Scientific and Practical Cyber Security Journal (SPCSJ) 8(4): 9 – 16 ISSN 2587-4667 Scientific Cyber Security Association (SCSA)

STATISTICAL ANALYSIS OF THE HAS03 HASH FUNCTION BASED ON THE SPONGE STRUCTURE

[1]K.T. Algazy, [1]K.S. Sakan, [1]D.S. Dyusenbayev
e-mail: kunbolat@mail.ru, kairat_sks@mail.ru, dimash_dds@mail.ru
[1]Institute of Information and Computing Technologies, Almaty, Kazakhstan

ABSTRACT. This work focuses on the statistical analysis of the HAS03 hash function, which is based on the Sponge structure. The algorithm involves defining a fixed-length internal state and includes the absorption and squeezing phases. The paper presents the results of statistical testing of the cryptographic hash function. The analysis was performed using the software implementation of the statistical test suite recommended by the U.S. National Institute of Standards and Technology (NIST). Additionally, one of the key requirements for cryptographic hash algorithms is the presence of the avalanche effect. The results of the analysis confirm that the sequences produced by this hashing algorithm fully meet the criteria for the avalanche effect and exhibit a high level of statistical security.

KEYWORDS: *hash function, sponge, NIST test suite, cryptographic avalanche effect.*

INTRODUCTION

Hashing emerged in the early days of computer science and was initially used for efficient data retrieval. Over time, its applications have significantly expanded, especially in the field of cryptography, where hashing has become a fundamental component for ensuring data integrity and security in the digital age. A hash function is a mathematical algorithm that transforms an input message into a fixed-size string, typically in the form of a hash value. This transformation is deterministic, ensuring that a given input message consistently produces the same hash output, and it is one-way, preventing the original input from being derived from the hash value (Fashim P., 2021).

The primary purpose of hashing is information verification. This task is critical in numerous scenarios: from password verification on websites to complex computations in blockchain technology. Since a hash represents a unique code for a specific data set, it can be used to determine whether the information matches the expected value. Consequently, programs can store hashes instead of the original data for comparison purposes. This approach is particularly useful for protecting confidential information and conserving storage space. For example, instead of storing passwords on a server, their hashes are stored; antivirus programs keep hashes of viruses in their databases rather than the actual virus samples; digital signatures use hashes for verification; cryptocurrency transaction information is stored as hashes.

Other, less common uses of hashing include searching for duplicates in large datasets, generating identifiers (IDs), and constructing special data structures. An example of such a structure is a hash table, where an element’s identifier is its hash, which also determines the element's position in the table.

There are numerous methods for generating hashes. These can include formulas based on multiplication, division, and other mathematical operations, as well as algorithms of varying complexity (Chiambarasan N.R., 2021). However, if a hash is used for data protection, its function must be cryptographic, possessing certain properties.

The operation of a cryptographic hash function typically involves several stages. Data is divided into parts and passed through a compression function, which reduces the information to a smaller number of bits. Such a function must be cryptographically secure, meaning its output must be practically impossible to forge. The main properties of cryptographic hash functions include irreversibility, determinism, uniqueness, and diversity (Peiser S.C., 2020).

LITERATURE REVIEW

In the article (Rasool M., 2020), a novel approach is presented to address current security issues by using a generalized Collatz process to create a chaos-based hash function. By leveraging the unpredictable behavior of the Collatz sequence, the proposed hash function enhances ergodicity and entropy properties, making it highly suitable for cryptographic applications.

Hash functions are extensively used in measuring high-speed network traffic. In the work (Ying Hu, 2020), the authors introduce a practical development of hash functions for IPv6 measurement. This development is based on an entropic analysis of IPv6 network data and an automated multi-objective genetic programming (GP) method. Three fitness functions are used as optimization goals: active flow estimation, uniformity, and the avalanche effect, with active flow estimation being the primary objective for this specific measurement task.

Recent attacks on modern hash functions have raised doubts about the adequacy of standard hash function construction principles. The paper (Regenscheid A., 2007) examines a multiplication-based hash function construction in the group of 2 x 2 matrices over a finite field, proposed by Zemor and Tillich.

A hash chain is constructed by repeatedly hashing the initial value. The study (Lee D., 2007) investigates the complexity of compromising the security properties of hash functions through hash chain attacks using probabilistic algorithms. It is demonstrated that each hash function has a vulnerability index that measures its inherent susceptibility to hash chain attacks.

SHA-256 is a secure cryptographic hash function. Its output should not exhibit any detectable properties. The paper (Bouam M., 2021) describes three-bit strings whose hashes by SHA-256 are nonetheless correlated in a non-trivial way: the first half of their hashes XORs to zero. These were found using a brute force method without exploiting any cryptographic weaknesses in the hash function itself.

The works (Luo P., 2016) focus on the differential error analysis of the SHA-3 algorithm family, specifically the SHA3-224 and SHA3-256 algorithms. The authors developed an error injection model and enhanced the realism of the attack for various SHA-3 implementation architectures. Later, the authors extended the application of this developed approach to an attack based on error injection.

RESEARCH METHODOLOGY

A one-way function is a fundamental concept in cryptography and plays a crucial role in ensuring the security of hash functions. A one-way function is a function that is easy to compute in one direction but extremely difficult to invert without specific information. In the context of hash functions, this property ensures that hashing data can be done quickly and efficiently, but it is impossible to recover the original data from its hash (Levin A., 2003).

Let's provide a formal definition of a hash function. Let $\{0,1\}^m$ be the set of all binary strings of length m, and $\{0,1\}^*$ be the set of all finite-length binary strings. Then, a hash function h is a transformation of the form $h: \{0,1\}^* \to \{0,1\}^m$, where m is the length of the hash output.

There are various approaches to constructing cryptographic hash functions today. Among these, the sponge construction is a versatile cryptographic primitive used to create hash functions. The main idea behind it is the flexibility, security, and efficiency of the construction, making it suitable for various cryptographic tasks (Beirendonck M., 2019).

The sponge construction is characterized by two parameters:

- Absorption rate (r): The part of the state that is updated with each block of input data.
- Capacity (c): The part of the state that remains unchanged for each block of input data. The total length of the sponge state is $b=r+c$.

The avalanche effect is a key property of cryptographic algorithms that characterizes their resistance to analysis and cracking. This property denotes that a slight change in the input data (e.g., changing one bit) should cause significant changes in the output data, affecting a large number of bits. For hash functions, the avalanche effect plays a critically important role. Changing one bit in the original message results in a drastically different hash. This property ensures that two similar messages will have

completely different hashes, making it difficult to predict or match the original message from its hash (Upadhyay D., 2022).

The avalanche effect parameter is an important indicator of the quality of cryptographic algorithms. Its use allows evaluating and comparing the robustness of various algorithms against cryptanalysis, thereby providing a high degree of data protection. The avalanche effect parameter is defined by the following formula: $\varepsilon_i = |2k_i - 1|$, 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 ith in the input, and ε_i is the avalanche parameter. The ideal value for cryptographic hash algorithms is 0.5, indicating that, on average, changing one bit of input data alters half of the output bits.

RESULTS AND DISCUSSION

Development of a hashing algorithm

The hashing algorithm runs over a working range of 1024 bit (that is, the size of the internal state is 1024 bits). The original plaintext is divided into blocks of 512 bits. If the total plaintext length is not a multiple of 512, i.e. the length of the last block is less than 512 bits, then it is padded. As a complement, a bit sequence is used, the first and last positions of which are ones, and all the other positions are filled with zeros. The algorithm is based on sponge construction, the general scheme of which is shown in Figure 1. The internal state is divided into two parts. The developed algorithm uses an internal state consisting of two 512-bit parts A and V.

At the stage of absorption, a consecutive plaintext block A_i is combined with both parts of the internal state of the algorithm using the *XOR* operation and written instead of the first part, and the second part remains unchanged: $A'_i = A_i \oplus A_{i-1} \oplus V_i$, $V'_i = V_{i-1}$, $i = \overline{0, n-1}$. As the initial internal state, one can choose a sequence consisting only of zeros (in the general case, it is possible to choose any fixed sequence) with a length of 1024 bit. Using the function f, we transform the data obtained as a result of these operations. After all the plaintext blocks have been processed, we move on to the squeezing stage. At this stage, the input to the function f is the internal state $A_{n-1} \parallel V_{n-1}$ and the first 64 bits of the output are chosen as the hash value. This is repeated until a hash value of the required length is obtained. That is, to get a 256- or 512-bit hash value, one need to repeat this procedure 4 or 8 times, respectively.

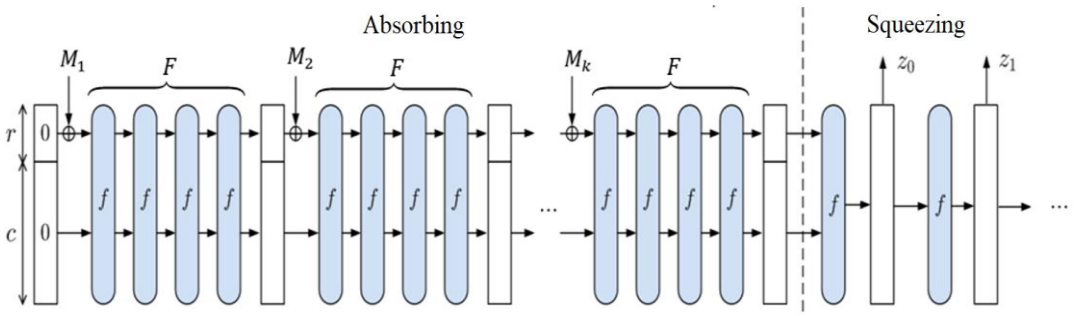

Figure 1. *Sponge construction scheme*

Function f. The structure of the internal state used by the function f is a 4 × 4 square matrix whose elements are 64-bit words (see Fig.2). W_i, C_i $i = \overline{0,7}$ are 64-bit words, W_i are plaintext data (A_j in the general scheme of the algorithm), C_i is the second part involved in the transformation (V_j in the general scheme).

W0	W1	W2	W3
W4	W5	W6	W7
C0	C1	C2	C3
C4	C5	C6	C7

Figure2. *The structure of the internal state used by the function f*

The function f consists of transformations X, S, R and P, where X is performed twice. Let's describe each of these transformations.

X-transformation performs the addition of the corresponding elements of the first and second columns of the matrix using the xor operation and places the result in the first column, i.e. $W'_0 = W_0 \oplus W_1$, $W'_4 = W_4 \oplus W_5$, $C'_0 = C_0 \oplus C_1$, $C'_4 = \mathrm{C}_4 \oplus \mathrm{C}_5$. The same transformation is used to get the elements of the second and third columns: $W'_1 = W_1 \oplus W_2$, $W'_5 = W_5 \oplus W_6$, $C'_1 = C_1 \oplus C_2$, $C'_5 = \mathrm{C}_5 \oplus \mathrm{C}_6$, $W'_2 = W_2 \oplus W_3$, $W'_6 = W_6 \oplus W_7$, $C'_2 = C_2 \oplus C_3$, $C'_6 = \mathrm{C}_6 \oplus \mathrm{C}_7$. The fourth column does not change.

S-transformation. Each of the values $W_i, C_i\ i = \overline{0,7}$ consists of 8 bytes, and their total number is 16. Then the total number of bytes processed by the function f is 128. As a result of the transformation S, these 128 bytes will be replaced by other bytes using S-box, i.e. S: $\{0,1\}^{1024} \to \{0,1\}^{1024}$.

After the S-transformation, the X-transformation is performed on the first and fourth columns, and the result is written as the new elements of the fourth column: $W''_3 = S(W'_0) \oplus S(W'_3)$, $W''_7 = S(W'_4) \oplus S(W'_7)$, $\mathrm{C}''_3 = S(\mathrm{C}'_0) \oplus S(\mathrm{C}'_3)$, $\mathrm{C}''_7 = S(\mathrm{C}'_4) \oplus S(\mathrm{C}'_7)$.

R-transformation. Let's denote each of the 16 internal state words as X_n, $n = \overline{0,15}$. Then, for $n =$ 0,1,2,3, the $SHRL(X_n)$ operation is performed, and in other cases, the $ROTRL(X_n)$ operation is performed. Here $SHRL(X_n)$ is a logical shift operation of the 64-bit argument by $n + 3$ bits to the right if n is even and to the left if n is odd. Similarly, $ROTRL(X_n)$ is an operation to rotate a 64-bit argument $n + 3$ to the right if n is even, and to the left if n is odd.

P-transformation. The main purpose of the transformation is to mix the input elements. Each byte of the input sequence is moved to a different location in a specific order (Table 1).

Table 1 *- Permutation table for P-transformation.*

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	16	32	48	64	80	96	112	1	17	33	49	65	81	97	113
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
2	18	34	50	66	82	98	114	3	19	35	51	67	83	99	115
32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
4	20	36	52	68	84	100	116	5	21	37	53	69	85	101	117
48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
6	22	38	54	70	86	102	118	7	23	39	55	71	87	103	119
64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
8	24	40	56	72	88	104	120	9	25	41	57	73	89	105	121
80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
10	26	42	58	74	90	106	122	11	27	43	59	75	91	107	123
96	**97**	**98**	**99**	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**
12	28	44	60	76	92	108	124	13	29	45	61	77	93	109	125
112	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**	**127**
14	30	46	62	78	94	110	126	15	31	47	63	79	95	111	127

The function f consists of repeating the above transformations 14 times, except that the R-transformation is not performed in the last round.

Let's show the results of the transformations used in the function f using a simple example. Let M be a binary sequence 512 bits long, consisting only of zeros. Processing begins with the absorption stage.

STUDY RESULTS

One of the universal attacks carried out on all cryptographic algorithms is exhaustive search or brute-force attack. If the hash value of the message M_1 is $H(M_1)$, then the attacker needs to find a message

M_2 that satisfies $H(M_1) = H(M_2)$. If the length of the resulting hash is n, then the complexity of this method is $O(2^n)$. Usually, when developing algorithms, special attention is paid to this condition and the algorithm is constructed in such a way that the required level of security is ensured. In the HAS03 algorithm, the length of the resulting hash value is 256 and 512 bits, so using the brute force method is inefficient.

Another universally valid attack technique used for hash functions is the "birthday paradox". This cryptographic analysis method estimates how many messages need to be examined to detect a collision with a probability greater than 0.5. To find a collision, one needs to generate two sets, each containing $2^{n/2}$ messages and calculate their hash values. According to the birthday paradox, among them there are messages with the same hash values with a probability of more than 0.5. However, this method requires a large amount of memory. Even if the hash length is 256, modern computers will not be able to implement this method. To find a collision of the second kind, it is necessary to make calculations in the amount of at least 2^{128}.

The avalanche effect is one of the mandatory properties of cryptographic algorithms. A small change in the input of block ciphers and cryptographic hash functions should significantly change the output value (for example, change half of the output bits).

To check the avalanche effect of the constructed hashing algorithm, consider the above example, i.e. as the source text, we choose a binary sequence consisting only of zeros (Zima V.M., 2000). By inverting one of the zero bits in each position of this sequence, we will get a new 512-bit text. The probability of changing k_i, $i = \overline{0,511}$, is calculated, where k_i is the probability that the ith bit in the output value will change if the ith element of the input value is inverted from the original output value (see Fig.3).

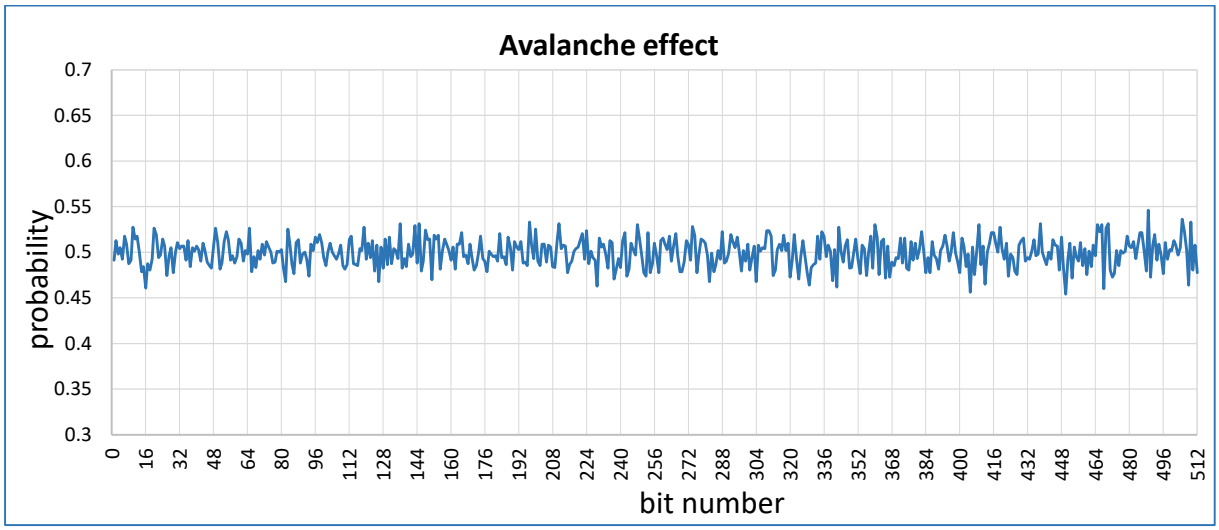

Figure 3. *Avalanche effect of the 6th round*

The closer the value of each k_i is to 0.5, the better the avalanche effect. Figure 4 shows a diagram of the improvement in the avalanche effect from round to round. In the sixth round, the algorithm shows a good result.

Evaluation of the statistical properties of hash values

For any hash value *h(M)* of a hashing algorithm, one of the essential conditions is the presence of properties of pseudorandom sequences.

Therefore, an evaluation of the HAS03 hashing algorithm was conducted using a set of statistical tests from the National Institute of Standards and Technology (NIST). The purpose of the testing is to determine the degree of deviation of the sequence of hash values from truly random sequences.

Each test in the NIST suite evaluates randomness according to specific criteria by calculating the p-value. Small p-values indicate that if the null hypothesis H_0 is true, the probability of obtaining the same or more extreme test statistics is very low. If the test yields a p-value $p > \alpha = 0{,}01$, it means that the examined sequence is random with a confidence level of 99%. Here, $p \in [0,1]$ and α is the significance level, i.e., the probability of rejecting the null hypothesis H_0.

The statistical tests using the NIST suite were conducted as follows. A random *.zip file of size 31,250 KB was selected for testing. According to the HAS03 algorithm description, each consecutive 64-byte block of the hashed message is processed, and the resulting 64-byte hash value is sequentially written to a new file. In our case, 500,000 hash values were obtained from 500,000 message blocks. The output file with the *.hash extension was split into 100 files, each 3,125 KB in size. These files were then tested using the NIST statistical test suite.

Figure 4 presents the results of the pseudo-randomness analysis of the hash value sequences generated by the HAS03 algorithm using NIST tests. Two significance levels, α, were considered, and the number of successfully passed NIST tests is shown.

The experiment results indicated that all tests were passed successfully. According to NIST recommendations, a test is considered successful if at least 96 out of 100 sequences pass.

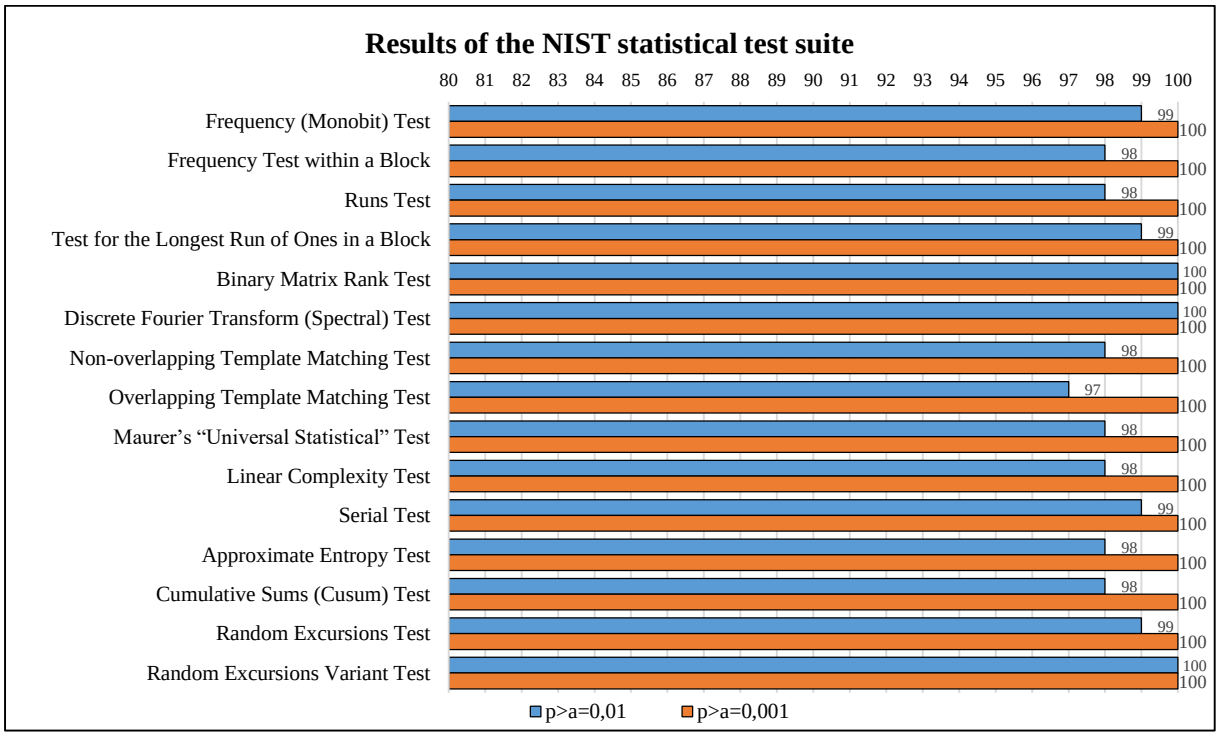

***Figure 4**. NIST statistical test results*

CONCLUSIONS

This article provides a brief overview of research on the development and analysis of cryptographic hash functions, as well as a study of the current state of security and known structures of these functions. The developed hashing algorithm, HAS03, is based on the sponge construction and transforms plaintext of arbitrary length, divided into 512-bit (64-byte) blocks, into a hash value of either 256 or 512 bits. Research has been conducted to confirm the cryptographic properties of the developed algorithm, including evaluations of the avalanche effect of the HAS03 hashing algorithm. The algorithm demonstrates a significant avalanche effect after just the 6th round of hashing. For experimental rigor, the avalanche criterion was applied to the analysis after the 8th, 10th, 12th, and 14th rounds of hashing, confirming the necessary degree of avalanche effect propagation in the HAS03 algorithm.

Additionally, the NIST testing results showed no deviations in the sequences generated by the HAS03 hashing algorithm. Therefore, it can be concluded that this algorithm provides a high level of statistical security.

A promising direction for future research is a comprehensive study of the HAS03 hashing algorithm in terms of its efficiency and security, specifically the search for collisions and preimages using cryptanalysis methods. This will help identify its vulnerabilities and develop corresponding protective measures.

ACKNOWLEDGMENTS

This work was supported by Shota Rustaveli National Science Foundation of Georgia (SRNSFG) - CG-24-220.
The research work was funded by the Ministry of Science and Higher Education of Kazakhstan and carried out within the framework of the project AP14870719 “Development and study of post-quantum cryptography algorithms based on hash functions” at the Institute of Information and Computational Technologies.

REFERENCES

Farshim, Pooya, and Stefano Tessaro. "Password Hashing and Preprocessing." In *Advances in Cryptology – EUROCRYPT 2021*, edited by Anne Canteaut and François-Xavier Standaert, vol. 12697 of *Lecture Notes in Computer Science*. Cham: Springer, 2021.

Chilambarasan, N. R., and A. Kangaiammal. "Matyas–Meyer–Oseas Skein Cryptographic Hash Blockchain-Based Secure Access Control for E-Learning in Cloud." In *Inventive Systems and Control*, edited by V. Suma, J. I. Z.

Chen, Zubair Baig, and Hui Wang, vol. 204 of *Lecture Notes in Networks and Systems*. Singapore: Springer, 2021. https://doi.org/10.1007/978-981-16-1395-1_65.

Peiser, S. C., L. Friborg, and R. Scandariato. "JavaScript Malware Detection Using Locality Sensitive Hashing." In *ICT Systems Security and Privacy Protection*, edited by Marko Hölbl, Kai Rannenberg, and Tomaž Welzer, vol. 580 of *IFIP Advances in Information and Communication Technology*. Cham: Springer, 2020. https://doi.org/10.1007/978-3-030-58201-2_10.

Rasool, M., and S. B. Belhaouari. "From Collatz Conjecture to Chaos and Hash Function." *Chaos, Solitons & Fractals* 176 (2023): 114103. https://doi.org/10.1016/j.chaos.2023.114103.

Hu, Ying, Guang Cheng, Yongning Tang, and Feng Wang. "A Practical Design of Hash Functions for IPv6 Using Multi-Objective Genetic Programming." *Computer Communications* 162 (2020): 160–68. https://doi.org/10.1016/j.comcom.2020.08.013.

Regenscheid, Andrew. "An Algebraic Hash Function Based on SL2." 2007. https://doi.org/10.31274/rtd-180813-15958.

Lee, D. "Hash Function Vulnerability Index and Hash Chain Attacks." In *2007 3rd IEEE Workshop on Secure Network Protocols*, 1–6. Beijing, China: IEEE, 2007. https://doi.org/10.1109/NPSEC.2007.4371616.

Bouam, M., Christophe Bouillaguet, C. Delaplace, and C. Nous. "Computational Records with Aging Hardware: Controlling Half the Output of SHA-256." *Parallel Computing* 106 (2021): 102804. https://doi.org/10.1016/j.parco.2021.102804.

Luo, P., Y. Fei, L. Zhang, and A. A. Ding. "Differential Fault Analysis of SHA3-224 and SHA3-256." In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 4–15. IEEE, 2016. https://doi.org/10.1109/FDTC.2016.17.

Levin, L. A. "The Tale of One-Way Functions." *Problems of Information Transmission* 39, no. 1 (2003): 92–103. https://doi.org/10.1023/A:1023634616182.

Beirendonck, M., L. Trudeau, P. Giard, and A. Balatsoukas-Stimming. "A Lyra2 FPGA Core for Lyra2REv2-Based Cryptocurrencies." In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. Sapporo, Japan: IEEE, 2019. https://doi.org/10.1109/ISCAS.2019.8702498.

Upadhyay, D., N. Gaikwad, M. Zaman, and S. Sampalli. "Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications." *IEEE Access* 10 (2022): 112472–86. https://doi.org/10.1109/ACCESS.2022.3215778.

Zima, V. M. *Security of Global Network Technologies*. BHV-Petersburg, 2000.