A new hashing algorithm - HAS01: development, cryptographic properties and inclusion in graduate studies

Nursulu Kapalova†‡, Dilmukhanbet Dyusenbayev† & Kairat Sakan†‡

Institute of Information and Computational Technologies, Almaty, Republic of Kazakhstan†
Al-Farabi Kazakh National University, Almaty, Republic of Kazakhstan‡

ABSTRACT: A new hashing algorithm - HAS01 is considered in this article. This algorithm is part of the hash function of the SHA-3 family, and was developed on the basis of the cryptographic sponge construction. The difference between the proposed scheme and the classical one is that at the absorption stage, the function *f*, which is part of the function *F*, is called more than once. Besides, when generating a hash value, each element of the sequence is determined by copying a certain column of the state matrix, i.e. the current state of the hash generated at the moment. The hashing algorithm HAS01 converts a plaintext *M* of arbitrary length, which is iteratively processed in 192-bit blocks, into a hash value of 256 or 512 bits. The maximum throughput, the simplicity of software and hardware implementation, and the required level of resistance to collisions were the main goals of the scientific work. To assess the reliability and conduct cryptographic analysis, HAS01 was software implemented in the C++ and Python programming languages. During the analysis, the *avalanche effect* was investigated, and the possibility of using the chaining technique and the differential cryptanalysis method to find collisions was considered. It is shown that the considered methods for finding collisions are not effective. Overall, HAS01 is efficient, simple to use and is recommended for inclusion in graduate studies.

Keywords: Hash function, cryptographic sponge construction, collision, differential cryptanalysis, chaining method

INTRODUCTION

When it comes to the development of technology in the modern world, including information systems, the first thing that comes to mind is the *Internet* and *data exchange*, and then the question immediately arises *how secure and safe they are*. Modern methods of processing, transferring and storing information have contributed to the emergence of threats associated with the possibility of causing such harm as the loss, modification and disclosure of personal data to illegitimate users. Therefore, ensuring the information security of computer systems and networks is one of the leading directions in the development of information technologies. Ways to protect data include the use of hardware devices and the implementation of specialised hardware and software. Protection of information by means of cryptographic transformations consists in changing its semantics with the help of special encryption algorithms or hardware solutions and key codes, that is, in bringing it to an implicit form. In addition to encryption algorithms, hash functions play an important role in cryptography. They match messages of arbitrary length with hash codes (digests) of fixed length. Initially, they were used to check the integrity of a message, but over time they have become used in almost all areas of the on-line world.

In the examples mentioned above, one not only needs to manage confidentiality and integrity, but sometimes authentication (hash with key) and many other functions, and this is done by standard dedicated hash functions - secured hash algorithms (SHA), such as SHA-224, SHA-256, SHA-384, SHA-512 (collectively referred to as SHA-2 [1]) and now SHA-3 [2] can also be counted on this list. SHA-2 is still popular for data integrity, privacy, digital signatures and many other uses [3]. In 2005, cryptanalysts identified a potential vulnerability in the SHA-1 algorithm, which is now the *de facto* standard for hash functions.

Modern scientific and practical achievements in cryptanalysis continue to reveal the shortcomings inherent in most popular hash functions, which encourages further targeted research in this area. Thus, there was a need for a new and more thorough study of the principles of constructing hash functions. Based on this, two research approaches have been identified by the cryptography community: 1) fixing existing designs by slightly modifying them to address a certain set of shortcomings; and 2) developing new hash function designs. In practice, both approaches have disadvantages and advantages, since existing designs have been carefully analysed and adjusted over time, and structurally new hash functions may well be subject to more attacks than already tested functions [4].

Based on international experience in the field of information security, in particular in the field of authentication, and drawing on the shortcomings revealed over time, it can be argued that research work on hash functions is timely and

relevant. As for the reliability of each specific algorithm, it requires a comprehensive study and a subsequent conclusion on the advisability of further analysis.

LITERATURE REVIEW AND PROBLEM STATEMENT

Hash functions solve the problem of the volume of incoming data, and in the modern world of digital technologies, algorithms that work with compact data are in high demand. The hash function mechanism is also used to reduce the time required to generate and verify a signature, as well as to reduce its length. From the point of view of the reliability of the applied algorithms and confidence in them, when it comes to information security, each state seeks to develop its own national standards in the field of cryptography. In many countries, cryptographic standards are domestic products, including those for hash functions and cryptographic hashing algorithms.

At the moment, a common algorithm for calculating hash values is the US standard called Keccak (SHA-3) [5]. In 2015, the National Institute of Standards and Technology (NIST) chose the algorithm proposed by the Keccak team as the standard SHA-3 hash function. SHA-3 consists of a sponge construction. The completely new sponge construction proved to be resistant to the available methods of attacking existing hash functions. The previous hash functions SHA-1 and SHA-2 had a similar structure, and for them, cryptographers in different years presented quite a lot of attack methods and ways to find collisions [6-8]. The SHA-3 algorithm is fundamentally different in sponge architecture and is a simple iterative construction of two phases. In the absorbing phase, the initial state is first set from a zero vector up to $b = r + c$ bits in size (b is the *width*). Then, a fragment of the original message is added to modulo 2 with a fragment of the initial state of size r (r is *bitrate*), and the remaining state part of capacity c (c is *capacity*) remains unaffected. The result is processed by the function f - multi-round keyless pseudo random permutation (PRP). This process is repeated until the blocks of the original message are exhausted. Then comes the squeezing phase, where the hash of the specified length is output in a certain order.

The algebraic analysis of the finalists of the SHA-3 competition based on SAT solvers was presented in earlier studies [9][10]. The paper by Morawiecki and Srebrny describes an SAT-based preimage search attack on smaller versions of the Keccak algorithm [9]. Preimages for truncated versions of Keccak have been found, particularly, a preimage for a 3-round Keccak-f with 40 unknown message bits. The studies presented by Morawiecki et al made it possible to perform a preimage search attack for the 4-round Keccak algorithm with the complexity of 2^{506} and determine the distinguisher on a 5-round Keccak-f permutation with the complexity of 2^{15} [10].

Suryawanshi et al [11] describe a method for analysing the Keccak algorithm based on the combination of two approaches: the SymSum property [12] and the idea of using linear structures to develop an efficient security analysis method in terms of computational complexity and number of rounds. The authors managed to carry out an attack on 9 rounds of the SHA-3 version with the attack complexity of 2^{64} [11].

The work by Luo et al describes an attack on the θth iteration of the first round of MAC-Keccak implemented on an FPGA [13]. The authors have generated several different side-channel leak models and implemented attacks based on them. The study showed that the use of non-masked hardware implementation of SHA-3 is vulnerable to side-channel attacks based on power analysis [13].

According to Choi and Seo, software implementations of SHA-3 still do not provide sufficient performance for various applications [14]. Therefore, in their work, they offer an optimised software implementation of SHA-3 in the GPU environment. As a result of applying multiple methods, such as SHA-3 internal process optimisation and PTX built-in optimisation, the SHA-3 algorithm on the RTX2080Ti GPU achieves a maximum throughput of 88.51 Gb/s, and with the use of CUDA asynchronous stream, 171.62 Gb/s. Here, for comparison, other GPUs (GTX1070 and GTX1080) are considered, with which SHA-3 can be effectively used for blockchain applications and the key generation process in lattice-based cryptosystems [14].

The purpose of this current work is to design a new hashing algorithm HAS01 based on the sponge construction, which, taking into account the basic requirements for hash functions, allows for improving their quality indicators, as well as identifying the algorithm's strengths and weaknesses. To achieve this goal, it is necessary to solve the following tasks:

- Development of the HAS01 hashing algorithm;
- Development of a software product for the implementation of HAS01 hash functions and obtaining experimental results;
- Application of various research methods to evaluate the quality of HAS01 hash functions.

METHODOLOGY

Properties to Determine the Quality of a Hash Function

A mathematical transformation H defined by a hash function is called hashing. An input data M is called an input array, key or message. The result of the transformation H is called a hash, hash code, hash value, hash sum, message summary or digest. Traditionally, the cryptographic strength of a hash function is determined by the complexity of solving the following main tasks:

1. Hash function inversion (preimage attack). Given the value $h \in V_n$, find a message $M \in V_*$ such that $h = H(M)$.
2. Building a collision. Find two different messages $M, M' \in V_*$ satisfying the condition $H(M) = H(M')$. An unordered pair of distinct messages $\{M, M'\}$ for which $H(M) = H(M')$ is called a collision of the function *H*.
3. Building a second preimage (second preimage attack). Given a message $M \in V_*$ find a message $M' \in V_*$ different from *M* such that $H(M) = H(M')$.

Hash functions are built according to an iterative scheme when the original message is divided into blocks of a certain size, and a series of transformations are performed on them using both reversible and irreversible operations. As a rule, a compressing function is included in the hashing transformation. The input of each hashing cycle is the output of the previous cycle, as well as the next block of the message.

Avalanche Effect

For the analysis of the avalanche effect, an avalanche criterion is used, requiring an average change of 50% of the bits in the output sequence for each bit in the input sequence. It is estimated by the formula $\varepsilon_a = |2 * k_i - 1|$, where i is the number of the changed bit in the input sequence, k_i is the probability of changing half of the bits in the output sequence when changing the *i*th bit at the input, and ε_a is the avalanche parameter [15].

Finding Collisions by Chaining Method

There are several approaches to resolving collisions, they are based on one important requirement: with a slight change in the argument, a significant change in the function itself must occur. Thus, the hash value should not give information even about individual bits of the argument. The most famous of the popular methods is the chaining method (external or open hashing). The advantages of the chaining method are: 1) it is efficient and has a clear structure; 2) it is convenient to use when the number of collisions per hash value is unknown; and 3) the search for the desired value will occur in the minimum possible time. The disadvantages are: 1) it uses a lot of memory, i.e. ~n^2 memory cells are allocated to store *n* hash values; and 2) the running time of the method is $O(n^2)$.

The technology of linking elements is that the elements of the set (field of hashes), which correspond to the same hash, are linked into a chaining list. The chaining technique is that a certain range is created - a field of hashes, in which the desired hash is then searched. If no matches are found, then a new hash is created from the desired hash, and the search process is repeated. This algorithm is executed until a match is found. In this case, the desired hash, that is, the hash that gives the collision, is in the previous cell.

Method of Differential Cryptanalysis

Differential cryptanalysis was proposed in 1990 by Israeli specialists Eli Biham and Adi Shamir, and is a method of cryptanalysis of symmetric block ciphers and other cryptographic primitives, in particular, hash functions [16]. It is included in the class of statistical attacks and is based on the study of the transformation of the differences between encrypted values in different rounds of encryption. As a difference, as a rule, the operation of bitwise summation modulo 2 is used, although there are attacks with the calculation of the difference over different modules.

RESULTS OF DEVELOPMENT AND SECURITY ASSESSMENT OF THE NEW HASHING ALGORITHM

Development of the HAS01 Hashing Algorithm

The HAS01 hashing algorithm is based on the sponge construction, the execution of which takes place in two phases: *absorbing* and *squeezing*. HAS01 converts arbitrarily long plaintext consisting of 192-bit (or 24-byte) blocks into a hash value of 256 or 512 bits. A detailed description of the HAS01 hash function is given in an earlier publication by Sakan et al [17]. Figure 1 shows the general scheme of the HAS01 hashing algorithm:

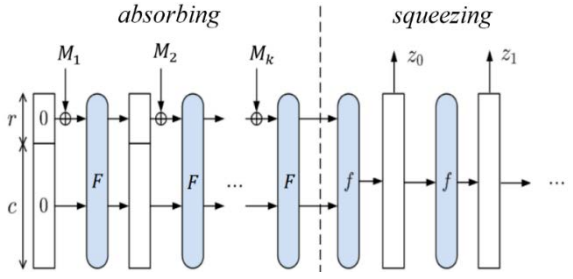

Figure 1: Sponge construction used by the HAS01 hashing algorithm.

The input data block M_i has a size of 24 bytes and can be represented as a byte sequence $m_1, m_2, \ldots, m_{24}$ or as a [3 × 8] matrix. The external state of the r_i hash has the same size as the size of the input data block (24 bytes) and can be represented as a [3 × 8] matrix. The internal state of the c_i hash has a size of 40 bytes and can be represented as a [5 × 8]

matrix. Thus, the complete state of the y_i hash is the union of the external and internal state, that is, $y_i = r_i \,||\, c_i$, and can be represented as an [8 × 8] matrix of 64 bytes. The hash value *h(M)* is the byte sequence $z_0, z_1, \ldots, z_l$. With $l = 4$, the size of the hash value is 256 bits, with $l = 8$, the size of the hash value is 512 bits.

Mathematical Description of HAS01

First, the hashing algorithm divides the plaintext message *M* into 192-bit (24-byte) blocks $M_1, M_2, \ldots, M_k$. Initially, all *b* bits of the state are set to zero, and the input message *M* is padded and divided into blocks of *r* bits each. If the length of the message *M* is not a multiple of the block length *r*, then the last block is padded with one followed by zeros. In the case when the length of *M* is a multiple of 192 bits, another 192-bit block is added to the end of *M*, consisting of zero bits, except for the first bit, which is equal to one:

$$M = M_1||M_2||M_3|| \ldots ||M_{k-1}||M_k. \quad [1]$$

The initial state of the hash y_0 contains only zero values.

In the *absorbing* phase, the following transformations are performed:

$$F(M_1, y_0) = f\left(f\left(f\left(f((M_1 \oplus 0)||0)\right)\right)\right) = y_1 = r_1||c_1$$
$$F(M_2, y_1) = f\left(f\left(f\left(f((M_2 \oplus r_1)||c_1)\right)\right)\right) = y_2 = r_2||c_2$$
$$\ldots$$
$$F(M_k, y_{k-1}) = f\left(f\left(f\left(f((M_k \oplus r_{k-1})||c_{k-1})\right)\right)\right) = y_k = r_k||c_k$$

In the *squeezing* phase, the following transformations are performed:

$$z_1 \leftarrow f(y_k) = x_1$$
$$z_2 \leftarrow f(x_1) = x_2$$
$$\ldots$$
$$z_l \leftarrow f(y_{l-1}) = x_l$$
$$h(M) = z_1||z_2|| \ldots ||z_l$$

The function $f(X)$ transforms a 64-byte [8 × 8] matrix *A* containing the current hash state into a 64-byte [8 × 8] matrix A' containing the new hash state. The function *f* is a composition of functions f_1, f_2 and f_3:

$$f(A) = f_1 \circ f_3 \circ f_2 \circ f_1 \circ f_3 \circ f_1 \circ f_2 \circ f_1(A), \quad [2]$$

where the function f_1 performs a non-linear matrix transformation, function f_2 is a matrix transposition, and function f_3 serves to transform the matrix by rows.

When the function f_1 is executed, the elements of matrix *A* are transformed row by row from left to right and from top to bottom in accordance with Algorithm 1. In Algorithm 1, the SBox operation is used, which is an S-box substitution as per Table 1.

Table 1: S-box substitution.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	A5	04	A6	A7	F7	C6	A4	12	5F	C8	C7	D1	F6	D4	7E	7B
1	0B	EF	13	AD	94	5B	4C	8A	0C	FC	CE	1C	9B	76	19	F3
2	21	68	53	96	2D	D0	A1	89	3D	9C	DA	6D	51	AF	E1	E9
3	A2	E3	09	FE	C3	3F	AA	1E	BA	DD	9F	1D	28	54	8E	92
4	E7	D5	43	33	DE	81	3C	97	32	EC	1F	72	74	CD	B3	60
5	3A	95	39	FA	1A	0E	C1	05	DF	CC	A0	8D	87	58	83	D3
6	26	FD	86	7C	20	4B	08	36	45	DC	3B	79	22	BE	AB	14
7	2A	03	99	2C	6B	E5	F9	5C	B0	85	5D	B2	30	80	ED	DB
8	57	8F	9D	A9	D6	B8	EE	24	CB	84	B7	D8	69	A8	6F	50
9	BD	F1	01	38	F8	40	4E	BF	9E	0D	91	C9	7D	F4	47	07
A	B9	63	6E	0F	EB	70	D9	6A	7A	2B	A3	CF	44	65	F5	00
B	98	35	C2	41	27	1B	62	AC	67	23	88	10	B6	8C	4D	C0
C	64	3E	5A	E8	34	D7	9A	16	B4	29	D2	37	73	F2	6C	46
D	06	E6	CA	C4	EA	7F	18	E0	B5	31	FB	FF	71	17	AE	02
E	B1	15	25	78	BB	F0	61	93	11	4F	56	82	8B	42	59	48
F	2F	E2	66	4A	0A	90	2E	75	BC	C5	E4	55	52	77	49	5E

The pseudocode of the function f_1:

for i = 0 to 7 do
  for j = 0 to 6 do
    $a'_{(i+1) \bmod 8,j} \leftarrow \mathrm{SBox}\left(\left((a_{i,j} \oplus a_{(i+1) \bmod 8,j}) \ll\right) \oplus \left((a_{i,j+1} \oplus a_{(i+1) \bmod 8,j+1}) \gg 5\right)\right)$
  end for
  $a'_{(i+1) \bmod 8,7} \leftarrow \mathrm{SBox}\left(\left((a_{i,7} \oplus a_{(i+1) \bmod 8,7}) \ll 3\right) \oplus \left((a_{i,0} \oplus a_{(i+1) \bmod 8,0}) \gg 5\right) \oplus 7\right)$
end for

The function f_2 performs the transposition of the [8 × 8] matrix *A*, consisting of the elements $a_{i,j}$ of the set $GF(2^8)$, taking values from 0 to 255. The pseudocode of the function f_2:

for i = 0 to 6 do
  for $j = i + 1$ to 7 do
    $a_{i,j} \leftrightarrow a_{j,i}$
  end for
end for

The function f_3 transforms the [8 × 8] matrix *A* into the [8 × 8] matrix *B* by row:

$b_{i,k} = \oplus_{j=0}^{7}((\left(\left(a_{i,j}[x] \times x^{8-k}\right) mod(x^8 + 1)\right) \& 1) \times x^{7-j})$.

The pseudocode of the function f_3:

for i = 0 to 7 do
  for j = 0 to 7 do
    for k = 0 to 7 do
      $b_{i,j} = b_{i,j} \oplus \left((a_{i,k} \gg j) \& 0x01\right) \ll k$
    end for
  end for
end for

As mentioned above, the HAS01 hashing algorithm is based on the sponge construction. The difference is that in the *absorbing* phase, function *f*, which is part of function *F*, is called more than once. Besides, when forming a hash value $h(M) = z_0, z_1, \ldots, z_l$, each element of the sequence z_i is obtained by copying a certain column of the state matrix, the current state of the hash generated at the moment, and in the classic sponge scheme, the external state of the hash is copied.

According to the sponge construction, the *absorbing* phase is performed first, and then the *squeezing* phase. In the *absorbing* phase, the operation of bitwise addition of the next block M_i, $i = \overline{1,k}$ of the original message M with the external state of the hash r_i, formed at the moment, is performed.

The length of the original message block, as well as the length of the external state of the r_i hash is 24 bytes. After that, the function $F(y_i)$ performs the above transformations of the current state y_i of the hash, including the internal state c_i and the external state r_i. The *absorbing* phase ends after the conversion of the last block M_k of the original message M.

In the *squeezing* phase, the elements z_i of the hash value $i = \overline{1,l}$. are formed. With $l = 4$, the size of the hash value is 256 bits, with $l = 8$, the size of the hash value is 512 bits. At each iteration of the *squeezing* phase, the elements z_i are obtained by copying the sixth column of the current hash state matrix if the hash size is 256 bits, and by copying the fourth column if the hash size is 512 bits. Thus, if the hash value is 256 bits long, then the number of iterations that will be performed in the *squeezing* phase is four, and if the hash value is 512 bits long, then the number of iterations is eight.

RESULTS OF HAS01 ANALYSIS

The brute-force method evaluates the occurrence of at least one collision in a birthday paradox attack with a probability of 0.5, while iterating 3.4 x 10^{38} for a 256-bit hash value and 1.15 x 10^{77} for 512-bit hash value.

Implementation Features of the Hashing Function HAS01 and the Experimental Data

The HAS01 function was implemented as the ISL_HASH 1.0 software product in the C++ programming language in the Qt Creator 4.15.2 integrated development environment using the Qt library version 5.15.2 (open source edition). Figure 2 shows the main program window, and the results of experimental performance measurements are shown in Table 2.

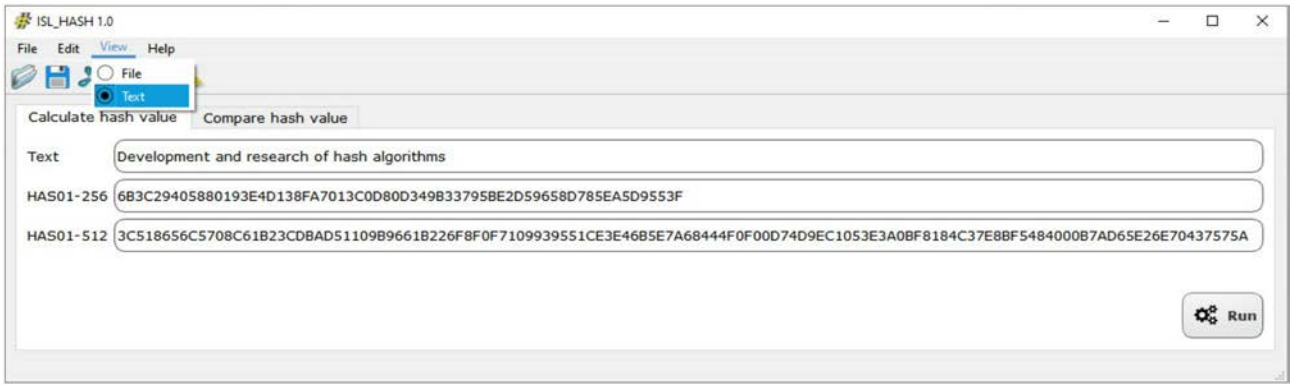

Figure 2: Main window of the ISL_HASH 1.0 data hashing program.

Table 2: Comparison of hash algorithm modes by execution speed.

HAS01 algorithm modes	Execution speed (MB/sec)
HAS01-256	5.66
HAS01-512	5.71

In addition, using the obtained implementations, experiments were carried out and time measurements of the processing speed of one message were obtained using various configurations of computer systems. To calculate the average processing time for one block of data, the same block of data was hashed 1,000 times. In multi-core processor systems, the experiment was carried out using only one core. The results of experimental measurements are shown in Table 3.

Table 3: Experimental results of software implementations.

Algorithm	PC parameters	Language	Max t	Min t	Average time
HAS01-256	4CPUx3GHz, 4GB RAM	Python	0.0331	0.0141	0.0196
	Ryzen 5 3600, 3GHz, 4GB RAM		0.0158	0.0155	0.0156
	intel i9 10900k, 3GHz, 4GB RAM		0.0127	0.0124	0.0126
HAS01-512	4CPUx3GHz, 4GB RAM	Python	0.0497	0.0213	0.0288
	Ryzen 5 3600, 3GHz, 4GB RAM		0.0346	0.0339	0.0347
	intel i9 10900k, 3GHz, 4GB RAM		0.0281	0.0274	0.0277

Study of the Avalanche Effect

The bit variance of the HAS01 hash algorithm for the *F*-function is shown in Figure 3.

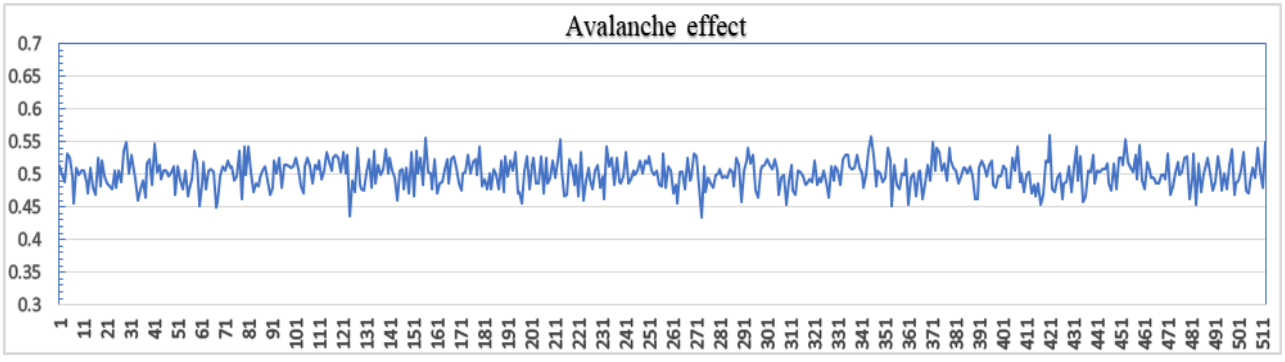

Figure 3: *F*-Function bit change probability diagram.

512-bit blocks are selected as inputs to the *F*-function, each of which differs from the first by only one bit (all bits are different). The experiment showed that all values of the probability k_i of matching the corresponding bits of the cipher text during the transformation (hashing) of these blocks using the *F*-function lie in the interval (0.427; 0.572).

As is known, the range of variation of the avalanche parameter ε_a lies in the range from 0 to 1, inclusive. The closer the value of the avalanche parameter to zero, the more the avalanche effect manifests itself in the algorithm. According to the results of the experiments, it was found that the values of the avalanche parameter are in the range (0.13; 0.01), and their average value is 0.035. The experimental values of ε_a are shown in Figure 4. Thus, the bit variance of the *F*-function and the values of the avalanche parameter ε_a indicate that the *F*-function fully satisfies the avalanche criterion.

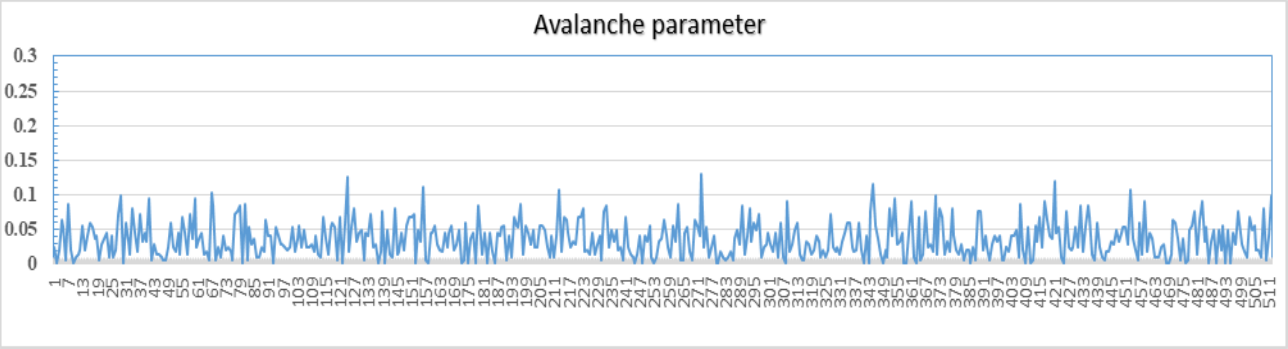

Figure 4: *F*-function avalanche parameter diagram.

Collision Search by the Chaining Method

For analysis, a field of one million hashes was created. To generate the field, it took 7185.0434 seconds or roughly two hours (on both the Intel i9 10900k and the Ryzen 5 3600). The resulting field took up 386 MB of memory. A fragment of the field is shown in Figure 5.

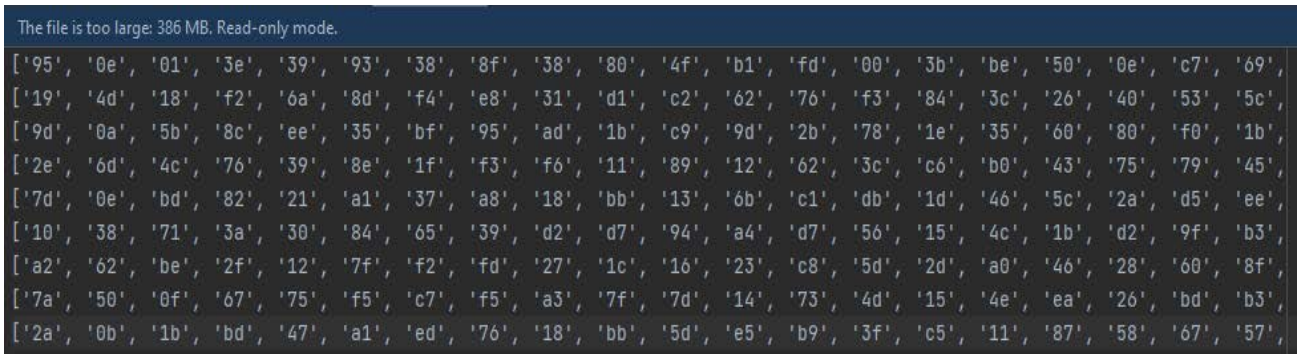

Figure 5: Field of hashes.

After that, 10 thousand hashes from hashes were created. Their generation time did not exceed 827.2601 seconds or approximately 13.8 minutes on both systems (Intel i9 10900k, Ryzen 5 3600). A fragment of the received hash field is shown in Figure 6. The amount of memory used for this field did not exceed 2.56 MB.

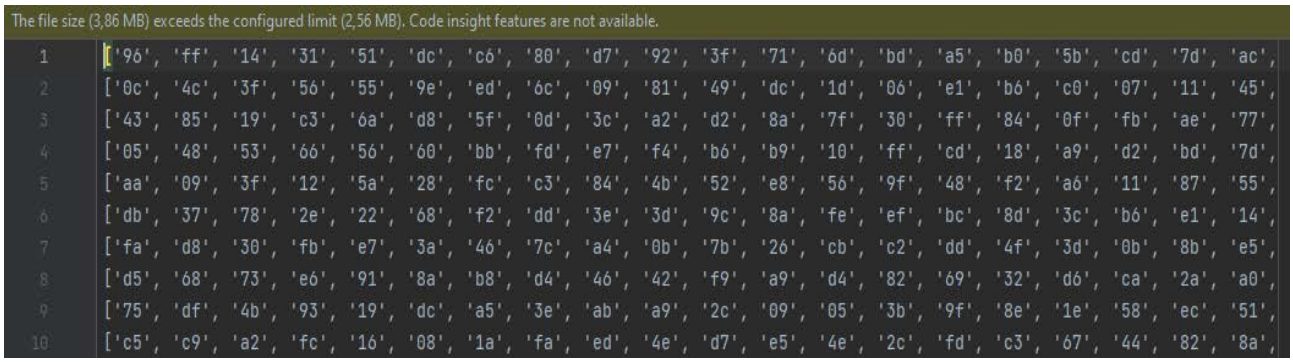

Figure 6: Hash field from hashes.

The matching process took 16239.0426 seconds or approximately 4.5 hours. In the case of an increase in the number of hashes from hashes to 20 thousand, the comparison time increases to 11 hours. At the moment, no collisions have been found for the HAS01 function using the chaining method.

Finding Collisions by Differential Cryptanalysis

For the HAS01 algorithm, the non-linear transformation is the function f_1, while the functions f_2 and f_3 are linear. The non-linearity of the function f_1 is based on the repeated application of a non-linear substitution using an S-box.

To determine the degree of influence of one non-zero bit at the input to the f_1 transformation on the output state, a matrix was constructed that reflects the formation of each output byte depending on the initial values and values obtained as a result of the operation of the f_1 transformation. Since the matrix turned out to be quite voluminous, it is presented in the form of two fragments. Figure 7 shows the f_1 transformation output matrix elements.

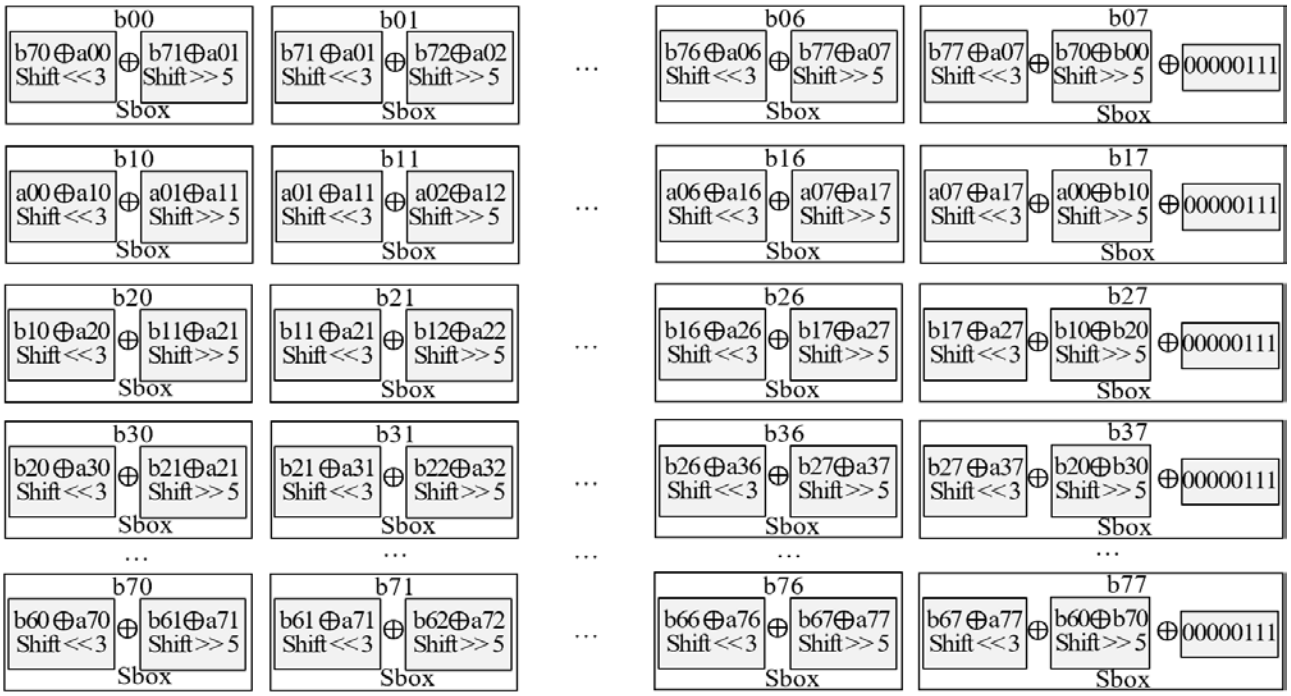

Figure 7: The f_1 transformation output matrix.

Figure 7 show that only the second row of the matrix is obtained from the original bytes of the input state. For all subsequent rows, including the first row, new bytes are formed both on the basis of the input state bytes and the basis of previously received bytes in the f_1 function. In this case, each byte formed by the f_1 goes through the S-box transformation. Thus, it turns out that in the first row of the output matrix, all generated bytes depend on previous states. The revealed regularities will be extended in the same way to the change of bits in the difference between two texts. As a consequence, this will lead to a large number of transformations in the S-box, which will rapidly reduce the probability of success in the analysis.

Figure 8 shows how changing one status bit (highlighted in red) affects and changes other bytes in the output of the f_1 function.

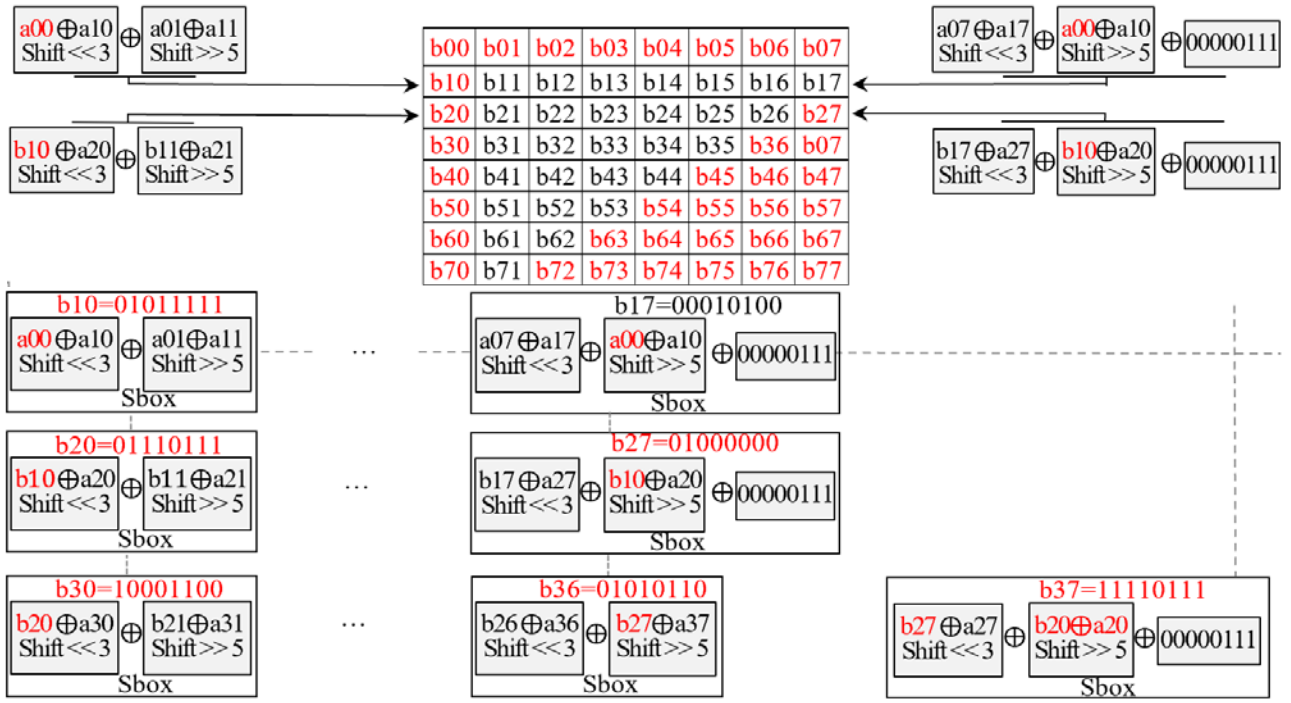

Figure 8: Effect of one bit on the output state change for the f_1 function.

From Figure 8, it can be seen that a change in one bit immediately affected the output for 36 S-boxes. Even taking into account the use of the most probable values of the differences following the analysis of the substitution S-box, it turns out that the probability of changing the difference under the considered scheme is $(1/4)^{36}$. Further, an avalanche effect of the influence of the affected blocks on the formation of a new state will be observed. Thus, differential cryptanalysis is not applicable to the analysis of the HAS01 hashing algorithm.

JOINT RESEARCH WORK WITH STUDENTS

The Institute of Information and Computational Technologies together with Al-Farabi Kazakh National University (KazNU), both in Almaty, Kazakhstan, conduct training of scientific and pedagogical personnel. In accordance with the plan of interaction with the Faculty of Information Technology for students studying in the specialty *information security systems*, seminars and round tables are organised, and relevant researchers are supervisors of undergraduates and doctoral students [18]. In particular, to assess the practicality of the HAS01 algorithm, students carried out computational and experimental studies, as a result of which the following points were revealed:

1. The simplicity of the structure of the algorithm makes it accessible for study and analysis by almost all students;
2. Due to the use of elementary mathematical operations and generally accepted cryptographic primitives, the software implementation of the algorithm in different programming languages takes students much less time than it is required for other algorithms;
3. Experimental data obtained by students show that HAS01 has sufficient performance required for further research work using conventional PCs (Table 2, Table 3);
4. Scientific supervisors decided that the study of the HAS01 hashing algorithm can become a research work component for graduate theses.

CONCLUSIONS

During the research work outlined in this article, the following main results were obtained:

A new hashing algorithm HAS01 based on the sponge construction has been developed. It converts arbitrarily long plaintext with 192-bit blocks into a hash value of 256 or 512 bits. The main difference of this algorithm is that at the absorption stage, the function *f*, which is part of the function *F*, is called more than once.

The hashing algorithm HAS01 has been implemented in software. As expected, the C++ version (Table 2) showed a multiple speedup compared to the Python version; namely, almost 5 times faster than the average processing speed per data block.

The reliability of the hashing algorithm HAS01 has been examined. Analysis of the *avalanche effect* confirmed its presence to the required extent.

For HAS01, a chaining method was considered to find collisions. It was shown that for a field of 1,000,000 hashes (or $10^6 \approx 2^{19,93}$ hashes) and 20,000 double hashes, the analysis time is 11 hours without a positive result. Thus, for a complete enumeration of the field of various hashes, consisting of 256 bits, at best, it takes 1.25 x 10^{56} years of computing on a PC with an Intel i9 10900k processor, 3GHz, 4GB RAM, if memory is not taken into account. Indeed, the running time of the chaining method can be estimated as $O(n^2)$, and an increase in the size of the hash field generates a quadratic computational complexity to find collisions.

For differential cryptanalysis of the HAS01 algorithm, it was shown that a change in one bit affects 36 S-boxes, which leads to a rapid decrease in the probability of finding a collision and makes the differential cryptanalysis method unsuitable.

ACKNOWLEDGMENTS

The research work presented in this article was carried out within the framework of the project OR11465439 - *Development and Research of Hashing Algorithms of Arbitrary Length for Digital Signatures and Assessment of their Strength* in the Institute of Information and Computational Technologies, Almaty, Kazakhstan.

REFERENCES

1. Sanadhya, S. and Sarkar, P., Attacking step reduced SHA-2 family in a unified framework. *IACR Cryptology ePrint Archive*, 271 (2008).
2. Regenscheid, A., Perlner, R., Chang, S., Kelsey, J., Nandi, M. and Paul, S., Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, 7620 (2009), 14 June 2022, https://doi.org/10.6028/NIST.IR
3. Bussi, K., Dey, D., Kumar, M. and Dass, B.K., Neeva: a lightweight hash function. *IACR Cryptology ePrint Archive*, 042 (2016).
4. Al-Kuwari, S., Davenport, J.H. and Bradford, R.J., Cryptographic hash functions: recent design trends and security notions. *Proc. of 6th China Inter. Conf. on Information Security and Cryptology (Inscrypt'10)*, Science Press of China, 133-150 (2010).
5. Dworkin, M., SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Infor. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD (2015), 16 June 2022, https://doi.org/10.6028/NIST.FIPS.202.

6. Leurent, G. and Peyrin, T., SHA-1 is a shambles: first chosen-prefix collision on SHA-1 and application to the PGP web of trust. *Proc. 29th USENIX Secur. Symp.*, Capkun, S. and Roesner, F. (Eds), Berkeley, CA, USA: USENIX Association, 1839-1856 (2020).
7. Stevens, M., Bursztein, E., Karpman, P., Albertini, A. and Markov, Y., The First Collision for Full SHA-1. Advances in Cryptology - CRYPTO 2017. Lecture Notes in Computer Science, Katz, J. and H. Shacham, H. (Eds), Santa Barbara, CA, USA: Springer, 10401, 570-596 (2017).
8. Sanadhya, S.K. and Sarkar, P., New collision attacks against up to 24-step SHA-2. Progress in Cryptology - INDOCRYPT 2008. Lecture Notes in Computer Science, Chowdhury, D.R., Rijmen, V. and Das, A. (Eds), Kharagpur, India: Springer, 5365, 91-103 (2008).
9. Morawiecki, P. and Srebrny, M., A SAT-based preimage analysis of reduced Keccak hash functions. *Infor. Process. Lett.* 113 (**10-11**), 392-397 (2013).
10. Morawiecki, P., Pieprzyk, J. and Srebrny, M., Rotational Cryptanalysis of Round-Reduced Keccak. FSE. Lecture Notes in Computer Science, Springer, 8424, 241-262 (2013).
11. Suryawanshi, S., Saha, D. and Sachan, S., New Results on the SymSum Distinguisher on Round-Reduced SHA3. In: Nitaj, A., Youssef, A. (Eds), Progress in Cryptology - AFRICACRYPT 2020. AFRICACRYPT 2020. Lecture Notes in Computer Science, Springer, Cham., 12174 (2020).
12. Saha, D., Kuila, S. and Chowdhury, D.R., Symsum: symmetric-sum distinguishers against round reduced SHA3. *IACR Trans. Symmetric Cryptol.*, **1**, 240-258 (2017).
13. Luo, P., Fei, Y., Fang, X., Ding, A.A., Leeser, M. and Kaeli, D.R., Power analysis attack on the hardware implementation of MAC-Keccak on FPGAs. *Proc. Inter. Conf. on ReConFigurable Computing and FPGAs*, 1-7 (2014).
14. Choi, H. and Seo, S.C., Fast implementation of SHA-3 in GPU Environment. *IEEE Access*, 9, 144574-144586 (2021).
15. Vergili, I. and Yucel, M.D., Avalanche and bit independence properties for the ensembles of randomly chosen n×x S-Boxes. *Turkish J. of Electrical Engng. and Computer Sciences*, 9, **2**, 137-145 (2008).
16. Biham, E. snd Shamir, A., *Differential Cryptanalysis of the Data Encryption Standard*. Springer New York (1993).
17. К.С. Сакан, Д.С. Дюсенбаев, К.Т. Алгазы, О.А. Лизунов, Хомпыш Ардабек, Разработка и анализ алгоритма хеширования *HAS01*. Сборник статей IV международной научно-технической конференции *Минские научные чтения-2021*. Minsk, **3**, 181-187 (2021) (in Russian).
18. Syamsuddin, I., Evaluation of NgeXTEA - a cryptography learning module. *Global J. Engng. Educ.*, 20, **3**, 196-200 (2018).

BIOGRAPHIES

Nursulu Kapalova received her Master's degree in mathematics from Al-Farabi Kazakh National University (KazNU), Almaty, Kazakhstan in 2002 and her Candidate of Technical Sciences degree (Almaty, Kazakhstan) in 2009. Currently, she is a leading researcher in the Information Security Laboratory at the Institute of Information and Computing Technology, Almaty, Kazakhstan, and an associate professor at the Department of Information Systems at KazNU. Her area of scientific work is development and research in the field of information protection.

Dilmukhanbet Dyusenbayev graduated from the Faculty of Mechanics and Mathematics of Al-Farabi Kazakh National University (KazNU), Almaty, Kazakhstan, with the specialty mathematics in 1994. Between 1994 and 1998, he worked as a researcher at the Research Institute of Informatics and Management, Almaty, Kazakhstan. Later, he taught mathematics at the Republican School of Physics and Mathematics. In 2007-2009, he studied computer science and computer engineering at Bauman Moscow State Technical University (MVTU, Moscow, Russia). For several years, he worked in the field of information in the state structure. Since 2015, he has been working as a researcher at the Scientific Institute of Information and Computing Technologies in Almaty, Kazakhstan. In addition, since 2019, he has been working as a senior lecturer at the Faculty of Mechanics and Mathematics at KazNU. His field of scientific research is information protection in the public and private sectors.

Kairat Sakan graduated from the Faculty of Mechanics and Mathematics of Al-Farabi Kazakh National University (KazNU), Almaty, Kazakhstan, majoring in mathematics and applied mathematics in 2001. In 2001-2002, he worked as a teacher at the Department of Applied Mathematics and Mathematical Modelling 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 protection in the state structure for several years. Since 2018, he has been working as a mathematician in the Information Protection Laboratory at the Scientific Institute of Information and Computing Technologies, Almaty, Kazakhstan. Currently, he is a doctoral student at KazNU, majoring in information security systems. The field of scientific research is information protection in the public and private sectors.