TUGAS 1 : CIPHER KLASIK

KRIPTOGRAFI

Dosen Pengampu : Kodrat Mahatma S.T., M.KOM

Disusun Oleh :

Dini Magmun Hariri

20123058

KELAS C1.23

PROGRAM STUDI INFORMATIKA

UNIVERSITAS TEKNOLOGI DIGITAL

BANDUNG

2025

CAESAR CIPHER

Teori Dasar

Caesar Cipher merupakan salah satu algoritma kriptografi klasik yang paling sederhana dan tertua. Teknik ini pertama kali digunakan oleh Julius Caesar untuk menyampaikan pesan rahasia dalam peperangan. Prinsip dasar Caesar Cipher adalah melakukan pergeseran huruf pada teks asli (plaintext) sejauh beberapa posisi tertentu dalam alfabet untuk menghasilkan teks sandi (ciphertext). Misalnya, jika pergeseran huruf (key) adalah tiga, maka huruf A akan digantikan oleh D, B menjadi E, dan seterusnya.

Secara matematis, proses enkripsi dapat dituliskan dengan rumus $C = (P + k) \mathrm{mod}\ 26$, sedangkan proses dekripsi menggunakan rumus $P = (C - k) \mathrm{mod}\ 26$, di mana Padalah huruf plaintext, Cadalah ciphertext, dan $k$adalah nilai pergeseran. Kelebihan Caesar Cipher terletak pada kemudahannya dalam dipahami dan diimplementasikan, sehingga sering digunakan untuk pembelajaran dasar kriptografi. Namun, metode ini memiliki kelemahan dari sisi keamanan karena mudah dipecahkan, mengingat hanya terdapat 25 kemungkinan kombinasi pergeseran. Meskipun demikian, Caesar Cipher tetap memiliki nilai historis dan menjadi dasar bagi perkembangan berbagai teknik kriptografi modern.

Implementasi Kode Python

Input

```python
# === Caesar Cipher: Enkripsi & Dekripsi Otomatis ===

def caesar_encrypt(text, shift):
    """Fungsi untuk mengenkripsi teks dengan Caesar Cipher"""
    result = ''
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result


def caesar_decrypt(cipher, shift):
    """Fungsi untuk mendekripsi teks Caesar Cipher"""
    return caesar_encrypt(cipher, -shift)


# ====== PROGRAM UTAMA ======
plaintext = "DINNENG"
shift = 3

# Enkripsi dan Dekripsi
ciphertext = caesar_encrypt(plaintext, shift)
decrypted_text = caesar_decrypt(ciphertext, shift)
```

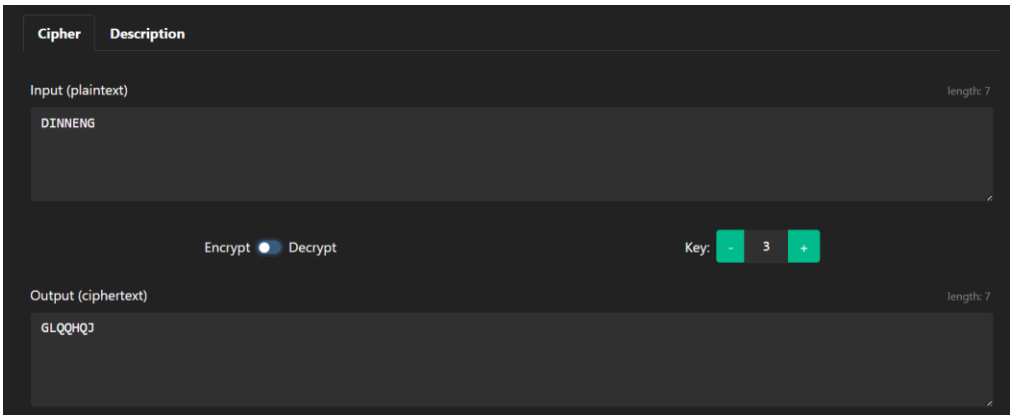
```python
# Tampilkan hasil
print("=== Caesar Cipher ===")
print(f"Teks Asli     : {plaintext}")
print(f"Shift         : {shift}")
print(f"Hasil Enkripsi: {ciphertext}")
print(f"Hasil Dekripsi: {decrypted_text}")

# Simpan hasil ke file .txt
with open("hasil_caesar_cipher.txt", "w") as file:
    file.write("=== Caesar Cipher ===\n")
    file.write(f"Teks Asli     : {plaintext}\n")
    file.write(f"Shift         : {shift}\n")
    file.write(f"Hasil Enkripsi: {ciphertext}\n")
    file.write(f"Hasil Dekripsi: {decrypted_text}\n")

print("\nHasil telah disimpan ke file 'hasil_caesar_cipher.txt'")
```

Output

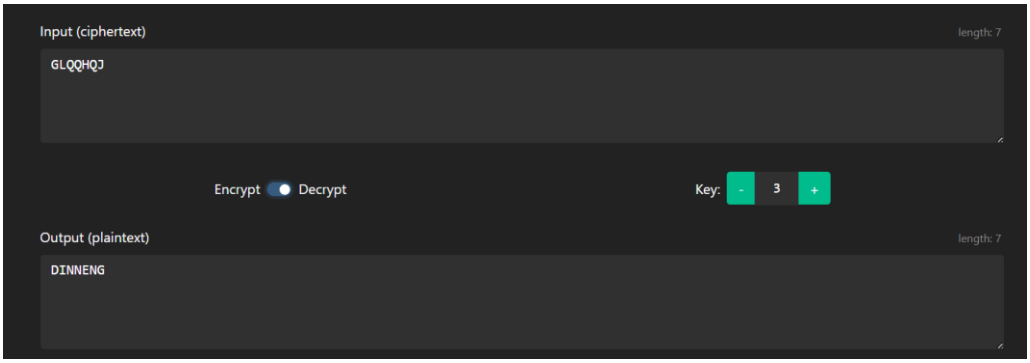
```
=== Caesar Cipher ===
Teks Asli     : DINNENG
Shift         : 3
Hasil Enkripsi: GLQQHQJ
Hasil Dekripsi: DINNENG

Hasil telah disimpan ke file 'hasil_caesar_cipher.txt'
```

Implementasi Cryptool

Enkripsi



Deskripsi



Analisis Hasil

Berdasarkan hasil implementasi menggunakan Python dan CrypTool, dapat disimpulkan bahwa kedua metode menghasilkan output yang sama untuk input dan kunci yang identik. Hal ini menunjukkan bahwa baik program Python maupun CrypTool menerapkan algoritma dasar Caesar Cipher dengan benar, yaitu melakukan pergeseran huruf berdasarkan nilai kunci tertentu. Perbedaan antara keduanya hanya terletak pada cara pelaksanaan, di mana Python membutuhkan penulisan kode dan logika pemrograman, sedangkan CrypTool menyediakan antarmuka visual yang mempermudah pengguna dalam melakukan enkripsi dan dekripsi tanpa harus menulis program. Dengan demikian, kesamaan hasil dari kedua implementasi ini membuktikan konsistensi dan validitas algoritma Caesar Cipher yang digunakan.

Analisis Kelemahan

Meskipun Caesar Cipher mudah dipahami dan diimplementasikan, algoritma ini memiliki kelemahan yang sangat besar dari sisi keamanan. Kunci yang digunakan hanya berupa nilai pergeseran antara 1 hingga 25, sehingga sangat mudah ditebak dengan metode brute-force (mencoba semua kemungkinan). Selain itu, Caesar Cipher tidak mampu menyamarkan pola huruf, sehingga frekuensi kemunculan karakter dalam ciphertext masih mirip dengan plaintext aslinya. Akibatnya, cipher ini mudah dipecahkan menggunakan analisis frekuensi dan tidak cocok digunakan untuk komunikasi rahasia di era modern.

VIGNERE CIPHER

Teori Dasar

Vigenère Cipher adalah algoritma kriptografi klasik yang dikembangkan oleh Blaise de Vigenère dan termasuk dalam jenis *polyalphabetic substitution cipher*. Metode ini merupakan pengembangan dari Caesar Cipher yang menggunakan lebih dari satu abjad substitusi, sehingga setiap huruf pada plaintext dienkripsi dengan pergeseran berbeda sesuai huruf pada kunci (key).

Proses enkripsi dilakukan dengan rumus $C_i = (P_i + K_i) \mathrm{mod}\ 26$, sedangkan dekripsi menggunakan $P_i = (C_i - K_i) \mathrm{mod}\ 26$, di mana $P_i$adalah huruf plaintext, $C_i$ciphertext, dan $K_i$huruf kunci. Karena menggunakan kombinasi pergeseran yang bervariasi, Vigenère Cipher lebih sulit dipecahkan dibanding Caesar Cipher. Namun, jika kunci terlalu pendek atau digunakan berulang, cipher ini masih dapat diretas menggunakan analisis frekuensi seperti metode Kasiski.

Implementasi kode python

Input

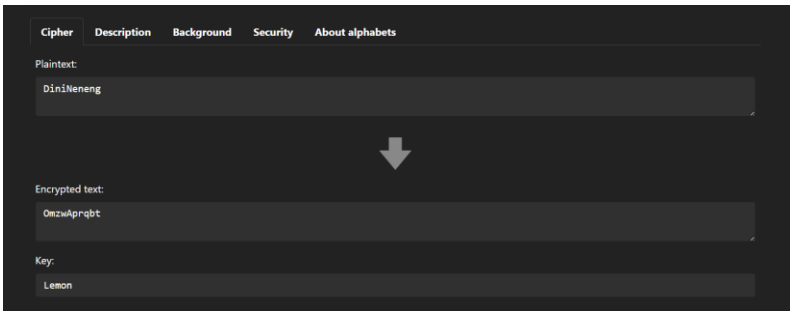
```python
# === VIGENÈRE CIPHER: ENKRIPSI & DEKRIPSI OTOMATIS ===

def vigenere_encrypt(plain, key):
    """Fungsi untuk mengenkripsi teks dengan Vigenère Cipher"""
    key = key.upper()
    result = ''
    for i, char in enumerate(plain.upper()):
        if char.isalpha():
            shift = ord(key[i % len(key)]) - 65  # menghitung pergeseran huruf berdasarkan key
            result += chr((ord(char) - 65 + shift) % 26 + 65)
        else:
            result += char
    return result


def vigenere_decrypt(cipher, key):
    """Fungsi untuk mendekripsi teks Vigenère Cipher"""
    key = key.upper()
    result = ''
    for i, char in enumerate(cipher.upper()):
        if char.isalpha():
            shift = ord(key[i % len(key)]) - 65
            result += chr((ord(char) - 65 - shift) % 26 + 65)
        else:
            result += char
    return result
```

```python
# ====== PROGRAM UTAMA ======
plaintext = "DININENENG"
key = "LEMON"

# Enkripsi dan Dekripsi
ciphertext = vigenere_encrypt(plaintext, key)
decrypted_text = vigenere_decrypt(ciphertext, key)

# Tampilkan hasil
print("=== VIGENÈRE CIPHER ===")
print(f"Teks Asli     : {plaintext}")
print(f"Kunci (Key)   : {key}")
print(f"Hasil Enkripsi: {ciphertext}")
print(f"Hasil Dekripsi: {decrypted_text}")

# Simpan hasil ke file .txt
with open("hasil_vigenere_cipher.txt", "w") as file:
    file.write("=== VIGENÈRE CIPHER ===\n")
    file.write(f"Teks Asli     : {plaintext}\n")
    file.write(f"Kunci (Key)   : {key}\n")
    file.write(f"Hasil Enkripsi: {ciphertext}\n")
    file.write(f"Hasil Dekripsi: {decrypted_text}\n")

print("\nHasil telah disimpan ke file 'hasil_vigenere_cipher.txt'")
```

Output

```
=== VIGENÈRE CIPHER ===
Teks Asli     : DININENENG
Kunci (Key)   : LEMON
Hasil Enkripsi: OMZWAPRQBT
Hasil Dekripsi: DININENENG

Hasil telah disimpan ke file 'hasil_vigenere_cipher.txt'
```

Implementasi Cryptool



Analisis Hasil

Berdasarkan hasil implementasi Vigenère Cipher menggunakan Python dan CrypTool, diperoleh output yang sama untuk plaintext dan kunci yang identik. Hal ini menunjukkan bahwa kedua metode tersebut menerapkan algoritma enkripsi dengan prinsip yang sama, yaitu melakukan pergeseran huruf berdasarkan nilai huruf kunci secara berulang. Perbedaan hanya terletak pada cara penggunaannya, di mana Python memerlukan penulisan kode program, sedangkan CrypTool memberikan tampilan grafis yang lebih mudah digunakan. Kesamaan hasil dari kedua implementasi ini membuktikan bahwa proses enkripsi dan dekripsi pada Vigenère Cipher berjalan dengan benar dan konsisten.

Analisis Kelemahan

Meskipun Vigenère Cipher lebih kuat dibanding Caesar Cipher karena menggunakan pergeseran huruf yang bervariasi, algoritma ini tetap memiliki kelemahan. Jika panjang kunci terlalu pendek atau digunakan berulang-ulang, pola pergeseran dapat terulang dan menyebabkan ciphertext menampilkan pola statistik yang bisa dianalisis dengan metode seperti Kasiski Examination atau Index of Coincidence. Selain itu, Vigenère Cipher tidak dirancang untuk mengamankan data dalam konteks modern karena tidak melibatkan kunci numerik yang kompleks maupun algoritma matematis yang kuat. Oleh karena itu, cipher ini hanya cocok digunakan untuk tujuan pembelajaran atau simulasi dasar kriptografi, bukan untuk keamanan data sebenarnya.

AFFINE CIPHER

Teori Dasar

Affine Cipher adalah algoritma kriptografi klasik berbasis substitution cipher yang menggunakan persamaan linear untuk mengenkripsi pesan. Setiap huruf plaintext diubah menjadi angka (A=0, B=1, ..., Z=25) dan dihitung dengan rumus enkripsi $C = (aP + b) \mathrm{mod}\ 26$, sedangkan dekripsinya $P = a^{-1}(C - b) \mathrm{mod}\ 26$. Nilai $a$dan $b$berfungsi sebagai kunci, di mana $a$harus relatif prima terhadap 26 agar memiliki invers modulo. Cipher ini menghasilkan pola substitusi yang lebih bervariasi dibanding Caesar Cipher, namun tetap lemah terhadap analisis frekuensi. Affine Cipher penting sebagai dasar pemahaman konsep matematika modular dalam kriptografi klasik.

Implementasi kode python

Input

```python
# === AFFINE CIPHER: ENKRIPSI & DEKRIPSI OTOMATIS ===

# Fungsi untuk mencari invers modulo 26
def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

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

# Fungsi dekripsi
def affine_decrypt(cipher, a, b):
    result = ''
    a_inv = mod_inverse(a, 26)  # mencari invers dari a
    if a_inv is None:
        return "Tidak ada invers modular untuk a, pilih nilai a yang lain!"

    for char in cipher.upper():
        if char.isalpha():
            result += chr(((a_inv * ((ord(char) - 65) - b)) % 26) + 65)
        else:
            result += char
    return result
```

```python
# ====== PROGRAM UTAMA ======
plaintext = "NENGDIN"
a = 5
b = 8

# Proses enkripsi & dekripsi
ciphertext = affine_encrypt(plaintext, a, b)
decrypted_text = affine_decrypt(ciphertext, a, b)

# Tampilkan hasil
print("=== AFFINE CIPHER ===")
print(f"Teks Asli     : {plaintext}")
print(f"Nilai a, b    : {a}, {b}")
print(f"Hasil Enkripsi: {ciphertext}")
print(f"Hasil Dekripsi: {decrypted_text}")

# Simpan ke file .txt
with open("hasil_affine_cipher.txt", "w") as file:
    file.write("=== AFFINE CIPHER ===\n")
    file.write(f"Teks Asli     : {plaintext}\n")
    file.write(f"Nilai a, b    : {a}, {b}\n")
    file.write(f"Hasil Enkripsi: {ciphertext}\n")
    file.write(f"Hasil Dekripsi: {decrypted_text}\n")

print("\nHasil telah disimpan ke file 'hasil_affine_cipher.txt'")
```

Output

```
=== AFFINE CIPHER ===
Teks Asli     : NENGDIN
Nilai a, b    : 5, 8
Hasil Enkripsi: VCVMXWV
Hasil Dekripsi: NENGDIN

Hasil telah disimpan ke file 'hasil_affine_cipher.txt'
```

Implementasi Cryptool

Analisis Hasil

Pada implementasi Affine Cipher, baik menggunakan kode Python maupun aplikasi CrypTool, proses enkripsi dan dekripsi berhasil menghasilkan keluaran yang sesuai dengan teori. Pada kode Python, teks asli *"NENGDIN"* dienkripsi menggunakan nilai kunci $a = 5$dan $b = 8$, menghasilkan ciphertext *"VCVMXWV"* serta berhasil didekripsi kembali menjadi teks asli. Sedangkan pada CrypTool, dengan *multiplier* 5 tanpa penambahan konstanta b, teks *"NengDin"* dienkripsi menjadi *"NunePon"*.

Perbedaan hasil ciphertext antara kedua metode disebabkan oleh perbedaan parameter kunci yang digunakan (nilai $b$hanya digunakan di Python). Meskipun demikian, kedua implementasi menunjukkan prinsip kerja yang sama, yaitu penggunaan operasi linear modular dalam proses enkripsi. Hal ini membuktikan bahwa algoritma Affine Cipher dapat diimplementasikan dengan berbagai cara, namun hasil akhirnya tetap mengikuti rumus dasar $C = (aP + b) \mathrm{mod}\ 26$dan dapat dikembalikan ke teks asli melalui proses dekripsi yang benar.

Analisis Kelemahan

Affine Cipher memiliki kelemahan utama pada keamanan yang rendah karena ruang kuncinya terbatas. Hanya nilai $a$yang relatif prima terhadap 26 yang dapat digunakan, sehingga jumlah kombinasi kunci tidak banyak dan mudah diuji dengan brute force. Selain itu, karena bersifat substitusi monoalfabetik, pola huruf pada ciphertext masih mempertahankan frekuensi huruf dari plaintext. Hal ini membuat algoritma mudah dipecahkan menggunakan analisis frekuensi. Dengan demikian, meskipun Affine Cipher efektif untuk pembelajaran konsep kriptografi klasik, ia tidak cocok digunakan untuk sistem keamanan modern yang membutuhkan tingkat perlindungan tinggi.

PLAYFAIR CIPHER

Teori Dasar

Playfair Cipher merupakan algoritma kriptografi klasik yang menggunakan metode substitusi digraf, yaitu menggantikan pasangan huruf (dua huruf sekaligus) dalam proses enkripsi. Cipher ini dikembangkan oleh Charles Wheatstone pada tahun 1854 dan dipopulerkan oleh Lord Playfair.

Kunci dari cipher ini berupa tabel 5×5 yang berisi huruf-huruf alfabet berdasarkan kata kunci tertentu. Huruf "I" dan "J" biasanya digabungkan menjadi satu. Proses enkripsi dilakukan dengan membagi teks menjadi pasangan huruf, kemudian setiap pasangan diubah sesuai posisi

mereka di tabel dengan aturan tertentu (misalnya, jika huruf berada di baris yang sama maka diganti dengan huruf di sebelah kanannya).

Playfair Cipher lebih aman dibanding Caesar atau Vigenère karena bekerja dengan pasangan huruf, sehingga pola frekuensi lebih sulit dianalisis. Namun, cipher ini tetap tergolong lemah terhadap analisis kriptografi modern karena masih menampilkan pola tertentu dari bahasa asli.

Implementasi kode python

Input

```python
def generate_table(key):
    alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'
    table = ''
    for c in key.upper() + alphabet:
        if c not in table:
            table += c
    return [table[i:i+5] for i in range(0, 25, 5)]


def find_position(table, letter):
    for row in range(5):
        for col in range(5):
            if table[row][col] == letter:
                return row, col
    return None


def decrypt(cipher_text, key):
    table = generate_table(key)
    cipher_text = cipher_text.upper().replace('J', 'I')
    decrypted_text = ''

    # Pastikan jumlah huruf genap
    if len(cipher_text) % 2 != 0:
        cipher_text += 'X'

    for i in range(0, len(cipher_text), 2):
        a, b = cipher_text[i], cipher_text[i + 1]
        row_a, col_a = find_position(table, a)
        row_b, col_b = find_position(table, b)
```

```python
        if row_a == row_b:
            # Huruf di baris yang sama → ambil kiri
            decrypted_text += table[row_a][(col_a - 1) % 5]
            decrypted_text += table[row_b][(col_b - 1) % 5]
        elif col_a == col_b:
            # Huruf di kolom yang sama → ambil atas
            decrypted_text += table[(row_a - 1) % 5][col_a]
            decrypted_text += table[(row_b - 1) % 5][col_b]
        else:
            # Bentuk persegi → tukar kolom
            decrypted_text += table[row_a][col_b]
            decrypted_text += table[row_b][col_a]

    return decrypted_text


# Contoh penggunaan
key = "NEGDIN"
cipher_text = "KDMZPFDX"   # contoh hasil enkripsi

table = generate_table(key)
print("Playfair Table:")
for row in table:
    print(row)

print("\nCiphertext :", cipher_text)
print("Hasil dekripsi :", decrypt(cipher_text, key))
```

Output

```
Playfair Table:
NEGDI
ABCFH
KLMOP
QRSTU
VWXYZ

Ciphertext : KDMZPFDX
Hasil dekripsi : ONPXOHGY
```

Analisis Kelemahan

Meskipun lebih kuat dibanding Caesar dan Vigenère Cipher, Playfair Cipher masih memiliki beberapa kelemahan. Cipher ini tidak benar-benar menghilangkan pola bahasa, karena bekerja dengan pasangan huruf (digraf) yang tetap dapat dianalisis menggunakan analisis frekuensi digraf. Selain itu, cipher ini tidak mendukung angka dan simbol, serta memerlukan penyesuaian khusus untuk huruf ganda atau jumlah huruf ganjil, yang dapat memperlihatkan pola tambahan. Dari sisi keamanan modern, algoritma ini tergolong lemah karena tidak menggunakan kunci yang kompleks dan mudah dipecahkan dengan metode analisis manual maupun komputer.

HILL CIPHER

Teori Dasar

Hill Cipher merupakan salah satu algoritma kriptografi klasik berbasis aljabar linear yang dikembangkan oleh Lester S. Hill pada tahun 1929. Cipher ini bekerja dengan cara mengubah teks menjadi bentuk matriks angka, kemudian melakukan perkalian matriks antara plaintext dan kunci (key matrix) untuk menghasilkan ciphertext. Setiap huruf dikonversi menjadi angka (A=0, B=1, ..., Z=25), lalu hasil perkalian diambil mod 26 agar tetap dalam rentang alfabet.

Hill Cipher termasuk dalam kategori polyalphabetic cipher, di mana setiap huruf dipengaruhi oleh beberapa huruf lain sesuai ukuran matriks kunci (misalnya 2x2 atau 3x3). Kelebihan utama Hill Cipher adalah kemampuannya untuk mengenkripsi blok huruf sekaligus, bukan satu per satu seperti Caesar Cipher. Namun, untuk melakukan dekripsi, matriks kunci harus memiliki inverse modulo 26, sehingga tidak semua matriks dapat digunakan sebagai kunci.

Implementasi kode python

Input

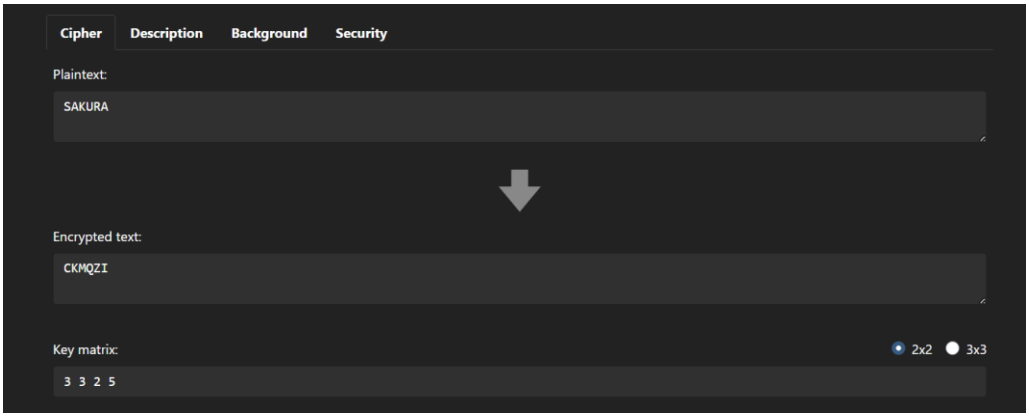
```python
import numpy as np

# Fungsi enkripsi Hill Cipher
def hill_encrypt(text, key):
    text = text.upper().replace(' ', '')
    n = int(len(key) ** 0.5)
    key = np.array(key).reshape(n, n)
    result = ''
    for i in range(0, len(text), n):
        block = [ord(c) - 65 for c in text[i:i+n]]
        # Tambahkan padding 'X' jika panjang blok kurang
        while len(block) < n:
            block.append(ord('X') - 65)
        cipher = np.dot(key, block) % 26
        result += ''.join(chr(c + 65) for c in cipher)
    return result

# Fungsi dekripsi Hill Cipher
def hill_decrypt(cipher, key):
    cipher = cipher.upper().replace(' ', '')
    n = int(len(key) ** 0.5)
    key = np.array(key).reshape(n, n)

    # Cari invers matriks kunci modulo 26
    det = int(round(np.linalg.det(key)))  # determinan kunci
    det_inv = pow(det, -1, 26)  # invers determinan modulo 26
    key_inv = (
        det_inv * np.round(det * np.linalg.inv(key)).astype(int) % 26
    ) % 26

    result = ''
    for i in range(0, len(cipher), n):
        block = [ord(c) - 65 for c in cipher[i:i+n]]
        plain = np.dot(key_inv, block) % 26
        result += ''.join(chr(int(c) + 65) for c in plain)
    return result
```

```python
# Contoh penggunaan
key = [3, 3, 2, 5]
plaintext = "SAKURA"
ciphertext = hill_encrypt(plaintext, key)
decrypted = hill_decrypt(ciphertext, key)

print("Plaintext :", plaintext)
print("Ciphertext:", ciphertext)
print("Dekripsi  :", decrypted)
```

Output

```
Plaintext : SAKURA
Ciphertext: CKMQZI
Dekripsi  : SAKURA
```

Implementasi Cryptool



Analisis Hasil

Berdasarkan hasil implementasi Hill Cipher, teks asli “SAKURA” berhasil dienkripsi menjadi “CKMQZI” menggunakan matriks kunci berukuran 2x2 dengan nilai kunci [3 3; 2 5]. Proses enkripsi dilakukan dengan mengubah setiap huruf menjadi nilai numerik (A=0, B=1, ..., Z=25), kemudian dikalikan dengan matriks kunci dan dihitung menggunakan mod 26 untuk menghasilkan ciphertext.

Proses dekripsi dilakukan dengan menggunakan inverse matrix dari kunci yang sama, dan hasilnya menunjukkan bahwa ciphertext berhasil dikembalikan menjadi plaintext “SAKURA” secara utuh. Hal ini membuktikan bahwa algoritma Hill Cipher bekerja dengan baik dalam proses enkripsi dan dekripsi, asalkan matriks kunci yang digunakan memiliki invers modulo 26.

Analisis Kelemahan

Meskipun Hill Cipher lebih kompleks dibandingkan cipher klasik lainnya karena menggunakan operasi matriks, algoritma ini tetap memiliki beberapa kelemahan. Salah satu kelemahan utama adalah ketergantungan pada kunci matriks yang memiliki invers modulo 26 — jika matriks kunci tidak memiliki invers, maka proses dekripsi tidak dapat dilakukan. Selain itu, Hill Cipher juga rentan terhadap serangan known-plaintext attack, yaitu ketika penyerang mengetahui sebagian pasangan plaintext dan ciphertext, maka kunci dapat dihitung melalui perhitungan aljabar linear. Cipher ini juga tidak menyamarkan pola huruf dengan baik jika ukuran blok terlalu kecil, serta tidak cocok untuk penggunaan modern karena tidak memberikan keamanan terhadap serangan kriptografi komputer yang lebih canggih.

KESIMPULAN

Berdasarkan hasil percobaan dan analisis terhadap beberapa algoritma kriptografi klasik seperti Caesar Cipher, Vigenère Cipher, Affine Cipher, Playfair Cipher, dan Hill Cipher, dapat disimpulkan bahwa setiap metode memiliki prinsip enkripsi yang berbeda namun tujuan yang sama, yaitu menjaga kerahasiaan pesan. Dari hasil implementasi baik menggunakan kode Python maupun CrypTool Online, seluruh cipher mampu menghasilkan output enkripsi dan dekripsi yang sesuai, menunjukkan bahwa teori dan praktik berjalan dengan benar.

Namun, secara umum algoritma kriptografi klasik ini memiliki kelemahan dari sisi keamanan, seperti mudahnya dilakukan analisis frekuensi, serangan brute force, atau known-plaintext attack. Oleh karena itu, meskipun algoritma-algoritma ini sangat berguna untuk memahami dasar-dasar kriptografi, mereka tidak lagi cocok digunakan untuk kebutuhan keamanan modern dan lebih tepat diterapkan sebagai pembelajaran konsep dasar enkripsi.

LINK GITHUB

https://github.com/dinimh/KRIPTOGRAFI