The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 2025

Volume 38, Pages 655-666

IConTES 2025: International Conference on Technology, Engineering and Science

Generation Algorithms of Invertible Linear and Nonlinear Finite Automata with Memory

Gulmira Shakhmetova
L.N. Gumilyov Eurasian National University

Khasenov Altay
L.N. Gumilyov Eurasian National University

Zhanat Saukhanova
L.N. Gumilyov Eurasian National University

Altynbek Sharipbay
L.N. Gumilyov Eurasian National University

Alibek Barlybayev
L.N. Gumilyov Eurasian National University

Raykul Sayat
L.N. Gumilyov Eurasian National University

Abstract: In the context of rapid digital technology development, information security has become one of the key factors for the sustainable functioning of both society and the state. Modern methods of information protection rely on cryptographic algorithms, whose effectiveness is determined not only by their mathematical rigor but also by the degree of innovation in the applied models. One of the promising directions in this field is the use of finite automata models, which possess significant theoretical potential and broad opportunities for practical implementation. This study is devoted to the description of algorithms for generating invertible linear and nonlinear finite automata with input-output memory, applied in the FAPKC cryptosystem series. Unlike previously published works that are limited to purely mathematical descriptions of such models, this paper presents concrete procedures for their construction and implementation. Furthermore, a set of statistical tests was conducted to evaluate the randomness and cryptographic strength of the generated automata. The experimental results confirm their cryptographic robustness and demonstrate the feasibility of applying the proposed algorithms in modern cryptographic protocols.

Keywords: Cryptography, Finite automata, Cryptosystem, Invertible automata, Algorithm generation

Introduction

With the rapid development of digital technologies and the global digitalization of all spheres of activity, ensuring information security is of particular importance. The flow of data circulating in public and corporate networks is constantly increasing, which, in turn, increases the risks of unauthorized access, leakage, and modification of information. To counter these threats, cryptographic methods are used, which form the foundation of modern data protection systems.

The effectiveness of cryptographic algorithms is determined not only by the rigor of their mathematical foundations but also by the originality of the models used. Traditional approaches based on linear algebra, number theory, and combinatorial transformations are currently facing limitations due to increasing computing power and the development of quantum computing (Focardi, 2018). Therefore, alternative cryptographic constructs based on finite-state machine theory, which enable the implementation of highly complex transformations with relatively low computational costs, are becoming increasingly relevant (Shakhmetova et al., 2024).

Finite automata models offer several advantages: they are formally defined, easily amenable to mathematical analysis, and their structures are well adapted to hardware and software implementations (Satybaldina et al., 2011). The use of reversible linear and nonlinear finite-state machines with input and output memory opens up new possibilities for constructing robust encryption systems that provide both diffusion and data confusion at the level of elementary state transitions.

This study aims to develop and describe algorithms for generating such automata implemented in the FAPKC (Finite Automata Public Key Cryptosystem) series of cryptosystems (Tao & Chen, 1999; Kodada, 2022). Unlike previously published works, which are limited to the theoretical aspects of describing automaton models (Meskanen, 2001; Tao, 2008), this paper presents specific procedures for their construction, handling, and software implementation.

Particular attention is paid to the practical verification of the resulting models: a set of statistical tests was conducted to assess the randomness of the output sequences and their resistance to certain attacks. The experimental results demonstrate the high cryptographic strength of the proposed algorithms, confirming their feasibility for use in cryptography. Thus, this study not only expands theoretical understanding of the capabilities of finite automata models in cryptography but also makes a practical contribution to the development of methods for constructing and implementing cryptographically secure automaton converters.

Method

Linear and Nonlinear Finite Automata with Input-Output Memory

In classical automata theory, a finite automaton (FA) is a quintuple $M =< X, Y, S, \delta, \lambda >$, where X is a non-empty finite set of the input alphabet, Y is a non-empty finite set of the output alphabet, Q is a finite set of states, , $\delta: Q \times X \to Q$ is the transition function, and $\lambda: Q \times X \to Y$ is the output function. The behavior of a finite automaton, in the classical sense, depends only on the previous state and the current input. However, many cryptographic systems use finite automata with memory (Tao, 2008), (Kodada, 2022). The states of such an automaton are represented as a combination of the last k output and h input symbols, which leads to a finite automaton model with input-output memory of order (h, k) (Kohavi & Jha, 2009).

An important property of finite automata used for cryptographic purposes is reversibility. Reversibility refers to the ability to uniquely reconstruct the input sequence given a known initial state and output sequence. In particular, in real-world cryptosystems, a weakly reversible finite automata (WIFA) model is often used, where input reconstruction is not possible immediately, but only after a fixed number of steps—the so-called delay τ (Katerinsky, 2013). This behavior ensures one-way transformation at the encryption stage, while maintaining the ability to reverse the transformation given the necessary amount of information**.** Further, we will consider the sets X and Y as l-dimensional linear spaces over the field $GF(2) = \{0,1\}$. Let $y(i) \in Y$ be the output data at time i, and $(i) \in X$ be the column vector of the input data, then the automaton M_0 can be represented in the following form:

$$y(i) = \sum_{j=1}^{k_0} A_j\, y(i-j) + \sum_{j=0}^{h_0} B_j x(i-j),\ i = 0,1,2,\ldots \quad (1)$$

The automaton M_0 belongs to the class of finite automata with memory of order (h_0, k_0). This means that to uniquely determine its initial state, knowledge of at least the last h_0 input and k_0 output signals is required. Set of $< x(-1), x(-2), \ldots, x(-h_0), y(-1), y(-2), \ldots, y(-k_0) >$ describes the initial state of the automaton M_0.

In expression (1), A_j and B_j are linear matrices of size $(l\ x\ l)$ that completely determine the structure of the finite automaton M_0. All operations involved in formula (1) are performed in a Galois field (GF(2)), where standard addition and multiplication rules are used.

To construct a reversible linear FA with input-output memory, linear R_a^{-1}/R_b^{-1} transformations are used (Meskanen, 2001), in the following order:

$$G_\tau(i) \xrightarrow{R_b^{-1}[r_\tau]} G'_{\tau-1}(i) \xrightarrow{R_a^{-1}[P_{\tau-1}^{-1}]} G_{\tau-1}(i) \xrightarrow{R_b^{-1}[r_{\tau-1}]} \cdots \xrightarrow{R_b^{-1}[r_1]} G'_0(i) \xrightarrow{R_a^{-1}[P_0^{-1}]} G_0(i), \quad (2)$$

where $G_\tau(i) \in MP(q)$ is echelon matrix.

A linear finite automaton M_0^* is considered to be inverse to an automaton with delay τ if and only if from the system of matrices, A_j and B_j it is possible to select a corresponding set of matrices A_j^* , where $(j = 0 \dots \tau_0 + k_0)$, for which the equality holds:

$$x(i) = \sum_{j=1}^{h_0} B_j^* x(i-j) + \sum_{j=0}^{\tau_0+k_0} A_j^* y(i-j), \; i = 0, 1, \dots \quad (3)$$

For an arbitrary initial state $< x(-1), x(-2), \dots, x(-h_0), y(-1), y(-2), \dots, y(-k_0) >$ of the automaton M_0 and any input sequence $x(0), x(1), \dots, x(n), x(n+\tau) \in X$, the elements $x(0), x(1), \dots, x(n)$ can be uniquely reconstructed using formula (3). Thus, the automaton M_0^*, defined by expression (3), correctly defines the inverse automaton for M_0 with delay τ. To construct an inverse linear FA with input-output memory, the R_a/R_b transformation methods are used in the following order:

$$G_0(i) \xrightarrow{R_a[P_0]} G'_0(i) \xrightarrow{R_b[r_1]} G_1(i) \xrightarrow{R_a[P_1]} \cdots \xrightarrow{R_a[P_{\tau-1}]} G'_{\tau-1}(i) \xrightarrow{R_b[r_\tau]} G_\tau(i) \quad (4)$$

where $G_\tau(i) \in MP(q)$ is a matrix constructed for an invertible FA.

Nonlinear finite automata with input-output memory occupy a special place among finite-state cryptographic models, as they provide a high level of data diffusion and entanglement, introducing nonlinearity into linear transformations. Therefore, studying their inversion algorithms and analyzing the complexity of the corresponding procedures are of practical importance in the design of secure encryption transformations.

Nonlinear finite automata M_1 with input-output memory (h_1, k_1) has the following representation:

$$y(i) = \sum_{j=0}^{k_1} F_j\, y(i-j) + \sum_{j=1}^{k_1-\epsilon} F'_j s(y(i-j), \dots, y(i-j-\epsilon)) + \sum_{j=0}^{h_1} D_j\, x(i-j), \quad (5)$$

$i = 0,1,2, \dots$ где $s(x(i-j), \dots, x(i-j-\epsilon)$ nonlinear function, and ϵ – small positive integer.

The inverse nonlinear finite automata M_1^* is given by the formula:

$$x(i) = \sum_{j=1}^{h_1} D_j^*\, x(i-j) + \sum_{j=1}^{k_1+\tau-\epsilon} F_j'^* s(y(i-j), \dots, y(i-j-\epsilon)) + + \sum_{j=0}^{k_1+\tau} F_j^*\, y(i-j), \quad (6)$$

$i = 0,1,2, \dots$

Implementation of Algorithms

In this section, we describe our implemented program and some of the main algorithms for generating reversible linear and nonlinear finite automata. The software implementation is implemented as the NLFTEncrypt.exe executable module, developed in Python using the NumPy and SciPy libraries, as well as the TaoUtilities auxiliary module, which contains a set of functions for performing arithmetic operations in the Galois field $\mathbb{F}_2$. The program is built according to object-oriented design principles and includes four main classes that interact with each other during key generation and finite automata inversion (Figure 1).

LTaoKeygen is a key generation class for linear finite automata with input-output memory. It implements the full key construction cycle: from generating the height vector of the echelon matrix to generating all the coefficient matrices required for encryption and decryption. NLTaoKeygen is a key generation class for nonlinear finite automata. Structurally, it is similar to LTaoKeygen, but includes modified key generation algorithms and additional procedures that account for the nonlinear relationship between inputs and outputs. LTao is a linear automaton class containing the *encrypt()* and *decrypt()* functions, which use generated coefficient matrices to perform forward and inverse transformations. NLTao is an implementation of a nonlinear finite automaton, providing encryption and decryption of data based on nonlinear expressions. The main function __nonLinearFunc() implements the nonlinear transformation used to generate output vectors.

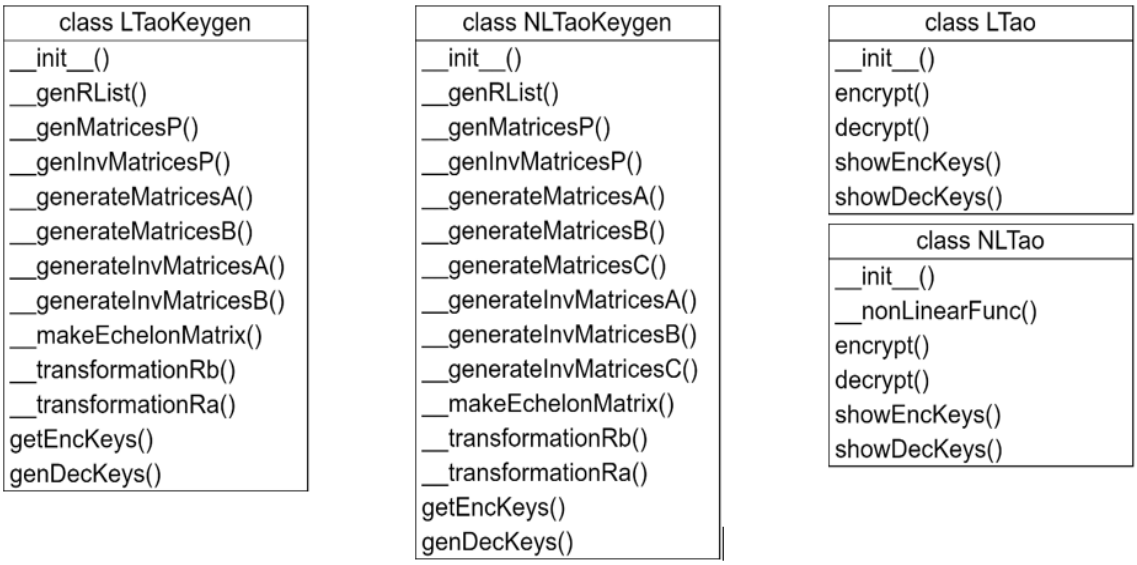


Figure 1. UML class diagrams of the program

The key construction methods are the algorithms for constructing coefficient matrices, the preceding algorithms for constructing height vectors, non-singular matrices P and P^{-1}, as well as transformation algorithms according to the rules R_a and R_b. The R_a transformation transforms the input matrix by multiplying it by the nonsingular matrix Pi. The R_a transformation rule has two variations: the direct transformation and the inverse transformation. The first is used when constructing the decryption key, while the second is used when constructing the encryption key and is denoted as R_a^{-1}. The inverse transformation R_a^{-1} differs from the direct transformation by the matrix P by which the multiplication occurs. The inverse transformation uses the direct matrix Pi, while the direct transformation uses the inverse matrix Pi^{-1}.

It is worth noting an important nuance in the R_a^{-1} transformation: the echelon matrix initially contains undefined elements. According to (Tao et al., 1997), the undefined element u has its own rules for multiplication and addition with numbers from the set {0, 1}. These rules are described as follows: $x \times u = u, 0 \times u = 0, y + u = u$, where: $x \in \{1, u\}, y \in \{0,1, u\}$. For the described arithmetic with undefined elements of the echelon matrix, a small class EchelonElementsCalculate was created, which is shown in Figure 2 below. The EchelonElementsCalculate class provides two static public methods: add and mul, which replace the addition and multiplication operators for values from the set $\{0, 1, u\}$. Within the program code, the value of u was replaced by the number 9. Thus, the EchelonElementsCalculate class can work with all elements of the echelon matrix, which, in turn, can be any number from a small group of numbers- {0, 1, 9}.

```
class EchelonElementsCalculate:

    @staticmethod
    def add(value1, value2):
        if value1 in (0, 1) and value2 in (0, 1):
            return value1 ^ value2
        return value1 if value2 in (0, 1) else value2

    @staticmethod
    def mul(value1, value2):
        if value1 in (0, 1) and value2 in (0, 1):
            return value1 & value2
        if value1 in (0, 1):
            return value2 if value1 == 1 else 0
        if value2 in (0, 1):
            return value1 if value2 == 1 else 0
        return value1
```

Figure 2. Operations with undefined elements

The R_b transformation also has two functions: a forward transformation and an inverse transformation. The forward R_b transformation shifts matrices to the left, while the inverse R_b^{-1} transformation shifts them to the right. If matrix expansion is necessary, the matrix dimension increases by the shift length; if not, the shifted elements that extend beyond the matrix boundary are lost. In all cases of shifting, the shifted elements are replaced by zeros.

Figure 3 shows the implementation of the R_b transformation, which uses the shift_rows_matrix function for shifting. This function is adapted for the implemented algorithms in that it immediately uses the *vectorLength* value as the shift length. This value is the state machine parameter m, where m is the length of the vector being processed. Also, instead of rowIndices, the *shiftRow* value is specified — this is the r_{k+1} value used for the R_b rule.

```
def transformationRb(matrix: np.ndarray, vectorLength: int,
                     shiftRow: int, leftShift: bool = False,
                     appending: bool = False):

    row2Shift = list(range(shiftRow, len(matrix)))

    transformedMatrix = shift_rows_matrix(
        matrix=matrix,
        rowsIndices=row2Shift,
        shiftLength=vectorLength,
        leftShift=leftShift,
        appending=appending
    )

    return transformedMatrix
```

Figure 3. R_b transformation

Finite automata construction focuses on constructing coefficient matrices (1), (3), (5), and (6). Structural coefficient matrices are the keys for encryption and decryption of data in automata cryptography.

An invertible linear finite automaton has two coefficient matrices: matrices $A = \{A_1, \ldots, A_{k_0}\}$ and матрицы $B = \{B_0, \ldots, B_{h_0}\}$. The number of matrices depends on the input and output memory. If the finite automaton has no output memory, then only the B matrices remain. The matrices are generated as a single rectangular matrix, combining the matrices of each input or output parameter. However, after a series of transformations or if certain conditions are met, the matrices are divided in a specific manner into equal square matrices.

Constructing the overall matrix B is a more complex task compared to matrix A due to the use of the echelon matrix and its transformations, which were discussed earlier. The generated overall matrix B has dimensions of m rows and $m \times (h + 1)$ columns. Construction of matrix B begins with two separate matrix elements: one identity matrix and h arbitrary matrices. Combining these two matrices yields an incomplete matrix B. To complete it, it is necessary to transform it into an echelon matrix and perform τ cycles of successive transformations R_b^{-1} and R_a^{-1} with the values ri and Pi, where i=τ− 1, …, 0. It is also worth considering one important rule: the value of the input memory h must not be less than the value of the finite state machine delay τ, h ≥ τ.

```
def generateMatricesB(self, vectorLength, rList, matricesP, inputMemory, taoDelay):
        identityMatrix = np.eye(vectorLength, vectorLength, dtype=np.uint8)

        randomMatrices = []
        for _ in range(inputMemory):
            randomMatrices.append(np.random.randint(0, 2, size=(vectorLength, vectorLength), dtype=np.uint8))

        stackedMatrices = [identityMatrix] + randomMatrices

        echelonMatrix = makeEchelonMatrix(stackedMatrices, rList, vectorLength)

        for i in range(taoDelay - 1, -1, -1):
            echelonMatrix = transformationRb(echelonMatrix, rList[i])
            echelonMatrix = transformationRa(echelonMatrix, matricesP[i])

        return echelonMatrix
```

Figure 4. Matrix generation function B

Figure 4 shows the implementation of the *genMatricesB* function, which constructs the general matrix B. Figure 4 shows that the function makes extensive use of the previously presented *transformationRa* and *transformationRb* functions. The construction of the inverse automaton is performed after the formation of the reversible automaton. This is crucial, since the process of constructing the decryption key is based on the use of the encryption key. Therefore, the structural matrices of the coefficients A and B must be pre-generated before constructing their inverse matrices A^* and B^*. The generation of the inverse matrices $A^*$and $B^*$is very similar, but differs only in the order of applying the R_a/R_b transformations, as can be seen in the code for constructing the inverse matrix B in Figure 5.

```
def generateInvMatricesB(matricesB: np.ndarray, invMatricesP, rList, taoDelay):
    matrix = matricesB.copy()

    for i in range(taoDelay):
        matrix = transformationRa(matrix, invMatricesP[i])
        matrix = transformationRb(matrix, rList[i], True)

    return matrix
```

Figure 5. Inverse matrix generation function B^*

The process of constructing a nonlinear finite automaton is largely similar to that of generating a linear automaton, but has several fundamental differences. A key feature of a nonlinear automaton is the presence of additional coefficient matrices, which must be generated separately. These matrices are used to process the results of nonlinear functions that accept elements of the input or output memory, depending on the type of memory used. Thus, the transition to considering nonlinear finite automata allows us to expand the model's functionality and explore the impact of nonlinear transformations on the cryptographic security of the system.

A nonlinear finite automaton with input-output memory, as seen in expression (5), has a nonlinear function s. In our case, the nonlinear function is defined as component-wise multiplication modulo 2 of two vectors. The procedure for constructing the structural matrices responsible for the nonlinear component involves generating arbitrary matrices of dimension $m \times m$. Thus, to construct a general rectangular matrix B, it is necessary to construct $h - 1$ or $k - 1$ arbitrary matrices of dimension $m \times m$. The parameter influencing the number of matrices is determined by the presence of input memory or output memory.

Algorithm LTao.encrypt(encKeys, data, vectorLength, taoDelay, inputMemory, outputMemory)

Encrypting process in LTao class

Input: keys to encrypt, encKeys contain coefficient matrices with elements ∈ $\mathbb{F}_2$
Input: bytes data, data ∈ {0, …, 255}
Input: length of processing vector, *vectorLength* ∈ ℕ
Input: delay of finite automata, *taoDelay* ∈ ℕ
Input: input memory, *inputMemory* ∈ ℕ
Input: output memory, *outputMemory* ∈ ℕ
Output: encrypted bytes data, data ∈ {0, …, 255}

```
1:    bitData ← bit_data_converter(data, vectorLength)
2:    delayData ← vectors with vectorLength zeros taoDelay times
3:    bitData ← bitData + delayData
4:    xBuffer ← vectors with vectorLength ones inputMemory times
5:    yBuffer ← vectors with vectorLength ones outputMemory times
6:    encBitData ← intager array[length of bitData][vectorLength]
7:    for each xCurrent in bitData
8:        xBuffer ← insert vectorLength vector with zeros in the start
9:        yCurrent ← vectorLength vector with zeros
10:       for (i = 0; i < length of encKey[0]; i++)
11:           yCurrent ← gf2_multiply(encKey[0][i], yBuffer) XOR yCurrent
12:       end for
13:       for (i = 0; i < length of encKey[1]; i++)
14:           yCurrent ← gf2_multiply(encKey[1][i], xBuffer) XOR yCurrent
15:       end for
16:       xBuffer ← pop the last element
17:       yBuffer ← pop the last element
18:       yBuffer ← insert vectorLength vector with zeros in the start
19:       encBitData ← insert yCurrent in the end
20:   end for
21:   encData ← bit_vector_to_int(encBitData)
22:   return encData
```

Figure 6. Encryption algorithm using a linear FA

After implementing the linear and nonlinear finite automata construction modules, we moved on to the data encryption and decryption stage, based on formulas (1), (3), (5), and (6). Each automaton type is implemented as a separate class (LTao, NLTao) with encrypt and decrypt functions accepting bit streams in various formats. For data preprocessing, the bit_data_converter function is used, converting input values into binary form suitable for subsequent processing by the finite automaton. This article presents algorithms implementing encryption/decryption using linear finite automata.

The *LTao.encrypt* algorithm in Figure 6 implements the data encryption process using a linear finite automaton and represents a transformation of the input byte stream based on binary operations in the Galois field $\mathbb{F}_2$. The input is the coefficient matrices of an invertible direct linear finite automaton, the vector length parameters, the delay, and the values of the automaton's input and output memory. The algorithm processes the input data step by step, converting them to binary form and generating an expanded stream taking into account the delay and memory state. During the computation, the state buffers *xBuffer* and *yBuffer*, which model the automaton's input and output memory, are iteratively updated. For each input data vector, the coefficient matrices A and B are sequentially multiplied by the corresponding buffers, followed by a bitwise XOR operation. This produces an encrypted bit sequence, which is converted back to an integer representation and returned as an array of encrypted bytes.

For the decryption process, the *LTao.decrypt* algorithm was implemented (Figure 7), which shows the process of decrypting data encrypted using a linear finite automaton. The algorithm is based on the coefficient matrices A^* and B^*, which are the inverse of the matrices used in encryption, and whose elements belong to the Galois field $\mathbb{F}_2$. The input is the encrypted data, the vector length parameters, the delays, and the values of the automaton's input and output memory.

```
Algorithm LTao.decrypt(decKeys, data, vectorLength, taoDelay, inputMemory, outputMemory)
Decrypting process in LTao class
Input: keys to decrypt, decKeys contain coefficient matrices with elements ∈ 𝔽₂
Input: bytes data, data ∈ {0, …, 255}
Input: length of processing vector, vectorLength ∈ ℕ
Input: delay of finite automata, taoDelay ∈ ℕ
Input: input memory, inputMemory ∈ ℕ
Input: output memory, outputMemory ∈ ℕ
Output: encrypted bytes data, data ∈ {0, …, 255}
    bitData ← bit_data_converter(data, vectorLength)
    delayData ← bitData from the start to taoDelay
    bitData ← bitData from taoDelay to the end
    xBuffer ← vectors with vectorLength ones inputMemory times
    yBuffer ← vectors with vectorLength ones outputMemory times
    yBuffer ← yBuffer + delayData
    yBuffer ← reverse()
    decBitData ← intager array[length of bitData][vectorLength]
    for each yCurrent in bitData
       yBuffer ← insert vectorLength vector with zeros in the start
       xCurrent ← vectorLength vector with zeros
       for (i = 0; i < length of decKey[0]; i++)
           xCurrent ← gf2_multiply(decKey[0][i], yBuffer) XOR xCurrent
       end for
       for (i = 0; i < length of decKey[1]; i++)
           xCurrent ← gf2_multiply(decKey[1][i], xBuffer) XOR xCurrent
       end for
       xBuffer ← pop the last element
       yBuffer ← pop the last element
       xBuffer ← insert vectorLength vector with zeros in the start
       decBitData ← insert xCurrent in the end
   end for
   decData ← bit_vector_to_int(decBitData)
  return decData
```

Figure 7. Decryption algorithm using a linear FA

The implementation of encryption and decryption algorithms using nonlinear finite automata is similar to the algorithms described previously. The only distinguishing feature is the presence of a nonlinear component. The encryption function in a nonlinear automaton has an additional variable in the block — *bufferForFunc*. This buffer stores the vectors that pass through the nonlinear function before being multiplied by the coefficient matrix.

Results and Discussion

To assess the functional viability of the implemented algorithms, comprehensive tests were conducted, including testing the correct operation of the key generation, encryption, and decryption software modules, as well as an

analysis of the statistical strength of the encrypted data. The experiments performed comprise a set of basic cryptographic tests aimed at verifying the system's ability to provide a sufficient level of data obfuscation while maintaining the ability to correctly recover data from the encrypted representation.

To clearly demonstrate cryptographic transformations, an image was used as the object. The initial data consists of the image's pixel values, represented as a byte stream, which are then encrypted using a finite automaton. This process was implemented using the *Pillow* library, which provides convenient image processing tools in the Python programming environment (Pillow, n.d.). Using the *tobytes()* method of the *Image* class object, the pixel values are extracted in binary form, after which they are passed to the encryption function. The result of the finite automata acting as an encryption mechanism is shown in Figure 8.

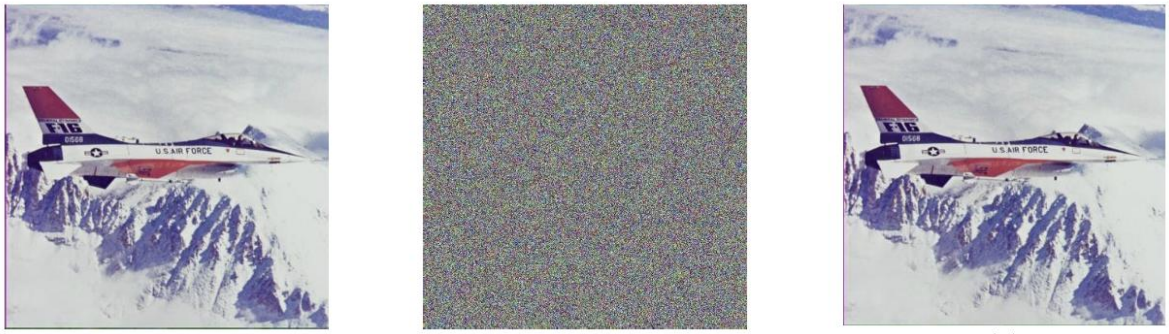


Figure 8. (a) Original, (b) Encrypted and (C) Decrypted image

Statistical Test

A statistical test from the National Institute of Standards and Technology (NIST) was performed (Almaraz, 2024). The NIST test performs bit-by-bit processing of the data and analyzes the distribution of bits across the entire stream using a set of statistical procedures, each aimed at identifying possible patterns in the sequence. The result of each calculation is a *p-value* ranging from 0 to 1; if the p-value is <0.01, the sequence is considered non-random, otherwise it is random. Four encrypted images were used for the experiment (Figure 9): one with a resolution of 512x512 pixels (4.2.05) and three with a resolution of 256x256 pixels (4.1.04–4.1.06), presented in .tiff format and containing color information.

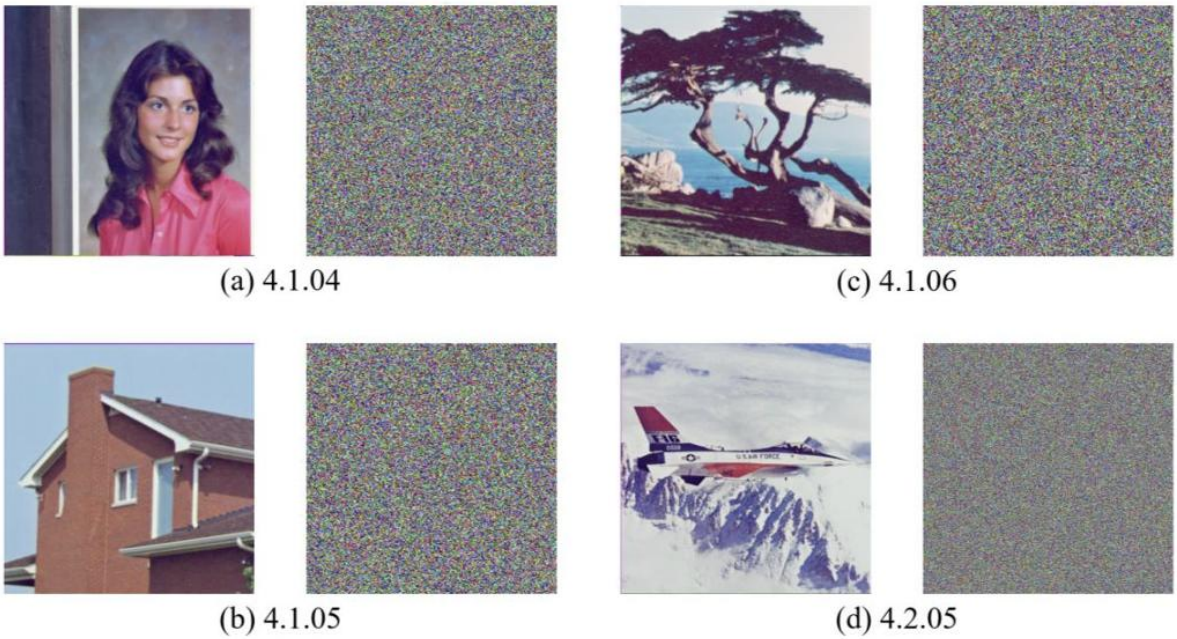


Figure 9. Original and encrypted images

The images were encrypted using a nonlinear finite-state machine with input-output memory. The NIST benchmark calculations use the first 1 million bits of image data, equivalent to 125,000 bytes. The NIST benchmark results are shown in Table 1.

Table 1. Test results NIST

Name of test	4.1.04	4.1.06	4.1.05	4.2.05
Frequency Test	0.2165 (true)	0.8462 (true)	0.7566 (true)	0.1160 (true)
Block Frequency Test	0.5587 (true)	0.6801 (true)	0.7218 (true)	0.6602 (true)
Run Test	0.7245 (true)	0.9936 (true)	0.1331 (true)	0.3146 (true)
Longest Run of Ones	0.8008 (true)	0.1601 (true)	0.5295 (true)	0.3786 (true)
Binary Matrix Rank Test	0.4271 (true)	0.7054 (true)	0.2936 (true)	0.1206 (true)
Discrete Fourier Transform (Spectral) Test	0.2748 (true)	0.8328 (true)	0.3216 (true)	0.7760 (true)
Non-overlapping Template Matching Test	0.7212 (true)	0.5440 (true)	0.0841 (true)	0.7985 (true)
Overlapping Template Matching Test	0.1090 (true)	0.1404 (true)	0.6235 (true)	0.2754 (true)
Universal Statistical Test	0.3271 (true)	0.9034 (true)	0.1256 (true)	0.7157 (true)
Linear Complexity Test	0.8717 (true)	0.3860 (true)	0.7438 (true)	0.0972 (true)
Serial Test 1	0.4772 (true)	0.3326 (true)	0.9797 (true)	0.8152 (true)
Serial Test 2	0.2522 (true)	0.5649 (true)	0.9008 (true)	0.2478 (true)
Approximate Entropy Test	0.2038 (true)	0.6227 (true)	0.9222 (true)	0.6078 (true)
Cumulative Sums (Forward)	0.4193 (true)	0.5264 (true)	0.2724 (true)	0.2273 (true)
Cumulative Sums (Backward)	0.4193 (true)	0.5264 (true)	0.2724 (true)	0.2273 (true)
Random Excursion & Variant Tests	p>0.01 (true)	p>0.01 (true)	p>0.01 (true)	p>0.01 (true)

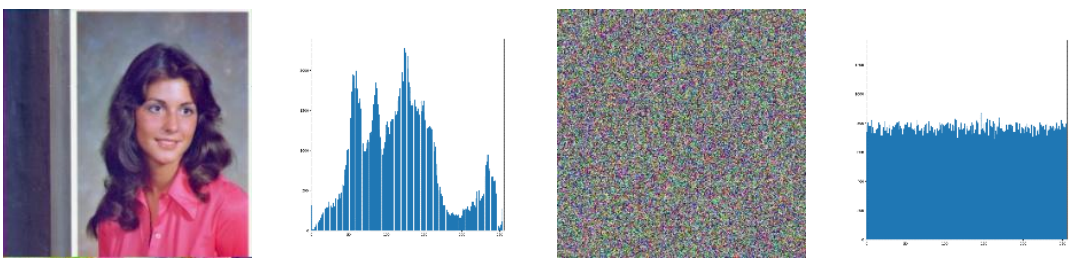
The results for all four data sets show that the p-values for all tests exceed the critical level of 0.01, confirming the absence of statistically significant deviations. Therefore, it can be concluded that the encrypted images exhibit a high degree of randomness and meet the cryptographic strength requirements.

Graphic Tests

Graphical tests are an analysis method based on visualizing encryption results. This approach considers histogram tests, as well as UACI and NPCR metrics, used to assess the sensitivity of a cryptosystem to minor changes in the original data. These tests use encrypted images, in which histograms reflect the distribution of pixel intensities, while UACI and NPCR tests quantify the degree of difference between the original and modified images after encryption.

Histogram

The histogram of an encrypted image is a graphical distribution of the frequencies of luminance values or color components of pixels across the entire image. In the context of cryptographic analysis, it is used to assess the uniformity of the intensity distribution after encryption (Cardona-López et al., 2024). For original (unencrypted) images, the histogram typically has pronounced peaks and patterns reflecting the structure and content of the image (Figure 10b). After encryption, provided the algorithm is working correctly, the histogram of the encrypted image should approach a uniform distribution, where all luminance levels occur with approximately equal frequency (Figure 10d).



(a) (b) (c) (d)

Figure 10. (a) original image 4.1.04, (b) histogram of (a), (c) encrypted image, (d) histogram of (c)

UACI and NPCR tests

The Number of Pixels Change Rate (NPCR) metric characterizes the percentage of changed pixels in the encrypted image when only one pixel in the input image changes. This parameter is used to assess the sensitivity of an encryption algorithm to minimal variations in the source data and, therefore, the cryptosystem's resistance to differential attacks (Asghar, 2023). The formal expression for calculating NPCR is presented in the formula:

$$NPCR = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{i=1}^{H} D(C1_{i,j}, C2_{i,j}), \tag{7}$$

C1 and C2 two encrypted images, W and H size of image. $D(C1_{i,j}, C2_{i,j})$ calculate:

$$D(C1_{i,j}, C2_{i,j}) = \begin{cases} 0, if C1_{i,j} = C2_{i,j} \\ 1, if C1_{i,j} \neq C2_{i,j} \end{cases}. \tag{8}$$

The Unified Average Changing Intensity (UACI) metric reflects the average difference in intensity between two encrypted images, expressed as a percentage. These images are generated from the original and a slightly modified original, allowing one to evaluate the sensitivity of a cryptosystem to minimal changes in the input data (Zhang, 2021). The UACI calculation analyzes the byte intensity values of each color channel (red, green, and blue) separately, providing a more accurate assessment of the uniformity and unpredictability of pixel distribution.

$$UACI = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} |C1_{i,j} - C2_{i,j}|, \tag{9}$$

where $C1_{i,j}$ and $C2_{i,j}$ pixel intensity values of two encrypted images.

The results of the tests performed are shown in Table 2.

Table 2. Results of NPCR and UACI

Name of test	Chanel	4.1.04	4.1.06	4.1.05	4.2.05
NPCR	RGB	99.592590	99.603780	99.596151	99.617004
UACI	Red	33.507888	33.313413	33.502179	33.550233
	Green	33.546938	33.597089	33.440636	33.572581
	Blue	33.280017	33.345552	33.523266	33.492114

The NPCR test determines how much the encryption result changes with minimal input data change, expressed as a percentage. A pass on the NPCR test is a score of 99% or higher. The ideal UACI test result, indicating that images have a high intensity difference, is 33% (Loukhaoukha et al., 2013). As can be seen from Table 2, all images meet these criteria and demonstrate good results.

Conclusion

In this study, algorithms for the generation of invertible linear and nonlinear finite automata with input–output memory were developed and implemented as a software module forming the basis of the FAPKC cryptosystem series. The research included detailed formalization of the automaton construction procedures, design of key generation methods, and implementation of encryption and decryption algorithms based on linear and nonlinear automata structures. The developed software, created in Python using the NumPy and SciPy libraries, provided a complete computational environment for modeling automaton-based cryptographic transformations and their inversions.

Comprehensive experimental testing confirmed the correctness and reliability of the proposed algorithms. The NIST statistical tests demonstrated high randomness and uniformity of encrypted data, as all *p*-values exceeded the 0.01 threshold, confirming the absence of statistical regularities. Graphical analyses, including histogram, NPCR, and UACI evaluations, also supported the cryptographic strength of the system. The NPCR values exceeded 99%, and UACI values were close to the theoretical ideal of 33%, indicating high sensitivity to minor input changes and effective diffusion properties. Overall, the performed work proves the efficiency of using finite automata with input–output memory for building secure and computationally efficient cryptographic systems. The obtained results can serve as a foundation for further research aimed at optimizing nonlinear transformations,

improving key management, and integrating automaton-based encryption into modern communication and data protection systems.

Scientific Ethics Declaration

* The authors declare that the scientific ethical and legal responsibility of this article published in EPSTEM journal belongs to the authors.

Conflict of Interest

* The authors declare that they have no conflicts of interest

Funding

* This research is funded by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan (Grant No. AP19677422).

Acknowledgements or Notes

* This article was presented as an oral presentation at the International Conference on Technology, Engineering and Science (www.icontes.net) held in Antalya/Türkiye on November 12-15, 2025.

* The authors would like to thank the conference committee and the reviewers who reviewed the article for their valuable contributions.

References

Almaraz Luengo, E. (2024). Statistical tests suites analysis methods. Cryptographic recommendations. *Cryptologia, 48*(3), 219-251.

Asghar, L., Ahmed, F., Khan, M. S., Arshad, A., & Ahmad, J. (2023). Noise-crypt: Image encryption with non-linear noise, hybrid chaotic maps, and hashing. In *2023 International Conference on Engineering and Emerging Technologies (ICEET)* (pp. 1-5). IEEE.

Cardona-López, M. A., Chimal-Eguía, J. C., Silva-García, V. M., & Flores-Carapia, R. (2024). Statistical analysis of the negative–positive transformation in image encryption. *Mathematics, 12*(6), 908.

Focardi, R., & Luccio, F. L. (2018, September). Neural cryptanalysis of classical ciphers. In *ICTCS* (pp. 104-115).

Katerinsky, D. A. (2013). On the reversibility of finite automata with finite delay. *Applied Discrete Mathematics. Appendix, 6*, 35–36.

Kodada, B. (2022). FSAaCIT: Finite state automata based one-key cryptosystem and chunk-based indexing technique for secure data de-duplication in cloud computing. *Authorea Preprints*.

Kohavi, Z., & Jha, N. K. (2009). *Switching and finite automata theory*. Cambridge University Press.

Loukhaoukha, K., Nabti, M., & Zebbiche, K. (2013, May). An efficient image encryption algorithm based on blocks permutation and Rubik's cube principle for iris images. In *2013 8th International Workshop on Systems, Signal Processing and Their Applications (WoSSPA)* (pp. 267–272). IEEE.

Meskanen, T. (2001). *On finite automaton public key cryptosystems*. Turku Centre for Computer Science.

Pillow. (n.d.). *Pillow 11.2.1 Documentation*. Retrieved from https://pillow.readthedocs.io/en/stable/index.html

Satybaldina, D., Sharipbayev, A., & Adamova, A. (2011). Implementation of the finite automaton public key cryptosystem on FPGA. In *WOSIS* (pp. 167–173).

Shakhmetova, G., Barlybayev, A., Saukhanova, Z., Sharipbay, A., Raykul, S., & Khassenov, A. (2024). Enhancing visual data security: A novel FSM-based image encryption and decryption methodology. *Applied Sciences, 14*(11), 4341.

Tao, R. (2008). *Finite automata and application to cryptography*. Springer.

Tao, R., & Chen, S. (1999). The generalization of public key cryptosystem FAPKC4. *Chinese Science Bulletin, 44*(9), 784-790.

Tao, R., Chen, S., & Chen, X. (1997). FAPKC3: A new finite automaton public key cryptosystem. *Journal of Computer Science and Technology, 12*(4), 289-305.

Zhang, Y. (2021). Statistical test criteria for sensitivity indexes of image cryptosystems. *Information Sciences, 550*, 313-328.

Author(s) Information

Gulmira Shakhmetova
L.N. Gumilyov Eurasian National University
Pushkin street 11, Astana, Kazakhstan
Contact e-mail: *sh_mira2004@mail.ru*

Khassenov Altay
L.N. Gumilyov Eurasian National University
Pushkin street 11, Astana, Kazakhstan

Zhanat Saukhanova
L.N. Gumilyov Eurasian National University
Pushkin street 11, Astana, Kazakhstan

Altynbek Sharipbay
L.N. Gumilyov Eurasian National University
Pushkin street 11, Astana, Kazakhstan

Alibek Barlybayev
L.N. Gumilyov Eurasian National University
Pushkin street 11, Astana, Kazakhstan

Raykul Sayat
L.N. Gumilyov Eurasian National University
Pushkin street 11, Astana, Kazakhstan

To cite this article:

Shakhmetova, G., Altay K., Saukhanova, Z., Sharipbay A., Barlybayev A., & Sayat, R. (2025). Generation algorithms of invertible linear and nonlinear finite automata with memory. *The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 38,* 655-666.