

CSE 721: Introduction to Cryptography

Project Report

Classical Cryptography Tool and Hill Cipher Cracker



Inspiring Excellence

Submitted to
Md Sadek Ferdous, PhD
Professor
Department of Computer Science and Engineering

Submitted by
Mekhala Mariam Mary
Student ID: 17101368

A. S. M. Sabiqul Hassan
Student ID: 1000055716

1 Introduction

Cryptography plays a fundamental role in ensuring confidentiality, integrity, and security of information in communication systems. Before the advent of modern cryptographic algorithms, a variety of classical encryption techniques were developed to protect sensitive messages from unauthorized access.

The cryptographic algorithms implemented in this project include the Caesar Cipher, Affine Cipher, Playfair Cipher, and the Hill Cipher using a 2×2 key matrix. Each cipher represents a different stage in the evolution of classical cryptography, ranging from simple monoalphabetic substitution to more mathematically complex polygraphic encryption schemes. In addition to these encryption and decryption modules, a Hill Cipher cracker tool has been developed to recover the secret key using known plaintext-ciphertext pairs, thereby illustrating a practical cryptanalytic attack.

Overall, this project provides a structured and practical exploration of classical cryptography and cryptanalysis. By combining theoretical explanations with implementation-level details, the tool serves as an educational platform for understanding both the functionality and the inherent weaknesses of classical encryption techniques, reinforcing key concepts that are essential for the study of modern cryptography and information security.

This project focuses on the design and implementation of a comprehensive classical cryptography tool that supports both encryption and decryption operations using multiple well-known classical cipher systems. The objectives of this project are threefold:

- The project aims to design and implement a user-friendly software tool that allows users to interactively select a cryptographic algorithm, specify encryption or decryption operations, and provide the required keys and input messages.
- The project seeks to explore and explain the underlying mathematical principles governing classical cryptographic systems, including substitution, modular arithmetic, and linear algebra. Through hands-on implementation, users gain insight into the strengths and limitations of these techniques.
- The project demonstrates the vulnerability of certain classical ciphers by implementing a cryptanalysis tool that performs a known plaintext attack on the Hill Cipher, highlighting the importance of cryptographic robustness and secure key management.

2 Background: Basics of Cryptography

Cryptography is the science of protecting information by transforming it into a secure form that prevents unauthorized access. It plays a vital role in ensuring confidentiality and secure communication in digital systems. The core concepts of cryptography include encryption, decryption, cryptographic keys, and algorithms.

Encryption is the process of converting readable data, known as plaintext, into an unreadable format called ciphertext using a cryptographic algorithm and a secret key. Mathematically, encryption can be represented as:

$$C = E(P, K)$$

Decryption is the reverse process of encryption, where ciphertext is transformed back into plaintext using a corresponding key:

$$P = D(C, K)$$

Where,

- P represents the plaintext,
- K represents the encryption key,
- E denotes the encryption algorithm,
- D denotes the decryption algorithm,
- C represents the resulting ciphertext.

In symmetric-key cryptography, the same key (or a related key) is used for both encryption and decryption. Classical cryptographic systems fall under this category and primarily rely on substitution and mathematical transformations such as modular arithmetic and matrix operations.

Classical cryptography, although insecure by modern standards, is essential for understanding fundamental cryptographic principles and the evolution of encryption techniques. Cryptanalysis, the study of breaking cryptographic systems, further highlights the limitations of classical ciphers and emphasizes the importance of strong cryptographic design.

3 System Architecture

The cryptographic application is designed using a modular and layered architecture, where each cryptographic component is implemented as an independent module with clearly defined responsibilities. This architectural approach improves code readability, scalability, and maintainability, while also enabling the cryptographic algorithms and cryptanalysis components to be developed, tested, and extended independently.

At a high level, the system is composed of three primary layers: the User Interface Layer, the Cipher Processing Layer, and the Cryptanalysis Layer in Figure 2.1. Each layer interacts with the others through well-defined interfaces, ensuring separation of concerns and minimizing interdependencies between modules.

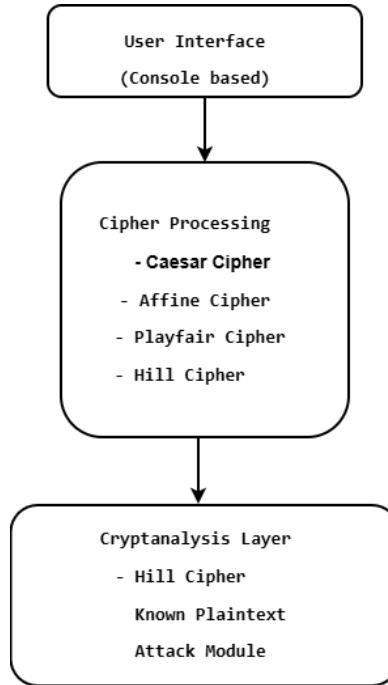


Figure 2.1 Conceptual Architecture Diagram

4 Development Language and Libraries

4.1 Programming Language

The combined use of Python, Google Colab Notebook, and supporting libraries provides a flexible development environment for implementing classical cryptographic algorithms and cryptanalysis techniques. Google Colab facilitates rapid experimentation and validation, while Python and its libraries ensure accurate and maintainable.

4.2 Libraries Used

In Table 4.2.1, the used libraries are described shortly.

Table 4.2.1: Libraries Used

Library	Purpose
numpy[3]	Matrix operations for Hill Cipher
math[4]	GCD calculation and determinant checks
Built-in Python[5]	Text processing and user interaction

5 Cipher Operations Description

This section describes the theoretical foundations and implementation details of the classical cryptographic ciphers implemented in this project. Each cipher is discussed in terms of its encryption and decryption process, mathematical formulation, and practical realization in the developed tool. Different cipher operations are shortly described in Table 5.1.

Table 5.1: summary of Caesar, Affine, Playfair, and Hill Cipher

Cipher	Type	Key Size
Caesar	Monoalphabetic	1 integer
Affine	Monoalphabetic	2 integers
Playfair	Digraph	Keyword
Hill (2×2)	Polygraphic	4 integers

5.1 Caesar Cipher

The Caesar Cipher is one of the earliest and simplest substitution ciphers. It works by shifting each letter of the plaintext by a fixed number of positions down the alphabet [1].

Let:

- (P) be the numerical representation of a plaintext character
- (C) be the numerical representation of the ciphertext character
- (k) be the secret key (shift value)

Encryption is defined as:

$$C = (P + k) \bmod 26$$

Decryption is defined as:

$$P = (C - k) \bmod 26$$

Because the Caesar Cipher uses a monoalphabetic substitution, it is highly vulnerable to brute-force and frequency analysis attacks [1].

The Caesar Cipher was implemented by first converting the input plaintext or ciphertext to uppercase and removing any non-alphabetic characters. Each character is mapped to a numerical value using Python's built-in `ord()` and `chr()` functions [5], where A is mapped to 0 and Z to 25. During encryption, the numerical value of each character is shifted forward by a fixed key value using modulo 26 arithmetic to ensure wrap-around at the end of the alphabet. Decryption applies the same logic by subtracting the key value instead of adding it. This cipher does not require any external libraries, as all operations are handled using built-in Python functions, making the implementation simple and efficient. This code from Figure 5.1.1 shows implementation of Caesar encryption and decryption mechanism.

```

# Caesar Cipher

def caesar_encrypt(text, key):
    result = ""
    for char in text:
        if char.isalpha():
            shift = (ord(char.upper()) - 65 + key) % 26
            result += chr(shift + 65)
        else:
            result += char
    return result

def caesar_decrypt(text, key):
    return caesar_encrypt(text, -key)

```

Figure 5.1.1 Coding implementation of Caesar Cipher

5.2 Affine Cipher

The Affine Cipher is an extension of the Caesar Cipher that uses two keys instead of one. It applies a linear transformation to each plaintext character.

Let:

- (α) and (β) be the keys such that $\gcd(\alpha, 26) = 1$
- (x) be the plaintext character value

Encryption:

$$y = (\alpha x + \beta) \bmod 26$$

Decryption:

$$x = \frac{1}{\alpha} (y - \beta) \bmod 26$$

Here, ($\frac{1}{\alpha}$) is the modular multiplicative inverse of (α) modulo 26. If (α) is not coprime with 26, decryption becomes impossible [1]-[2].

```

▶ # Affine Cipher

def modinv(a, m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    raise ValueError("No modular inverse")

def affine_encrypt(text, a, b):
    result = ""
    for char in text:
        if char.isalpha():
            result += chr(((a*(ord(char.upper())-65)+b)%26)+65)
        else:
            result += char
    return result

def affine_decrypt(text, a, b):
    a_inv = modinv(a, 26)
    result = ""
    for char in text:
        if char.isalpha():
            result += chr((a_inv*((ord(char.upper())-65)-b)%26)+65)
        else:
            result += char
    return result

```

Figure 5.2.1 Coding implementation of Affine Cipher

The Affine Cipher implementation extends the Caesar Cipher by introducing two keys, α and β . Before performing encryption or decryption, the program validates that the key α is coprime with 26 using the `gcd()` function from Python's math library [4]. Encryption is performed using the formula of y , where each plaintext character is first converted to its numerical equivalent. For decryption, the modular multiplicative inverse of α is computed manually, and the inverse transformation $(y - \beta)mod\ 26$ is applied. The use of the math library ensures correct key validation, while all character processing relies on Python's built-in functions[5].This code from Figure 5.2.1 shows implementation of Affine encryption and decryption mechanism.

5.3 Playfair Cipher

The Playfair Cipher is a digraph substitution cipher that encrypts pairs of letters instead of individual letters, increasing resistance to frequency analysis [1].

A 5×5 matrix is generated using a keyword:

- Duplicate letters are removed.
- The letters I and J are treated as a single character.
- Remaining letters of the alphabet are filled sequentially.

Encryption rules:

1. If both letters are in the same row, each letter is replaced by the letter to its right.
2. If both letters are in the same column, each letter is replaced by the letter below it.
3. Otherwise, letters are replaced using the rectangle rule.

Decryption applies the inverse of these operations.



Playfair Cipher

```
def playfair_matrix(key):
    key = "".join(dict.fromkeys(key.upper().replace("J", "I")))
    matrix = []
    for c in key:
        if c not in matrix:
            matrix.append(c)
    for c in "ABCDEFGHIJKLMNPQRSTUVWXYZ":
        if c not in matrix:
            matrix.append(c)
    return [matrix[i:i+5] for i in range(0, 25, 5)]
```

Figure 5.3.1 A 5×5 matrix of Playfair Cipher

The Playfair Cipher was implemented by generating a 5×5 key matrix based on a user-provided keyword. Duplicate letters are removed from the keyword, and the letters 'I' and 'J' are treated as a single character to fit the alphabet into the matrix in Figure 5.3.1. The plaintext is preprocessed by removing spaces, converting to uppercase, inserting a filler character (typically 'X') between repeated letters in a digraph, and padding the plaintext if its length is odd. Encryption and decryption are performed on pairs of letters according to the standard Playfair rules involving row shifts, column shifts, and rectangle substitutions in Figure 5.3.2 and Figure 5.3.3 respectively. The key matrix is stored as a two-dimensional Python list, and all operations are handled using built-in Python data structures without the need for external libraries.

```
▶ def playfair_encrypt(text, key):
    matrix = playfair_matrix(key)
    text = text.upper().replace("J","I").replace(" ", "")
    # prepare digraphs
    i = 0
    digraphs = []
    while i < len(text):
        a = text[i]
        b = ''
        if i+1 < len(text):
            b = text[i+1]
            if a == b:
                b = 'X'
                i += 1
            else:
                i += 2
        else:
            b = 'X'
            i += 1
        digraphs.append(a+b)
    result = ""
    for a,b in digraphs:
        ax, ay, bx, by = -1,-1,-1,-1
        for row in range(5):
            for col in range(5):
                if matrix[row][col]==a:
                    ax, ay = row, col
                if matrix[row][col]==b:
                    bx, by = row, col
        if ax==bx:
            result += matrix[ax][(ay+1)%5] + matrix[bx][(by+1)%5]
        elif ay==by:
            result += matrix[(ax+1)%5][ay] + matrix[(bx+1)%5][by]
        else:
            result += matrix[ax][by] + matrix[bx][ay]
    return result
```

Figure 5.3.2 Encryption of Playfair Cipher

```

def playfair_decrypt(text, key):
    matrix = playfair_matrix(key)
    result = ""
    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        ax, ay, bx, by = -1, -1, -1, -1
        for row in range(5):
            for col in range(5):
                if matrix[row][col]==a:
                    ax, ay = row, col
                if matrix[row][col]==b:
                    bx, by = row, col
        if ax==bx:
            result += matrix[ax][(ay-1)%5] + matrix[bx][(by-1)%5]
        elif ay==by:
            result += matrix[(ax-1)%5][ay] + matrix[(bx-1)%5][by]
        else:
            result += matrix[ax][by] + matrix[bx][ay]
    return result

```

Figure 5.3.3 Decryption of Playfair Cipher

5.4 Hill Cipher (2×2 Matrix)

▶ # Hill Cipher (2x2)

```

def hill_encrypt(text, key_matrix):
    text = text.upper().replace(" ", "")
    if len(text) % 2 != 0:
        text += "X"
    result = ""
    for i in range(0, len(text), 2):
        vec = np.array([[ord(text[i])-65], [ord(text[i+1])-65]])
        enc = np.dot(key_matrix, vec) % 26
        result += chr(enc[0,0]+65) + chr(enc[1,0]+65)
    return result

def hill_decrypt(text, key_matrix):
    text = text.upper().replace(" ", "")
    det = int(np.round(np.linalg.det(key_matrix))) % 26
    det_inv = modinv(det, 26)
    key_inv = det_inv * np.round(det * np.linalg.inv(key_matrix)).astype(int) % 26
    result = ""
    for i in range(0, len(text), 2):
        vec = np.array([[ord(text[i])-65], [ord(text[i+1])-65]])
        dec = np.dot(key_inv, vec) % 26
        result += chr(int(dec[0,0])+65) + chr(int(dec[1,0])+65)
    return result

```

Figure 5.4.1 Hill Cipher (2×2 Matrix)

The Hill Cipher is a polygraphic substitution cipher based on linear algebra. In this project, a 2×2 key matrix is used [1]-[2].

Let:

- (K) be a 2×2 key matrix
- (P) be a plaintext vector
- (C) be a ciphertext vector

If the matrix K-1 is applied to the ciphertext, then the plaintext is recovered

Encryption:

$$C = E(K, P) = PK \bmod 26$$

Decryption:

$$P = D(K, C) = CK^{-1} \bmod 26 = P K K^{-1} = P$$

The key matrix must be invertible modulo 26, which requires:

$$\gcd(\det(K), 26) = 1$$

The Hill Cipher was implemented using a 2×2 key matrix and relies heavily on matrix arithmetic in Figure 5.4.1. The plaintext is converted into numerical vectors and grouped into digraphs, with padding added if necessary. Matrix multiplication is performed modulo 26 to generate the ciphertext. The numpy library [3] is used to represent matrices and perform efficient matrix multiplication. Before encryption or decryption, the determinant of the key matrix is calculated, and its invertibility modulo 26 is verified using the `gcd()` function from the math library. For decryption, the modular inverse of the key matrix is computed manually and then applied to the ciphertext vectors.

6 Input–Output Use Case

This section demonstrates the encryption and decryption processes of four classical cryptographic algorithms using a common plaintext: “MEKHALA”. Using a single plaintext across all ciphers enables clear comparison of cipher behavior, key usage, and transformation characteristics.

6.1 Caesar Cipher Use Case

To demonstrate monoalphabetic substitution using a fixed shift value.

Encryption

Input Parameters

- Cipher: Caesar Cipher
- Operation: Encryption
- Plaintext: MEKHALA
- Key (Shift): 3

Encryption Process

Each letter in the plaintext is shifted forward by 3 positions in the alphabet.

Letter M E K H A L A

Shifted P H N K D O D

```
Classic Cipher Tool
1. Caesar Cipher
2. Affine Cipher
3. Playfair Cipher
4. Hill Cipher (2x2)
5. Exit
Select Cipher: 1
Encrypt or Decrypt (E/D): E
Enter text: Mekhala
Enter key (integer): 3
Ciphertext: PHNKDOD
```

```
Classic Cipher Tool
1. Caesar Cipher
2. Affine Cipher
3. Playfair Cipher
4. Hill Cipher (2x2)
5. Exit
Select Cipher: 1
Encrypt or Decrypt (E/D): D
Enter text: PHNKDOD
Enter key (integer): 3
Plaintext: MEKHALA
```

Figure 6.1.1 output of Caesar Cipher

Figure 6.1.1 displays the output for encrypting “Mekhala” with key=3 and decrypted answer for “” with key=3.

Decryption

Input Parameters

- Ciphertext: PHNKDOD
- Key (Shift): 3

6.2 Affine Cipher Use Case

To demonstrate linear transformation using multiplicative and additive keys.

Encryption

Input Parameters

- Cipher: Affine Cipher
- Operation: Encryption
- Plaintext: MEKHALA
- Key:
 - $\alpha = 5$ & $\beta = 8$
 - $(gcd(5,26)=1 \rightarrow \text{valid key})$

Encryption Formula

$$y = (5P + 8) \bmod 26$$

Figure 6.2.1 displays the output for encrypting “Mekhala” with $\alpha = 5$ & $\beta = 8$ and decrypted answer for “QCGRILI” with $\alpha = 5$ & $\beta = 8$.

```
*** Classic Cipher Tool
1. Caesar Cipher
2. Affine Cipher
3. Playfair Cipher
4. Hill Cipher (2x2)
5. Exit
Select Cipher: 2
Encrypt or Decrypt (E/D): E
Enter text: Mekhala
Enter key a (coprime with 26): 5
Enter key b: 8
Ciphertext: QCGRILI

Classic Cipher Tool
1. Caesar Cipher
2. Affine Cipher
3. Playfair Cipher
4. Hill Cipher (2x2)
5. Exit
Select Cipher: 2
Encrypt or Decrypt (E/D): D
Enter text: QCGRILI
Enter key a (coprime with 26): 5
Enter key b: 8
Plaintext: MEKHALA
```

Figure 6.2.1 output of Affine Cipher

Decryption

Input Parameters

- Ciphertext: QCSXRIL
- Key: $\alpha = 5$ & $\beta = 8$

6.3 Playfair Cipher Use Case

To demonstrate digraph substitution encryption.

Encryption

Input Parameters

- Cipher: Playfair Cipher
- Operation: Encryption
- Plaintext: MEKHALA
- Keyword: MONARCHY

Plaintext Preprocessing

- Plaintext: MEKHALA
- Digraph formation:

ME | KH | AL | AX

- No repeated letters in digraphs
- No padding required

Key Matrix Generation

M O N A R
C H Y B D
E F G I/J K
L P Q S T
U V W X Z

Encryption Result

Applying Playfair rules:

Figure 6.3.1 shows the output of encrypted text for “Mekhala” with key = MONARCHY and decrypted plaintext of “CLFDMSBA” with same key.

```

...
Classic Cipher Tool
1. Caesar Cipher
2. Affine Cipher
3. Playfair Cipher
4. Hill Cipher (2x2)
5. Exit
Select Cipher: 3
Encrypt or Decrypt (E/D): E
Enter text: Mekhala
Enter keyword: MONARCHY
Ciphertext: CLFDMSBA

Classic Cipher Tool
1. Caesar Cipher
2. Affine Cipher
3. Playfair Cipher
4. Hill Cipher (2x2)
5. Exit
Select Cipher: 3
Encrypt or Decrypt (E/D): D
Enter text: CLFDMSBA
Enter keyword: MONARCHY
Plaintext: MEKHALAX

```

Figure 6.3.1 Output of Playfair Cipher

Decryption

Input Parameters

- Ciphertext: CLFDMSBA
- Keyword: MONARCHY

6.4 Hill Cipher (2×2) Use Case

To demonstrate polygraphic substitution using linear algebra and matrix operations.

Encryption

Input Parameters

- Cipher: Hill Cipher
- Operation: Encryption
- Plaintext: MEKHALA
- Key= $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$

```

...
    Classic Cipher Tool
    1. Caesar Cipher
    2. Affine Cipher
    3. Playfair Cipher
    4. Hill Cipher (2x2)
    5. Exit
Select Cipher: 4
Encrypt or Decrypt (E/D): E
Enter text: Mekhala
Enter 2x2 key matrix row-wise (integers 0-25):
Row 1: 3 3
Row 2: 2 5
Ciphertext: WSZDHDR

    Classic Cipher Tool
    1. Caesar Cipher
    2. Affine Cipher
    3. Playfair Cipher
    4. Hill Cipher (2x2)
    5. Exit
Select Cipher: 4
Encrypt or Decrypt (E/D): D
Enter text: WSZDHDR
Enter 2x2 key matrix row-wise (integers 0-25):
Row 1: 3 3
Row 2: 2 5
Plaintext: MEKHALAX

```

Figure 6.4.1 Outcome of Hill Cipher

Plaintext Processing

Plaintext digraphs:

ME | KH | AL | AX

Decryption

Input Parameters

- Ciphertext: WSZDHDR
- Key= $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$

Key= $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$ is used in Figure 6.4.1 to get the ciphertext and the plaintext for “Mekhala” and “WSZDHDR” repectively. This padding case shows the plaintext as “MEKHALAX”,

Table 6.4.1 Outcomes for Usecase

Cipher	Plaintext	Ciphertext	Key Type
Caesar	MEKHALA	PHNKDOD	Shift (3)
Affine	MEKHALA	QCSXRIL	(a=5, b=8)
Playfair	MEKHALA	CLFDMSBA	MONARCHY
Hill	MEKHALA	WSZDHDR	2×2 Matrix

7 Hill Cipher Cracker (Known Plaintext Attack)

```
# Known Plaintext Attack

def hill_cipher_kpa(plaintext, ciphertext):
    P_nums = text_to_numbers(plaintext)
    C_nums = text_to_numbers(ciphertext)

    if len(P_nums) < 4 or len(C_nums) < 4:
        raise ValueError("Need at least 4 characters for known plaintext attack.")

    P = [
        [P_nums[0], P_nums[2]],
        [P_nums[1], P_nums[3]]
    ]

    C = [
        [C_nums[0], C_nums[2]],
        [C_nums[1], C_nums[3]]
    ]

    P_inv = matrix_mod_inv_2x2(P)
    if P_inv is None:
        return None, False

    K = matrix_mul_2x2(C, P_inv)
    return K, True
```

Figure 7.1 Know Plaintext Attack

The cracker accepts known plaintext and ciphertext as input. Both inputs are converted to uppercase and filtered to remove non-alphabetic characters. Each character is then mapped to its numerical equivalent using the standard mapping A=0, B=1,..., Z=25. The text is divided into digraphs to match the 2×2 Hill Cipher structure.

The determinant of the plaintext matrix is calculated using NumPy’s determinant function [3]. The determinant is then reduced modulo 26, and the math library’s GCD [4] function is used to

check whether the determinant is coprime with 26. If the determinant is not invertible modulo 26, the attack cannot proceed using that plaintext selection.

When the plaintext matrix is invertible, the modular inverse of the determinant is computed. The adjugate of the plaintext matrix is calculated, and all values are reduced modulo 26. This step produces the modular inverse of the plaintext matrix, which is essential for recovering the secret key.

The secret key matrix is recovered by multiplying the ciphertext matrix with the inverse of the plaintext matrix using NumPy's matrix multiplication function. The result is reduced modulo 26 to ensure valid alphabetic values. This recovered matrix represents the original encryption key used in the Hill Cipher.

▶ # Verification

```
def encrypt_hill_2x2(plaintext, key):
    nums = text_to_numbers(plaintext)
    result = ""

    for i in range(0, len(nums), 2):
        if i + 1 >= len(nums):
            break

        p = [nums[i], nums[i+1]]
        c1 = (key[0][0] * p[0] + key[0][1] * p[1]) % MOD
        c2 = (key[1][0] * p[0] + key[1][1] * p[1]) % MOD

        result += num_to_char(c1)
        result += num_to_char(c2)

    return result
```

Figure 7.2 Verification for the retrieved key

The Known Plaintext Attack (KPA) exploits the linear nature of the Hill Cipher. If an attacker knows at least two plaintext–ciphertext block pairs, the key matrix can be recovered.

Figure 7.1 illustrates the coding implementation of know plaintext attack and the key retrieval. To verify the retrieved key we used Figure 7.2. To verify correctness, the recovered key matrix is used to re-encrypt the known plaintext. If the resulting ciphertext matches the known ciphertext, the key recovery is considered successful. This verification step ensures that the attack result is accurate. The outcome is showed in Figure 7.3.

```
...
--- Hill Cipher Known Plaintext Attack Tool ---
Enter known plaintext (at least 4 letters): HELP
Enter corresponding ciphertext (at least 4 letters): HIAT
Recovered Key Matrix:
[[3, 3], [2, 5]]
Verification: HIAT
```

Figure 7.3 output of Hill Cipher Cracker

8 Conclusion

This project demonstrates the implementation of classical cryptographic algorithms and highlights the vulnerability of the Hill Cipher against known plaintext attacks.

9 GitHub Repository

Repository Link: <https://github.com/SabiqulHassan13/CSE721-Project-Update/>

The repository contains:

- CSE 721 project part1
- CSE 721 project part2
- CSE 721 project instruction file
- README with execution instructions

Acknowledgement

This project was developed with the assistance of online resources which was used to understand cryptographic algorithms and implementation strategies.

Reference

- [1] Trappe, W. and Washington, L.C. (2020) Introduction to cryptography: With coding theory. Hoboken, NJ: Pearson Education.
- [2] Stallings, W. (2023) Cryptography and network security: Principles and practice. Harlow, United Kingdom: Pearson Education Limited.
- [3] NumPy Developers, “NumPy Documentation,” <https://numpy.org/doc/>.
- [4] Python Software Foundation, “math — Mathematical functions,” <https://docs.python.org/3/library/math.html>.
- [5] Python Software Foundation, “Built-in Functions — Python 3 Documentation,” <https://docs.python.org/3/library/functions.html>.