, K.; Dyusenbayev, D.; Sakan, K. Design of substitution nodes (S-Boxes) of a block cipher intended for preliminary encryption of confidential information. *Cogent Eng.* **2022**, *9*, 2080623. [\[CrossRef\]](#)