IMPLEMENTATION AND TESTING OF HASH FUNCTION BASED ON MODIFIED SKEIN ALGORITHM

UDC 004.021

DOI: https://doi.org/10.35546/2313-0687.2018.24.16-25

BARYBIN Oleksii
candidate of science (technic), Head of the Department of Radiophysics and Cybersecurity, Vasyl' Stus Donetsk National University, Vinnytsia.
E-mail: o.barybin@donnu.edu.ua. ORCID ID: 0000-0002-0897-4454.

TKACHENKO Vira
candidate of science (physics and mathematics), associate professor, associate professor of the department of general physics and didactics of physics, Donetsk national university of the name of Vasyl' Stus, Vinnitsa. **E-mail:** v.tkachenko@donnu.edu.ua. ORCID ID: 0000-0001-6064-5474.

ZHABSKA Yelyzaveta
master`s program student of specialty 122 Computer Science, Vasyl' Stus Donetsk National University, Vinnytsia.
E-mail: jabska.e@donnu.edu.ua. ORCID ID: 0000-0002-9917-3723.

PETROVA Iryna
master`s program student of specialty 122 Computer Science, Vasyl' Stus Donetsk National University, Vinnytsia.
E-mail: petrova.i@donnu.edu.ua. ORCID ID: 0000-0001-6870-7502.

Abstract. Context and objective. The article shows that the original Skein hash algorithm, which was developed as part of the competition for the new standard SHA-3 of the National Institute of Standards and Technology USA (NIST) and was one of five finalists can be modified to improve its operation efficiency in systems with multicore processors, which are used in almost all modern desktop and mobile devices. Analysis of the few publications that dedicated to such improvement of the original algorithm shows that all the authors focus on improving efficiency by optimizing the algorithm computations parallelization because computing speed was one of its main weaknesses. The simplification the original algorithm was not considered before and became the main purpose of the article.

Research methods. Analysis of the original algorithm showed that the basic premise for the modifications is flexibility for adjusting to build a hash function, particularly hash function is based on block encryption algorithm with adjustable block size. Block size and key are fixed, but they do not determine the size of the original hash function string, which complicates the method of attack, based on finding the key length. Alternating nature of the output line allows you to make hash function very flexible: it is possible to use a hash function and applications in devices with limited memory. Including mobile devices.

Results. Modification for Skein based on the fact that the original algorithm is assumed that the hash process can be represented as a multilevel tree structure. The basic idea is that the message on the first hashing level is divided into blocks, for each of the blocks calculated parameters for the second and higher levels become the input values for the next level. A significant simplification has been proposed no speed up the algorithm – use the only single-level hash that actually means abandoning the use of tree structure calculations.

Hash function based on the modified algorithm is implemented with the use of the programming language Python. Test results according to NIST SP 800-22 standard for set of 1,000 files with hash function results showed the percentage of files that have been tested successfully from 98.0 to 99.6% that indicates that the hash function under consideration meets random and pseudorandom generator integers and hash functions conditions.

The scientific novelty and practical significance. It is the first time then simplification of the structure for Skein hash algorithm calculations effectiveness improvement was proposed and it was shown that the hash function based on a simplified algorithm comply NIST SP 800-22.

Keywords: *hash functions, Skein, statistical methods of analysis of hash functions.*

Introduction.

Cryptographic hash function is an essential and common tool used to perform a variety of tasks: authentication, verification of data integrity, file protection, generation of associative arrays, finding duplication in a series of data sets, generation of unique identifiers for data sets etc. Hash function is a function that converts the arbitrary input data set to the output fixed size bit length string. The latter and simplicity of hash functions calculation are of their two main practical advantages. There are developed many hash algorithms with different properties (bit, computational complexity, cryptographical secure etc.). The best-known algorithms to obtain hash values are MD5, SHA, RIPEMD, TIGER [1].

In general, hash algorithms are such that it is almost impossible to find two messages with the same hash value. But it is impossible to completely avoid this, because it can results in collisions that denotes to forming the same strings as a result of hash functions with different input data [1]. In the early 2000s the cryptanalysts revealed the possibility of collision in such algorithms as MD4, MD5, and SHA-0 [2, 3]. Moreover, in is known the published method for finding collisions using SHA-1 algorithm.

Even though the new generation of standardized SHA-2 hash algorithms was ready to replace SHA-1, in late 2007 NIST decided to start a 4-year global process of developing a new hash algorithm, so called Standard SHA-3, which was developed through public competition [4]. In 2010 the 5 finalists were determined: BLAKE, Grostl, JH, Keccak and Skein. The results of a comparative analysis of the proposed algorithms can be found, for example, in [5, 6]. Keccak algorithm was chosen for the new standard and in [6] it is indicated that the choice was extremely difficult and each of the finalists had their advantages and disadvantages. Accordingly, each of the abovementioned algorithms can be used for further development and implementation.

Based on experts prediction [5] fourth generation hash algorithms is not expected until 2030, but the issue of implementation and testing of existing hash functions and their modifications is a topical research direction.

Skein is one of the promising hash algorithms and as mentioned earlier in 2010 was in the top five in the competition NIST [6]. The main feature of this algorithm is to obtain optimal performance when working with critical applications that require customized implementations efficiently on multicore processors. Given the fact that almost all modern computer systems are equipped with such processors the generation and study of the Skein algorithm modifications are promising.

Studies on the implementation of the Skein algorithm was carried out in most cases in the areas of its computation parallelization. Specifically, the authors [2] created a separate thread for each branch of calculations and showed that the effectiveness of the algorithm implementation in Java and C did not differ. The authors of [7] showed that it is possible to effectively calculate the hash value by parallelization of computations using a special tool «PLUTO». In general, in [5] it was indicated that in the implementation of this algorithm computation speed was one of its main weaknesses. As seen from the above works the authors did not consider the possibility of improving performance by simplifying the implementation of the original algorithm.

The objective of the study.

Consider all the above mentioned, the objective of the article is to propose, implement and verify modification for Skein algorithm. To achieve the objective following scientific and technological tasks must be settled:

- proposals generation for Skein algorithm modifications, which should be based on consideration of the principles of its construction;

- software implementation for hash function based on the modified algorithm;

- testing the implemented hash function.

Modified Skein hash function algorithm.

The general structure of the Skein algorithm. Principles of this algorithm, its structure, analysis of its cryptographic security are described in detail in [5, 6, 8-11], so let's consider only general information.

Skein main advantage is that it provides an opportunity to build a hash function with parameters that can be adjusted. Hash function is based on block encryption algorithm with adjustable block size. Block size and key are fixed, but they do not determine the size of the hash function output string, which complicates the attack methods based on finding the key length. Alternating the output string allows you to make hash function very flexible: it is possible to use a hash function in devices with limited memory, including mobile devices.

Skein has three components [8]:

- Threefish. Threefish is the tweakable block cipher at the core of Skein, defined with a 256-,512-, and 1024-bit block size.
- Unique Block Iteration (UBI). UBI is a chaining mode that uses Threefish to build a compression function that maps an arbitrary input size to a fixed output size.
- Optional Argument System. This allows Skein to support a variety of optional features without imposing any overhead on implementations and applications that do not use the features. An extra set of arguments allows Skein maintain a set of additional features that allow you to use not only as Skein hash function (such as encryption algorithm) without significant changes in the source code implementation.

UBI and Threefish are independent and can be used separately.

When used as a hash function, the message type is the only optional input type used. The output of configuration UBI becomes a precomputed initial chaining value. This is the simplest use of Skein. With the variable output size it becomes a drop-in replacement for almost any existing hash function.

Algorithm modification. To simplify the original algorithm, consider some functions in more detail.

Tweak function is presented as a description of individual components-bytes in original algorithm. It was designed as an auxiliary one for the values generation, which is formed as a unique unit for each 128-bit string. Tweak Fields indicated in Fig. 1 [5], and their description specified in Table 1.

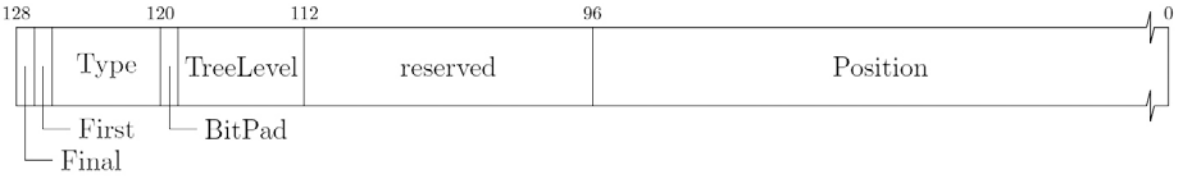

Figure 1 – The fields in the Tweak value

Tweak function takes as input the original message, hash tree level, type of Tweak and value lists for the respective blocks which are converted into binary form. Based on this input parameters a string to be form in the manner specified in Fig. 1. The following string is converted to decimal form. The function returns the number in decimal form.

In the original algorithm it is assumed that the hashing process can be represented as a multilevel tree structure. The basic idea is that the first level hash message is divided into blocks, each block is calculated from UBI, on the second and higher levels of UBI the values are used as input parameters. Block size and number of required levels specified in Tweak and Configuration_string function, which is described below.

Implementers of tree hashing have a number of decisions to make. There are three parameters to choose: the leaf node

size, the fan-out, and the maximum tree height. For efficiency, a larger leaf node size and fan-out is better; it reduces the number of nodes and thus the overhead. But large leaf nodes and high fan-out make some uses less efficient.

To speed up the algorithm a significant simplification has been proposed – use only single-level hashing that actually means abandoning the use of tree structure calculations.

Table 1

The fields in the Tweak value

Name	Position	Description
Position	0- 95 bits	The number of bytes in the string processed so far (including this block)
Reserved	96-111 bits	Reserved for future use, must be zero
TreeLevel	112-118 bits	Level in the hash tree, zero for non-tree computations
BitPad	119 bit	Set if this block contains the last byte of an input whose length was not an integral number of bytes. 0 otherwise.
Type	120-125 bits	Type of Tweak (key, block configuration, messages, etc.)
First	126 bit	Set for the first block of a UBI compression
Final	127 bit	Set for the last block of a UBI compression

Configuration_string function in original algorithm is presented as a description of individual components-bytes. It was designed as an auxiliary for the formation of values string configuration. According to the proposed modification of certain bytes are assigned fixed value (see last column of Table. 2).

Таблиця 2

The Fields in the configuration value

Offset	The number of bytes	Name	Description	Description modifications
0	4	Schema identi fi er	Constant	-
4	2	Version number	-	-
6	2	Reserved	Reserved bytes filled with zeros	-
8	8	Output length	Converted to bytes additional parameter ToBytes, defines the desired length of hash sum	-
16	1	Tree leaf size enc.	Size of tree leaf	zero
17	1	Tree fan-out enc.	Tree branching	zero
18	1	Max. tree height	The maximum level of the tree	zero
19	13	Reserved	Reserved bytes filled with zeros	-

An implementer that needs the hash function to process data at a very high data rate can use a leaf node size of a few kilobytes and a maximum tree height of 2. This allows multiple processors to each work on its own leaf node, with one processor doing the second level of the tree. Increasing the leaf node size makes this more efficient, but it increases the amount of memory needed for buffering, and will tend to increase latency.

Limiting the tree height is useful when memory-limited devices are involved. When computing a tree hash incrementally, the implementation must store data for each level of the tree. Limiting the tree height allows a fixed allocation of memory for small devices.

Thus, modifications of the algorithm consist of two simplifications:

- abandoning the use of multi-level hashing,

- several bytes in Tweak and Configuration_string have fixed values and do not depend on the size of the incoming message.

Software implementation of hash function based on the modified algorithm. The above-mentioned changes are implemented using software programming language Python. Graphical interface was created that allowed to choose a text file with message, for which the hash function value was calculated. Hash result is displayed in a separate GUI field and recorded in a separate file.

Preliminary analysis of hash function execution indicates that changing even one character in the initial report led to changes in the value of the final sequence, which is one of the requirements for hash functions in general.

Complete performance testing for modified algorithm has at least two stages:

1. Compliance Testing for statistical input and output data applicable to cryptographic hash functions. This must be done because simplification could lead to malfunction of the algorithm.

2. Implementation effectiveness testing. In particular, calculation speed comparison with known counterparts. This phase will be conducted in a separate study, because it is time-consuming, as can be seen, for example, in [12, 13].

Compliance Testing for statistical input and output data.

Hash function assessment methodology. The NIST STS 800-22 tests package. The basis for the software implementation of statistical tests in this work is NIST SP 800-22 «A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications» [14].

NIST SP 800-22 highlights aspects of selection and testing of random and pseudorandom generators for integers and hash functions. It was suggested during the competition for the new US national standard block encryption which was used for the analysis of statistical properties of the candidates for the new block cipher. Test suite contains 15 statistical tests that are designed to test the hypothesis of randomness for binary sequences of any length. All tests are focused on detecting various defects of randomness.

Files from the NIST SP 800-22 with a programming code can be found at NIST web-site. This code is written in the programming language C and in [15] observed its following imperfections:

1. Incorrect implementation of the statistical test suite. Because test suit was compiled during testing without any revisions in the source code and was tested on reference sequences that were delivered with the test, it does not guarantee the absence of errors and impropriety in the code. During testing process there were error messages associated with the release of calculated values beyond significance limit. Although these errors were observed only in some samples (usually low accuracy calculations), they could in some way influence the overall test results.

2. Incorrect programming code implementation. Although the authors of the studies were carried fullest extent to study the reliability and reproducibility of research results relevant for code, it does not exclude errors in the code (for example, relating to incorrect alignment of data types) that lead to distortion and deterioration for test results.

3. Imperfection of statistical methods. Statistical methods imperfection include the preposition that sequence under analyzes considered to be the general sample from which the user must select a partial sample. In some cases, partial sample properties differ from those of a general sample.

4. The use of tests for hash functions has its own specific. Tests that have been developed are used for binary sequences, while the majority of hash functions calculate hash in hexadecimal, octal and other numerical systems. Changing the number system can affect statistics for input sequence.

Because of the above-mentioned drawbacks the implementation of selected tests in the programming language Python became in this work a separate task and among the 15 tests proposed by NIST 8 most often used test were chosen for the implementation to study hash function under consideration. Thus, in this paper were implemented the following tests:

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. Tests for the Longest-Run-of-Ones in a Block,
4. The Linear Complexity Test,
5. The Serial Test,
6. The Approximate Entropy Test,
7. The Cumulative Sums Test,
8. The Cumulative Cusums Test.

For some tests implemented in the suite, it was assumed that the size of the sequence length n is large (on the order of 10^3 to 10^7). For such a large sample were ap-

plied asymptotic methods. Most tests can be used for smaller values of *n*. However, when applying for smaller values of *n* asymptotic methods would be unacceptable and they should be replaced exact distributions that are usually difficult to calculate.

Hash function test results. To assess the quality Skein-based hash functions a sample of 1000 files containing the results of hashing was tested.

Test binary matrices test and Maurer statistical test were not used for testing, since the minimum length of input sequences for these tests to be 38,912 bits and 387840 bits, respectively, and estimation of the time required to execute these tests showed significant duration.

Test results indicate the almost complete absence of deviation for sequences which were generated by simplified version of Skein hash function.

Table 3

Hash function testing results

Type of test	The percentage of files that have passed the test
The Frequency (Monobit) Test	98,6%
Frequency Test within a Block	99,6%
Tests for the Longest-Run-of-Ones in a Block	98,9%
The Linear Complexity Test	98,4%
The Serial Test	98,1%
The Approximate Entropy Test	98,0%
The Cumulative Sums Test	99,4%
The Cumulative Cusums Test	99,3%

Conclusions.

Further development of algorithms for hash functions associates with both the generation of new and improvement of existing algorithms.

It is shown that the Skein algorithm is one of the algorithms on which basis high quality hash functions can be build, and further development of the implementation feasibility for this algorithm lay in simplification of some features that make up the structure of the original algorithm. In particular, the following simplifications were proposed in this work:

- abandoning the use of multi-level hashing,

- several bytes in Tweak and Configuration_string have fixed values and do not depend on the size of the incoming message.

The first phase of implemented hash function testing was performed by analyzing statistical input and output data applicable to cryptographic hash functions. In particular NIST SP 800-22 «A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications» from NIST was used. To elude some shortcomings of program implementation for these tests proposed by NIST in the programming language C in this work software implementation has been established using language Python.

The testing results for set of 1,000 files with hash function results showed the percentage of files that have been tested successfully from 98.0 to 99.6% that indicates that the hash function under consideration meets random and pseudorandom generator integers and hash functions conditions.

REFERENCES:

1. Keith, M. M. (2017). Everyday Cryptography. Fundamental Principles and Applications. *Croydon : Oxford University Press*, 773.
2. Atighehchi, K., Enache, A., Muntean, T., Risterucci, G. (2010). An Efficient Parallel Algorithm For Skein Hash Functions. *Innsbruck, Austria: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, 1-11.
3. Bahi, J. M., Couchot, J. F., Guyeux, C. (2012). Quality Analysis of a Chaotic Proven Keyed Hash Function. *International Journal On Advances in Internet Technology*, 5(1), 26-33.

4. Hadedy, M. El., Gligoroski, D., Knapskog, S. J., Margala, M. (2010). Compact Implementation of BLUE MIDNIGHT WISH-256 Hash Function on Xilinx FPGA Platform. *Journal of Information Assurance and Security*, 5, 626-636.
5. Al-shaikhli, I. F., Alahmad, M. A., Munthir, K. (2013). Hash Function of Finalist SHA-3 : Analysis Study. *International Journal of Advanced Computer Science and Information Technology (IJACSIT)*, 2(2), 37-48.
6. Chang, S., Perlner, R., Burr, W. E., Turan, M. S., Kelsey, J. M., Paul, S., Bassham, L. E. (2012). Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition. *NIST*, 84.
7. Shivaprasad, S., Ramaswamy, A., Raviprasad, R.T., Sadanandam, M. (2016). Optimization of Skein Hash Function Using Pluto Tool. *International Journal of Applied Engineering Research.* 11(5), 3624-3631.
8. .Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J. The Skein Hash Function Family. Retrieved from http://www.skein-hash.info/sites/default/files/skein1.3.pdf.
9. Aumasson, J.-P., Çalık, Ç., Meier, W., Özen, O., Phan, R. C.-W., Varıcı, K. (2009). Improved Cryptanalysis of Skein. *Tokyo, Japan: International Conference on the Theory and Application of Cryptology and Information Security ASIACRYPT,* 542-559.
10. Goldhamer, J. (2010). SHA-3 Submission: Skein Hash Function Family. Insight on skein hash function proposal. Retrieved from https://koclab.cs.ucsb.edu/teaching/cren/project/2010/goldhamer.pdf.
11. Bellare, M., Kohno, T., Lucks, S., Ferguson, N., Schneier, B., Whiting, D., Callas, J., Walker, J. Provable Security Support for the Skein Hash Family. Retrieved from http://www.skein-hash.info/sites/default/files/skein-proofs.pdf.
12. Mouha, N., Raunak, M.S., Kuhn, D.R., Kacker, R. (2018). Finding Bugs in Cryptographic Hash Function Implementations. *IEEE Transactions on Reliability*, 99, 1-15.
13. Wang, D., Jiang, Y., Song, H., He, F., Gu, M., Sun, J. (2017). Verification of Implementations of Cryptographic Hash Functions. *IEEE Access*, 5, 7816-7825.
14. (2010). Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *Special Publication 800-22 Revision 1a, NIST*.
15. Antonov, A.V., Bzot, V. B., (2013) Variant jeffektivnoj realizacii metoda postroenija hesh-funkcij na osnove haoticheskih otobrazhenij s peremennymi parametrami i parallel'noj organizaciej vychislenij. *Sistemi obrobki informacii*, 1, 187-191.

БАРИБІН Олексій Ігорович
кандидат технічних наук, завідувач кафедри радіофізики та кібербезпеки, Донецький національний університет імені Василя Стуса, Вінниця. E-mail: o.barybin@donnu.edu.ua. ORCID ID: 0000-0002-0897-4454.

ТКАЧЕНКО Віра Сергіївна
кандидат фізико-математичних наук, доцент, доцент кафедри загальної та дидактики фізики, Донецький національний університет імені Василя Стуса, Вінниця. E-mail: v.tkachenko@donnu.edu.ua. ORCID ID: 0000-0001-6064-5474

ЖАБСЬКА Єлизавета Олегівна
студентка СО «Магістр» спеціальності 122 Комп'ютерні науки, Донецький національний університет імені Василя Стуса, Вінниця. E-mail: jabska.e@donnu.edu.ua. ORCID ID: 0000-0002-9917-3723.

ПЕТРОВА Ірина Ігорівна
студентка СО «Магістр» спеціальності 122 Комп'ютерні науки, Донецький національний університет імені Василя Стуса, Вінниця. E-mail: petrova.i@donnu.edu.ua. ORCID ID: 0000-0001-6870-7502.

РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ГЕШ-ФУНКЦІЇ НА ОСНОВІ МОДИФІКОВАНОГО АЛГОРИТМУ SKEIN

Анотація. Актуальність та мета статті. У статті показано, що оригінальний алгоритм гешування Skein, який був розроблений в рамках конкурсу на новий стандарт SHA-3 Національного інституту стандартів і технологій США (NIST) і був одним з п'яти фіналістів може бути модифіковано для підвищення ефективності його роботи в системах з багатоядерними процесорами, якими комплектуються практично всі сучасні настільні та мобільні пристрої. Аналіз нечисленних публікацій, які присвячені розвитку можливостей оригінального алгоритму, показує, що всі автори зосереджують увагу на підвищенні ефективності виконання алгоритму шляхом оптимізації розпаралелювання обчислень, які скла-

дають структуру алгоритму. При цьому саме швидкість обчислень при імплементації цього алгоритму була однією з основних його недоліків. Спрощення оригінального алгоритму раніше не розглядалась, що стало метою статті.

Методи дослідження. Аналіз оригінального алгоритму виявив, що базовою передумовою запропонованої модифікації є те, що він дає можливість побудувати геш-функцію з параметрами, які можна налаштовувати. Геш-функція будується на основі блокового алгоритму шифрування з регульованим розміром блоку. Розміри блоку та ключа є фіксованими, але вони не визначають розмір вихідного рядка геш-функції, що ускладнює методи атак, які ґрунтуються на знаходженні довжини ключа. Змінний розмір вихідного рядка дозволяє зробити геш-функцію дуже гнучкою: з'являється можливість використовувати геш-функцію у додатках та пристроях з обмеженим обсягом пам'яті. Зокрема на мобільних пристроях.

Результати. Запропоновано модифікацію Skein засновану на тому, що в оригінальному алгоритмі передбачається, що процес гешування може бути представлений у вигляді багаторівневої деревоподібної структури. Основна ідея полягає у тому, що на першому рівні гешування повідомлення розбивається на блоки, для кожного з блоків обчислюється відповідні параметри, на другому та вищих рівнях вже отримані значення цих параметрів використовуються у якості вхідних. Для пришвидшення роботи алгоритму було запропоноване суттєве спрощення – використання лише однорівневого гешування, що фактично означає відмову від застосування деревоподібної структури обчислень.

Геш-функцю, яка заснована на модифікованому алгоритмі, реалізовано із використання мови програмування Python. Результати тестування відповідно до стандарту NIST SP 800-22 на виборці, яка складалася з 1000 файлів з результатами гешування, показали відсоток файлів, які пройшли тестування, від 98,0 до 99,6%, що свідчить про те, що реалізована геш-функція на основі модифікованого алгоритма Skein відповідає вимогам до випадкових і псевдовипадкових генераторів цілих чисел та геш-функцій.

Наукова новизна та практична значимість. Вперше запропоновано пришвидшити ефективність імплементації алгоритму гешування Skein шляхом спрощення структури обчислень та показано, що геш-функція заснована на спрощеному алгоритмі відповідає стандарту NIST SP 800-22.

Ключові слова: *геш-функції, Skein, статистичні методи аналізу геш-функцій.*

БАРЫБИН Алексей Игоревич

кандидат технических наук, заведующий кафедрой радиофизики и кибербезопасности, Донецкий национальный университет имени Васыля Стуса, Винница. E-mail: o.barybin@donnu.edu.ua. ORCID ID: 0000-0002-0897-4454.

ТКАЧЕНКО Вера Сергеевна

кандидат физико-математических наук, доцент, доцент кафедры общей и дидактики физики, Донецкий национальный университет имени Васыля Стуса, Винница. E-mail: v.tkachenko@donnu.edu.ua. ORCID ID: 0000-0001-6064-5474

ЖАБСКАЯ Елизавета Олеговна

студентка СО «Магистр» специальности 122 Компьютерные науки, Донецкий национальный университет имени Васыля Стуса, Винница. E-mail: jabska.e@donnu.edu.ua. ORCID ID: 0000-0002-9917-3723.

ПЕТРОВА Ирина Игоревна

студентка СО «Магистр» специальности 122 Компьютерные науки, Донецкий национальный университет имени Василия Стуса, Винница. E-mail: petrova.i@donnu.edu.ua. ORCID ID: 0000-0001-6870-7502.

РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ХЭШ-ФУНКЦИИ НА ОСНОВЕ МОДИФИЦИРОВАННОГО АЛГОРИТМА SKEIN

Аннотация. Актуальность и цель статьи. В статье показано, что оригинальный алгоритм хеширования Skein, который был разработан в рамках конкурса на новый стандарт SHA-3 Национального института стандартов и технологий США (NIST) и был одним из пяти финалистов может быть модифицирован для повышения эффективности

его работы в системах с многоядерными процессорами, которыми комплектуются практически все современные настольные и мобильные устройства. Анализ немногочисленных публикаций, посвященных развитию возможностей оригинального алгоритма, показывает, что все авторы сосредоточены на повышении эффективности выполнения алгоритма путем оптимизации распараллеливания вычислений, которые составляют структуру алгоритма. При этом именно скорость вычислений при имплементации этого алгоритма была одной из основных его недостатков. Упрощение оригинального алгоритма ранее не рассматривалась, что стало целью статьи.

Методы исследования. Анализ оригинального алгоритма выявил, что базовой предпосылкой предложенной модификации является то, что он дает возможность построить хеш-функцию с параметрами, которые можно настраивать. Хеш-функция строится на основе блочного алгоритма шифрования с регулируемым размером блока. Размеры блока и ключа являются фиксированными, но они не определяют размер исходной строки хеш-функции, что усложняет методы атак, основанных на нахождении длины ключа. Переменный размер исходной строки позволяет сделать хеш-функцию очень гибкой: появляется возможность использовать хеш-функцию в приложениях и устройствах с ограниченным объемом памяти. В частности, на мобильных устройствах.

Результаты. Предложена модификация Skein основанная на том, что в оригинальном алгоритме процесс хеширования может быть представлен в виде многоуровневой древовидной структуры. Основная идея заключается в том, что на первом уровне хеширования сообщение разбивается на блоки, для каждого из блоков исчисляется соответствующие параметры, на втором и высших уровнях уже полученные значения этих параметров используются в качестве входных. Для ускорения работы алгоритма было предложено существенное упрощение – использование только одноуровневого хеширования, что фактически означает отказ от применения древовидной структуры вычислений.

Хеш-функция, которая основана на модифицированном алгоритме, была реализована с использованием языка программирования Python. Результаты тестирования в соответствии со стандартом NIST SP 800-22 на выборке, состоящей из 1000 файлов с результатами хеширования, показали процент файлов, которые прошли тестирование, от 98,0 до 99,6%, что свидетельствует о том, что реализованная хеш-функция на основе модифицированного алгоритма Skein соответствует требованиям к случайным и псевдослучайным генераторам чисел и хеш-функциям.

Научная новизна и практическая значимость. Впервые предложено ускорить эффективность имплементации алгоритма хеширования Skein путем упрощения структуры вычислений и показано, что хеш-функция основана на упрощенном алгоритме соответствует стандарту NIST SP 800-22.

Ключевые слова: *хеш-функции, Skein, статистические методы анализа хеш-функций.*

ЛІТЕРАТУРА:

1. Keith M. M., Everyday Cryptography. Fundamental Principles and Applications: Second edition. Croydon : Oxford University Press, 2017.773 p.
2. Atighehchi K., Enache A., Muntean T., Risterucci G. An Efficient Parallel Algorithm For Skein Hash Functions. Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems. Innsbruck, Austria, 2010. – P. 1-11.
3. Bahi J. M., Couchot J.-F., Guyeux C. Quality Analysis of a Chaotic Proven Keyed Hash Function. International Journal On Advances in Internet Technology. 2012. Vol. 5. № 1. P.26-33.
4. Hadedy M. El., Gligoroski D., Knapskog S. J., Margala M. Compact Implementation of BLUE MIDNIGHT WISH-256 Hash Function on Xilinx FPGA Platform. Journal of Information Assurance and Security. 2010. Vol. 5. P. 626-636.
5. Al-shaikhli I. F., Alahmad M. A., Munthir K. Hash Function of Finalist SHA-3 : Analysis Study. International Journal of Advanced Computer Science and Information Technology (IJACSIT). 2013. Vol. 2, № 2. P. 37-48.
6. Chang S., Perlner R., Burr W. E., Turan M. S., Kelsey J. M., Paul S., Bassham L. E. Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition. NIST. 2012. 84 p.
7. Shivaprasad S., Ramaswamy A., Raviprasad R.T., Sadanandam M. Optimization of Skein Hash Function Using Pluto Tool. International Journal of Applied Engineering Research. 2016. Vol. 11, № 5. P. 3624-3631.
8. Ferguson N., Lucks S., Schneier B., Whiting D., Bellare M., Kohno T., Callas J., Walker J. The Skein Hash Function Family. URL: http://www.skein-hash.info/sites/default/files/skein1.3.pdf.

9. Aumasson J.-P., Çalık Ç., Meier W., Özen O., Phan R. C.-W., Varıcı K. Improved Cryptanalysis of Skein. International Conference on the Theory and Application of Cryptology and Information Security ASIACRYPT. Tokyo, Japan 2009. P. 542-559.
10. Goldhamer J. SHA-3 Submission: Skein Hash Function Family. Insight on skein hash function proposal. 2010. P. 1-3. URL: https://koclab.cs.ucsb.edu/teaching/cren/project/2010/goldhamer.pdf.
11. Bellare M., Kohno T., Lucks S., Ferguson N., Schneier B., Whiting D., Callas J., Walker J. Provable Security Support for the Skein Hash Family Version 1.0. URL: http://www.skein-hash.info/sites/default/files/skein-proofs.pdf.
12. Mouha N., Raunak M. S., Kuhn D. R., Kacker R. Finding Bugs in Cryptographic Hash Function Implementations. IEEE Transactions on Reliability. 2018. Vol. 99. P. 1-15.
13. Wang D., Jiang Y., Song H., He F., Gu M., Sun J. Verification of Implementations of Cryptographic Hash Functions. IEEE Access. 2017. Vol. 5. P. 7816-7825.
14. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Special Publication 800-22 Revision 1a, NIST. 2010.
15. Antonov A. V., Bzot V. B. Variant jeffektivnoj realizacii metoda postroenija hesh-funkcij na osnove haoticheskih otobrazhenij s peremennymi parametrami i parallel'noj organizaciej vychislenij. Sistemi obrobki informacïï. 2013. Vip. 1. S. 187-191.