Systems of Linear Equations and Matrices Application in Cryptography: The Hill Cipher

I Made Wiweka Putera - 13523160[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523160@mahasiswa.itb.ac.id, wiwekaputera@gmail.com

***Abstract*— Cryptography has a crucial role in securing data by preventing unauthorized access and ensuring the confidentiality and integrity of data. Among various cryptographic techniques that have been developed, the Hill Cipher stands out as an early method founded on linear algebra principles. This paper explores the Hill Cipher, delving into its implementation and demonstrates how systems of linear equations and matrices are utilized for encrypting and decrypting messages. Although modern encryption systems have evolved, leaving the Hill Cipher behind, its foundational concepts continue to influence contemporary cryptographic methods. This paper highlights the educational value of the Hill Cipher, providing valuable insights into the application of linear algebra in cryptography and paving the way for understanding more advanced encryption techniques used today.**

***Keywords*—Hill Cipher, Linear Algebra, Systems of Linear Equations, Matrices**

I. Introduction

Introduced by Lester S. Hill in 1929, the Hill Cipher is a symmetric encryption algorithm that utilizes matrix multiplication and matrix inversion for both encryption and decryption processes. As a multi-letter cipher, it encrypts blocks of letters simultaneously, enhancing security by diffusing the plaintext over the entire ciphertext. Originally designed to secure text-based communications, the Hill Cipher employs an $n \times n$ matrix as the key, with common implementations using 2×2 and 3×3 matrices.

By using systems of linear equations and matrices, Hill Cipher allows for cryptographic encryption schemes that are both efficient and secure. The Hill Cipher was one of the first cryptographic algorithms to utilize these concepts, paving the way for future advancements in the field. The ability to represent and manipulate data using matrices gives advantages for developing robust encryption methods.

This paper explores the application of systems of linear equations and matrices used in the Hill Cipher, providing a comprehensive analysis of its theoretical foundations and practical implementations.

II. Theoretical Basis

2.1 Cryptography

Cryptography is the study of mathematical techniques for securing and achieving confidentiality of information. The term cryptography originated from Greek, meaning "secret writing". Cryptography involves transforming plaintext into ciphertext using various algorithms and keys, ensuring that only authorized parties can access the original information.

Plaintext refers to the original, readable message or data that is intended to be encrypted. It is the information in its natural, unencrypted form, which can be easily understood by anyone who has access to it. Examples of plaintext include text documents, emails, and any other type of data that needs to be kept secret.

Ciphertext is the result of the encryption process. It is the scrambled, unreadable version of the plaintext that has been transformed using an encryption algorithm and a key. Ciphertext is designed to be unintelligible to anyone who does not possess the decryption key. The purpose of converting plaintext into ciphertext is to protect the information from unauthorized access and ensure that it can only be read by someone with the correct decryption key.

Classical cryptography focused on creating and using codes, also known as ciphers, to enable two parties to communicate safely, by not leaking meaningful plaintext data to an adversary that could intercept their communications. In modern terminology, these codes are referred to as encryption schemes. The security of classical encryption schemes relied on a secret key shared in advance by the communicating parties and kept unknown to the adversary. This scenario is known as the private-key setting.

In private-key encryption, two parties share a key and use it to communicate secretly. One party encrypts the plaintext message using the key, resulting in ciphertext that is then sent to the receiver. The receiver then uses the same key to decrypt the ciphertext back into the original plaintext.

2.2 Symmetric Encryption

Symmetric encryption is a classical cryptographic

technique where the same key is used for both encryption and decryption. This method is efficient and requires less computational power compared to asymmetric encryption. However, the challenge arises in securely sharing the secret key between the communicating parties.

Encryption Process:

1. The plaintext is transformed into ciphertext using an encryption algorithm and a secret key.
2. The same key is used to decrypt the ciphertext back into plaintext.

Examples of Symmetric Encryption Algorithms:

1. Hill Cipher: Utilizes matrix multiplication and matrix inversion for encryption and decryption.
2. AES (Advanced Encryption Standard): A widely used symmetric encryption algorithm known for its security and efficiency.

Given the focus of this paper on the Hill Cipher, which is a symmetric encryption algorithm, the following sections will delve deeper into the principles and applications of symmetric encryption.

2.3 Systems of Linear Equations

Systems of Linear Equations are mathematical models that represent relationships between variables using linear equations. In the context of cryptography, linear systems are used to describe the transformation of plaintext into ciphertext and vice versa.

A linear system consists of a set of linear equations that can be represented in matrix form as $Ax = b$, where A is the matrix of coefficients, x is the vector of variables, and b is the vector of constants. Methods for solving linear systems include Gaussian elimination, iterative methods, and matrix inversion.

In cryptographic algorithms like the Hill Cipher, linear systems are used in the encryption and decryption processes. The relationship between plaintext and ciphertext can be modeled as a linear system, where the key matrix acts as the transformation matrix.

2.4 Hill Cipher

The Hill Cipher, created by Lester S. Hill in 1929, is a classical cryptographic technique designed to be invulnerable to the frequency analysis attacks. Unlike simple substitution ciphers, the Hill Cipher uses matrix multiplication for both encryption and decryption, making it a polyalphabetic cipher. This means that the same letter in the plaintext can be encrypted to different letters in the ciphertext, depending on its position within a block of text.

The Hill Cipher is categorized as a block cipher because it processes plaintext in fixed-size blocks. Each character in a block affects other characters during encryption and decryption, ensuring that the same character is not consistently stored in the same ciphertext character. This inter-block dependency increases the security of the cipher.

The theoretical basis of the Hill Cipher involves modulo arithmetic applied to matrices. The key to the Hill Cipher is an $n \times n$ invertible matrix K, where n is the block size. The matrix K must have an inverse K^{-1} because the decryption process depends on this inverse matrix.

Encryption Process:

1. Alphabet Mapping:
   Characters are mapped to numerical values, usually with ($A \rightarrow 1$, $B \rightarrow 2$, ..., $Z \rightarrow 26$).
2. Key Matrix:
   Create an $n \times n$ key matrix K. This matrix must be invertible, meaning it has a non-zero determinant and an inverse matrix K^{-1}.
3. Plaintext Blocking:
   Divide the plaintext P into blocks of size n. If the number of letters in the plaintext is not a multiple of n, padding characters (e.g., 'X') are added to the plaintext to complete the final block.
4. Matrix Multiplication:
   For each plaintext block P, convert it to a column vector. Multiply the plaintext vector by the key matrix K and take the result modulo 26 (or 256 for ASCII characters) to obtain the ciphertext vector (C): $[C = PK \ mod \ 26]$
5. Ciphertext Conversion:
   Convert the numerical values in the ciphertext vector back into characters to form the final ciphertext.

Decryption Process:

1. Inverse Key Matrix:
   Calculate the inverse of the key matrix K^{-1}.
2. Ciphertext Blocking:
   Divide the ciphertext into blocks of size n.
3. Matrix Multiplication:
   For each ciphertext block C, convert it to a column vector. Multiply the ciphertext vector by the inverse key matrix K^{-1} and take the result modulo 26 (or 256 for ASCII characters) to obtain the plaintext vector (C): $[C = CK^{-1} \ mod \ 26]$
4. Plaintext Conversion:
   Convert the numerical values in the plaintext vector back into characters to form the original plaintext.

Handling Non-Multiples of Block Size:

If the number of letters in the plaintext is not a multiple of the block size n, padding characters are added to the plaintext to complete the last block. For example, if the key matrix K is 2×2 and the plaintext is 5 letters long, one padding character is added to make the plaintext 6 letters long. The padding character is usually chosen to be a character that does not change the meaning of the message, such as 'X'.

Handling Numbers:

To accommodate numerical digits in the Hill Cipher, the character-to-number mapping is extended to include numbers in addition to letters. For example, digits '0' to '9' can be mapped to numerical values 27 to 36, respectively. This extension allows the Hill Cipher to encrypt and decrypt plaintext containing both letters and numbers. During encryption and decryption, these numerical values are processed together with the alphabetic characters, preserving the cipher's block-based and position-dependent variability. By extending the mapping in this

way, the Hill Cipher becomes capable of handling a wider variety of plaintexts that include both alphabetic and numerical characters.

2.5 Hill Cipher in Modern Cryptography

Today, the Hill Cipher is not widely used in modern cryptography. While it was an important step in the early days of encryption by using linear algebra and matrices, it has some weaknesses that make it unsuitable for today's security needs. One of the main issues is that the Hill Cipher can be easily broken if an attacker knows some of the plaintext and its corresponding ciphertext. This makes it vulnerable to attacks that modern encryption methods are designed to resist.

Modern encryption techniques, like AES (Advanced Encryption Standard), offer much stronger security and are built to protect data against a wide range of attacks. These newer methods are more complex and provide better protection for sensitive information in our digital world.

Despite not being used for serious security purposes today, the Hill Cipher is still valuable for educational reasons. It helps students understand basic ideas in linear algebra and how they can be applied to encrypt messages. Learning about the Hill Cipher gives a good foundation for studying more advanced and secure encryption methods used in today's technology. In summary, while the Hill Cipher is outdated for practical use, it remains a useful tool for learning and understanding the basics of cryptography.

III. IMPLEMENTATION

The Hill Cipher is implemented using Python and is written in a single file. The program performs encryption and decryption operations, receiving user inputs through main menu, and handles key matrix validations. Below is a detailed explanation of each part of the code, which are paired with corresponding screenshots for better understanding.

```python
import numpy as np

def char_to_num(char):
    if char == ' ':
        return 0
    return ord(char) - ord('A') + 1

def num_to_char(num):
    if num == 0:
        return ' '
    return chr(num + ord('A') - 1)
```

Fig. 3.1 Code Implementation (1)
Source: Author's document

1. Importing Libraries and Character Mapping

   The code starts by importing the NumPy library, which is needed to handle matrix operations efficiently.

   a) char_to_num function converts each character in the plaintext to a corresponding numerical value, mapping 'A' to 1, 'B' to 2, up to 'Z' mapping to 26, and space mapping to 0.
   b) num_to_char function reverses this process by converting numerical values back to their respective characters, with 0 mapping to space.

```python
def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None
```

Fig. 3.2 Code Implementation (2)
Source: Author's document

2. Calculating Modular Inverse

   The mod_inverse function computes the modular inverse of a given number a under modulo m. It iterates through possible values to find an integer x such that (a * x) % m == 1. If no such x exists within the range, the function returns None, indicating that the inverse does not exist.

```python
def inverse_key_matrix(K):
    det = int(np.round(np.linalg.det(K)))
    det_inv = mod_inverse(det, 26)
    if det_inv is None:
        raise ValueError("Key matrix is not invertible")

    adjugate = np.array([[K[1, 1], -K[0, 1]], [-K[1, 0], K[0, 0]]])
    K_inv = (det_inv * adjugate) % 26
    return K_inv
```

Fig. 3.3 Code Implementation (3)
Source: Author's document

3. Inverting the Key Matrix

   The inverse_key_matrix function calculates the inverse of the key matrix K required for decryption.

   a) First, it computes the determinant of K and then finds its modular inverse with respect to 26.
   b) If the determinant does not have an inverse modulo 26, the function raises a ValueError, indicating that the key matrix cannot be inverted, thus, invalid for the cipher operation.
   c) Otherwise, it computes the adjugate matrix, multiplies it by the determinant's inverse, and applies modulo 26 to obtain the final inverse key matrix K_inv.

```python
def matrix_multiply_mod(matrix1, matrix2, mod):
    res = [[0 for _ in range(len(matrix2[0]))] for _ in range(len(matrix1))]
    for i in range(len(matrix1)):
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):
                res[i][j] += matrix1[i][k] * matrix2[k][j]
            res[i][j] %= mod
    return res
```

Fig. 3.4 Code Implementation (4)
Source: Author's document

4. Matrix Multiplication with Modulo Operation

The matrix_multiply_mod function performs matrix multiplication between matrix1 and matrix2, followed by a modulo operation on each element of the resulting matrix.

a) It uses nested loops to multiply corresponding elements and accumulate the results.
b) After each row-column multiplication, it applies the modulo operation to ensure the results stay within the bounds defined by mod (usually 26 for alphabetical characters).

```python
def encrypt_hill_cipher(plaintext, key_matrix):
    plaintext = plaintext.upper().replace(' ', '')
    if len(plaintext) % key_matrix.shape[0] != 0:
        plaintext += 'X' * (key_matrix.shape[0] - len(plaintext) % key_matrix.shape[0])

    P = [char_to_num(char) for char in plaintext]
    P = np.array(P).reshape(-1, key_matrix.shape[0])

    C = []
    for i in range(P.shape[0]):
        col = P[i, :].reshape(-1, 1)
        encrypted_col = matrix_multiply_mod(col.T, key_matrix, 26)
        C.extend(np.array(encrypted_col).flatten())

    ciphertext = ''.join(num_to_char(num) for num in C)

    print("\n--- Mapping of Original to Ciphered Characters ---")
    for i in range(len(plaintext)):
        print(f"{plaintext[i]} -> {ciphertext[i]}")

    return ciphertext
```

Fig. 3.5 Code Implementation (5)
Source: Author's document

5. Encrypting the Plaintext

The encrypt_hill_cipher function handles the encryption process by performing the following steps:

a) Text Preparation: Converts the plaintext to uppercase and removes any spaces to ensure consistency.
b) Padding: Adds padding characters ('X') if the plaintext length is not a multiple of the key matrix size, ensuring that the plaintext can be evenly divided into blocks matching the matrix dimensions.
c) Mapping: Translates the plaintext characters into their numerical equivalents using the char_to_num function.
d) Reshaping: Arranges the numerical plaintext into a matrix where each row corresponds to a block of the size of the key matrix.
e) Encryption: Multiplies each plaintext block by the key matrix using the matrix_multiply_mod function to produce ciphertext blocks.
f) Conversion: Converts the resulting numerical ciphertext back into characters using the num_to_char function.
g) Debugging: Prints each plaintext character to its corresponding ciphertext character mapper for a better understanding of the encryption process.
h) Output: Returns the final ciphertext as a string.

```python
def decrypt_hill_cipher(ciphertext, key_matrix):
    K_inv = inverse_key_matrix(key_matrix)
    C = [char_to_num(char) for char in ciphertext]
    C = np.array(C).reshape(-1, key_matrix.shape[0])

    P = []
    for i in range(C.shape[0]):
        col = C[i, :].reshape(-1, 1)
        decrypted_col = matrix_multiply_mod(col.T, K_inv, 26)
        P.extend(np.array(decrypted_col).flatten())

    plaintext = ''.join(num_to_char(num) for num in P)
    return plaintext.rstrip()
```

Fig. 3.6 Code Implementation (6)
Source: Author's document

6. Decrypting the Ciphertext

The decrypt_hill_cipher function handles the decryption to obtain the original plaintext from the ciphertext by performing following steps:

a) Inversion: Calculates the inverse of the key matrix K_inv using the inverse_key_matrix function. This inverse will be used to decrypt the ciphertext back to plaintext.
b) Mapping: Converts the ciphertext characters into their numerical counterparts using the char_to_num function.
c) Reshaping: Arranges the numerical ciphertext into a matrix that matches the key matrix size, ensuring each block aligns correctly for decryption.
d) Decryption: Multiplies each ciphertext block by the inverse key matrix K_inv using the matrix_multiply_mod function to produce plaintext blocks.
e) Conversion: Translates the numerical plaintext back into characters using the num_to_char function.
f) Cleaning: Removes all padding characters that were added during the encryption process.
g) Output: Returns the decrypted plaintext as a string.

```python
def main():
    # Input for key size
    n = int(input("Enter the key size (n): "))

    # Input for the key matrix
    print(f"Enter your {n}x{n} key matrix (each row in a new line, columns separated by space):")
    rows = []
    for _ in range(n):
        row = list(map(int, input().split()))
        rows.append(row)
    key_matrix = np.array(rows)

    while True:
        print("\nSelect an operation:")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Find Inverse Key Matrix")
        print("4. Exit")
        choice = input("Enter your choice (1/2/3/4): ")

        if choice == '1':
            # Input for plaintext
            plaintext = input("Enter plaintext to encrypt: ")
            # Perform encryption
            ciphertext = encrypt_hill_cipher(plaintext, key_matrix)
            print("\n--- Encryption Result ---")
            print("Ciphertext:", ciphertext)

        elif choice == '2':
            # Input for ciphertext
            ciphertext = input("Enter ciphertext to decrypt: ")
            # Perform decryption
            decrypted_text = decrypt_hill_cipher(ciphertext, key_matrix)
            print("\n--- Decryption Result ---")
            print("Decrypted Text:", decrypted_text)

        elif choice == '3':
            # Find and display the inverse key matrix
            try:
                K_inv = inverse_key_matrix(key_matrix)
                print("\n--- Inverse Key Matrix ---")
                print(K_inv)
            except ValueError as e:
                print(e)

        elif choice == '4':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice. Please select 1, 2, 3, or 4.")

if __name__ == "__main__":
    main()
```

Fig. 3.7 Code Implementation (7)
Source: Author's document

7. Main Function

The main function serves as the primary interface for user interaction, guiding users through the encryption and decryption processes:

a) Accepting User Input:
- Key Size: Prompts the user to enter the size of the key matrix (n), determining the block size for encryption and decryption.
- Key Matrix: Instructs the user to input the key matrix row by row, with each element separated by a space. It collects these rows into a list and converts them into a NumPy array for matrix operations.

b) Menu-Driven Interface:
- Encryption Option: Allows the user to input plaintext, which is then encrypted using the encrypt_hill_cipher function. The resulting ciphertext is displayed.
- Decryption Option: Enables the user to input ciphertext, which is decrypted back to plaintext using the decrypt_hill_cipher function. The decrypted text is displayed.
- Inverse Key Matrix Option: Provides the user with the inverse of the current key matrix, calculated using the inverse_key_matrix function. If the key matrix is not invertible, an error message is shown.
- Exit Option: Offers the user the ability to terminate the program gracefully.

c) Validation: Ensures that the user selects a valid option from the menu. If an invalid choice is made, it prompts the user to select a valid option again.

IV. Result Discussion

The testing results were obtained from a series of test cases that covered various scenarios that might occur when using the Hill Cipher. This testing process was conducted to evaluate the functionality and reliability of the Hill Cipher implementation in meeting user needs. The tests included different types of input texts and key matrix sizes to ensure that the cipher works correctly under various conditions.

1. Program Entry

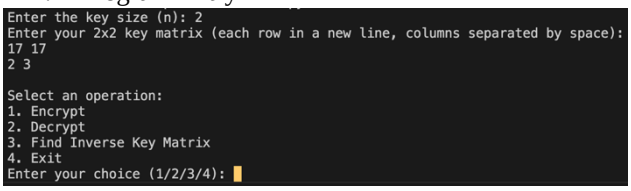

Fig. 4.1 Screenshot of Program Entry
Source: Author's document

As shown above, a key size of 2, indicating a 2 × 2 key matrix is given as an input.

2. Encryption

Fig. 4.2 Screenshot of Encryption Process
Source: Author's document

Explanation:
- Plaintext Processing: The plaintext "HELLO WORLD" was converted to uppercase and spaces were removed, resulting in "HELLOWORLD".

- Padding: Since the key matrix size is 2x2 and "HELLOWORLD" has 10 characters (which is already a multiple of 2), no padding was needed.
- Encryption Steps:
  - Each pair of characters (HE, LL, OW, OR, LD) was converted to their numerical equivalents.
  - The key matrix was used to encrypt each pair, resulting in the ciphertext "PUTFOLEWDH".
- Mapping Display: The program printed a mapping of each plaintext character to its corresponding ciphertext character for verification.

3. Decryption

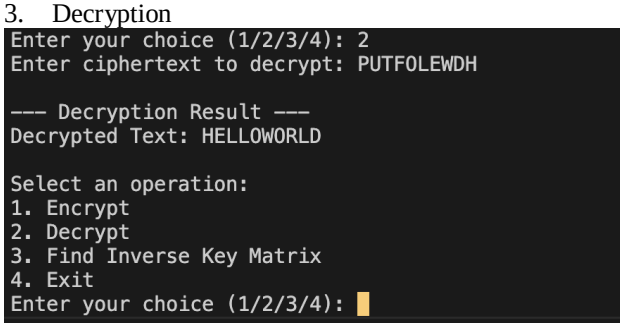

Fig. 4.3 Screenshot of Decryption Process
Source: Author's document

Explanation:

- Ciphertext Processing: The ciphertext "PUTFOLEWDH" was entered for decryption.
- Decryption Steps:
  - The inverse of the key matrix was calculated.
  - Each pair of ciphertext characters was converted back to numerical values.
  - The inverse key matrix was used to decrypt each pair, resulting in the plaintext "HELLOWORLD".
- Result Verification: The decrypted text accurately matched the original plaintext, confirming the correctness of both encryption and decryption processes.

4. Calculating Inverse Key Matrix

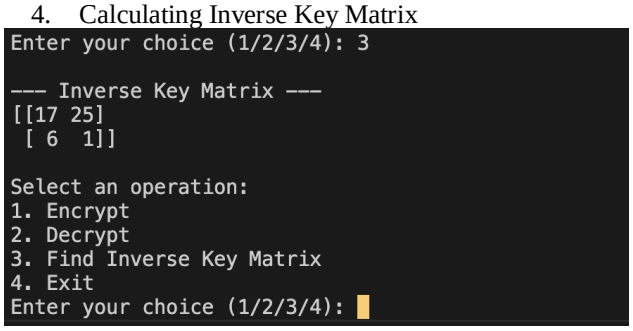

Fig. 4.4 Screenshot of Inverse Key Calculation Process
Source: Author's document

Explanation: The program calculated the inverse of the provided key matrix

V. Conclusion

The Hill Cipher marked a significant milestone in modern cryptography by incorporating linear algebra into the encryption process. By utilizing matrix multiplication and vector spaces, it demonstrated how algebraic techniques could enhance security against basic frequency analysis attacks. Although the cipher is vulnerable to known-plaintext attacks and does not meet the stringent security requirements of contemporary encryption systems, it effectively highlighted both the strengths and limitations of algebraic approaches in cryptography. The foundational use of linear algebra in the Hill Cipher has paved the way for the development of more advanced block ciphers and continues to serve as an invaluable educational tool for understanding the mathematical principles underlying modern encryption techniques.

VII. Acknowledgment

All praise and gratitude to God for the blessings and guidance throughout the process of learning and writing. Additionally, sincere thanks are extended to the lecturers of the IF2123 course, particularly Dr. Judhi Santoso, M.Sc., and Mr. Arrival Dwi Sentosa, S.Kom., M.T., for their invaluable support and teachings. The author also wishes to thank family and friends for their unwavering support throughout the entire semester. It is hoped that this paper will be beneficial to both the author and others.

References

[1] H. Anton and C. Rorres, Elementary Linear Algebra: Applications Version, 11th ed. Hoboken, NJ, USA: John Wiley & Sons, 2013.
[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-03-Sistem-Persamaan-Linier-2023.pdf Diakses pada 27 Desember 2024
[3] J. Katz and Y. Lindell, Introduction to Modern Cryptography, 2nd ed. Boca Raton, FL, USA: CRC Press, 2015.

STATEMENT OF ORIGINALITY

I hereby declare that this paper is my own writing, not an adaptation, or translation of someone else's paper, and not plagiarized

Bandung, 2 Januari 2025

I Made Wiweka Putera - 13523160