Encryption Methods and Algorithms Based on Domestic Standards in Open-Source Operating Systems

MADJIT MALIKOVICH KARIMOV[1], NIZOMIDDIN NAJMIDDIN UGLI OCHILOV[1],
ABDIQAHHAR EGAMOVICH TANGIROV[2]
[1]State Testing Center under the Cabinet of Ministers Republic of Uzbekistan,
12 Bagishamol St., Yunusabda paradise,100084, Tashkent,
REPUBLIC OF UZBEKISTAN
[2]Department of "Mathematics and Informatics", Tashkent Institute of Textile and Light Industry,
5 Shokhjakhon St., Yakkasaray paradise, Tashkent,
REPUBLIC OF UZBEKISTAN

Abstract: - The paper describes the principles and methods underlying the creation of an application in secure operating systems, which provides reliable data encryption. The research aims to analyze and indicate the specifics of encryption methods and algorithms based on domestic standards in open-source operating systems. Cryptanalysis was used in the article, as this avoids vulnerabilities identified in previously created implementations. In the article, the authors draw attention to the fact that 7-Zip uses CBC encryption (concatenation of encrypted text blocks), but the Counter Mode is supported. The same support was provided in the encrypt implementation. Since the key expansion function initially fills the special array created by p7zip with round keys using a unique property of the domestic standard, only one round encryption function was created (performed both during encryption and decryption). This method is also used in various modes. In many cases, initialization time deviations depending on the selected mode are insignificant. The created cryptographic module was tested to meet the domestic standard, which contains several test cases. It was confirmed during the tests that the created module implements the algorithm of the domestic standard. The article shows a way to implement a fairly convenient graphical interface for accessing the cryptographic module, which enables the user not to call the command line and remember the sequence and types of parameters passed to p7zip. This implementation also takes into account the verification of the correctness of decryption and the reading of other error codes.

Key-Words: - lossless compression, cryptanalysis, concatenation of ciphertext blocks, initialization vector.

Received: April 22, 2022. Revised: December 13, 2022. Accepted: January 22, 2023. Published: February 20, 2023.

1 Introduction

The open-source operating system ensures the protection of processed and stored data through Security Policies, kernel settings, and additional software. One of the requirements is to support encryption in accordance with the law. It was required to create an application that performs encryption following the domestic standard, the only encryption algorithm allowed for use when working with information containing state secrets, [1], [2].

However, the lack of open-source software available in the public domain that performed the assigned tasks, while having sufficient performance, with open source, necessitated the development of a cryptographic module.

2 Analysis of Research on Encryption Methods and Algorithms in Open-Source Operating Systems

The main problem in creating a cryptographic application is to provide all verification procedures — splitting the data stream into blocks, padding to the desired length, and creating a pseudo-random sequence initializing vector (IV). As a result, it was decided to use ready-made software, in which these procedures were successfully implemented, [3], [4], [5], [6], [7].

For block ciphers, the use of IV is described by modes of operation. Randomization is also required for other primitives such as generic hash functions and derived message authentication codes. An initialization vector (IV) is introduced in CBC, CFB, and OFB encryption. Moreover, both the sender and the receiver must have the same IV at the

beginning of the communication session. The IV does not have to be secret at all and can be transmitted along with the first ciphertext block. What's important is that this value should be unpredictable in CBC and CFB modes and unique in OFB mode. Unpredictability in CBC and CFB modes can be achieved in several ways. For example, the value of a counter (say a message counter) can be converted using the same function. GPC can also be used to generate a pseudo-random sequence of the desired length.

As encryption in everyday work is more often applied to text information, especially to documents in DOC, and DOCX format, the presence of standard expected sections in them creates a danger in the reliability of the password used for encryption. To eliminate this effect, it is recommended to pre-compress the information, thus reducing data redundancy. For this reason, the archiver was chosen as the main application, [8], [9].

The encryption algorithm in WinZip versions earlier than 9 is not strong enough with the current computing power, but it also has a 64-bit key size. In this case, it is necessary to correct not only the encryption module, but also key extensions, check it for statistical properties, and so on. Moreover, the encryption block is one byte, [10], [11].

The most popular archiver WinRAR is proprietary, which means that it does not have open source, thereby automatically excluding this software from consideration.

7-Zip is a free, highly compressed file archiver. It supports multiple compression algorithms and multiple data formats, including the proprietary 7z format with the highly efficient LZMA (Lossless Compression Algorithm) compression algorithm, [8]. The encryption algorithm used is AES with a block size of 128 bits, and a key size of 256 bits. As a result, the p7zip version, developed for the Linux OS family, was chosen (Table 1).

Table 1. Comparative table of archiver programs

	7zip	ACE	WinZip	RAR	WinRAR	Squeeze	UHARC
Compressing directories	+	+	+	+	+	+	+
Creation of self-extracting (SFX) archives	+	+	+	+	+	+	-
Changing the contents of the archive	+	+	+	+	+	+	+
Encryption mode	+	+	+	+	+	+	+
Archive recovery	-	+	+	+	+	+	+
Breaking the archive into parts	+	+	+	+	+	+	-
Console version	+	+	+	+	+	+	+
Graphic version (GUI)	+	+	+	+	+	+	+
Asymmetry	+	+	+	+	+	+	-
RAM requirements, MB	5	14	8	8	21	8	21
Available	+	-	+	-	-	-	+

3 Overview of Encryption Methods and Algorithms in Open-Source Operating Systems

Modifications and cryptanalysis. As the domestic standard has the same key size as AES-256, it was decided not to change the procedure for creating a key from the password sequence. For this purpose, p7zip repeatedly uses the SHA-2 algorithm, which is used because of rather good statistics, [6], as a pseudo-random sequence generator.

However, the procedure for expanding the key itself is strikingly different for AES and the domestic standard, so the entire module located in the AES.c file was changed to the encrytp.c module. 7-Zip itself performs text splitting into blocks of 128 bits and padding to the desired length according to the rules, [12], [13]. The encrytp.c implementation resizes the block by 64 bits. As 128 is a multiple of 64, this transformation is done by simply changing the constants, as well as doubling the number of blocks.

7-Zip uses CBC mode encryption (ciphertext block concatenation mode), but the Counter mode is supported. The same support was provided in the encrytp.c implementation. As the key expansion

function initially fills a special array created by p7zip with round keys, using a unique property of the domestic standard, only one round encryption function was created (performed both during encryption and decryption).
This method is also applied in various modes. In many cases, the deviations in the initialization time depending on the selected mode are negligible. The initialization time increases in CBC and CFB modes (ciphertext feedback mode) (see Figure 1). Cryptographic strength is provided for this method in cases of choosing encryption in the SHS mode, [14], [15], [16].

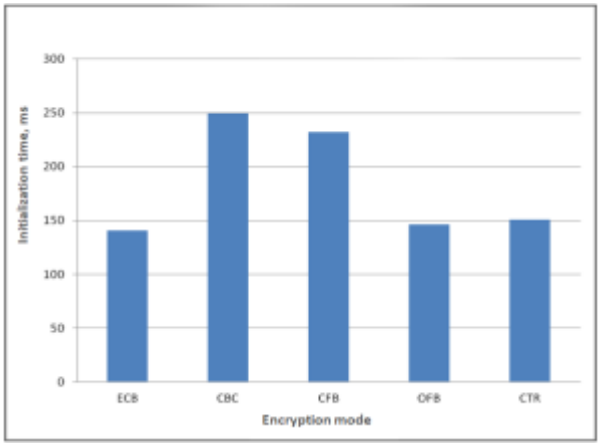

Fig. 1: Results of measuring the algorithm initialization time

4 Overview of the Results of the Analysis of Encryption Methods and Algorithms In Open-Source Operating Systems

The first block of the message is IV — the initialization vector (see Figure 2). When encrypting, the first block is added to the second block (bitwise by the module of 2). The resulting block is encrypted and the result is stored as an output second block (the first output block is still IV). The output second block is added by the module of 2 to the third block, the resulting third block is encrypted and stored as the output third block, etc. In the implemented module, the received output blocks are stored in the same memory, in which the input blocks with the same sequence number were stored, [17], [18].

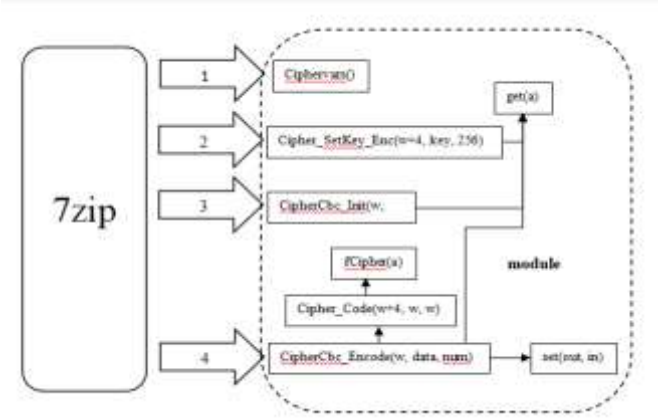

Fig. 2: Encryption process

When decrypting, the sequence is different:
The second block is decrypted (see Figure 3). The result is added to the first block (IV) bitwise by the module of 2, the resulting value is stored as the second output block. In this case, the encrypted value of the second block is stored in a temporary variable. Then the third block is added to a temporary variable bitwise by the module of 2, decrypted, and saved as the third output block, and the original encrypted value of the third block is stored in a temporary variable, and so forth, [19].

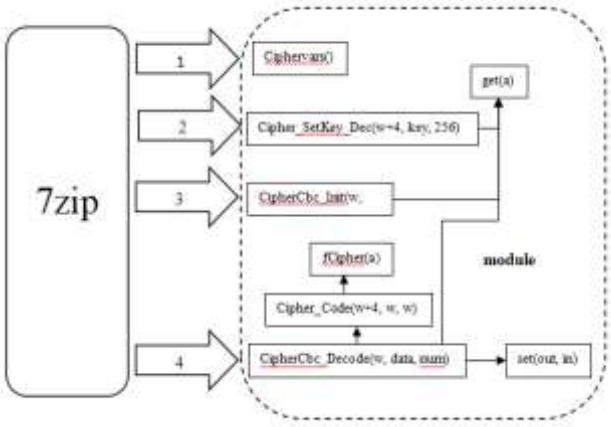

Fig. 3: Decryption process

The only security issue may be changing the length of the initialization vector from 128 bits to 64 bits. To confirm the validity of this transition, the following theorem was proved:
Theorem. If α_(m+n) – a random variable of length m+n bits — is uniformly distributed, then α_m – a random variable of length m bits, obtained from α_(m+n) by discarding the last n bits, is also uniformly distributed, [20], [21].
Proof. Suppose the opposite, α_m is unevenly distributed. Take all possible sequences of length n bits and denote them by β_n. Denote by η – a

random variable obtained from a random variable α_m by adding to it at the end of all possible β_n. Since β_n is not a random variable, η will have the same distribution as α_m. The resulting random variable η will take values, among which the values of the random variable α_(m+n). Then it is possible to associate each value of α_(m+n) with the same number of values of η, obtained by discarding the random sequence and adding the same number of deterministic different values. Therefore, we get a contradiction, as α_(m+n) is uniformly distributed. Therefore, the theorem is proved.

So, the generated random vector p7zip after sprinkling 64 bits from the end has the same uniform distribution function. That is, the durability corresponds to that of AES. As the IV (initialization vector) is not secret, the problem of reducing the enumeration does not affect the robustness of the algorithm.

Cryptanalysis by side channels was also taken into account during the development. So, if the message about the rejection of the correct password depends on the stage at which it was found, the bits of the correct encryption key can be found. In this implementation, the refusal message is generated only after the complete decryption of the message using the entered password. The incorrect key is identified by checking the last message block, which is formed according to the established rules, [22], [23]. If it does not correspond to those required for the message of this structure (the number of blocks, the number of added bits at the end of the message), an error occurs.

5 Testing Encryption Methods and Algorithms in Open-Source Operating Systems

5.1 Adaptation for an Open-Source Operating System

In operating systems of the Linux family, the super user has unlimited rights. But the system administrator may not have a sufficient level of access to some information that is processed on computers under his/her control. The developed cryptographic module is used for this purpose, [24], [25].

The developed module uses the get and set methods that convert a 4-byte sequence into a 32-bit unsigned number and a 32-bit unsigned number into a 4-byte sequence, respectively. These methods are also implemented in the developed module, they are not considered external functions. The sequence of writing bytes in a 32-bit number fully complies with GOST 28147-89. 8-bit shifts are used to insert byte data into an unsigned 32-bit integer.

It is mandatory to use cryptanalysis methods when developing such modules, as this prevents the detection of vulnerabilities in previously created applications. Because of world practice, it is also necessary to create uniform requirements for checking such an implementation of symmetric block data encryption algorithms and other cryptographic algorithms.

5.1.1 Test no. 1

Description: Files are encrypted using the State Standard 28147-89 algorithm based on the 7zip archiver, in which the **Aes.c**. file is modified, [9].

Description: The file encryption function is checked.

Preconditions:

1. The user is authorized in the system.
2. There is a non-empty text file in the user's home folder (for example, readme.txt).

Actions:

1. The user creates an archive with a password, into which he places the file (in this case readme.txt, you can specify a whole list of files together):

7za a –p arj.7z readme.txt

After pressing Enter, the archiver prompts you to enter a password. After entering it and pressing Enter, the archiver asks to repeat the password. After entering the password again and pressing Enter, the archiver creates the specified archive.

2. The user copies the archive to another folder (say archtest):

cp arj.7z archtest /

3. The user enters the specified folder:

cd archtest /

Postconditions:

1. The archtest folder contains the arj.7z archive, in which the encrypted state is a file (readme.txt).

5.1.2 Test no. 2

Decrypting files

Description:

Checking the file decryption with the same password as when entering. Demonstrates that the file is restored after unzipping, [19].

Preconditions:

1. Complete Test # 1 completed
2. The user is in the archtest folder

Actions:

1. Enter in the console:

7za x arj. 7z

At the invitation of the archiver to enter the password, enter the correct password (the one that was used for encryption).
Postcondition:
1. The archtest folder contains the source file (readme.txt).
An attempt to access without a known password
Description:
The test verifies the validity of encryption protection by the example of an attempt to enter a different keyword.
Precondition:
1. Test # 1 completed
2. If Test # 2 was performed, delete the unzipped file (readme.txt) from the archtest folder:rm readme.txt
Actions:
1. Enter the command:
2. 7za x arj.7z
At the invitation of the archiver, enter a different password (different from the one used for encryption)
Postcondition:
1. The archiver gives an unzipping error.
2. The folder contains an empty file (readme.txt). This item is optional.
As operations are performed on the entire volume of input data in several cycles, this algorithm has a linear complexity of O(n).

5.2 Checking the Compliance of the Encryption Function with GOST 28147-89

Description: Checking whether the implemented encryption function matches the test case given in State Standard R 34.11-94 for the encryption function specified by GOST 28147-89, [9]. Contains two examples.

tests/p7zip_gost/c/test.cpp

test.cpp content

```
#include "Types.h"
#include "GOST.h"
#include <stdio.h>

int main()
{Byte key[32] = {0x33, 0x20, 0x6d, 0x54, 0x32, 0x6c, 0x65, 0x68, 0x20, 0x65, 0x73, 0x69, 0x62, 0x6e, 0x73, 0x73, 0x79, 0x67, 0x61, 0x20, 0x74, 0x74, 0x67, 0x69, 0x65, 0x68, 0x65, 0x73, 0x73, 0x3d, 0x2c, 0x20};
Byte key2[32] = {0xa4, 0x24, 0x87, 0x34, 0x67, 0x76, 0xa6, 0xc1, 0x59, 0xde, 0x3d, 0x15, 0x50, 0x42, 0x88, 0x33, 0x65, 0x8c, 0x24, 0xe3, 0x8c, 0x3b, 0x41, 0x7d, 0x9a, 0xa0, 0x9c, 0x1c, 0xcf, 0x68, 0xd9, 0x56};
Byte in[64] = {0}, out[64], iv[32] = {0};
size_t n = 1;
int i;
UInt32 px[1000];
GOST_SetKey_Enc(px+4, key, 32);
printf("TEST1: input text = %8x %8x\n", get(in+4), get(in));
GOSTCbc_Init(px, iv);
GOSTCbc_Encode(px, in, 1);
printf("Encryption: %x %x\n", get(in+4), get(in));
GOST_SetKey_Enc(px+4, key2, 32);
set(in+4, 0x561c7de3);
set(in, 0x3315c034);
printf("TEST2: input text = %8x %8x\n", get(in+4), get(in));
GOSTCbc_Init(px, iv);
GOSTCbc_Encode(px, in, 1);
printf("Encryption: %x %x\n", get(in+4), get(in));
return 0;}
```

Precondition:
1. The tester is authorized as a user testmen.
2. The user is in his home directory. The pwd command will issue: / home / testmen.
Actions:
Compile: make test-gost
Run: ./bin/test-gost
Result:
TEST1: input text = 0 0
Encryption: 42abbcce 32bc0b1b
TEST2: input text = 561c7de3 3315c034
Encryption: 3cd1602d dd783e86
TEST 3: input text = 2998 1748
Encryption: 59 f3 4a fd 2f 56 fc 03 ad cb 99 5d 64 d3 f2 77
TEST 4: input text =1111 1111
Encryption: ed 98 ec 4c 05 74 ef 0b 6b 23 ed a5 39 b7 2a 3b
TEST 5: input text =5 3
Encryption: 27 4a 47 31 42 95 90 38 2e 3f 18 0f a6 29 68 cf
TEST 6: input text =a2d1 b45d
Encryption: dd 19 87 e2 34 a1 9d 56 21 d0 e6 d0 c9 a3 06 4a
TEST 7: input text =10101010 10101010
Encryption: 68 ad 46 f8 e0 20 3c cc 68 ad 46 f8 e0 20 3c cc
TEST 8: input text =842a3 8f89c
Encryption: 22 4a f5 2e 86 12 f6 59 53 41 3b d4 d1 0f 12 23
TEST 9: input text =9583 fdeac
Encryption: 84 9a e1 e6 aa 96 12 dc 23 9c aa ad a6 91 95 a1
TEST 10: input text =1234 56789

Encryption: bf d8 ff de b4 19 56 09 34 24 d6 d9 b7 a3 bd eb

The traditional data protection method has the advantages of long average secrecy time and low encryption efficiency, [26]. Because of this, many researchers choose more modern encryption algorithms. For example, the LRNG algorithm was chosen for analysis in [27]. The researchers determined that it is quite complex and includes a large state from three different storage pools, a complex mechanism for adding entropy from system events, and an extraction algorithm based on a shift register and several SHA-1 operations. This, in turn, complicates the implementation but does not prevent attacks on the direct security of LRNG. We chose DST 28147-89 encryption algorithm for research. It is a well-known 256-bit block cipher that is a likely alternative to AES-256 and triple DES, which, however, has a much lower implementation cost, [28]. Moreover, it is currently the only algorithm allowed to work with state secrets, [2]. To use it, the 7zip archiver was modified (with the AES encryption algorithm). Changes have been made to allow the State Standard algorithm to be implemented while keeping the external parameters the same as in AES. The created archiver successfully encrypts and decrypts files. Simple encryption (no archiving) was rejected due to the redundancy of symbolic information. This redundancy significantly weakens the properties of the algorithm. So, the first blocks in a txt document are the same for documents created on this computer by this user. Pre-archiving (compression) can significantly reduce the redundancy of files, and increase the statistical spread of byte values, which enhances the strength of the resulting cryptosystem. The user should be able to conveniently initiate encryption (see Figure 4) and decryption (see Figure 5) of his/her files. For this purpose, scripts have been created that draw a graphical interface (after all, p7zip is a console application).

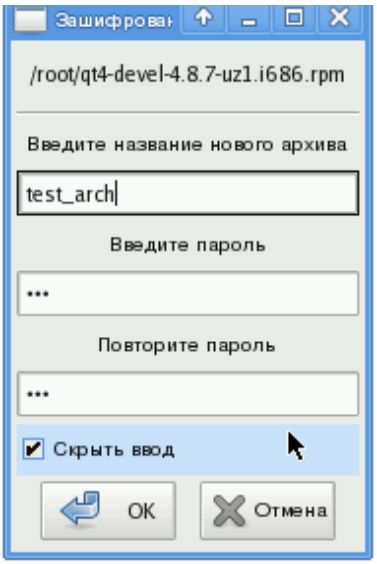

Fig. 4. File Encryption GUI

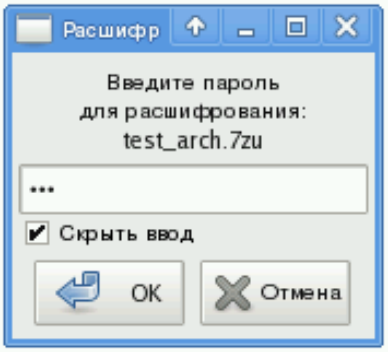

Fig. 5: File decryption graphical interface

6 Conclusion

The created cryptographic module was tested to meet the domestic standard, which contains several test cases. It was confirmed during the tests that the created module implements the algorithm of the domestic standard.

The use of cryptanalysis in the development of such modules is mandatory, as this avoids vulnerabilities that were identified in previously created implementations. It is also necessary to create uniform requirements for checking such implementations of both symmetric block data encryption algorithms and other cryptographic algorithms, taking into account world practice.

A fairly convenient graphical interface for accessing the cryptographic module was implemented, which enabled the user not to call the command line and remember the sequence and types of parameters passed to p7zip. This implementation also takes into account the verification of the correctness of decryption and the reading of other error codes.

References

[1] Luo Q. (ed.) *Advancing Computing, Communication, Control and Management*. Springer Berlin, Heidelberg, 2010.

[2] State Standard 28147-89. Information Processing Systems. Cryptographic Protection. Cryptographic Conversion Algorithm (76907). (Accessed 25 June 2022) Available online: https://dnaop.com/html/76907_3.html.

[3] Mao V. *Modern Cryptography: Theory and Practice*. M. Sudul (Ed.): Prentice Hall: New Jersey, USA, 2003.

[4] Stallings V. *Cryptography and Protection of Networks: Principles and Practice*, 7th Ed.; Pearson, NY, 2016.

[5] Ferguson N., Schneier B. *Practical Cryptography*. Publishing House "Williams", 1st Ed.; Wiley, 2003.

[6] Shannon C. E. Communication Theory in Secret Systems. *Bell System Technical Journal*, 1949, Vol. 28, No. 4, pp. 656-715.

[7] Ochilov N. Creating Secure File Systems in Open-Source Operating Systems. *WSEAS Transactions on Systems*, Vol. 21, pp. 221-232, 2022.
https://doi.org/10.37394/23202.2022.21.24

[8] Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Ed.; Wiley, 2015.

[9] Algorithm for Cryptographic Transformation GOST 28147 89. Domestic Data Encryption Standard. Available online: https://lawyersegezha.ru/en/ (Accessed 25 June 2022).

[10] ISO/IEC 18033-2, Information Technology — Security Techniques — Encryption Algorithms — Part 2: Asymmetric Ciphers. Available Online: https://www.iso.org/ (Accessed 25 June 2022)

[11] Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, November 26, 2001.

[12] National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standards Publication 198-1, July 2008. Available online: https://csrc.nist.gov/publications/detail/fips/198/1/final (accessed 25 June 2022).

[13] National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Information Processing Standards Publication 202, August 2015. Available online: https://csrc.nist.gov/publications/detail/fips/202/final (accessed 25 June 2022).

[14] Dworkin M. *Recommendation for Block Cipher Modes of Operation. Methods and Techniques*. NIST Special Publication 800-38A 2001 Edition. Available online: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf (accessed 25 June 2022).

[15] Dworkin M. *Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*. Addendum to NIST Special Publication 800-38A, October 2010. Available online: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a-add.pdf (accessed 25 June 2022).

[16] Rosulek M. *The Joy of Cryptography. Presents Modern Cryptography at a Level Appropriate for Undergraduates.* Oregon State University, Corvallis, Oregon, USA, 2021. Available online: https://joyofcryptography.com/pdf/book.pdf (accessed 25 June 2022).

[17] Yeomans J. S. A Bicriterion Approach for Generating Alternatives using Population-based Algorithms. *WSEAS Transactions on Systems*, Vol. 18, pp. 29-34, 2019.

[18] Holden J. *The Mathematics of Secrets: Cryptography from Caesar Ciphers to Digital Encryption.* Princeton University Press, Illustrated Edition, 2018.

[19] Grimes R. A. *Cryptography Apocalypse: Preparing for the Day When Quantum Computing Breaks Today's Crypto*. 1st Ed.; Wiley, 2019.

[20] Bertram L. A., van Dooble, G. Nomenclatura - *Encyclopedia of modern Cryptography and Internet Security - From AutoCrypt and Exponential Encryption to Zero-Knowledge-Proof Keys.* 1st Ed.; Books on Demand, 2019.

[21] Easttom W. *Modern Cryptography: Applied Mathematics for Encryption and Information Security.* 1st Ed.; Springer, 2020.

[22] Bock L. Modern Cryptography for Cybersecurity Professionals. Packt Publishing, 2021.

[23] Mihailescu M. I., Nita, S. L. *Pro Cryptography and Cryptanalysis with C++20: Creating and Programming Advanced Algorithms*. Apess: Berkeley, California, 2021.

[24] Baumslag G., Fine B., Kreuzer M., Rosenberger, G. *A Course in Mathematical Cryptography*. De Gruyter, 2015.

[25] Harvey J. *Cryptocurrency Investing for Beginners Part I: The Ultimate Guide to Cryptocurrency, History, Crypto Wallets and Cryptography*. Independently published, 2021.

[26] Zhao N. Improvement of Cloud Computing Medical Data Protection Technology Based on Symmetric Encryption Algorithm. *Journal of Testing and Evaluation*, Vol. 51, No. 1, 2022. https://doi.org/10.1520/JTE20210456

[27] Gutterman Z., Pinkas B., Reinman T. Analysis of the Linux random number generator. *2006 IEEE Symposium on Security and Privacy (S&P'06).* Berkeley/Oakland, CA, 2006. https://doi.org/10.1109/sp.2006.5

[28] Courtois N. T. Security Evaluation of GOST 28147-89 in View of International Standardisation. *Cryptologia*, 2012, Vol. 36, No. 1, pp. 2-13. https://doi.org/10.1080/01611194.2011.632807.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.