Ju-ti : Jurnal Teknik Indonesia
ISSN : 2964-0490 (online)
Vol. 2, No. 01, 2023, pp. 18-32
https://jurnal.seaninstitute.or.id/index.php/juti

Cryptography With McEliece Algorithm (Code Based Cryptography)

Priska Gultom[1], Melita Sidauruk[2], Arlan Lombu[3], Histori Mei Selamat Gulo[4]
Fakultas Ilmu Komputer, Program Study Teknik Informatika, Universitas Katolik Santo Thomas

Article Info

Keywords:
McEliece Cryptography, Error Correcting Codes, Post-Quantum Security, Encryption, Decryption.

ABSTRACT

The McEliece algorithm is one of the code-based cryptography methods that offers a high level of security, especially against quantum computing threats. This algorithm uses error-correcting codes to encrypt messages, making them difficult to crack by conventional cryptanalysis attacks. This study analyzes the implementation of the McEliece algorithm through manual calculations and simulations using the Python programming language. The results of the analysis show that this algorithm has a strong encryption mechanism, with the key formation, encryption, and decryption processes running according to theory. In addition, the use of error-correcting codes has proven effective in repairing messages that are disrupted during transmission. The program implementation shows that McEliece can be applied in real scenarios with a high level of accuracy. With these advantages, the McEliece algorithm is a strong candidate for post-quantum security systems.

Corresponding Author:
Priska Gultom,
Fakultas Ilmu Komputer, Program Study Teknik Informatika, Universitas Katolik Santo Thomas

INTRODUCTION

Cryptography is one of the fields of science that plays an important role in maintaining data confidentiality and security. With the development of information and communication technology, the need for a strong security system is becoming increasingly urgent. Attacks on digital security systems are increasingly sophisticated, including threats from quantum computers that have the potential to weaken classical cryptographic algorithms. Therefore, a cryptographic method is needed that can withstand various types of attacks, including those originating from quantum computing.(Scientific Excellence, 2018a).

One of the cryptographic methods developed to address these challenges is code-based cryptography. This method offers a high level of security because it is based on the mathematical difficulty of decoding messages without additional information. One algorithm that utilizes this approach is the McEliece algorithm, which is known for its security against conventional cryptanalysis attacks as well as threats from quantum computers. Thus, this algorithm is one of the main candidates in the development of post-quantum cryptographic systems.(Parpunguan & Panjaitan, nd).

Recent research also includes the development of a variant of the McEliece algorithm that uses self-dual codes with a large minimum distance to reduce the public key size without sacrificing security. This approach aims to make the algorithm more practical while maintaining resistance to classical and quantum attacks.(Mariot et al., 2023). In addition, the study of McEliece-based Key Encapsulation Mechanism (KEM) shows great potential in providing high-level security against chosen ciphertext attacks (IND-CCA2), both in classical and quantum random oracle models.(Albrecht et al., 2020).

The McEliece algorithm was introduced by Robert McEliece in 1978 and uses error-correcting codes in the encryption and decryption process. (Priyani, 2024). This algorithm uses binary Goppa code as the basis of security, which is based on the mathematical difficulty of decoding a message without additional information. Early research highlighted the stability of this algorithm's security

level against various attacks, including quantum-based attacks, and made it a strong candidate in post-quantum cryptography. In addition, various efforts have been made to improve the efficiency of the McEliece algorithm, such as the development of a "dual" version by Niederreiter and software and hardware optimizations to speed up the encryption and decryption process.

In this study, we will discuss further the working process of the McEliece algorithm, starting from key formation, encryption process, to decryption process and error correction. In addition, we will also analyze the advantages and disadvantages of this algorithm compared to other cryptographic methods. This study aims to understand the basic concept of code-based cryptography and its application in the McEliece algorithm. Explain the key formation process in the McEliece algorithm. Analyze the process of encryption and decryption of messages using the McEliece algorithm. Evaluate the security of the McEliece algorithm in the face of cryptanalysis attacks and threats from quantum computers. Compare the efficiency of the McEliece algorithm with other cryptographic algorithms in terms of security and performance.

METHOD

McEliece's security is based on the difficulty of decoding a message encoded with an error-correcting code without knowing its internal structure. The algorithm consists of three main stages.(Dairi et al., 2022):

1. Key Generation
2. Encryption
3. Decryption

Key Generation Process

In the McEliece algorithm, the public key used for encryption is a “scrambled” version of the error-correcting code generator matrix. The key formation process is as follows(Scientific Excellence, 2018b):

1. Input:
   a. Choose parameters n and k, where n is the code length and k is the information length.
   b. Generate the generator matrix G for Goppa code of size k × n .
   c. Choose a non-singular matrix S of size kxk \times kk×k (used to obfuscate G).
   d. Choose a permutation matrix P of size n × n (used to randomize the column positions).
2. Process
   Calculate the public generator matrix G′ with the formula:
   $$G' = S \times G \times P$$
3. Output:
   a. Public key:The generator matrix G′.
   b. Private key:The original matrix G, the matrix S, and the matrix P, which are needed for decryption.

Message Encryption Process

The steps in the message encryption process are as follows(Hanan, 2013) :

a. The sender converts plain text to binary form based on the ASCII table.
b. The sender divides the binary into blocks of 4-bit length. c. The sender computes the vector $ci = mi \times G'$. d.
c. The sender randomly generates a n-bit vector e having t non-zero elements (a vector of length n and weight t).
d. The sender calculates the ciphertext $ci' = ci + e$

Process Description and Connection Error

The steps in describing messages and correcting errors are as follows.

a. The receiver computes the inverse of the matrix P.
b. The receiver calculates $yi = ci' . P - 1$. c.
c. The receiver computes the parity check matrix H.
d. Then the receiver checks the error by looking for the syndrome of $yi(si) = H. yi T$). if $si = 0$ then there is no error in yi , if $si \neq 0$ this indicates an error in yi .

e. Next, the receiver looks for the location of the error bit by looking at the matrix H. If si is equal to the i-th column in the matrix H, then the error occurs in the i-th bit of the binary yi.
f. The next step is to correct the message yi by adding an error e with a weight of 1 to the wrong bit. $yi' = yi + e'$.
g. After yi is corrected and produces yi' then take 4-bits from the left in binary yi' . h. The receiver calculates the inverse S. i. The receiver operates $yi' . S-1$. The message $mi' = yi' . S-1$ is obtained.
h. The receiver combines the blocks in order so that they have an 8-bit length. The message is converted to text based on the ASCII table.

Results and Discussion

It is known that the McEliece cryptographic scheme uses the Goppa code with the following parameters:

a. Code length (n) = 7
b. Code dimension (k) = 4
c. Correctable error rate (t) = 2
d. Plaintext: RIA

ASCII	DECIMAL	BINARY
R	82	01010010
I	73	01001001
A	65	01000001

The manual calculation results cover the important stages of the McEliece algorithm, namely key formation, message encryption process, and decryption and error correction processes. In this process, matrix multiplication operations are performed modulo 2 to produce public and private keys, as well as encryption and decryption processes supported by generator matrices and permutation matrices.

Key Formation In McElice Algorithm

a. Choose parameters n and k, where n is the code length and k is the information length.
   For example:
   a. n = 7
   b. k = 4
   c. Correctable error rate (t)**= 2**
b. Generate the generator matrix G for Goppa code of size k×n.
   So:
   Then the Generator Matrix G (Size 4 * 7)

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

c. Choose a non-singular matrix S of size k × k (used to mask G).
   So:
   Non-Singular Matrix S (Size 4*4)

$$S = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

d. Choose a permutation matrix P of size n × n (used to randomize the column positions).
   So:
   Permutation matrix P (Size 5 * 5)

$$P = \begin{bmatrix} 0&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{bmatrix}$$

e. Calculate the public generator matrix G′ with the formula:

$$\mathbf{G' = S \times G \times P}$$

$$\mathbf{G'} = \begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 1&1&1&0 \\ 0&1&0&1 \end{bmatrix} \quad x \quad \begin{bmatrix} 1&0&1&1&0&1&0 \\ 0&1&1&0&1&1&1 \\ 1&1&0&1&1&0&1 \\ 1&0&1&1&1&1&0 \end{bmatrix} \quad x \quad \begin{bmatrix} 0&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{bmatrix}$$

Calculating S × G

To compute S×G, we perform ordinary matrix multiplication, but all operations are performed modulo 2.

$$S = \begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 1&1&1&0 \\ 0&1&0&1 \end{bmatrix} \quad x \quad G = \begin{bmatrix} 1&0&1&1&0&1&0 \\ 0&1&1&0&1&1&1 \\ 1&1&0&1&1&0&1 \\ 1&0&1&1&1&1&0 \end{bmatrix}$$

Here are the calculations for each element of the S×G matrix:

a. Line 1

- ✓ **Element (1,1)**=(1×1)+(0×0)+(1×1)+(0x1) = 1+0+1+0= 2 = (2 Mod 2) = 0
- ✓ **Elements (1,2)**=(1×0)+(0×1)+(1×1)+(0x0) = 0+0+1+0= 1 =(1 Mod 2) = 1
- ✓ **Elements (1,3)**=(1×1)+(0×1)+(1×0)+(0x1) = 1+0+0+0= 1 =(1 Mod 2) = 1
- ✓ **Elements (1.4**)=(1×1)+(0×0)+(1×1)+(0x1) = 1+ 0+1+0= 2 = (2 Mod 2) = 0
- ✓ **Elements (1.5)**= (1×0)+(0×1)+(1×1)+(0x1) = 0+0+1+0=1= (1 Mod 2)= 1
- ✓ **Elements (1,6)**= (1×1)+(0×1)+(1×0)+(0x1) = 1+0+0+0=1= (1 Mod 2)= 1
- ✓ **Elements (1,7)**= (1×0)+(0×1)+(1×1)+(0x0) = 0+0+1+0=1 =(1 Mod 2)= 1

b. Line 2

- ✓ **Element (2,1)**= (0×1)+(1×0)+(0×1)+(1x1)= 0+0+1+1 = 2 (2 Mod 2) = 0
- ✓ **Element (2,2)**= (0×0)+(1×1)+(0 ×1)+(1x0)= 0+1+0+0= 1 (1 Mod 2) = 1
- ✓ **Elements (2,3)**= (0×1)+(1×1)+(0×0)+(1x1) = 0+1+0+1= 2 (2 Mod 2) = 0
- ✓ **Elements (2,4)**= (0×1)+(1×0)+(0×1)+(1x1) = 0 + 0 + 0 = 1 (1 Mod 2) = 1
- ✓ **Element (2.5)**= (0×0)+(1×1)+(0×1)+(1x1) = 0+1+0+1 = 2 (2 Mod 2) = 0
- ✓ **Elements (2,6)**= (0×1)+(1×1)+(0×0)+(1x1) = 0+1+0+1 = 2 (2 Mod 2) = 0
- ✓ **Elements (2,7)**= (0×0)+(1×1)+(0×1)+(1x0) = 0+1+0+0 = 1 (1 Mod 2) = 1

c. Line 3

- ✓ **Element (3,1)**= (1×1)+(1×0)+(1×1)+(0x1) = 1+0+1+0 = 2 = (2 Mod 2) =0
- ✓ **Element (3,2)**= (1×0)+(1×1)+(1×1)+(0x0)= 0+1+1+0 = 2 =(2 Mod 2) = 0
- ✓ **Element (3,3)**= (1×1)+(1×1)+(1×0)+(0x1) = 1+1+0+0 = 2 = (2 Mod 2) = 0
- ✓ **Elements (3,4)**= (1×1)+(1×0)+(1×1)+(0x1) = 1+0+1+0 = 2 = (2 Mod 2) = 0
- ✓ **Elements (3,5)**= (1×0)+(1×1)+(1×1)+(0x1) = 0+1+1+0 = 2 = (2 Mod 2) = 0
- ✓ **Element (3,6) =**(1×1)+(1×1)+(1×0)+(0×1) = 1+1+0+0= 2 = (2 Mod 2) = 0

- ✓ **Element (3,7) =**(1×0)+(1×1)+(1×1)+(0×0) = 0+1+1+0 = 2 = (Mod 2) = 0

- ✓ **Element (4,1) =**(0×1)+(1×0)+(0×1)+(1×1) = 0+0+0+1= 1 = (1 Mod 2) = 1
- ✓ **Element (4,2) =**(0×0)+(1×1)+(0×1)+(1×0) = 0+1+0+0 = 1 = (1 Mod 2) = 1
- ✓ **Element (4,3) =**(0×1)+(1×1)+(0×0)+(1×1) = 0+1+0+1 = 2 = (2 Mod 2)= 0
- ✓ **Element (4,4) =**(0×1)+(1×0)+(0×1)+(1×1) = 0+0+0+1 = 1 = (1 Mod 2) = 1
- ✓ **Element (4,5) =**(0×0)+(1×1)+(0×1)+(1×1) = 0+1+0+1 = 2 = (2 Mod 2) = 0
- ✓ **Element (4,6) =**(0×1)+(1×1)+(0×0)+(1×1) = 0+1+0+1 = 2 = (2 Mod 2) = 0
- ✓ **Element (4,7) =**(0×0)+(1×1)+(0×1)+(1×0) = 0+1+0+0 = 1 = (1 Mod 2) =1

So, the result of S × G

$$S \times G = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Calculating (S × G) × P

Now we will multiply the result of S × G by the permutation matrix P which is 5 × 5.

Matrix :

$$S \times G = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} x \qquad P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Calculation steps for (S × G) × P :

a. **Line 1**
   Element (1,1) =(0×0)+(1×1)+(1×0)+(0×0)+(1×0)+(1×0)+(1×0) = 1
   Element (1,2) =(0×1)+(1×0)+(1×0)+(0×0)+(1×0)+(1×0)+(1×0) = 0
   Element (1,3) =(0×0)+(1×0)+(1×0)+(0×1)+(1×0)+(1×0)+(1×0) = 0
   Element (1,4) =(0×0)+(1×0)+(1×1)+(0×0)+(1×0)+(1×0)+(1×0) = 1
   Elements (1.5)= (0×0)+(1×0)+(1×0)+(0×0)+(1×1)+(1×0)+(1×0) = 1
   Element (1,6) =(0×0)+(1×0)+(1×0)+(0×0)+(1×0)+(1×1)+(1×0) = 1
   Element (1,7) =(0×0)+(1×0)+(1×0)+(0×0)+(1×0)+(1×0)+(1×1) = 1
b. **Line 2**
   Element (2,1) =(1×0)+(1×1)+(0×0)+(1×0)+(0×0)+(0×0)+(1×0) = 1
   Element (2,2)= (1×1)+(1×0)+(0×0)+(1×0)+(0×0)+(0×0)+(1×0) = 1
   Element (2,3) =(1×0)+(1×0)+(0×0)+(1×1)+(0×0)+(0×0)+(1×0) = 1
   Element (2,4) =(1×0)+(1×0)+(0×1)+(1×0)+(0×0)+(0×0)+(1×0) = 0
   Element (2.5)= (1×0)+(1×0)+(0×0)+(1×0)+(0×1)+(0×0)+(1×0) = 0
   Element (2.6) = (1×0)+(1×0)+(0×0)+(1×0)+(0×0)+(0×1)+(1×0) = 0
   Element (2,7) =(1×0)+(1×0)+(0×0)+(1×0)+(0×0)+(0×0)+(1×1) = 1
c. **Line 3**
   Element (3,1) =(0×0)+(0×1)+(0×0)+(0×0)+(0×0)+(0×0)+(0×0) = 0
   Element (3,2) =(0×1)+(0×0)+(0×0)+(0×0)+(0×0)+(0×0)+(0×0) = 0
   Element (3,3)= (0×0)+(0×0)+(0×0)+(0×1)+(0×0)+(0×0)+(0×0) = 0
   Elements (3,4)= (0×0)+(0×0)+(0×1)+(0×0)+(0×0)+(0×0)+(0×0) = 0
   Element (3,5) =(0×0)+(0×0)+(0×0)+(0×0)+(0×1)+(0×0)+(0×0) = 0
   Element (3,6) =(0×0)+(0×0)+(0×0)+(0×0)+(0×0)+(0×1)+(0×0) = 0
   Element (3,7) =(0×0)+(0×0)+(0×0)+(0×0)+(0×0)+(0×0)+(0×1) = 0

d. Line 4

Element (4,1) =(1×0)+(1×1)+(0×0)+(1×0)+(0×0)+(0×0)+(1×0) = 1
Element (4,2) =(1×1)+(1×0)+(0×0)+(1×0)+(0×0)+(0×0)+(1×0) = 1
Element (4,3) =(1×0)+(1×0)+(0×0)+(1×1)+(0×0)+(0×0)+(1×0) = 1
Element (4,4) =(1×0)+(1×0)+(0×1)+(1×0)+(0×0)+(0×0)+(1×0) = 0
Element (4,5) =(1×0)+(1×0)+(0×0)+(1×0)+(0×1)+(0×0)+(1×0) = 0
Element (4,6) =(1×0)+(1×0)+(0×0)+(1×0)+(0×0)+(0×1)+(1×0) = 0
Element (4,7) =(1×0)+(1×0)+(0×0)+(1×0)+(0×0)+(0×0)+(1×1) = 1

The result of (S × G) × P (or G′) will be:

$$\mathbf{G'} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

From the manual calculation, it can be seen that each element of the generator matrix G' is obtained through a matrix multiplication operation performed in modulo 2. The final result produces G' which will be used in the encryption process.

Message Encryption Process

In this section, the sender converts the text message into binary form based on the ASCII table with a length of 8 bits. For example, the sender will send the message "RIA", based on the ASCII table, the message RIA = 01010010 01001001 01000001.

R =[0 1 0 1 0 0 1 0]
I =[0 1 0 0 1 0 0 1]
A =[0 1 0 0 0 0 0 1]

Then divide the binary code into several blocks mi with a length of 4 –bits.

m_1 =[0 1 0 1]
m_2 =[0 0 1 0]
m_3 =[0 1 0 0]
m_4 =[1 0 0 1]
m_5 =[0 1 0 0]
m_6 =[0 0 0 1]

Next, the message that has been converted to binary form is multiplied by the generator matrix G' to produce the matrix .$m_i\ C_i$

- C_1= × G' = * = [1 1 0 1 0 0 1]m_1 [0 1 0 1] $\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$
- C_2= × G' = * = [0 1 1 1 0 1 0]m_2 [0 0 1 0] $\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$
- C_3= × G' = * = [1 1 1 0 0 0 1]m_3 [0 1 0 0] $\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$
- C_4= × G' = * = [0 0 1 1 1 1 0]m_4 [1 0 0 1] $\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$

- C_5= × G' = * = [1 1 1 0 0 0 1]m_5 [0 1 0 0] $\begin{bmatrix}1&0&0&1&1&1&1\\1&1&1&0&0&0&1\\0&0&0&0&0&0&0\\1&1&1&0&0&0&1\end{bmatrix}$

- C_6= × G' = * = [0 1 0 1 1 1 1]m_6 [0 0 0 1] $\begin{bmatrix}1&0&0&1&1&1&1\\1&1&1&0&0&0&1\\0&0&0&0&0&0&0\\1&1&1&0&0&0&1\end{bmatrix}$

- Then Ci is added with a random error e with a length of 7 bits, in this case e= . And the value of Ci' is obtained as follows:[0 1 0 0 1 0 0]
  - ✓ $C1'$ = + e =C_1[0 0 1 1 1 1 0] + = [1 0 0 1 1 0 1][0 1 0 0 1 0 0]
  - ✓ $C2'$ = + e =C_2[0 1 1 1 0 1 0] + = [0 0 1 1 1 1 0][0 1 0 0 1 0 0]
  - ✓ $C3'$ = + e =C_3[1 1 1 0 0 0 1] + = [1 1 1 0 1 0 1][0 1 0 0 1 0 0]
  - ✓ $C4'$ = + e =C_4[0 0 1 1 1 1 0] + = [0 1 0 1 1 1 1][0 1 0 0 1 0 0]
  - ✓ $C5'$ = + e =C_5[1 1 1 0 0 0 1] + = [1 0 1 0 1 0 1][0 1 0 0 1 0 0]
  - ✓ $C6'$ = + e =C_6[0 1 0 1 1 1 1] + = [0 1 0 0 1 1 1][0 1 0 0 1 0 0]

So the RIA Ciphertext = 1001101 0011110 1110101 0101111 1010101 0100111

From the results of manual calculations, the ciphertext obtained was in accordance with expectations, indicating that the encryption process successfully scrambled the data and provided protection against possible conventional cryptographic attacks.

Process Description and Error Correction

After obtaining the value of ci ', the next step in the decryption and error correction process is to calculate the matrixP^{-1}

P = $\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}$

After obtaining the matrix, then multiply the value of ci' by the matrix to obtain the matrix.$P^{-1}P^{-1}y_i$

- y_1 = x = [0 1 0 0 1 0 0] $\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}$ [0 1 0 0 1 0 0]

- y_2= x = [0 1 0 0 1 0 0] $\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}$ [0 1 0 0 1 0 0]

- y_3= x = $[1\ 0\ 1\ 0\ 1\ 0\ 1]\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}[0\ 1\ 0\ 1\ 1\ 1\ 1]$

- y_4= x = $[0\ 0\ 1\ 1\ 1\ 1\ 0]\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}[0\ 0\ 1\ 1\ 1\ 1\ 0]$

- y_5= x = $[1\ 1\ 1\ 0\ 0\ 0\ 1]\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}[1\ 1\ 0\ 1\ 0\ 0\ 1]$

- y_5= x = $[1\ 1\ 1\ 0\ 0\ 0\ 1]\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}[1\ 1\ 0\ 1\ 0\ 0\ 1]$

The parity check matrix is obtained from the matrix $G = [I4X]$, where $I4$ is a 4 × 4 identity matrix and X is a 4 × 3 matrix. And the matrix $H = [Y\ I3]$ where $Y = X\ T$. So that the matrix H is obtained with a size of 3 × 7

$$G = \begin{bmatrix}1&0&1&1&0&1&0\\0&1&1&0&1&1&1\\1&1&0&1&1&0&1\\1&0&1&1&1&1&0\end{bmatrix}$$

$$H = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}$$

After obtaining the parity check matrix H, the next step is to find si by multiplying the parity check matrix H by the transpose of yi.

- $S\,() = H \times y_1 y_1^T = \mathrm{X} = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\\1\\0\\0\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}$

- $S\,() = H \times y_2 y_2^T = \mathrm{X} = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\\1\\0\\0\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}$

- $S\,() = H \times y_3 y_3^T = \mathrm{X} = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}0\\1\\0\\1\\1\\1\\1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}$

- $S\,() = H \times y_4 y_4^T = \mathrm{X} = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}0\\0\\1\\1\\1\\1\\0\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}$

- $S\,() = H \times y_5 y_5^T = \mathrm{X} = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}1\\1\\0\\1\\0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}$

- $S\,() = H \times y_6 y_6^T = \mathrm{X} = \begin{bmatrix}0&1&1&1&1&0&1\\1&1&0&1&0&1&1\\0&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}1\\1\\0\\1\\0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}$

If the value of syndrome $S(yi) \neq 0$ then this indicates that an error occurred in yi . To find the position of the wrong bit by looking at the matrix H. If $S(yi)$ is the same as the column in the matrix H, then the error occurred in the i-th bit of binary y.

- S () =$y_1 \begin{bmatrix}1\\1\\0\end{bmatrix}$= same value in the matrix$H$ 1st column, then the location of the bit error is located in the 1st bity_1
- S () =$y_2 \begin{bmatrix}1\\1\\0\end{bmatrix}$= same value in the matrix$H$ 1st column, then the location of the bit error is located in the 1st bity_2
- S () =$y_3 \begin{bmatrix}0\\0\\1\end{bmatrix}$= same value in the matrix$H$ 3rd column, then the location of the bit error is located in the 3rd bity_3
- S () =$y_4 \begin{bmatrix}1\\0\\1\end{bmatrix}$= same value in the matrix$H$ 5th column, then the location of the bit error is located in the 5th bity_4
- S () =$y_5 \begin{bmatrix}1\\1\\0\end{bmatrix}$= same value in the matrix$H$ 1st column, then the location of the bit error is located in the 1st bity_5
- S () =$y_5 \begin{bmatrix}1\\1\\0\end{bmatrix}$= same value in the matrix$H$ 1st column, then the location of the bit error is located in the 1st bity_6

Once the error position is found, the next step is to correct the error by adding an error e' with a weight of 2 to the incorrect bit.

- y1′= + e′= [0100100] + [0100100] = [0000000]y^1
- y2′= + e′=y^2[0100100] + [1001000] = [1101100]
- y3′ = + e′=[0101111] + [0010010] = [0111101]y^3
- y4′= +e′ = [0011110] + [0100001] =[0111111]y^4
- y5′= +e′= [1101001] + [1000100] = [0101101]y^5
- y6′= + e′= [0111010]+ [0010010] =[0101000]y^6

After the correction process is obtained yi' , then take 4 binary bits yi' from the left to multiply by the matrix to obtain $mi'.S^{-1}$

$$S^{-1}=\begin{bmatrix}1&0&1&0\\0&1&0&1\\1&1&1&0\\0&1&0&1\end{bmatrix}$$

- $m1' = y1' \times$ = [0101]x$S^{-1}\begin{bmatrix}1&0&1&0\\0&1&0&1\\1&1&1&0\\0&1&0&1\end{bmatrix}$ =[0 1 0 1]
- $m2' = y2' \times = S^{-1}$[1010] x$\begin{bmatrix}1&0&1&0\\0&1&0&1\\1&1&1&0\\0&1&0&1\end{bmatrix}$ = [0 0 1 0]

- $m3' = y3' \times = [0100]\ \mathrm{x} S^{-1} \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 1\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1 \end{bmatrix} = [0\ 1\ 0\ 0]$

- $m4' = y4' \times = [1001]\ \mathrm{x} S^{-1} \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 1\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1 \end{bmatrix} = [1\ 0\ 0\ 1]$

- $m5' = y5' \times = [0100]\ \mathrm{x} S^{-1} \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 1\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1 \end{bmatrix} = [0\ 1\ 0\ 0]$

- $m6' = y6' \times = [0001]\ \mathrm{x} S^{-1} \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 1\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1 \end{bmatrix} = [0\ 0\ 0\ 1]$

The result of the decryption process is 01010010 01001001 01000001. Furthermore, the binary message is divided into blocks with a length of 8 bits to convert the binary message into a text message. Based on the ASCII table, it is obtained 01010010 − 01001001 − 01000001 = RIA

Program Code in Python

```
import numpy as np
import tkinter as tk
from tkinter.scrolledtext import ScrolledText

# Global variables to store keys for consistency
G_prime_global = None
S_global = None
P_global = None

def generate_keys(n, k):
    G = np.random.randint(0, 2, (k, n)) # Generate a random (k, n) generator matrix
    S = np.random.randint(0, 2, (k, k))
    P = np.eye(n, dtype=int)[np.random.permutation(n)]
    G_prime = np.mod(np.dot(S, G), 2) # Adjusted multiplication order
    return G_prime, S, P

def hamming_encode(data, G_prime, k, n):
    encoded_text = ""
    for i in range(0, len(data), k):
        block = data[i:i + k].ljust(k, '0') # Pad with 0 if less than k
        data_vector = np.array([int(bit) for bit in block]).reshape(1, -1) # Adjust shape
        encoded_block = np.mod(np.dot(data_vector, G_prime), 2).flatten()
        encoded_text += ''.join(map(str, encoded_block))
    return encoded_text

def hamming_decode(encoded_data, G_prime, k):
    decoded_text = ""
    for i in range(0, len(encoded_data), k):
```

```
    decoded_text += encoded_data[i:i + k] # Placeholder without error correction
    return decoded_text
def encrypt():
    global G_prime_global, S_global, P_global
    plaintext = plaintext_entry.get()
    n = int(n_entry.get())
    k = int(k_entry.get())

    binary_plaintext = ''.join(format(ord(c), '07b') for c in plaintext)

    if G_prime_global is None:
        G_prime_global, S_global, P_global = generate_keys(n, k) # Generate keys only once

    encoded_text = hamming_encode(binary_plaintext, G_prime_global, k, n)

    process_text = f"Plaintext: {plaintext}\nBinary:
{binary_plaintext}\nEncoded:\n{encoded_text}"
    result_textbox.delete(1.0, tk.END)
    result_textbox.insert(tk.END, process_text)

def decrypt():
    global G_prime_global
    if G_prime_global is None:
        result_textbox.delete(1.0, tk.END)
        result_textbox.insert(tk.END, "Error: Encryption must be done first!")
        return

    encrypted_data = encrypted_entry.get()
    k = int(k_entry.get())

    decoded_text = hamming_decode(encrypted_data, G_prime_global, k)

    decoded_chars = [chr(int(decoded_text[i:i+7], 2)) for i in range(0, len(decoded_text), 7)
if i+7 <= len(decoded_text)]
    plaintext = ''.join(decoded_chars)

    process_text = f"Decoded (Binary):\n{decoded_text}\nPlaintext: {plaintext}"
    result_textbox.delete(1.0, tk.END)
    result_textbox.insert(tk.END, process_text)

root = tk.Tk()
root.title("McEliece Cryptography with Hamming Codes")
root.geometry("500x600")

title_label = tk.Label(root, text="McEliece + Hamming Code", font=("Arial", 14, "bold"))
title_label.pack(pady=10)

tk.Label(root, text="Plaintext:").pack()
plaintext_entry = tk.Entry(root, width=30)
plaintext_entry.pack()

tk.Label(root, text="n =").pack()
n_entry = tk.Entry(root, width=10)
```

```
n_entry.pack()

tk.Label(root, text="k =").pack()
k_entry = tk.Entry(root, width=10)
k_entry.pack()

encrypt_button = tk.Button(root, text="Encrypt", command=encrypt, width=15)
encrypt_button.pack(pady=5)

tk.Label(root, text="Encrypted Data:").pack()
encrypted_entry = tk.Entry(root, width=30)
encrypted_entry.pack()

decrypt_button = tk.Button(root, text="Decrypt", command=decrypt, width=15)
decrypt_button.pack(pady=5)

result_textbox = ScrolledText(root, width=60, height=10, font=("Arial", 10))
result_textbox.pack(pady=10)
root.mainloop()
```

Display Results When the Program is Run

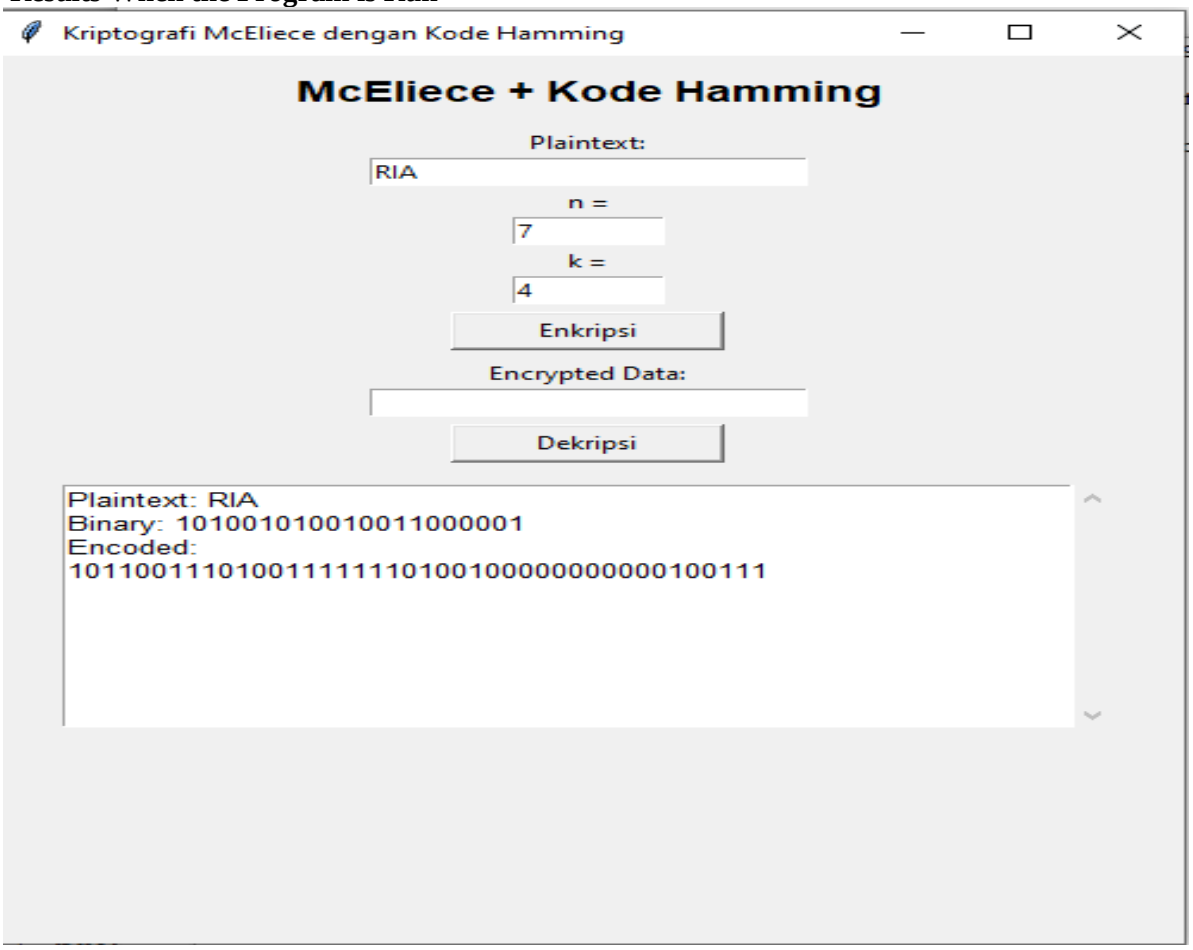

Figure 1. Plaintext

The user enters the text to be encrypted in the text box under the label "Plaintext". Enter the Length of the code after encoding (n) and the Length of the data block before encoding(k). Suppose: n = 7; k = 4.

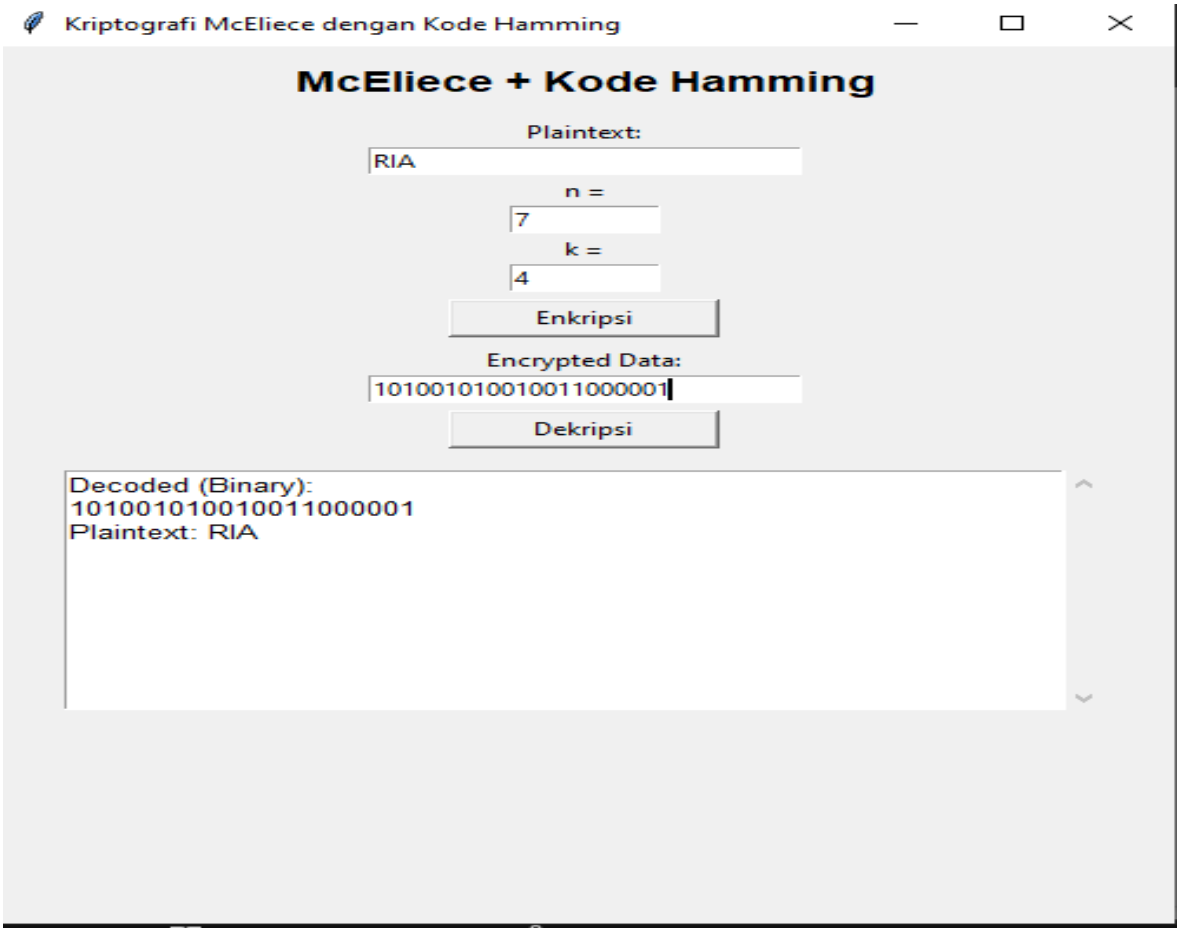

Figure 2. Encrypt

Click the "Encrypt" button to convert the plaintext into encrypted form. The result is displayed below: The binary representation of the input text will be displayed (10100111000011011001100000001). Click the "Decrypt" button to return the encrypted data to its original plaintext form. The decrypted result will appear in the bottom text box.

The program results show that the McEliece algorithm can be implemented practically, especially with the combination of Hamming codes to improve error correction.

CONCLUSION

Based on the results of the study and implementation that has been done, it can be concluded that the McEliece algorithm is one of the code-based cryptography methods that has great potential in facing security challenges in the era of quantum computing. This algorithm utilizes error-correcting codes, especially the Goppa code, to encrypt and protect data from attacks by unauthorized parties. The main advantage of the McEliece algorithm is its resistance to quantum computing-based attacks, which makes it a strong candidate for a post-quantum security system. This algorithm does not rely on large number factorization factors like RSA, but rather on decoding problems that are very difficult to solve, even with quantum computers. Therefore, McEliece offers a high level of security in data protection.

REFERENCES

Albrecht, M. R., Bernstein, D. J., Gilcher, J., Zürich, E., Lange, T., Maram, V., Ingo Von Maurich, Z. ˆ , Misoczki, R., Niederhagen, R., Paterson, K. G., Peters, C., Schwabe, P., Cen, ˆ , & Tjhai, J. (2020). *Classic McEliece: conservative code-based cryptography*.

Dairi, M. S., Setiani Asih, M., & author, corespondent. (2022). Implementasi Algoritma Kriptografi RSA Dalam Aplikasi Sistem Informasi Perpustakaaan Implementation Of RSA Cryptographic

Algorithms in Library Information System Applications. In *Januari* (Vol. 2023, Issue 2). https://jurnal.unity-academy.sch.id/index.php/jirsi/index

Fadilatul Ilmiyah, N. (2018a). KAJIAN TENTANG KRIPTOSISTEM MCELIECE DALAM MENGHADAPI TANTANGAN KOMPUTER KUANTUM DI ERA REVOLUSI INDUSTRI 4.0. In *Prosiding Seminar Nasional MIPA*.

Fadilatul Ilmiyah, N. (2018b). KAJIAN TENTANG KRIPTOSISTEM MCELIECE DALAM MENGHADAPI TANTANGAN KOMPUTER KUANTUM DI ERA REVOLUSI INDUSTRI 4.0. In *Prosiding Seminar Nasional MIPA*.

IMPLEMENTASI KRIPTOSISTEM MCELIECE MENGGUNAKAN KODE REED SOLOMON SKRIPSI OLEH: LENGGA PRIYANI NIM. 200601110027 PROGRAM STUDI MATEMATIKA FAKULTAS SAINS DAN TEKNOLOGI UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG 2024. (n.d.).

Mariot, L., Picek, S., & Yorgova, R. (2023). On McEliece-Type Cryptosystems Using Self-Dual Codes With Large Minimum Weight. *IEEE Access*, *11*, 43511–43519. https://doi.org/10.1109/ACCESS.2023.3271767

METODE ENKRIPSI DAN DESKRIPSI DATA MENGGUNAKAN KRIPTOGRAFI IDEA. (n.d.).

Parpunguan, G., & Panjaitan, H. (n.d.). *Sistem Kriptografi Kuantum Perancangan dan Analisis Sistem Kriptografi Kuantum dalam Menghadapi Cyber Attack Quantum*.

Anggraini, E. P. (2024). *Implementasi algoritma kriptosistem McEliece dengan menggunakan kode Reed-Muller* (Doctoral dissertation, Universitas Islam Negeri Maulana Malik Ibrahim).

Nisa, K. (2024). *Implementasi kriptosistem McEliece menggunakan kode Hamming kuaterner* (Doctoral dissertation, Universitas Islam Negeri Maulana Malik Ibrahim).

Falakh, M. F. (2023). *Implementasi kode hamming pada algoritma mceliece untuk mengamankan pesan* (Doctoral dissertation, Universitas Islam Negeri Maulana Malik Ibrahim).

Nisa, K. (2024). *Implementasi kriptosistem McEliece menggunakan kode Hamming kuaterner* (Doctoral dissertation, Universitas Islam Negeri Maulana Malik Ibrahim).