Cyber Security Project Report -23BCE518

-23BCE530

PROJECT DEFINITION: Cipher Block Chaining (CBC) Encryption and Decryption using Hill Modulo Cipher

1. Abstract

Encryption is a critical aspect of cybersecurity, ensuring data security and confidentiality. This project implements an encryption and decryption system using the Hill Modulo Cipher in Cipher Block Chaining (CBC) mode. The system processes plaintext data, applies matrix-based encryption, and ensures security through block chaining. The project is developed in Python and aims to demonstrate the combination of classical cryptography with modern encryption modes.

2. Introduction

With the increasing threat to digital data security, encryption plays a crucial role in securing information. The Hill Cipher is a symmetric-key cryptosystem based on linear algebra, while Cipher Block Chaining (CBC) is a widely used block cipher mode that introduces dependency between ciphertext blocks to enhance security. This project integrates these techniques to create a robust encryption-decryption system.

3. Objectives

- Implement the Hill Cipher for encryption and decryption.
- Integrate CBC mode to improve security.
- Develop a Python-based implementation for demonstration.
- Ensure correct handling of matrix operations and block chaining.

3.2 Step-by-Step Working of the Project

1. Preprocessing the Input

- The plaintext (message) is taken as input.
- All spaces and special characters are removed (only uppercase letters A-Z are used).
- If the length of the plaintext is not a multiple of the key matrix size, padding is added (e.g., 'X' is added to the end).

Encryption Process 2. Setting Up Key Matrix and IV

- A **key matrix** is chosen (must be invertible under modulo 26 for decryption to work).
- An **Initialization Vector (IV)** is chosen, which is a random block of the same size as the key matrix.

3. Applying Cipher Block Chaining (CBC)

For each block of plaintext:

1. **XOR Operation with Previous Cipher Block or IV**
   - The first block is XORed with the IV.
   - For subsequent blocks, the plaintext block is XORed with the previous ciphertext block.
   - XOR is performed modulo 26 (since we are working with letters A-Z).
2. **Encrypt Using Hill Cipher** o Convert the XORed text into a numerical vector (A=0, B=1, ..., Z=25). o Multiply with the key matrix. o Take modulo 26 to ensure values stay within A-Z.
   - Convert back to letters to get the encrypted block.
3. **Append Encrypted Block to Ciphertext** o The encrypted block is added to the final ciphertext.
   - This block is also used as the previous block for the next round.

Decryption Process

4. Using the Inverse Key Matrix

- The inverse of the key matrix is precomputed for decryption.
- CBC mode is reversed step by step. **5. Decrypt Each Cipher Block**

For each block of ciphertext:

1. **Decrypt Using Hill Cipher** o Convert the encrypted block to a numerical vector.
   - Multiply by the **inverse key matrix**.
   - Take modulo 26.
   - Convert back to letters to get the decrypted XORed block.
2. **Reverse the XOR Operation** o XOR the decrypted block with the previous ciphertext block (or IV for the first block).
   - This recovers the original plaintext block.

3. **Append to Plaintext**
   - The recovered block is added to the final plaintext.
4. **Repeat Until Entire Ciphertext is Decrypted** ○ The process continues for all ciphertext blocks.
   - After decryption, padding (if any) is removed to get the original plaintext.

Example Walkthrough

Encryption Example

Input: lua

CopyEdit

Plaintext: "HELLO"

Key Matrix: [[6, 24], [1, 16]]

IV: "XY"

Steps:

1. **Plaintext Processing:** HELLO → (in numbers) → [7, 4, 11, 11, 14]
2. **Split into blocks (size 2)**: HE, LL, OX (added X for padding)
3. **First Block (HE) XOR with IV (XY)** ○ H = 7, E = 4 ○ X = 23, Y = 24
   - XOR: (7+23 mod 26, 4+24 mod 26) → (4, 2) → "EC"
4. **Encrypt with Hill Cipher** → Generates encrypted block.
5. **Next Block (LL) XOR with previous encrypted block.**
6. **Repeat until all blocks are encrypted.**

Output:

vbnet

CopyEdit

Ciphertext: "RZMGPO"

Decryption Example

Input: lua

CopyEdit

Ciphertext: "RZMGPO"

Key Matrix: [[6, 24], [1, 16]]

Inverse Key Matrix: [[? ?], [? ?]]

IV: "XY"

Steps:

1. **Split Ciphertext into Blocks** → "RZ", "MG", "PO"
2. **Decrypt first block (RZ) using Hill Cipher inverse** → Produces "EC"
3. **Reverse XOR with IV (XY)** → Produces "HE"
4. **Next block (MG) decrypts to an intermediate step.**
5. **XOR with previous ciphertext block** → Produces next plaintext block.
6. **Repeat for all blocks.**

Final Decrypted Output:

vbnet

CopyEdit

Plaintext: "HELLO"

Advantages of CBC with Hill Cipher

✓ **Stronger Security** – Each ciphertext block depends on the previous one, making pattern detection difficult.
✓ **Resistant to Frequency Attacks** – Identical plaintext blocks produce different ciphertext blocks.
✓ **Modular Design** – Can be implemented with different matrix sizes and IVs.

4. Methodology

4.1 Hill Cipher

The Hill Cipher uses a key matrix to transform plaintext into ciphertext using modular arithmetic. The decryption process requires computing the modular inverse of the key matrix.

4.2 Cipher Block Chaining (CBC)

CBC mode introduces an Initialization Vector (IV) and ensures that each ciphertext block depends on the previous one, preventing identical plaintext blocks from encrypting into identical ciphertext blocks.

4.3 Implementation Workflow

1. Convert plaintext into numerical format.
2. Apply CBC encryption by XORing with the previous ciphertext block or IV.

3. Use the Hill Cipher transformation.
4. For decryption, reverse the Hill Cipher and remove the XOR effect.
5. Convert numerical values back to plaintext.

5. Implementation

5.1 Project Folder Structure

```
CBC_Hill_Cipher/
|— cipher.py        # Hill Cipher and CBC functions
|— utils.py        # Helper functions (if needed)
|— main.py         # Main script to run encryption/decryption
|— requirements.txt  # Dependencies (if any)
|— README.md       # Project documentation
```

5.2 Required Dependencies pip

install numpy

5.3 Cipher Implementation

```
Cbc.py
import numpy as np
from src.cipher import hill_encrypt, hill_decrypt

def cbc_encrypt(plaintext_numbers, key_matrix, iv_numbers):
   """Encrypt using Hill Cipher in CBC mode"""
ciphertext = []
   prev_cipher_block = iv_numbers

   for i in range(0, len(plaintext_numbers), 2):
      block = plaintext_numbers[i:i+2]

      # XOR with previous ciphertext block (CBC mode)
      block = [(block[j] + prev_cipher_block[j]) % 26 for j in range(2)]

      # Encrypt with Hill Cipher
      encrypted_block = hill_encrypt(block, key_matrix)
ciphertext.extend(encrypted_block)

      prev_cipher_block = encrypted_block  # Update IV

   return ciphertext

def cbc_decrypt(ciphertext_numbers, key_matrix, iv_numbers):
   """Decrypt using Hill Cipher in CBC mode"""
decrypted_text = []
   prev_cipher_block = iv_numbers
```

```
    for i in range(0, len(ciphertext_numbers), 2):
```

```
        block = ciphertext_numbers[i:i+2]

        # Decrypt with Hill Cipher
        decrypted_block = hill_decrypt(block, key_matrix)

        # XOR with previous ciphertext block (CBC mode)
        decrypted_block = [(decrypted_block[j] - prev_cipher_block[j]) % 26 for j in range(2)]
decrypted_text.extend(decrypted_block)

        prev_cipher_block = block  # Update IV

    return decrypted_text
```

```
Cipher.py import
numpy as np
from src.utils import mod_inverse, MODULO

def hill_encrypt(block, key_matrix):
   """Encrypts a 2-letter block using the Hill Cipher."""
block_vec = np.array(block).reshape(-1, 1)
   encrypted_vec   =   np.dot(key_matrix,   block_vec)   %   MODULO
return encrypted_vec.flatten().tolist()

def hill_decrypt(block, key_matrix):
   """Decrypts a 2-letter block using the Hill
Cipher."""     det =
int(round(np.linalg.det(key_matrix)))     det_inv =
mod_inverse(det, MODULO)

   if det_inv is None:
      raise ValueError("Key matrix is not invertible in mod 26!")

   key_inv = (det_inv * np.round(det * np.linalg.inv(key_matrix)).astype(int)) % MODULO
block_vec = np.array(block).reshape(-1, 1)
   decrypted_vec   =   np.dot(key_inv,   block_vec)   %   MODULO
return decrypted_vec.flatten().tolist()
```

```
Gui.py
import tkinter as tk
from tkinter import messagebox import
numpy as np
from .cbc import cbc_encrypt, cbc_decrypt
from .utils import text_to_numbers, numbers_to_text, pad_text, remove_padding
import os

class       HillCipherGUI(tk.Tk):
def __init__(self):
```

```
        super().__init__()
        self.title("Hill Cipher CBC Mode")
self.geometry("600x400")

        # Hardcoded key matrix from the original project
self.key_matrix = np.array([[3, 3], [2, 5]])

        # Set up assets directory relative to the script location
script_dir = os.path.dirname(os.path.abspath(__file__))
        self.assets_dir = os.path.normpath(os.path.join(script_dir, "..", "assets"))

        # Initialize the GUI widgets
self.create_widgets()

    def create_widgets(self):
        """Create and arrange all GUI widgets."""
        # Plaintext input
        tk.Label(self, text="Plaintext:").grid(row=0, column=0, sticky="e", padx=5, pady=5)
self.plaintext_text = tk.Text(self, height=5, width=50)
        self.plaintext_text.grid(row=0, column=1, columnspan=2, padx=5, pady=5)

        # IV input
        tk.Label(self, text="IV (2 letters):").grid(row=1, column=0, sticky="e", padx=5, pady=5)
self.iv_entry = tk.Entry(self, width=5)
        self.iv_entry.grid(row=1, column=1, sticky="w", padx=5, pady=5)

        # Buttons
        tk.Button(self, text="Encrypt", command=self.encrypt).grid(row=2, column=0, padx=5,
pady=5)
        tk.Button(self, text="Decrypt", command=self.decrypt).grid(row=2, column=1, padx=5,
pady=5)
        tk.Button(self, text="Load from File", command=self.load_from_file).grid(row=2, column=2,
padx=5, pady=5)
        tk.Button(self, text="Save Results", command=self.save_results).grid(row=2, column=3,
padx=5, pady=5)

        # Encrypted text output
        tk.Label(self, text="Encrypted Text:").grid(row=3, column=0, sticky="e", padx=5, pady=5)
self.encrypted_text = tk.Text(self, height=5, width=50)
        self.encrypted_text.grid(row=3, column=1, columnspan=2, padx=5, pady=5)
self.encrypted_text.config(state="disabled")

        # Decrypted text output
        tk.Label(self, text="Decrypted Text:").grid(row=4, column=0, sticky="e", padx=5, pady=5)
self.decrypted_text = tk.Text(self, height=5, width=50)
        self.decrypted_text.grid(row=4, column=1, columnspan=2, padx=5, pady=5)
self.decrypted_text.config(state="disabled")

    def encrypt(self):
```

```
    """Encrypt the plaintext using the Hill Cipher in CBC mode."""
try:
```

```
        # Get and clean input
        plaintext = self.plaintext_text.get("1.0", "end-1c").upper()            iv =
self.iv_entry.get().upper()

        # Validate IV          if len(iv) != 2 or not iv.isalpha():               raise
ValueError("IV must be exactly 2 uppercase letters.")

        # Prepare plaintext
        plaintext = pad_text(plaintext)
        plaintext_nums = text_to_numbers(plaintext)          iv_nums =
text_to_numbers(iv)

        # Perform encryption
        ciphertext_nums = cbc_encrypt(plaintext_nums, self.key_matrix, iv_nums)
ciphertext = numbers_to_text(ciphertext_nums)

        # Display encrypted text          self.encrypted_text.config(state="normal")
self.encrypted_text.delete("1.0", "end")          self.encrypted_text.insert("1.0",
ciphertext)          self.encrypted_text.config(state="disabled")

      except Exception as e:
        messagebox.showerror("Error", str(e))

   def decrypt(self):
     """Decrypt the encrypted text using the Hill Cipher in CBC mode."""        try:
        # Get encrypted text and IV
        ciphertext = self.encrypted_text.get("1.0", "end-1c").upper()          iv =
self.iv_entry.get().upper()

        # Validate IV               if len(iv) != 2 or not
iv.isalpha():
          raise ValueError("IV must be exactly 2 uppercase letters.")

        # Prepare ciphertext
        ciphertext_nums = text_to_numbers(ciphertext)          iv_nums =
text_to_numbers(iv)

        # Perform decryption
        decrypted_nums = cbc_decrypt(ciphertext_nums, self.key_matrix, iv_nums)
decrypted_text = remove_padding(numbers_to_text(decrypted_nums))

        # Display decrypted text          self.decrypted_text.config(state="normal")
self.decrypted_text.delete("1.0", "end")          self.decrypted_text.insert("1.0",
decrypted_text)          self.decrypted_text.config(state="disabled")
```

```
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def load_from_file(self):
        """Load plaintext from assets/plaintext.txt into the plaintext area."""
try:            with open(os.path.join(self.assets_dir, "plaintext.txt"), "r") as
f:
                plaintext = f.read().upper()
self.plaintext_text.delete("1.0", "end")
self.plaintext_text.insert("1.0", plaintext)        except
FileNotFoundError:            messagebox.showerror("Error", "File not
found: plaintext.txt")        except Exception as e:
            messagebox.showerror("Error", str(e))

    def save_results(self):
        """Save encrypted and decrypted text to files in the assets directory."""        try:
with open(os.path.join(self.assets_dir, "encrypted.txt"), "w") as f:
f.write(self.encrypted_text.get("1.0", "end-1c"))            with
open(os.path.join(self.assets_dir, "decrypted.txt"), "w") as f:
f.write(self.decrypted_text.get("1.0", "end-1c"))            messagebox.showinfo("Success",
"Results saved to encrypted.txt and decrypted.txt")        except Exception as e:
            messagebox.showerror("Error", str(e))

if __name__ == "__main__":
    app = HillCipherGUI()
    app.mainloop()
```

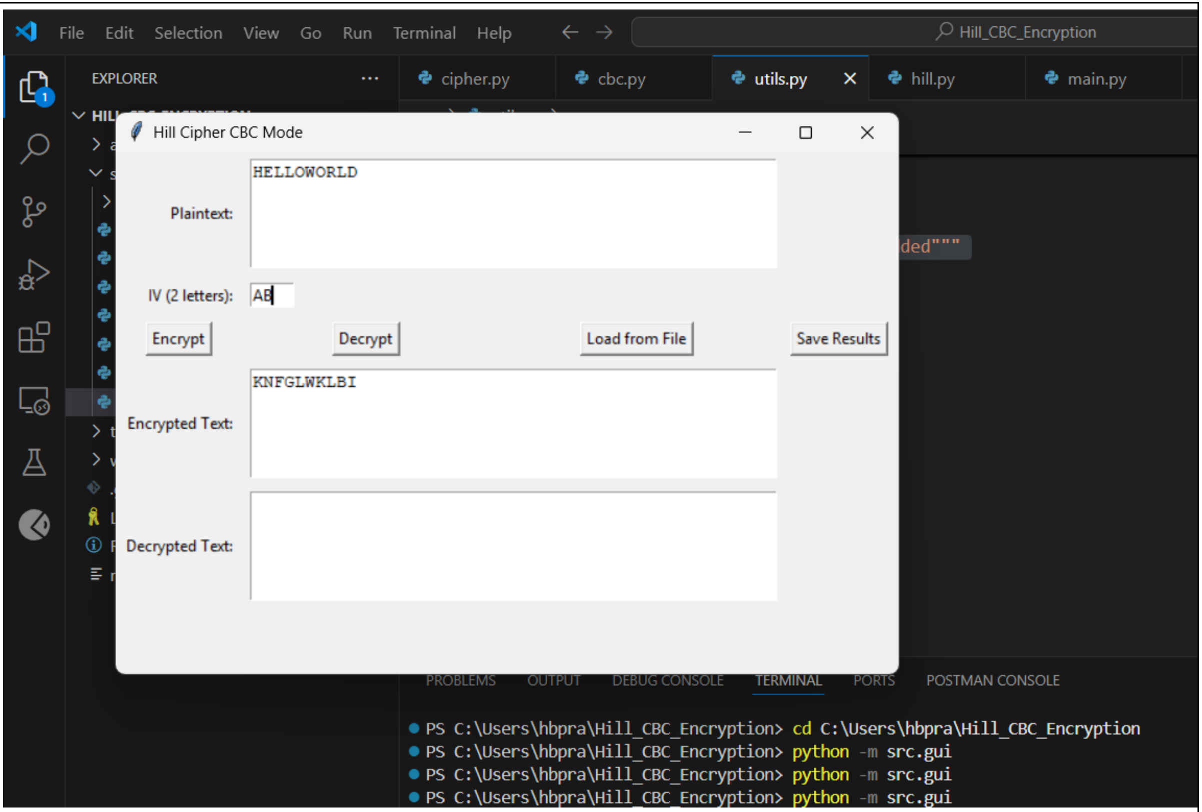
```
Hill.py
import numpy as np
from src.utils import text_to_numbers, numbers_to_text, matrix_mod_inverse, MODULO

def hill_encrypt(block, key_matrix):
    """Encrypt a 2-letter block using the Hill Cipher"""
block_vector = np.array(block).reshape(2, 1)
    encrypted_vector = (key_matrix @ block_vector) % MODULO
return encrypted_vector.flatten().tolist()

def hill_decrypt(block, key_matrix):
    """Decrypt a 2-letter block using the Hill Cipher"""
inverse_key_matrix = matrix_mod_inverse(key_matrix)
    block_vector = np.array(block).reshape(2, 1)
    decrypted_vector = (inverse_key_matrix @ block_vector) % MODULO
return decrypted_vector.flatten().tolist()
```

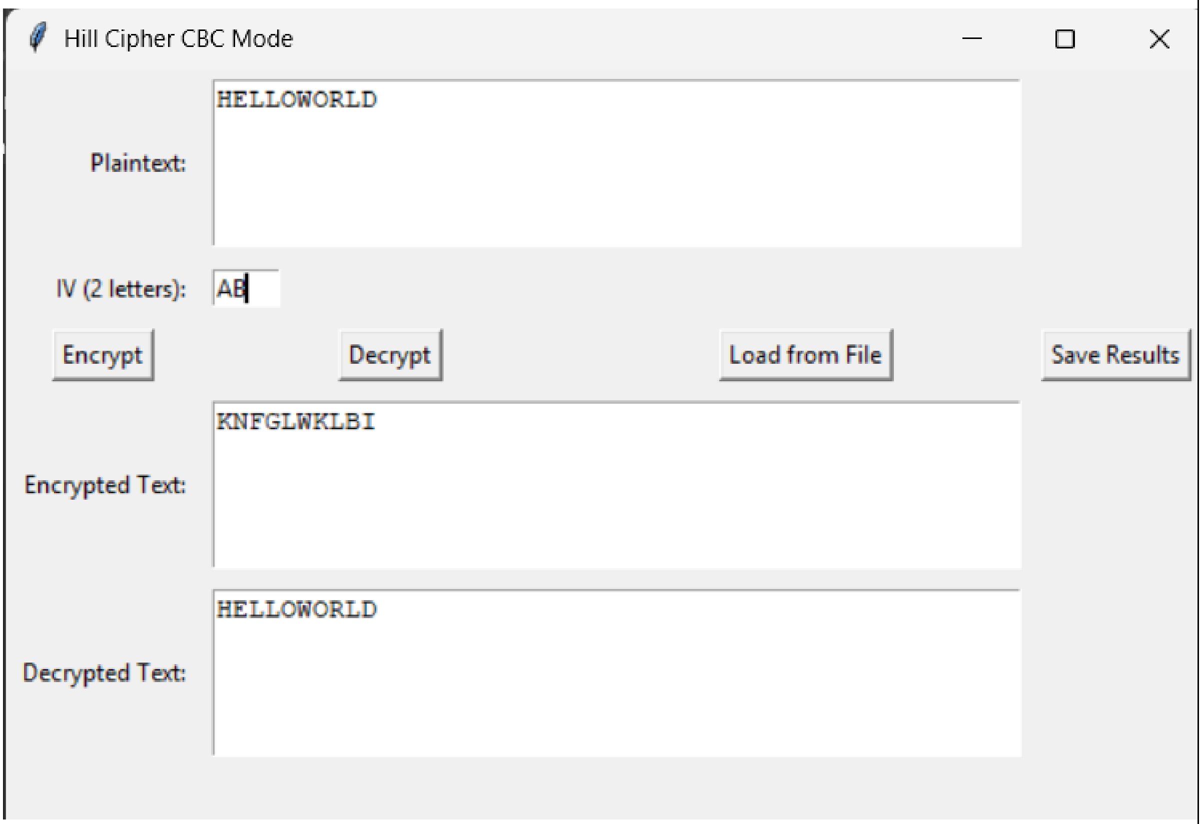
```
Main.py import
sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))

import numpy as np
from src.cbc import cbc_encrypt, cbc_decrypt
from src.utils import text_to_numbers, numbers_to_text, pad_text, remove_padding

# Define a 2x2 Key Matrix (Example)
KEY_MATRIX = np.array([[3, 3],
               [2, 5]])

# Get user input
plaintext = input("Enter plaintext (uppercase letters only): ").upper() iv
= input("Enter 2-letter Initialization Vector (IV): ").upper()

# Ensure plaintext is padded for 2x2 encryption plaintext
= pad_text(plaintext)

# Convert text to numerical representation
plaintext_nums = text_to_numbers(plaintext) iv_nums
= text_to_numbers(iv)

# Encrypt using CBC mode
ciphertext_nums = cbc_encrypt(plaintext_nums, KEY_MATRIX, iv_nums) ciphertext
= numbers_to_text(ciphertext_nums)

# Decrypt using CBC mode decrypted_nums =
cbc_decrypt(ciphertext_nums, KEY_MATRIX, iv_nums) decrypted_text =
remove_padding(numbers_to_text(decrypted_nums))

# Output results
print(f"Original  : {plaintext}") print(f"Encrypted
: {ciphertext}")
print(f"Decrypted : {decrypted_text}")
```

```
Utils.py import numpy as np

MODULO = 26  # Alphabet size

def text_to_numbers(text):
   """Convert text to a list of numbers (A=0, B=1, ..., Z=25)"""     return [ord(char) - ord('A') for char
in text]
```

```
def numbers_to_text(numbers):
    """Convert a list of numbers to text (0=A, 1=B, ..., 25=Z)"""
    return ''.join(chr(num % MODULO + ord('A')) for num in numbers)

def mod_inverse(a, m=MODULO):
    """Find the modular inverse of 'a' under modulo 'm'"""
a = a % m     for x in range(1, m):        if (a * x) % m == 1:
return x
    raise ValueError(f"No modular inverse for {a} under mod {m}")

def pad_text(text):
    """Pad text to make its length even (for 2x2 Hill Cipher)"""
if len(text) % 2 != 0:
        text += 'X'
    return text

def remove_padding(text):
    """Remove   padding   character   'X'   if   added"""
return text.rstrip('X')
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE

PS C:\Users\hbpra\Hill_CBC_Encryption> cd C:\Users\hbpra\Hill_CBC_Encryption
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui
```


File Edit Selection View Go Run Terminal Help
Hill_CBC_Encryption
EXPLORER
cipher.py
cbc.py
utils.py
hill.py
main.py
Hill Cipher CBC Mode
Plaintext:
HELLOWORLD
IV (2 letters):
AB
Encrypt
Decrypt
Load from File
Save Results
Encrypted Text:
KNFGLWKLBI
Decrypted Text:
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
POSTMAN CONSOLE
PS C:\Users\hbpra\Hill_CBC_Encryption> cd C:\Users\hbpra\Hill_CBC_Encryption
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui
PS C:\Users\hbpra\Hill_CBC_Encryption> python -m src.gui


Hill Cipher CBC Mode
Plaintext:
HELLOWORLD
IV (2 letters):
AB
Encrypt
Decrypt
Load from File
Save Results
Encrypted Text:
KNFGLWKLBI
Decrypted Text:
HELLOWORLD

6. Results & Observations

- The implemented system successfully encrypts and decrypts text using the Hill Cipher and CBC mode.
- The encryption process introduces randomness due to CBC, making it resistant to pattern attacks.
- The system correctly handles different plaintext lengths by padding with 'X'.

7. Conclusion

This project successfully integrates the Hill Cipher with Cipher Block Chaining (CBC) mode to enhance encryption security. The CBC mode ensures that even if two plaintext blocks are identical, their encrypted versions differ, adding an extra layer of security. This demonstrates how classical encryption techniques can be enhanced with modern cryptographic methods.

8. Future Enhancements

- Implementing larger matrix sizes for stronger encryption.
- Using a secure key management system.
- Enhancing the system with cryptographic libraries for better efficiency.