LAPORAN TUGAS BESAR 1
CIPHER CLASSIC

Laporan ini di susun untuk memenuhi salah satu tugas besar mata kuliah kriptografi yang di ampu oleh bapak Kodrat Mahatma S.T.,M.Kom

Di susun oleh :

Mita anggraeni (20123051)

Kelas C1.23

PROGRAM STUDI S1 INFORMATIKA

UNIVERSITAS TEKNOLOGI DIGITAL

2025

A. **TEORI SINGKAT CIPHER CLASSIC**

1. Caesar cipherr

Caesar Cipher adalah salah satu algoritma kriptografi klasik tertua yang menggunakan teknik substitution cipher. Setiap huruf pada plainteks digeser sejauh k posisi dalam alfabet untuk membentuk cipherteks. Misalnya, jika k=3, maka A → D, B → E, dan seterusnya. Meskipun sederhana, cipher ini mudah diserang menggunakan analisis frekuensi karena ruang kunci yang kecil.

2. Vigenere cipher

Vigenère Cipher merupakan pengembangan dari Caesar Cipher yang menggunakan kunci berbentuk kata untuk menentukan pergeseran setiap huruf. Algoritma ini termasuk dalam polyalphabetic substitution cipher, di mana setiap huruf plainteks dienkripsi menggunakan pergeseran berbeda berdasarkan huruf kunci. Hal ini membuatnya lebih tahan terhadap analisis frekuensi dibandingkan Caesar Cipher.

3. Affine cipher

Affine Cipher merupakan bentuk lain dari monoalphabetic substitution cipher yang menggunakan fungsi linear matematika untuk enkripsi, yaitu:

$$C=(aP+b)\bmod 26$$

dengan P adalah huruf plainteks (dalam bentuk numerik), C adalah huruf cipherteks, dan a serta b adalah kunci. Nilai a harus relatif prima dengan 26 agar enkripsi dapat dibalik. Cipher ini menggabungkan operasi multiplication dan addition untuk meningkatkan kompleksitas dibanding Caesar Cipher.

4. Playfair cipher

Playfair Cipher adalah algoritma digraph substitution cipher yang mengenkripsi dua huruf sekaligus menggunakan matriks 5x5 yang diisi dengan kata kunci. Cipher ini meningkatkan keamanan dibanding Caesar dan Vigenère karena analisis frekuensi menjadi lebih sulit dilakukan. Prosesnya melibatkan aturan baris, kolom, dan persegi panjang dalam tabel huruf.

5. Hill cipher

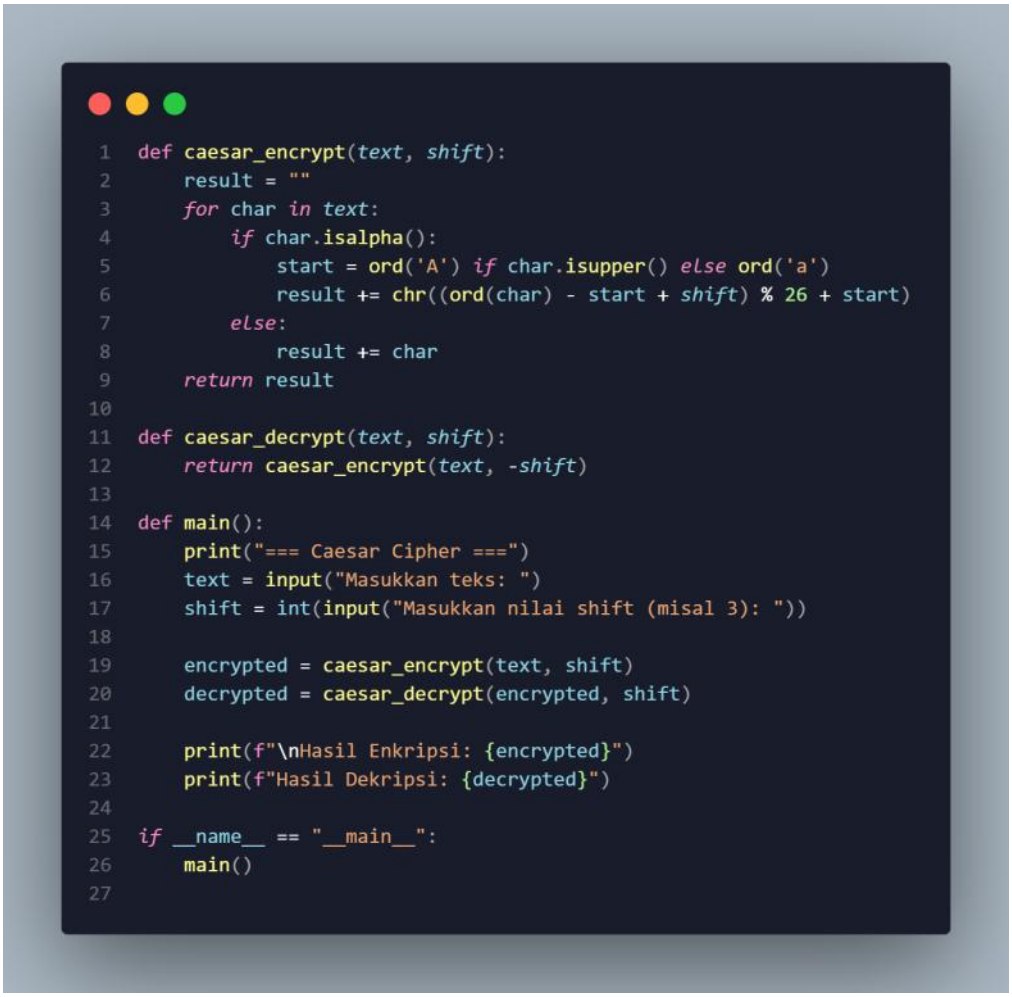
Hill Cipher merupakan polyalphabetic cipher berbasis aljabar linear yang menggunakan matriks kunci untuk melakukan enkripsi blok huruf sekaligus. Setiap blok plainteks dikonversi ke vektor numerik dan dikalikan dengan matriks kunci menggunakan operasi modulo 26. Cipher ini memiliki keamanan yang lebih baik

dibanding cipher klasik lainnya karena melibatkan operasi matriks dan transformasi linear.

B. INPUT DAN OUTPUT CIPHER CLASSIC

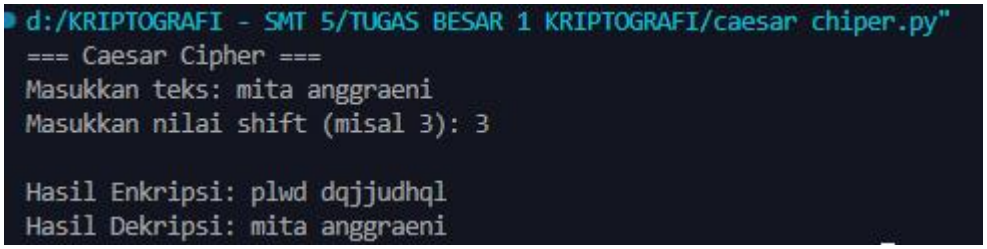
1. Caesar cipher

Input:



```python
def caesar_encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            start = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - start + shift) % 26 + start)
        else:
            result += char
    return result

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

def main():
    print("=== Caesar Cipher ===")
    text = input("Masukkan teks: ")
    shift = int(input("Masukkan nilai shift (misal 3): "))

    encrypted = caesar_encrypt(text, shift)
    decrypted = caesar_decrypt(encrypted, shift)

    print(f"\nHasil Enkripsi: {encrypted}")
    print(f"Hasil Dekripsi: {decrypted}")

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

Output:



```
d:/KRIPTOGRAFI - SMT 5/TUGAS BESAR 1 KRIPTOGRAFI/caesar chiper.py"
=== Caesar Cipher ===
Masukkan teks: mita anggraeni
Masukkan nilai shift (misal 3): 3

Hasil Enkripsi: plwd dqjjudhql
Hasil Dekripsi: mita anggraeni
```

2. Vigenere cipher

Input:

```python
def generate_key(text, key):
    key = list(key)
    if len(key) == len(text):
        return "".join(key)
    else:
        for i in range(len(text) - len(key)):
            key.append(key[i % len(key)])
    return "".join(key)

def vigenere_encrypt(text, key):
    cipher_text = []
    for i in range(len(text)):
        if text[i].isalpha():
            shift = (ord(key[i].lower()) - ord('a'))
            base = ord('A') if text[i].isupper() else ord('a')
            cipher_text.append(chr((ord(text[i]) - base + shift) % 26 + base))
        else:
            cipher_text.append(text[i])
    return "".join(cipher_text)

def vigenere_decrypt(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        if cipher_text[i].isalpha():
            shift = (ord(key[i].lower()) - ord('a'))
            base = ord('A') if cipher_text[i].isupper() else ord('a')
            orig_text.append(chr((ord(cipher_text[i]) - base - shift) % 26 + base))
        else:
            orig_text.append(cipher_text[i])
    return "".join(orig_text)

def main():
    print("=== Vigenere Cipher ===")
    text = input("Masukkan teks: ")
    key = input("Masukkan kunci (keyword): ")
    key_full = generate_key(text, key)

    encrypted = vigenere_encrypt(text, key_full)
    decrypted = vigenere_decrypt(encrypted, key_full)

    print(f"\nHasil Enkripsi: {encrypted}")
    print(f"Hasil Dekripsi: {decrypted}")

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

Output:

```
d:/KRIPTOGRAFI - SMT 5/TUGAS BESAR 1 KRIPTOGRAFI/vigenere chiper.py"
=== Vigenere Cipher ===
Masukkan teks: mita anggraeni
Masukkan kunci (keyword): kucing

Hasil Enkripsi: wcvi gxaiznkxc
Hasil Dekripsi: mita anggraeni
```

3. Affine cipher

Input:

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

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

def affine_decrypt(cipher, a, b):
    result = ""
    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        return "Nilai 'a' tidak memiliki invers modulo 26!"
    for char in cipher:
        if char.isalpha():
            start = ord('A') if char.isupper() else ord('a')
            result += chr(((a_inv * ((ord(char) - start) - b)) % 26) + start)
        else:
            result += char
    return result

def main():
    print("=== Affine Cipher ===")
    text = input("Masukkan teks: ")
    a = int(input("Masukkan nilai a (coprime dengan 26): "))
    b = int(input("Masukkan nilai b: "))

    encrypted = affine_encrypt(text, a, b)
    decrypted = affine_decrypt(encrypted, a, b)

    print(f"\nHasil Enkripsi: {encrypted}")
    print(f"Hasil Dekripsi: {decrypted}")

if __name__ == "__main__":
    main()

```

Output:

```
d:/KRIPTOGRAFI - SMT 5/TUGAS BESAR 1 KRIPTOGRAFI/affine chiper.py"
=== Affine Cipher ===
Masukkan teks: mita anggraeni
Masukkan nilai a (coprime dengan 26): 3
Masukkan nilai b: 8

Hasil Enkripsi: sgni ivaahiuvg
Hasil Dekripsi: mita anggraeni
```

4. Playfair cipher

Input:

```python
def generate_key_matrix(key):
    key = key.upper().replace("J", "I")
    matrix = []
    for char in key:
        if char not in matrix and char.isalpha():
            matrix.append(char)
    for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":
        if char not in matrix:
            matrix.append(char)
    return [matrix[i:i+5] for i in range(0, 25, 5)]

def find_position(matrix, char):
    if char == 'J':
        char = 'I'
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == char:
                return i, j
    return None

def process_text(text):
    text = text.upper().replace("J", "I")
    processed = ""
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else 'X'
        if a == b:
            processed += a + 'X'
            i += 1
        else:
            processed += a + b
            i += 2
    if len(processed) % 2 != 0:
        processed += 'X'
    return processed

def playfair_encrypt(text, key_matrix):
    text = process_text(text)
    result = ""
    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        row1, col1 = find_position(key_matrix, a)
        row2, col2 = find_position(key_matrix, b)
        if row1 == row2:
            result += key_matrix[row1][(col1 + 1) % 5]
            result += key_matrix[row2][(col2 + 1) % 5]
        elif col1 == col2:
            result += key_matrix[(row1 + 1) % 5][col1]
            result += key_matrix[(row2 + 1) % 5][col2]
        else:
            result += key_matrix[row1][col2]
            result += key_matrix[row2][col1]
    return result

def playfair_decrypt(text, key_matrix):
    result = ""
    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        row1, col1 = find_position(key_matrix, a)
        row2, col2 = find_position(key_matrix, b)
        if row1 == row2:
            result += key_matrix[row1][(col1 - 1) % 5]
            result += key_matrix[row2][(col2 - 1) % 5]
        elif col1 == col2:
            result += key_matrix[(row1 - 1) % 5][col1]
            result += key_matrix[(row2 - 1) % 5][col2]
        else:
            result += key_matrix[row1][col2]
            result += key_matrix[row2][col1]
    return result

def main():
    print("=== Playfair Cipher ===")
    key = input("Masukkan kunci: ")
    text = input("Masukkan teks: ")

    matrix = generate_key_matrix(key)
    encrypted = playfair_encrypt(text, matrix)
    decrypted = playfair_decrypt(encrypted, matrix)

    print(f"\nHasil Enkripsi: {encrypted}")
    print(f"Hasil Dekripsi: {decrypted}")

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

output:

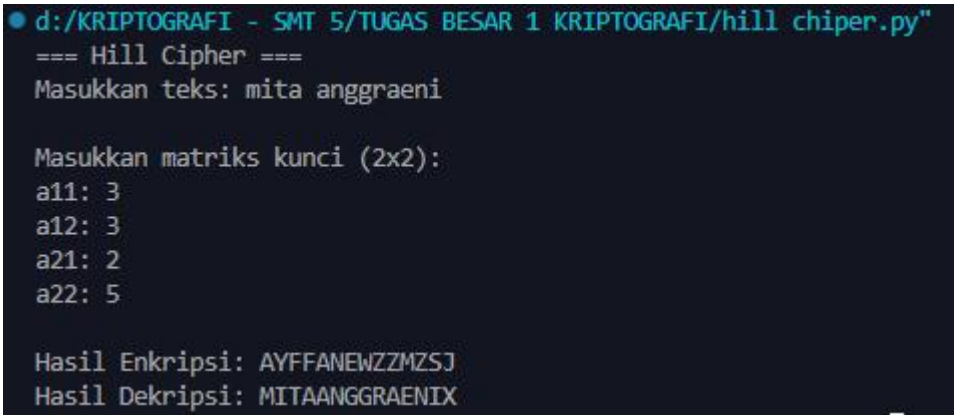
```
d:/KRIPTOGRAFI - SMT 5/TUGAS BESAR 1 KRIPTOGRAFI/playfair chiper.py"
=== Playfair Cipher ===
Masukkan kunci: kunci
Masukkan teks: mita

Hasil Enkripsi: OCPF
Hasil Dekripsi: MITA
```

5. Hill cipher

Input:

```python
import numpy as np

def mod_inverse_matrix(matrix, modulus):
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = pow(det % modulus, -1, modulus)
    matrix_mod_inv = (
        det_inv * np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
    )
    return matrix_mod_inv % modulus

def text_to_numbers(text):
    return [ord(c.upper()) - ord('A') for c in text if c.isalpha()]

def numbers_to_text(numbers):
    return ''.join(chr(n % 26 + ord('A')) for n in numbers)

def hill_encrypt(text, key_matrix):
    text_nums = text_to_numbers(text)
    while len(text_nums) % key_matrix.shape[0] != 0:
        text_nums.append(ord('X') - ord('A'))
    text_matrix = np.array(text_nums).reshape(-1, key_matrix.shape[0])
    encrypted = np.dot(text_matrix, key_matrix) % 26
    return numbers_to_text(encrypted.flatten())

def hill_decrypt(cipher, key_matrix):
    inv_key = mod_inverse_matrix(key_matrix, 26)
    cipher_nums = text_to_numbers(cipher)
    cipher_matrix = np.array(cipher_nums).reshape(-1, key_matrix.shape[0])
    decrypted = np.dot(cipher_matrix, inv_key) % 26
    return numbers_to_text(decrypted.flatten())

def main():
    print("=== Hill Cipher ===")
    text = input("Masukkan teks: ")

    print("\nMasukkan matriks kunci (2x2):")
    a11 = int(input("a11: ")); a12 = int(input("a12: "))
    a21 = int(input("a21: ")); a22 = int(input("a22: "))

    key_matrix = np.array([[a11, a12], [a21, a22]])

    encrypted = hill_encrypt(text, key_matrix)
    decrypted = hill_decrypt(encrypted, key_matrix)

    print(f"\nHasil Enkripsi: {encrypted}")
    print(f"Hasil Dekripsi: {decrypted}")

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

Output:



```
d:/KRIPTOGRAFI - SMT 5/TUGAS BESAR 1 KRIPTOGRAFI/hill chiper.py"
=== Hill Cipher ===
Masukkan teks: mita anggraeni

Masukkan matriks kunci (2x2):
a11: 3
a12: 3
a21: 2
a22: 5

Hasil Enkripsi: AYFFANEWZZMZSJ
Hasil Dekripsi: MITAANGGRAENIX
```

C. ANALISIS KELEMAHAN

1. Caesar Cipher sangat mudah dipecahkan karena hanya memiliki sedikit kemungkinan kunci dan pola hurufnya masih dapat dikenali dengan analisis frekuensi.
2. Affine Cipher memiliki struktur mirip dengan Caesar namun dengan dua kunci, tetapi tetap mudah diuraikan karena pola huruf masih bisa dianalisis dan ruang kuncinya terbatas.
3. Vigenère Cipher lebih kuat karena menggunakan kunci berulang, namun jika panjang kuncinya pendek atau berulang, pola huruf tetap bisa dianalisis dan ditebak.
4. Playfair Cipher mengenkripsi dua huruf sekaligus sehingga lebih kompleks, tetapi masih rentan terhadap serangan yang memanfaatkan sebagian teks asli. Sementara itu,
5. Hill Cipher menggunakan konsep matriks untuk mengenkripsi teks, namun jika sebagian ciphertext dan plaintext diketahui, kunci dapat dihitung kembali dengan metode matematika. Selain itu, proses perhitungannya cukup rumit untuk dilakukan secara manual.

D. KESIMPULAN

Dari kelima cipher klasik yang dianalisis, Caesar Cipher merupakan metode yang paling lemah karena memiliki ruang kunci yang sangat terbatas dan mudah dipecahkan dengan analisis frekuensi. Sementara itu, Hill Cipher menjadi yang paling direkomendasikan di antara cipher klasik karena memiliki kompleksitas matematis yang lebih tinggi dan tingkat keamanan yang lebih baik. Meskipun demikian, seluruh cipher klasik ini sudah tidak layak digunakan untuk melindungi data modern.

E. REFERENSI

- https://medium.com/bisa-ai/kriptografi-klasik-caesar-cipher-a33334fe2965
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2022-2023/02-Ragam-Cipher-Klasik-Bagian1-(2023).pdf
Git hub : https://github.com/mtaaagniii-arch