ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 2, 2025

DATA ENCRYPTION METHOD BASED ON THE MCELIECE CRYPTOSYSTEM AND THE REDUNDANT RESIDUE NUMBER SYSTEM

Vasyl Yatskiv, Serhii Kulyna, Stepan Ivasiev

West Ukrainian National University, 11, Lvivska Str, Ternopil, 46009, Ukraine.
Authors' e-mails: *jazkiv@ukr.net, sersks@gmail.com, stepanivasiev@gmail.com*

https://doi.org/10.23939/acps2025.02.214

Submitted on 30.09.2025

© Yatskiv V., Kulyna S., Ivasiev S. 2025

***Abstract*: This paper proposes a data encryption method based on a modified McEliece cryptosystem, in which classic Goppa codes are replaced by error-correcting codes from the Redundant Residue Number System (RRNS). The construction of the RRNS code's generator matrix and the formation of the public key as a composition of the *G* matrix, a scrambling matrix *S*, and a permutation matrix *P* have been described. An approach has been developed for selecting the RRNS moduli system, ensuring the necessary code parameters and the highest possible rank of the generator matrix. A study of the statistical and structural properties of the public key (entropy of elements, value distribution, density, rank), as well as their impact on the cryptosystem's resistance to ISD-type attacks, has been conducted. A software implementation of key generation and the encryption/decryption processes has been presented, along with experimental results from attack simulations for various RRNS code parameters. Based on the analysis, recommendations have been formulated regarding the choice of moduli, the structure of scrambling matrices, and code parameters to achieve a post-quantum security level with an acceptable public key size.**

***Index terms*: McEliece cryptosystem, Redundant Residue Number System (RRNS), error-correcting codes, public key, post-quantum cryptography, ISD attacks.**

I. INTRODUCTION

Ensuring data confidentiality and integrity is a fundamental task of modern information security. The development of quantum computing poses an existential threat to widely used asymmetric cryptosystems, as Shor's algorithm allows them to be broken in polynomial time. This vulnerability stimulates active research in the field of post-quantum cryptography (PQC) – the development of algorithms resistant to attacks by both classical and quantum computers.

One of the PQC candidates is the McEliece cryptosystem, proposed back in 1978. Its security is based on the hardness of the decoding problem in a general linear code, which remains NP-complete. The advantages of the McEliece system are high-speed encryption (which reduces to vector-matrix multiplication) and proven resistance. However, its practical application is limited by two significant drawbacks: a large public key size and considerable computational complexity of the decryption process, which requires efficient decoding algorithms for linear codes (such as Goppa codes). The relevance of this direction is confirmed by the NIST roadmap for standardizing post-quantum KEM schemes, where the Classic McEliece family is being considered among the next wave of candidates, in addition to the lattice-based algorithm standards already approved in 2024 [1].

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Analysis of scientific literature, especially in the context of the NIST PQC competition, allows us to identify three key research directions:

1. Optimization of key representation and reduction of their size.
2. Performance optimization and reduction of computational costs.
3. Hardware implementations and acceleration (FPGA, embedded systems).

The McEliece cryptosystem remains one of the most reliable post-quantum encryption schemes due to its resistance to known attacks; however, its practical application is complicated by the enormous size of the public key and significant computational costs. Over the last 5–7 years, researchers have been actively working on optimizing key representation, reducing their sizes, improving performance, and lowering computational costs, as well as on the development of efficient hardware implementations (FPGA, embedded systems, etc.).

In paper [2], a construction of the public key based on automorphisms of Goppa codes is proposed, which allows giving the matrix a special block structure. Using subgroups of additive and multiplicative automorphisms, the authors construct variants of Goppa codes with a structure suitable for compression and demonstrate the possibility of significantly reducing the public key size without loss of security. This research was the first to show the promise of using non-trivial automorphisms for key compression in the McEliece cryptosystem.

In paper [3], the authors proposed a new variant of the McEliece cryptosystem based on a non-binary code of orthogonal Latin squares (OLSC). Such a code has simpler encoding/decoding algorithms, which facilitates hardware implementation, and allows processing long

messages with smaller matrices, significantly reducing the key size. A parameterized coprocessor (FPGA) was developed that implements encryption and decryption; compared to the classic McEliece variant, it provides an approximately 3.3-fold speedup in operation [2]. This work demonstrates the possibility of reducing both keys and computational complexity by switching to a different class of codes.

A lightweight key agreement protocol based on the McEliece cryptosystem, optimized for resource-constrained devices (e.g., IoT), is presented in paper [4]. The proposed improvements made it possible to reduce the public key by approximately 80% (from ~1000 KB to ~200 KB) and cut computational costs by 66% (latency reduced from ~15 ms to ~5 ms). About 60% energy savings during crypto operations was also achieved, which is critically important for autonomous devices. This work confirms that even without compromising security, it is possible to significantly improve McEliece's metrics by optimizing the scheme and parameters for specific applications.

Optimization of performance and reduction of computational costs were achieved in paper [5]. The new implementation uses bitwise operations and FFT algorithms to accelerate calculations. Instead of parallel execution of many decodings, the author engaged internal parallelism within a single decoding, which allowed for improved decryption throughput at a higher security level. This result showed that significant acceleration of McEliece's operation can be achieved at a high security level even with purely software-based methods.

In paper [6], the possibility of implementing McEliece on constrained microcontrollers was demonstrated. The authors performed a constant-time implementation for the 32-bit ARM Cortex-M4, storing the large public key in the STM32 board's flash memory. For the smallest parameters (security level 1), a time of ~582 thousand cycles for encapsulation and 2.7 million cycles for decapsulation was achieved, which is 80 times faster for encryption and 17 times faster for decryption than the equivalent metric for the FrodoKEM scheme on the same platform. The implementation is also capable of performing key generation (for level 1) and decryption for the highest security level on this device. This work proved the practicality of Classic McEliece for IoT devices with limited memory, provided that memory usage and computations are optimized.

In [7], the first complete FPGA implementation of the Classic McEliece cryptosystem, compliant with the NIST specification, is presented. The design covers all stages: key generation (with expansion from a seed), encapsulation, and decapsulation, and is parameterized for different security levels. On a Xilinx Artix-7 FPGA, the following performance metrics were obtained: key generation in ~5.2–20 ms, encapsulation in 0.1–0.5 ms, and decapsulation in 0.7–1.5 ms (depending on the security level). By increasing parallelism, the design can be further accelerated at the cost of using more hardware area. These results are an order of magnitude better than previous implementations, demonstrating McEliece's suitability for high-performance applications with hardware acceleration.

Paper [8] focuses on accelerating one of the 'bottlenecks' of the McEliece algorithm – slow key generation. An algorithm and hardware co-design is proposed: specifically, a compact implementation of large-scale GF(2)-Gaussian elimination (with early singularity detection and memory-friendly task scheduling) and an optimized constant-time sorter for element permutation. As a result, the FPGA implementation achieved more than a 4-fold higher key generation throughput while simultaneously reducing the memory-time resource consumption by 9–14 times compared to previous FPGA solutions. This approach significantly mitigates the problems of slow key generation and large memory requirements that previously hindered the practical use of McEliece.

In [9], a universal hardware accelerator for Classic McEliece encapsulation, designed using High-Level Synthesis (HLS), is presented. The main innovation is the streaming processing of the public key directly from memory, which eliminates the need to store the 1 MB key in internal FPGA buffers. The accelerator performs encryption functions (encoder) and generates random errors, allowing the most demanding part of the algorithm (encapsulation) to be fully offloaded to the FPGA. Two design variants with different resource trade-offs are proposed. Record-breaking performance was achieved: depending on the parameters, the encryption time was reduced by 3.5–7.7 times compared to previous FPGA solutions. In conjunction with an embedded processor (Zynq SoC), this accelerator provided an approximately 2.2-fold faster protocol operation than an optimized 64-bit software implementation on a CPU. Thus, the work demonstrates the advantages of using CPU+FPGA co-design and HLS approaches to achieve a flexible yet high-performance solution.

A hybrid hardware-software implementation of McEliece was proposed in paper [10]. The public key is processed on the FPGA, while some parts of the algorithm run on an embedded processor. The solution complies with European security recommendations (ETSI). For a code length of 8192 bits, the authors achieved a decryption time of about 47.4 ms, which is a good result for FPGAs of that generation. This work became one of the first practical confirmations that a combined HW/SW solution can provide acceptable performance for the McEliece cryptosystem on modern hardware platforms.

The analysis of research showed that the main efforts were directed at improving the performance of the McEliece cryptosystem. New methods for public key compression have been proposed (structuring codes using automorphisms, using alternative code families), which allow for a multiple reduction in key size. Concurrently, optimization of software implementations has made it possible to speed up the cipher's operation

without loss of security. Significant results have been achieved in the field of hardware acceleration: modern FPGA implementations ensure all McEliece operations are performed in milliseconds, and specialized solutions for key generation and decryption eliminate the most critical bottlenecks. Embedded systems and microcontrollers are now capable of working with McEliece in real time thanks to memory and computation optimization. Thus, the shortcomings of the classic scheme (large keys, slow generation) are gradually being smoothed out, and the McEliece cryptosystem is becoming more attractive for practical implementation in post-quantum data protection protocols.

The analysis also showed a lack of fundamentally new approaches to improving the McEliece cryptosystem, particularly the use of redundant residue number system codes, which have high error-correcting capabilities but whose use requires a deep investigation of cryptographic resistance.

III. SCOPE OF WORK AND OBJECTIVES

The aim of this work is the development and analysis of a modified McEliece cryptosystem, in which traditional Goppa codes are replaced by error-correcting codes based on the Redundant Residue Number System. The research objective was to propose a complete cryptographic algorithm, including a key generation methodology (using generator, scrambling, and permutation matrices), as well as to demonstrate its practical implementation on a numerical example of encryption and decryption. Significant attention was paid to the analysis of cryptographic resistance; the work quantitatively assesses the complexity of Information Set Decoding (ISD) attacks against the proposed cryptosystem and determines the code parameters (number of moduli, information symbols, and errors) necessary to achieve modern levels of post-quantum security.

IV. DATA ENCRYPTION METHOD BASED ON THE MCELIECE-RRNS CRYPTOSYSTEM

A. RESIDUE NUMBER SYSTEM (RNS)

Let us consider a set of pairwise mutually prime positive modules [11]

$$M = \{m_1, m_2, \ldots, m_n\}, \quad (1)$$

$$gcd(m_i, m_j) = 1,\ I \neq j,\ m_i \geq 2. \quad (2)$$

The product of the moduli:

$$M = \prod_{i=1}^{n} m_i . \quad (3)$$

For every integer $x \in [0, M-1]$ is matched with the vector of residues

$$r(x) = (r_1, r_2, \ldots, r_n), r_i = x \bmod m_i . \quad (4).$$

Pair $(M, r(\cdot))$ defines a residual number system with a dynamic range $[0, M-1]$.

According to the Chinese remainder theorem (CRT), the mapping

$$\phi : x \to r(x), \quad (5)$$

is a bijection between $\{0, \ldots, M-1\}$ and the Cartesian product:

$$\prod Z_{m_i} . \quad (6)$$

A redundant residual number system is constructed by adding additional $n-k$ modules to k information modules.

Let

$M_I = \{m_1, m_2, \ldots, m_k\}$ – information modules,

$M_R = \{m_{k+1}, \ldots, m_n\}$ – redundant modules,

where all modules are collectively pairwise mutually prime.

Accordingly, the information range is:

$$M_I = \prod_{i-1}^{k} m_i , \quad (7)$$

verification range is:

$$M_R = \prod_{i=k+1}^{n} m_i . \quad (8)$$

Fool range is:

$$M = M_I \cdot M_R.$$

Then, an RRNS code will be defined as the encoding of a number that belongs to the range

$$x \in X = [0, M_I - 1], \quad (9)$$

in the form of a vector of all residues:

$$r(x) = (r_1, \ldots, r_k, r_{k+1}, \ldots, r_n), \quad (10)$$

$$r_i = x \bmod m_i \quad (11)$$

In this case:

the k first (r_1, …, r_k) components define the information vector in the RNS;

the remaining $n-k$ components (r_{k+1}, …, r_n) form the redundancy, which is used for error detection and correction.

B. THE DISTANCE AND CORRECTING CAPABILITY OF THE CPR CODE

In RRNS, a modular metric based on the number of positions in which the residues differ is usually considered. For two codewords

$$r(x) = (r_1, \ldots, r_n),\ r(y) = (S_1, \ldots, S_k), \quad (12)$$

determine the distance

$$d(r(x), r(y)) = |\{i \in \{1, .., n\} : r_i \neq S_i (\bmod m_i)\}| . \quad (13)$$

The minimum distance of the RRNS code:

$$d_{\min} = \min_{x \neq y} d(r(x), r(y)) . \quad (14)$$

Assuming that each error manifests as an arbitrary change in one of the residues r_i, then a code with a minimum distance d_{min} can detect up to d_{min-1} errors and correct:

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor, \quad (15)$$

errors in the residuals.

For an RRNS with an information range *[0, M_I -1]* and using the full set of moduli *M*, $d_{min}=n-k+1$ can be achieved under certain conditions (with a proper choice of moduli and range). In practice, for a given *n-k* it is often considered that the maximum number of errors the code corrects [11] is**:**

$$t = \left\lfloor \frac{n-k}{2} \right\rfloor. \quad (16)$$

V. THE ENCRYPTION ALGORITHM

The proposed asymmetric encryption algorithm consists of three stages [12]:

– private and public key generation;

– encryption;

– decryption.

Let us consider these stages in detail.

A. PRIVATE AND PUBLIC KEY GENERATION

Private key generation is performed according to the following algorithm.

1. Parameter selection: *k* - number of information bits, *n* - total code length, *t* - number of errors;

2. Selection of *n* prime moduli: m_i;

3. Matrix creation:

– A generator matrix *G* of size *(k × n)*;

– A scrambling matrix *S* (and its inverse S^{-1}) *(k × k)*;

A permutation matrix *P* (and its inverse P^{-1}) *(n×n)*);

Accordingly, the private key is defined by the matrices: *S, G, P.*

Public key generation:

$$G' = S \cdot G \cdot P. \quad (17)$$

Public key: *G', t.*

B. ENCRYPTION ALGORITHM

Message encryption is performed using the formula:

$$c = x \cdot G' + e. \quad (18)$$

where *x* – is the message; *e* – is the error vector.

C. THE DECRYPTION ALGORITHM

The decryption algorithm consists of the following steps:

1. We eliminate the influence of the permutation matrix *P* by multiplying the encrypted message by P^{-1}:

$$c' = c \cdot P^{-1}. \quad (19)$$

2. Error detection and correction.

To detect and correct errors, we use the Xiao algorithm [13]. The Xiao algorithm is a probabilistic method for identifying error positions in a Redundant Residue Number System (RRNS). Its main idea is to find a majority subset of error-free residues and, based on this, "discard" the positions containing errors. The algorithm consists of the following steps.

2.1. Initialization and Setup.

At this stage, the basic system parameters required for the algorithm's operation are set:

– data reading. The full list of moduli and the received list of residues (with errors) are loaded;

– code parameter definition: *k, t*;

– Information range calculation: M_R, the algorithm calculates M_R – the product of the first *l* moduli.

Since any uncorrupted number reconstructed from a subset of residues must necessarily be less than

$$M_R = \prod_{i=1}^{l} m_i \cdot \quad (20)$$

2.2. Random sampling of "consistent" subsets:

– Sampling launch. The algorithm executes a loop a certain number of times;

– *l* - subset formation. In each iteration, it randomly selects *l* ndices (positions) from the full set of *n* moduli.

2.3. Consistency check:

For this randomly selected *l* – subset (consisting of *l* – residues and *l* – moduli) the algorithm attempts to solve the system of equations using the Chinese Remainder Theorem (CRT);

If the CRT finds a unique solution *x*, the algorithm performs the main check: $x < M_R$;

If the solution *x* exists and it is less than the information range M_R, this *l* – subset is considered "consistent".

2.4. Statistics collection.

The algorithm maintains a list of indices for all *l* subsets that successfully passed the consistency check. In this case, if a random *l* – subset contains no erroneous positions, it will be consistent (yielding $x < M_R$). However, if it contains at least one error, it will, with high probability, be inconsistent (either the CRT will not find a solution, or $x > M_R$.

2.5. Frequency analysis of positions.

At this stage, the statistics collected in step 2 are analyzed.

1. Frequency counting. The algorithm creates a counter to track how many times each position (index from *0* to *n-1*) was part of "consistent" *l* – subsets.

2. Building the histogram. The algorithm iterates through the entire list of found consistent subsets and increments the counter for each index within them. In this process, uncorrupted positions will frequently form consistent subsets, so their counter will have a high value. Erroneous positions, conversely, will "break" the consistency, so they will rarely (or never) appear in this list, and their counter will have a low value.

2.6. Error determination by threshold value.

1. Finding the maximum. The algorithm finds the maximum counter value (i.e., the frequency with which the "most popular," likely "correct," position occurred).

2. Calculating the threshold. A threshold is set, for example, 50% of the maximum frequency.

3. Error identification. The algorithm iterates through all positions (indices). If the counter for a specific position is less than this threshold, it is marked as an error position.

The list of indices that failed the threshold check is the result of the algorithm's operation.

D. ERROR CORRECTION

We solve the system of equations in the field of integers:

$$y = x' \cdot G, \tag{21}$$

where y – is the known vector;
G is the generator matrix $(k \times n)$;
x'– is the unknown vector of length k.

It is necessary to find an integer solution x' such that

$$x' \cdot G = y, \tag{22}$$

$$y_i = \sum_i^k m_i \cdot G_{ij}, j = 1,...,n \cdot \tag{23}$$

E. REMOVING THE INFLUENCE OF THE SCRAMBLING MATRIX S.

To do this, we multiply the vector m' by the inverse matrix S^{-1}:

$$\mathrm{x} = x' \cdot S^{-1}.$$

VI. EXAMPLE DEMONSTRATION OF THE CRYPTOSYSTEM'S OPERATION

A. PUBLIC KEY GENERATION

Public key generation is performed according to the following algorithm.

1.1 Selection of public key parameters. Let, the number of information bits $k = 16$; total code length $n = 24$, number of errors $t = 4$, message $x = 1010101110101011$.

1.2 We select n coprime moduli: m_i [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 97].

1.3 Matrix Creation

Generator matrix G. The matrix G has the size $(k \times n)$, each row corresponds to one bit of information, and the columns are separated by moduli. Each row of the matrix G can be viewed as a set of residue vectors for the basis vectors in the message space.

Thus, the columns of the matrix G will be filled with the residues from the division of the numbers 2^0, 2^1, 2^2, …, 2^{k-1} by each of the moduli m_i [12].

$$G = \begin{bmatrix} (2^0 mod\ m_1) & (2^0 mod\ m_2) & \cdots & (2^0 mod\ m_n) \\ (2^1 mod\ m_1) & (2^1 mod\ m_2) & \cdots & (2^1 mod\ m_n) \\ \vdots & \vdots & \vdots & \vdots \\ (2^{k-1} mod\ m_1) & (2^{k-1} mod\ m_2) & (2^{k-1} mod\ m_n) & (2^{k-1} mod\ m_n) \end{bmatrix} \cdot \tag{24}$$

An example of the generator matrix G is shown in Fig. 1.

matrix_g24_1.txt

```
G=[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
   [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
   [0, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4],
   [0, 2, 3, 1, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8],
   [0, 1, 1, 2, 5, 3, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16],
   [0, 2, 2, 4, 10, 6, 15, 13, 9, 3, 1, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32],
   [0, 1, 4, 1, 9, 12, 13, 7, 18, 6, 2, 27, 23, 21, 17, 11, 5, 3, 64, 64, 64, 64, 64, 64],
   [0, 2, 3, 2, 7, 11, 9, 14, 13, 12, 4, 17, 5, 42, 34, 22, 10, 6, 61, 57, 55, 49, 45, 31],
   [0, 1, 1, 4, 3, 9, 1, 9, 3, 24, 8, 34, 10, 41, 21, 44, 20, 12, 55, 43, 37, 19, 7, 62],
   [0, 2, 2, 1, 6, 5, 2, 18, 6, 19, 16, 31, 20, 39, 42, 35, 40, 24, 43, 15, 1, 38, 14, 27],
   [0, 1, 4, 2, 1, 10, 4, 17, 12, 9, 1, 25, 40, 35, 37, 17, 21, 48, 19, 30, 2, 76, 28, 54],
   [0, 2, 3, 4, 2, 7, 8, 15, 1, 18, 2, 13, 39, 27, 27, 34, 42, 35, 38, 60, 4, 73, 56, 11],
   [0, 1, 1, 1, 4, 1, 16, 11, 2, 7, 4, 26, 37, 11, 7, 15, 25, 9, 9, 49, 8, 67, 29, 22],
   [0, 2, 2, 2, 8, 2, 15, 3, 4, 14, 8, 15, 33, 22, 14, 30, 50, 18, 18, 27, 16, 55, 58, 44],
   [0, 1, 4, 4, 5, 4, 13, 6, 8, 28, 16, 30, 25, 1, 28, 7, 41, 36, 36, 54, 32, 31, 33, 88],
   [0, 2, 3, 1, 10, 8, 9, 12, 16, 27, 1, 23, 9, 2, 9, 14, 23, 11, 5, 37, 64, 62, 66, 79]]
```

Fig. 1. The generator matrix G

Scrambling matrix S (and its inverse S^{-1}). To find the scrambling matrix, a program was developed that implements a random search method to find binary *(0-1)* unimodular matrices of a given size $k \times k$. A feature of the search is that the program only finds those matrices S, whose determinant equals ±1, and whose inverse matrix S^{-1} is also integer-valued.

To ensure precise arithmetic calculations (computation of the determinant and inverse matrix without rounding errors), the SymPy symbolic computation library is used.

The algorithm operates according to the following steps.

The program initializes parameters: size k, minimum (min_ones), and maximum (max_ones) number of ones.

In an infinite loop, a random binary matrix S of size $k \times k$is generated.

Each generated matrix undergoes sequential filtering:

- density check. It is verified that the total number of ones in the matrix w *(S)* is within the specified range [min_ones, max_ones];
- determinant check: *det (S)* is calculated. he search continues only if | *det (S)* | =1;
- inverse matrix check: S^{-1} is calculated and checked to ensure all its elements are integers.

If the matrix satisfies all three conditions, it is considered found, and the forward and inverse matrices are saved to the corresponding files S і S^{-1}.

The loop terminates when a matrix satisfying the given conditions is found.

The forward scrambling matrix for the given parameters is shown in Fig. 2, and the inverse in Fig. 3.

matrix_s16.txt

```
S=[
[1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0],
[0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0],
[1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0],
[1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
[1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1],
[0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
[0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0],
[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1],
[0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1],
[1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1],
[0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1],
[0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1],
[0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1],
[1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0]]
```

Fig. 2. Forward scrambling matrix

matrix_s16_inv.txt

```
S_inv=[
[0, 22, 15, 25, -9, -18, -6, -14, -10, 23, 17, 2, -24, 8, -6, -34],
[1, 33, 24, 39, -15, -29, -10, -22, -15, 36, 25, 2, -35, 13, -9, -53],
[0, 14, 10, 16, -6, -12, -4, -9, -6, 15, 10, 1, -15, 6, -4, -22],
[0, -29, -20, -33, 13, 24, 8, 19, 13, -31, -22, -2, 31, -11, 8, 45],
[0, 5, 4, 6, -2, -4, -1, -3, -3, 5, 4, 0, -6, 2, -1, -8],
[0, 9, 7, 11, -5, -8, -2, -7, -4, 11, 7, 0, -10, 4, -3, -15],
[0, 6, 5, 8, -3, -6, -2, -4, -3, 7, 4, 0, -6, 3, -2, -10],
[0, -15, -11, -18, 7, 13, 4, 10, 7, -16, -11, -1, 16, -6, 4, 24],
[0, -2, -1, -2, 0, 1, 1, 1, 1, -1, -2, -1, 2, 0, 0, 3],
[0, 13, 9, 15, -5, -11, -4, -8, -6, 13, 9, 1, -13, 5, -3, -20],
[0, -5, -5, -7, 3, 6, 1, 4, 3, -7, -3, 1, 6, -4, 2, 9],
[0, -10, -6, -11, 4, 8, 3, 6, 4, -10, -7, -1, 10, -4, 3, 15],
[0, -19, -15, -23, 9, 18, 5, 13, 9, -22, -14, 0, 21, -9, 6, 31],
[0, 5, 4, 6, -3, -5, -1, -4, -2, 6, 3, 0, -5, 3, -2, -8],
[0, 13, 9, 15, -5, -11, -4, -8, -6, 13, 10, 1, -14, 5, -3, -20],
[-1, -51, -37, -60, 23, 44, 15, 34, 23, -55, -38, -3, 54, -20, 14, 81]
```

Fig. 3. Inverse scrambling matrix

The matrix S must be a square matrix of size $k \times k$, where k is the number of information symbols (or the dimension of the information space). The matrix S must also be an invertible matrix in the field where the encryption is performed. This means that there exists an matrix S^{-1} such that $S \cdot S^{-1} = I$, where I is the identity matrix. Usually, the elements of matrix S are chosen from some finite field, for example, F_2 for binary codes or from a larger field for non-binary codes, depending on the chosen code. It is necessary that the choice of elements ensures invertibility.

The matrix S should be random to make the encryption more resistant to attacks based on knowledge of the code's structure. Randomness helps to hide the structure of the generator matrix G from attackers.

Permutation matrix P. The matrix P must be a square matrix of size $n \times n$, where n is the length of the codeword. In each row and each column, there must be exactly one *1*, and the rest of the elements must be *0*. This ensures that each element of the original vector's position is permuted to a unique new position. In other words, P is a unitary matrix in the context of permutations. A permutation matrix is always invertible, as its inverse matrix is the matrix that corresponds to the reverse permutation.

The calculated permutation matrix is shown in Fig. 4.

matrix_p24.txt

```
P=[
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Fig. 4. Permutation matrix P

B. PUBLIC KEY CREATION:

$$G' = S \cdot G \cdot P. \tag{25}$$

An example of the public key is shown in Fig. 5.

pub_key_SGP.txt

```
pub_key=[
[12, 38, 138, 15, 166, 14, 68, 103, 143, 119, 72, 37, 186, 112, 1, 49, 84, 51, 170, 203, 143, 132, 217, 64],
[9, 30, 164, 13, 168, 13, 47, 164, 195, 138, 124, 40, 234, 124, 0, 54, 94, 53, 212, 188, 127, 157, 253, 77],
[10, 36, 134, 17, 271, 21, 69, 149, 167, 117, 130, 49, 224, 160, 0, 68, 79, 37, 220, 258, 161, 148, 337, 76],
[9, 40, 118, 16, 174, 16, 67, 114, 115, 158, 101, 41, 277, 136, 1, 49, 77, 40, 205, 251, 162, 132, 222, 55],
[9, 32, 98, 9, 100, 11, 35, 117, 122, 89, 71, 31, 99, 66, 1, 37, 43, 30, 145, 113, 97, 91, 128, 54],
[8, 39, 100, 14, 190, 15, 73, 110, 126, 122, 70, 37, 169, 117, 0, 62, 48, 31, 186, 222, 179, 92, 232, 65],
[16, 54, 199, 20, 232, 21, 73, 182, 198, 187, 172, 54, 325, 171, 1, 77, 123, 62, 199, 211, 179, 213, 309, 99],
[9, 34, 108, 16, 187, 19, 49, 106, 121, 131, 96, 48, 281, 104, 0, 71, 88, 34, 165, 196, 189, 91, 243, 67],
[8, 31, 112, 19, 112, 18, 45, 130, 121, 133, 140, 41, 248, 119, 0, 44, 76, 38, 154, 171, 115, 126, 170, 57],
[17, 58, 214, 25, 320, 26, 80, 196, 222, 210, 139, 66, 335, 182, 1, 75, 123, 45, 314, 364, 272, 204, 411, 106],
[14, 57, 151, 20, 344, 25, 96, 189, 173, 187, 172, 58, 410, 185, 0, 98, 117, 51, 253, 319, 283, 200, 387, 90],
[12, 40, 111, 16, 280, 20, 68, 81, 90, 111, 83, 37, 227, 148, 1, 48, 79, 24, 141, 244, 163, 152, 328, 55],
[15, 58, 179, 17, 325, 23, 85, 203, 211, 188, 160, 59, 349, 180, 0, 96, 108, 51, 260, 280, 252, 199, 394, 102],
[15, 64, 153, 23, 349, 26, 112, 163, 153, 192, 137, 53, 382, 186, 0, 92, 119, 58, 247, 342, 293, 208, 382, 88],
[14, 46, 187, 25, 243, 22, 70, 172, 159, 195, 165, 46, 327, 178, 0, 51, 132, 54, 224, 296, 184, 229, 328, 90],
[11, 39, 164, 19, 293, 19, 76, 149, 166, 163, 120, 47, 323, 173, 1, 50, 110, 45, 282, 355, 217, 194, 359, 66]]
```

Fig. 5. Public key G'

C. PUBLIC KEY PARAMETER TESTING

Let us test the statistical characteristics of the public key: matrix rank, entropy, χ^2 - test for uniformity.

Public key test results:

Matrix size: 16 x 24.

Matrix rank:16.

The matrix has full rank (no linear dependencies were found).

Public key entropy: 7.4493 bits per symbol.

Maximum possible entropy for 204 unique values out of 360 elements: 7.6724 bits.

Total number of elements $N = 384$;

Number of unique values $f = 211$.

Results of the χ^2 -test for uniformity:

– χ^2 statistic: 147.8958;

– degrees of freedom *df*: 210;

– *p*-value: 0.999612.

As can be seen from Fig. 6, there are no obvious biases in the values of the public key elements **(χ^2 =147.8958)**.

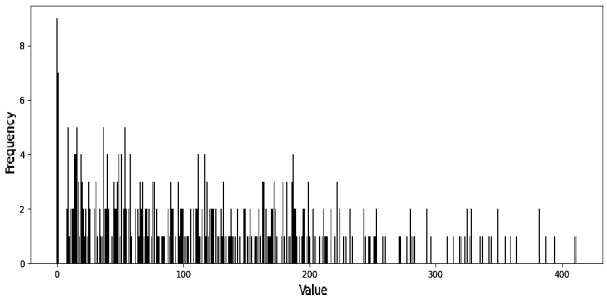

Fig. 6. Distribution of public key element values

The entropy is high (H ≈ 7.4493 bits/symbol) compared to the maximum possible entropy for 204 unique values out of 360 elements: H_{max} ≈7.6724 **бітів.**

D. ENCRYPTION

$$c = x \cdot G' + e. \tag{26}$$

Using the public key obtained in the previous step, we get the encrypted message c:

c = [118, 425, 1470, 177, 2273, 193, 666, 1500, 1581, 1509, 1298, 470, 2772, 1448, 4, 641, 960, 453, 2072, 2402, 1820, 1623, 2872, 765].

We add the error vector:

e = [0, 0, 29, 0, 0, 31, 0, 0, 0, 0, 0, 0, 37, 0, 0, 0, 0, 0, 0, 0, 0, 96, 0, 0].

We add errors at random positions (zero-indexed) [5, 2, 12, 21]:

position 5: +31 (was 193) -> 224;

position 2: +29 (was 1470) -> 1499;

position 12: +37 (was 2772) -> 2809;

position 21: +96 (was 1623) -> 1719.

The encrypted message with errors:

c = [118, 425, 1499, 177, 2273, 224, 666, 1500, 1581, 1509, 1298, 470, 2809, 1448, 4, 641, 960, 453, 2072, 2402, 1820, 1719, 2872, 765].

E. DECRYPTION

1. We eliminate the influence of the permutation matrix P, by multiplying the encrypted message c by the matrix P^{-1}:

$c' = c \cdot P^{-1}$ = [4,118, 224, 177, 425, 470, 666, 765, 641, 960, 453, 1509, 1448, 1581, 1500, 1499, 1719, 1298, 2072, 2402, 1820, 2872, 2273, 2809].

2. Using the Xiao algorithm, we find the error positions: error = [2, 15, 16, 23] and remove the columns from the matrix G, that correspond to the error positions. We write the new matrix to the variable G_{new}.

3. We remove the positions with errors from the file obtained in step 6.1.

y = [4, 118, 177, 425, 470, 666, 765, 641, 960, 453, 1509, 1448, 1581, 1500, 1298, 2072, 2402, 1820, 2872, 2273].

The next step is to solve the system of equations in the field of integers:

$$y = x' \cdot G_{new} \ , \tag{27}$$

where x' is the unknown vector of size $k \times 1$.

As a result of the calculations, we obtain the integer solution x': [4 6 4 4 6 6 5 6 6 4 6 3 3 6 4 5].

Check: $y = x' \cdot G_{new}$ = [4 118 177 425 470 666 765 641 960 453 1509 1448 1581 1500 1298 2072 2402 1820 2872 2273].

The difference between the vector (y) and the resulting product ($x' \cdot G_{new}$): [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0].

We remove the influence of the scrambling matrix S. To do this, we multiply the vector x' by the inverse matrix S^{-1}. As a result, we obtain the decrypted message:

x = [1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1].

For the McEliece-RRNS cryptosystem, besides the statistics of the elements, it is also important to conduct the following checks:

– matrix rank and structure;
– code properties (minimum distance, parameters k, n, t);
– structure of S, P;
– resistance to ISD/algebraic attacks, not just the statistics of the elements.

An ISD attack attempts to find k coordinates of the ciphertext c that do not contain errors and solve the system $x \cdot G' = y$ for these positions.

The probability that there are no errors in k randomly selected symbols among n symbols with t errors is $((n-t)|k)/(n|k)$. Thus, the total number of attempts for this attack is [12]:

$$W = \binom{n}{k} / \binom{n-t}{k} \cdot \alpha \cdot k^3 . \cdot \tag{28}$$

To investigate the resistance of the developed McEliece-RRNS cryptosystem to Information Set Decoding attacks, specialized software was developed. The program simulates the complete cryptographic cycle (key generation, encryption, error addition) and conducts a statistical experiment by repeatedly launching an ISD attack for given code parameters. The software allows changing the code parameters (n, k, t). Based on the program's results, estimates of the attack's computational complexity (number of iterations, execution time, number of operations on vectors/matrices) are automatically calculated. This made it possible to quantitatively assess the level of cryptographic resistance of the proposed cryptosystem for different sets of parameters [14].

The following code parameters were used in the cryptographic resistance calculation: m_i – prime numbers, n = 376, k = 256 (Fig. 7).

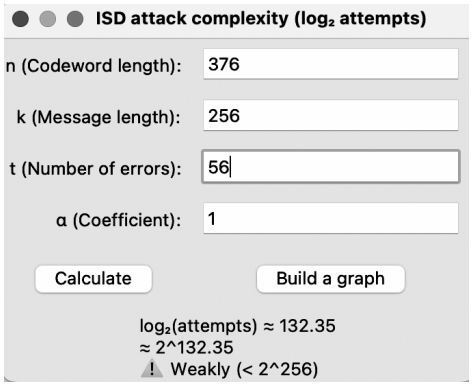

Fig. 7. Calculation of the number of errors for cryptographic resistance 2^{128}

In Fig. 8, the number of errors for cryptographic resistance 2^{196} is calculated with the same parameters.

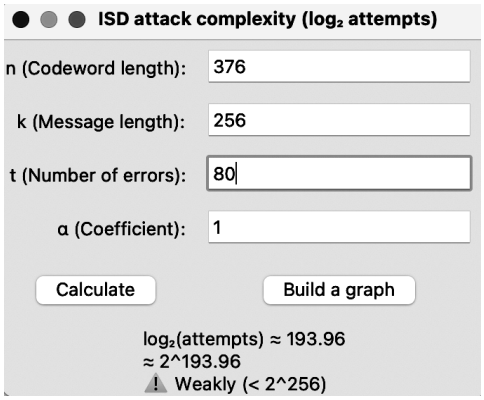

Fig. 8. Calculation of the number of errors for cryptographic resistance 2^{196}

In Fig. 9, the number of errors for cryptographic resistance 2^{256} is calculated with the same parameters.

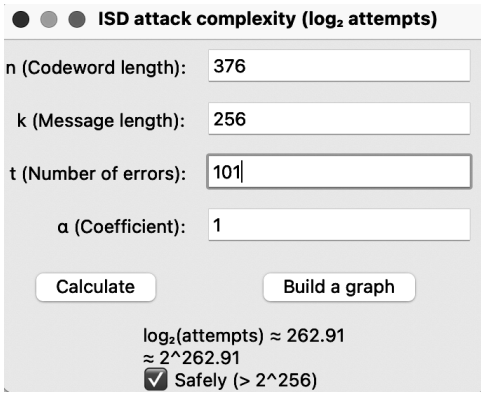

Fig. 9. Calculation of the number of errors for cryptographic resistance 2^{256}

Fig. 10 shows the dependence of the ISD attack complexity on the number of errors t for given parameters $n = 376$, $k = 256$.

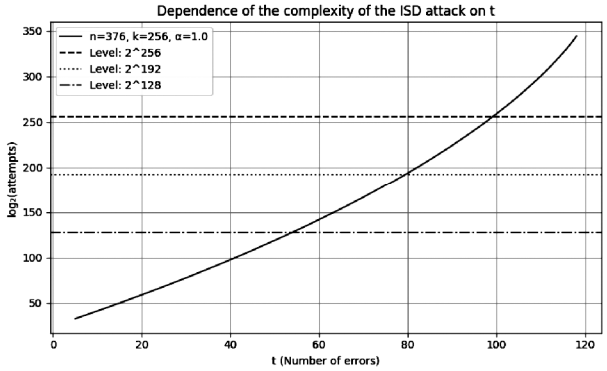

Fig. 10. Dependence of the ISD attack complexity on the number of errors t for given parameters n = 376, k = 256

The graph (Fig. 10) shows the required number of errors that must be introduced during encryption to ensure a given level of cryptographic resistance.

VII. CONCLUSION

This work proposed a modification of the McEliece cryptosystem based on error-correcting codes over a Redundant Residue Number System (RRNS), employing the standard scrambling – permutation public-key construction (G, S, P). We showed that this substitution enables the use of integer entries in the relevant matrices and naturally integrates the parameters (n, k, t) through the choice of the modulus set, while preserving the fundamental security principles of code-based cryptosystems.

An experimental assessment of statistical and structural properties of the generated public key indicated: full rank in the tested instance (no linear dependencies detected); high elementwise entropy ($\approx$ 7.449 bits/symbol versus a maximum of $\approx$ 7.672 bits/symbol); and no statistically significant deviation from uniformity under the χ^2 test (p-value $\approx$ 0.9996). These metrics mitigate the risk of trivial, pattern-based structural leakage; however, they are not, by themselves, sufficient to guarantee cryptographic security – decisive factors remain the code parameters and resistance to Information Set Decoding (ISD) and algebraic attacks.

We simulated ISD resistance while varying the number of errors t and the redundancy $r=n-k$. The resulting complexity curves confirm parameter regions in which the ISD cost reaches target post-quantum levels (e.g., $\geq 2^{128}$ operations) for acceptable public-key sizes.

Overall, the results support the viability of the McEliece-RRNS approach and outline practical parameter-selection guidelines that achieve post-quantum security levels with moderate memory and runtime overheads. Future work will focus on analyzing algebraic attacks targeting RRNS-based error-correcting codes and on optimizing the decryption algorithm.

VIII. CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

IX. DECLARATION ON GENERATIVE AI

During the preparation of this work, the authors used Claude in order to spelling check and translation. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Agrawal, R., Bu, L., & Kinsy, M. A. (2020, October). Quantum-proof lightweight McEliece cryptosystem co-processor design. In *2020 IEEE 38th International Conference on Computer Design (ICCD)* (pp. 73-79). IEEE. DOI: https://doi.org/10.1109/ICCD50377.2020.00029

[2] Chen, M. S., & Chou, T. (2021). Classic McEliece on the ARM cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021* (3), 125-148. DOI: https://doi.org/10.46586/tches.v2021.i3.125-148.

[3] Chen, P. J., Chou, T., Deshpande, S., Lahr, N., Niederhagen, R., Szefer, J., & Wang, W. (2022). Complete and improved FPGA implementation of classic McEliece. *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2022* (3), 71-113. DOI: https://doi.org/10.46586/tches.v2022.i3.71-113

[4] Chou, T. (2018). McBits revisited: Toward a fast constant-time code-based KEM. *Journal of Cryptographic Engineering, 8*, 95–107. DOI: https://doi.org/10.1007/s13389-018-0186-9

[5] Iqbal, S. S., & Zafar, A. (2025). Optimizing code-based cryptography for efficient and secure post-quantum key agreement. *Procedia Computer Science, 259*, 1034-1048. DOI: https://doi.org/10.1016/j.procs.2025.04.057

[6] Kostalabros, I. V., Ribes, J., Carril, X., Farras, O., Hernandez, C., & Moreto, M. (2025). Leveraging HLS to design a versatile & high-performance classic McEliece accelerator. *ACM Transactions on Embedded Computing Systems, 24* (5), 1-27. DOI: https://doi.org/10.1145/3698395

[7] Li, Z., Xing, C., & Yeo, S. L. (2019). Reducing the key size of McEliece cryptosystem from automorphism-induced Goppa codes via permutations. In D. Lin & K. Sako (Eds.), *Public-Key Cryptography – PKC 2019: Lecture Notes in Computer Science* (Vol. 11443). Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-17259-6_20

[8] López-García, M., & Cantó-Navarro, E. (2020, February). Hardware-software implementation of a McEliece cryptosystem for post-quantum cryptography. In *Future of Information and Communication Conference* (pp. 814-825). Springer. DOI: https://doi.org/10.1007/978-3-030-39442-4_60

[9] Moody, D., Perlner, R., Regenscheid, A., Robinson, A., & Cooper, D. (2024). *Transition to post-quantum cryptography standards* (NIST Internal Report 8547 ipd). National Institute of Standards and Technology. DOI: https://doi.org/10.6028/NIST.IR.8547.ipd

[10] Singh, H. (2019). *Code based cryptography: Classic McEliece* (arXiv:1907.12754). arXiv. DOI: https://doi.org/10.48550/arXiv.1907.12754

[11] Xiao, H., Garg, H. K., Hu, J., & Xiao, G. (2016). New error control algorithms for residue number system

codes. *Etri Journal, 38* (2), 326-336. DOI: https://doi.org/10.4218/etrij.16.0115.0575

[12] Yatskiv, V., Kulyna, S., Yatskiv, N., & Kulyna, H. (2020). Protected distributed data storage based on residue number system and cloud services. *2020 10th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 796-799). DOI:10.1109/ACIT49673.2020.9208849

[13] Yatskiv, V., Yatskiv, N., Ivasiev, S., Kulyna, S., Tsavolyk, T., & Yatskiv, I. (2025). The McEliece cryptosystem based on the redundant residue number system. *2025 15th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 573-577). IEEE. DOI:10.1109/ACIT65614.2025.11185887

[14] Zhu, Y., Zhu, W., Chen, C., Zhu, M., Li, Z., Wei, S., & Liu, L. (2023, July). McKeycutter: A high-throughput key generator of classic McEliece on hardware. In *2023 60th ACM/IEEE Design Automation Conference (DAC)* (pp. 1-6). IEEE. DOI: https://doi.org/10.1109/DAC56929.2023.10247918

Vasyl V. Yatskiv was born in Ivano-Frankivsk region, Ukraine, in 1972. He received a specialist degree in Process Automation at IvanoFrankivsk Technical University of Oil and Gas, Ukraine, 1996. Doctor of Philosophy in Computers, Systems and Networks, Lviv Polytechnic University, Ukraine, 2001. Associated professor, approved by the Ministry of Education and Science of Ukraine, 2004. Doctor of Engineering Science degree according to specialty Computer Systems and Components, Lviv Polytechnic National University in 2016. He is an author of more than 130 scientific articles and inventions. His research interests include secure data storage systems, code-based encryption methods, corrective codes in residue number system., ses for undergraduate students.

Stepan V. Ivasiev was born in Ternopil, Ukraine, in 1986. He obtained his B.S. and M.S. degrees in Software for Automated Systems at Ternopil National Economic University. In 2009, he defended his thesis for the degree of Candidate of Technical Sciences in the field of Computer Systems and Components and obtained a Ph.D. in Ternopil National Economic University. He is a co-author of 4 monographs and has published over 100 articles. His research interests include residue class systems, number theory, encoding, and factorization

Serhii V. Kulyna was born in 1987. He obtained his B.S. and M.S. degrees in Computer Engineering at Ternopil National Economic University. In 2023, he defended his Ph.D. thesis of Cybersecurity in Lviv Polytechnic University, Ukraine. Since 2018, he has been a lecturer at the Department of Cybersecurity.

He is an author of more than 30 scientific articles. His research interests include secure data storage systems, code-based encryption methods, corrective codes in residue number system.