Tugas Kecil 1 IF4020 Kriptografi

Semester I Tahun 2021/2022

Disusun oleh:

Reihan Andhika Putra	**13519043**
Ryo Richardo	**13519193**

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2021

1. **Source Program**

    a. Vigenere.py (kode untuk *Vigenere Cipher*, *Varian Vigenere Cipher*, dan *Extended Vigenere Cipher*)

```
from typing import List
import random
from Utility import alphabets

class VigenereCipher:
    """
    A class used for representing VigenereCipher and it's component.

    Attributes.
    -----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase
format.
    ciphertext : str
        Text you want to decrypt or plaintext after encrypted. In lowercase
format.
    key : str
        Key for encrypting or decrypting a text.
    """

    def __init__(self, key:str, plaintext: str ="", ciphertext:str="") -> None:
        """
        Constructor for VigenereCipher class. Either plaintext or ciphertext
must be empty at
        creation.

        Parameters
        ----------
        key : str
            Text you want to encrypt or ciphertext after decrypted.
        plaintext : str, optional
            Text you want to decrypt or plaintext after encrypted.
        cipherthex : int, optional
            Key for encrypting or decrypting a text.
        """

        # Input validation.
        if (plaintext != "" and ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")
        if (plaintext == "" and ciphertext == ""):
            raise Exception("Either plaintext or ciphertext must be filled")
        if (key ==""):
            raise Exception("Key must be filled!")

        self.plaintext = plaintext
        if (plaintext != ""):
            self.plaintext = self.normalizeText(plaintext)

        ciphertext = ciphertext.lower()
        self.ciphertext = ciphertext
        if (ciphertext != ""):
            self.ciphertext = self.normalizeText(ciphertext)

        key = key.lower()
        self.key = key
        if (key != ""):
            if (self.plaintext != ""):
                self.key = self.normalizeKey(self.plaintext, key)
            else:
                self.key = self.normalizeKey(self.ciphertext, key)

    def encrypt(self)->str:
        """
        Method to encrypt current plaintext with current key. Modify ciphertext
attribute also

        Return the capitalized ciphertext.
        """

        # Class validation.
        if (self.plaintext == "" or self.ciphertext != ""):
            raise Exception("Plaintext must be filled and ciphertext must be
empty")

        # Variable declaration.
        ciphertext:str = ""

        # Encrypt the plaintext.
        for (p,k) in zip(self.plaintext, self.key):
            ciphertext = ciphertext +
alphabets[(alphabets.find(p)+alphabets.find(k))%26]

        self.ciphertext = ciphertext
        return ciphertext.upper()


    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. Modify plaintext
attribute also

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext == ""):
            raise Exception("Plaintext must be empty and ciphertext must be
filled")

        # Variable declaration.
        plaintext:str = ""

        # Encrypt the plaintext.
        for (c,k) in zip(self.ciphertext, self.key):
            plaintext = plaintext + alphabets[(alphabets.find(c)-
alphabets.find(k))%26]

        self.plaintext = plaintext
        return plaintext


    @staticmethod
    def generateBasicVigenereTable()->List[str]:
        """
        Method to generate normal Vigenere Cipher encrypt table.
        Return the List of string representing normal Vigenere Cipher encrypt
table.
        """

        # Variable declaration.
        basicVigenereTable:List[str] = []

        # Loop to create Vigenere Cipher table.
        for i in range(26):
            if (i==0):
                basicVigenereTable.append(alphabets)
            else:
                basicVigenereTable.append(basicVigenereTable[i-
1][1:]+basicVigenereTable[i-1][0])

        return basicVigenereTable

    @staticmethod
    def generateRandomVigenereTable()->List[str]:
        """
        Method to generate random Vigenere Cipher encrypt table.

        Return the List of string representing random Vigenere Cipher encrypt
table.
        """

        # Variable declaration.
        randomVigenereTable:List[str] =
VigenereCipher.generateBasicVigenereTable()

        # Loop to create Vigenere Cipher table.
        for i in range(26):
            n1 = random.randint(0,25)
            n2 = random.randint(0,25)
            randomVigenereTable[n1], randomVigenereTable[n2] =
randomVigenereTable[n2], randomVigenereTable[n1]

        return randomVigenereTable

    @staticmethod
    def normalizeText(text:str)-> str:
        """
        Method to normalize text by removing space and punctuation.

        Return the normalized text.

        Parameters
        ----------
        text : str
            Text you want to normalize.
        """

        # Variable declaration.
        normalizedText:str

        # Remove number, punctuation, and space.
        normalizedText = "".join(filter(str.isalpha, text)).lower()
        return normalizedText

    @staticmethod
    def normalizeKey(text:str, key:str)-> str:
        """
        Method to normalize key by removing space, punctuation, and repeat the
key until it have
        same lenght with text. Text can be normalized plaintext or normalized
ciphertext.

        Return the normalized key.

        Parameters
        ----------
        text : str
            Normalized text (can be plaintext or ciphertext).
        key : str
            Key you want to normalize
        """

        # Variable declaration.
        normalizedKey:str

        # Remove number, punctuation, and space.
        normalizedKey = "".join(filter(str.isalpha, key)).lower()

        # Repeat key until it has same length with plaintext.
        normalizedKey =
(normalizedKey*(len(text)//len(normalizedKey)+1))[:len(text)]

        return normalizedKey

class FullVigenereCipher(VigenereCipher):
    """
    A class used for representing Full Vigenere Cipher and it's component. It's
basically same
    with Vigenere Cipher except it's use permutation of encrypt table for
encrypt and decrypt. So
    it will have new attribute in class

    Attributes.
    ----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase
format.
    ciphertext : str
        Text you want to decrypt or plaintext after encrypted. In lowercase
format.
    key : str
        Key for encrypting or decrypting a text.
    encryptTable: List[str]
        Encrypt table for encrypting and decrypting.
    """

    def __init__(self, key:str, plaintext: str ="", ciphertext:str="",
encryptTable:List[str]=[]) -> None:
        """
        Constructor for FullVigenereCipher class. Either plaintext or ciphertext
must be empty at
        creation.

        Parameters
        ----------
        key : str
            Text you want to encrypt or ciphertext after decrypted.
        plaintext : str
            Text you want to decrypt or plaintext after encrypted.
        cipherthex : int, optional
            Key for encrypting or decrypting a text.
        encryptTable : List[str], optional
        """

        super(FullVigenereCipher, self).__init__(key, plaintext, ciphertext)

        self.encryptTable = encryptTable
        if (len(encryptTable)==0):
            self.encryptTable = self.generateBasicVigenereTable()

    def encrypt(self)->str:
        """
        Method to encrypt current plaintext with current key and current encrypt
table.
        Modify ciphertext attribute also

        Return the capitalized ciphertext.
        """

        # Class validation.
        if (self.plaintext == "" or self.ciphertext != ""):
            raise Exception("Plaintext must be filled and ciphertext must be
empty")


        # Variable declaration.
        ciphertext:str = ""

        # Encrypt the plaintext.
        for (p,k) in zip(self.plaintext, self.key):
            ciphertext = ciphertext +
self.encryptTable[alphabets.find(k)][alphabets.find(p)]

        self.ciphertext = ciphertext
        return ciphertext.upper()


    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. Modify plaintext
attribute also

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")

        # Variable declaration.
        plaintext:str = ""

        # Encrypt the plaintext.
        for (c,k) in zip(self.ciphertext, self.key):
            plaintext = plaintext +
alphabets[self.encryptTable[alphabets.find(k)].find(c)]

        self.plaintext = plaintext
        return plaintext

class AutoKeyVigenereCipher(VigenereCipher):
    """
    A class used for representing Auto-Key Vigenere Cipher and it's component.
It's basically same
    with Vigenere Cipher except it's way for normalizing key and decrypt the
ciphertext.

    Attributes.
    ----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase
format.
    ciphertext : str
        Text you want to decrypt or plaintext after encrypted. In lowercase
format.
    key : str
        Key for encrypting or decrypting a text.
    """

    def __init__(self, key:str, plaintext: str ="", ciphertext:str="") -> None:
        """
        Constructor for AutoKeyVigenereCipher class. Either plaintext or
ciphertext must be empty at
        creation. If plaintext are empty then key will not be fully normalized
(must decrypt the
        ciphertext first to know the complete key), else if ciphertext are empty
then key will
        be normalized.

        Parameters
        ----------
        key : str
            Text you want to encrypt or ciphertext after decrypted.
        plaintext : str
            Text you want to decrypt or plaintext after encrypted.
        cipherthex : int, optional
            Key for encrypting or decrypting a text.
        """

        # Input validation.
        if (plaintext != "" and ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")
        if (plaintext == "" and ciphertext == ""):
            raise Exception("Either plaintext or ciphertext must be filled")
        if (key ==""):
            raise Exception("Key must be filled!")

        self.plaintext = plaintext
        if (plaintext != ""):
            self.plaintext = self.normalizeText(plaintext)

        ciphertext = ciphertext.lower()
        self.ciphertext = ciphertext
        if (ciphertext != ""):
            self.ciphertext = self.normalizeText(ciphertext)

        key = key.lower()
        self.key = key
        if (key != ""):
            if (self.plaintext != ""):
                self.key = self.normalizeKey(self.plaintext, key)
            else:
                self.key = self.normalizeText(key)

    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. While decrypting
it will also
        complete the key. Modify plaintext attribute.

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")

        # Variable declaration.
        plaintext:str = ""
        currPlainText:chr = ""

        # Encrypt the plaintext.
        # Notify that it will dinamically update the key.
        for (index, c) in enumerate(self.ciphertext):
            currPlainText = alphabets[(alphabets.find(c)-
alphabets.find(self.key[index]))%26]
            plaintext = plaintext + currPlainText
            if (len(self.key) < len(self.ciphertext)):
                self.key = self.key + currPlainText

        self.plaintext = plaintext
        return plaintext

    @staticmethod
    def normalizeKey(plaintext:str, key:str)-> str:
        """
        Method to normalize key by removing space, punctuation, and fill the key
with repeated text
        until it have same lenght with text. "text" must be normalized
plaintext. Otherwise you can
        complete the key while decrypting ciphertext.

        Return the normalized key.

        Parameters
        ----------
        plaintext : str
            Normalized plaintext.
        key : str
            Key you want to normalize
        """

        # Variable declaration.
        normalizedKey:str

        # Remove number, punctuation, and space.
        normalizedKey = "".join(filter(str.isalpha, key)).lower()

        # Repeat key until it has same length with plaintext.
        normalizedKey = (normalizedKey +
plaintext*(len(plaintext)//len(normalizedKey)+1))[:len(plaintext)]
        return normalizedKey

class ExtendedVigenereCipher:
    """
    A class used for representing Extended Vigenere Cipher and it's component.
It's basically
    Vigenere Cipher with 256 ASCII character.

    Attributes.
    ----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase
format.
    ciphertext : str
        Text you want to decrypt or plaintext after encrypted. In lowercase
format.
    key : str
        Key for encrypting or decrypting a text.
    """

    def __init__(self, key:str, plaintext: any ="", ciphertext:any="") -> None:
        """
        Constructor for VigenereCipher class. Either plaintext or ciphertext
must be empty at
        creation.

        Parameters
        ----------
        key : str
            Text you want to encrypt or ciphertext after decrypted.
        plaintext : str, optional
            Text you want to decrypt or plaintext after encrypted.
        cipherthex : int, optional
            Key for encrypting or decrypting a text.
        """

        # Input validation.
        if (plaintext != "" and ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")
        if (plaintext == "" and ciphertext == ""):
            raise Exception("Either plaintext or ciphertext must be filled")
        if (key ==""):
            raise Exception("Key must be filled!")

        self.plaintext = plaintext
        self.ciphertext = ciphertext
        self.key = key
        if (key != ""):
            if (self.plaintext != ""):
                self.key = self.normalizeKey(self.plaintext, key)
            else:
                self.key = self.normalizeKey(self.ciphertext, key)

    @staticmethod
    def normalizeKey(text:str, key:str)-> str:
        """
        Method to normalize key by repeat the key until it have same length with
text.
        Text can be plaintext or ciphertext.
        Return the normalized key.

        Parameters
        ----------
        text : str
            Normalized text (can be plaintext or ciphertext).
        key : str
            Key you want to normalize
        """

        # Variable declaration.
        normalizedKey:str

        # Repeat key until it has same length with plaintext.
        normalizedKey = (key*(len(text)//len(key)+1))[:len(text)]
        return normalizedKey

    def encrypt(self)->str:
        """
        Method to encrypt current plaintext with current key. Modify ciphertext
attribute also

        Return the ciphertext.
        """

        # Class validation.
        if (self.plaintext == "" or self.ciphertext != ""):
            raise Exception("Plaintext must be filled and ciphertext must be
empty")


        # Variable declaration.
        ciphertext:str = ""

        # Encrypt the plaintext.
        for (p,k) in zip(self.plaintext, self.key):
            ciphertext = ciphertext + chr((ord(p) + ord(k)) % 256)

        self.ciphertext = ciphertext
        return ciphertext

    def encryptByte(self)->bytearray:
        """
        Method to encrypt plaintext with current key as byte array.

        Return the ciphertext in byte array.
        """

        # Variable declaration.
        ciphertext:bytearray = self.plaintext

        # Encrypt the plaintext.
        for index, values in enumerate(ciphertext):
            ciphertext[index] = ((values + ord(self.key[index]))%256)

        self.ciphertext = ciphertext
        return ciphertext


    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. Modify plaintext
attribute also

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")

        # Variable declaration.
        plaintext:str = ""

        # Decrypt the plaintext.
        for (c,k) in zip(self.ciphertext, self.key):
            plaintext = plaintext + chr((ord(c) - ord(k)) % 256)

        self.plaintext = plaintext
        return plaintext

    def decryptByte(self)->bytearray:
        """
        Method to decrypt current bytearray ciphertext with current key. Modify
plaintext attribute also

        Return the plaintext.
        """

        # Variable declaration.
        plaintext:bytearray = self.ciphertext

        # Decrypt the plaintext.
        for index, values in enumerate(plaintext):
            plaintext[index] = ((values - ord(self.key[index]))%256)

        self.plaintext = plaintext
        return plaintext
```

b. Playfair.py (kode untuk *Playfair Cipher*)

```
import re
import random
from typing import List
from collections import OrderedDict

# alphabets is a string that represent all Indonesia alphabet.
alphabets:str = "abcdefghijklmnopqrstuvwxyz"

class PlayfairCipher:
    """
    A class used for representing PlayfairCipher and it's component.

    Attributes.
    ----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase
format.
    ciphertext : str
```

```
        Text you want to decrypt or plaintext after encrypted. In lowercase
format.
    key : str
        Key for encrypting or decrypting a text.
    """

    def __init__(self, key:str, plaintext: str ="", ciphertext:str="") -> None:
        """
        Constructor for PlayfairCipher class. Either plaintext or ciphertext
must be empty at
        creation.

        Parameters
        ----------
        key : str, optional
            Key for encrypting or decrypting a text. If none, generate randomly.
        plaintext : str, optional
            Text you want to encrypt or ciphertext after decrypted.
        cipherthex : int, optional
            Text you want to decrypt or plaintext after encrypted.
        """

        # Input validation.
        if (plaintext != "" and ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")
        if (plaintext == "" and ciphertext == ""):
            raise Exception("Either plaintext or ciphertext must be filled")
        if (key ==""):
            raise Exception("Key must be filled!")

        self.plaintext = plaintext
        if (plaintext != ""):
            self.plaintext = self.normalizeText(plaintext)

        self.ciphertext = ciphertext
        if (ciphertext != ""):
            self.ciphertext = self.normalizeText(ciphertext)

        self.key = key
        if (key != ""):
            self.key = self.normalizeKey(key)
        else:
            self.key = self.generateBasicPlayfairTable()

    def encrypt(self)->str:
        """
        Method to encrypt current plaintext with current key. Also modify
ciphertext attribute.

        Return the capitalized ciphertext.
        """

        # Class validation.
        if (self.plaintext == "" or self.ciphertext != ""):
            raise Exception("Plaintext must be filled and ciphertext must be
empty")

        # Variable declaration.
        ciphertext:str = ""
        letter1:int
        letter2:int
        # Encrypt the plaintext.
        for i in range (0, len(self.plaintext), 2):

            # Assign to variable
            letter1 = self.key.find(self.plaintext[i])
            letter2 = self.key.find(self.plaintext[i+1])

            # 2 chars in the same row
            if (letter1//5 == letter2//5):
                ciphertext += self.key[(letter1//5)*5 + ((letter1%5)+1)%5]
                ciphertext += self.key[(letter2//5)*5 + ((letter2%5)+1)%5]

            # 2 chars in the same column
            elif (letter1%5 == letter2%5):
                ciphertext += self.key[(letter1+5)%25]
                ciphertext += self.key[(letter2+5)%25]

            # Other places
            else:
                ciphertext += self.key[(letter1//5)*5 + letter2%5]
                ciphertext += self.key[(letter2//5)*5 + letter1%5]

        self.ciphertext = ciphertext
        return ciphertext.upper()


    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. Also modify
plaintext attribute.

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext == ""):
            raise Exception("Plaintext must be empty and ciphertext must be
filled")

        # Variable declaration.
        plaintext:str = ""
        letter1:int
        letter2:int

        # Decrypt the ciphertext.
        for i in range (0, len(self.ciphertext), 2):

            # Assign to variable
            letter1 = self.key.find(self.ciphertext[i])
            letter2 = self.key.find(self.ciphertext[i+1])

            # 2 chars in the same row
            if (letter1//5 == letter2//5):
                plaintext += self.key[(letter1//5)*5 + ((letter1%5)-1)%5]
                plaintext += self.key[(letter2//5)*5 + ((letter2%5)-1)%5]

            # 2 chars in the same column
            elif (letter1%5 == letter2%5):
                plaintext += self.key[(letter1-5)%25]
                plaintext += self.key[(letter2-5)%25]
            # Other places
            else:
                plaintext += self.key[(letter1//5)*5 + letter2%5]
                plaintext += self.key[(letter2//5)*5 + letter1%5]

        self.plaintext = plaintext
        return plaintext


    @staticmethod
    def generateBasicPlayfairTable()->str:
        """
        Method to generate normal Playfair Cipher encrypt table.

        Return the List of string representing normal Playfair Cipher encrypt
table.
        """

        # Variable declaration.
        basicPlayfairTable:str = ""
        alph:str = alphabets[:9] + alphabets[10:]
        num:int

        # Loop to create Playfair Cipher table.
        for count in range(25, -1, -1):

            # Generate random int
            num = random.randint(0, count)

            # Add new char to table
            basicPlayfairTable += alph[num]

            # Remove added char from alphabet
            alph = alph[:num] + alph[num+1:]

        return basicPlayfairTable


    @staticmethod
    def normalizeText(text:str)-> str:
        """
        Method to normalize text by removing space and punctuation, swap char
"j" to "i",
        add aditional char "x" for two consecutives same chars or unpaired
chars.
        Assumption: no 3 or more consecutive same char, no "x" at end of text.

        Return the normalized text.

        Parameters
        ----------
        text : str
            Text you want to normalize.
        """

        # Variable declaration.
        normalizedText:str

        # Remove number, punctuation, and space.
        normalizedText = "".join(filter(str.isalpha, text)).lower()
        # Swap char "j" to "i"
        normalizedText.replace("j", "i")

        # Add aditional char "x"
        normalizedText = re.sub(r'(.)\1', r'\1x\1', normalizedText)

        # Add char "x" at end of text if length is odd
        if (len(normalizedText) % 2 == 1):
            normalizedText += "x"

        return normalizedText

    @staticmethod
    def normalizeKey(key:str)-> str:
        """
        Method to normalize key by removing space, punctuation, duplicates, and
char "j",
        also complete key with the rest of alphabet if length < 25.

        Return the normalized key.

        Parameters
        ----------
        key : str
            Key you want to normalize
        """

        # Variable declaration.
        normalizedKey:str

        # Remove number, punctuation, and space.
        normalizedKey = "".join(filter(str.isalpha, key)).lower()

        # Remove char "j"
        normalizedKey = normalizedKey.replace("j", "")

        # Remove duplicates string
        normalizedKey = "".join(OrderedDict.fromkeys(normalizedKey))

        # Complete the key with the rest of alphabet
        for letter in alphabets:
            if (letter not in normalizedKey and letter !='j'):
                normalizedKey += letter

        return normalizedKey

    def keyToMatrix(self)-> list:
        """
        Method to transform string key to matrix.

        Return the matrix key.

        """

        # Variable declaration.
        matrix = [["" for i in range (5)] for j in range (5)]
        i:int = 0

        # Loop for each letter
        for j in range (25):
            # Put to matrix
            matrix[i][j%5] = self.key[j]

            # Next row
            if (j % 5 == 4):
                i += 1

        return matrix
```

c. Affine.py (kode untuk *Affine Cipher*)

```
from re import X
from typing import List

from Utility import alphabets, relativePrime, modularInverse

class AffineCipher:
    """
    A class used for representing Affine Cipher and it's component.

    Attributes.
    ----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase
format.
    ciphertext : str
        Text you want to decrypt or plaintext after encrypted. In lowercase
format.
    b : int
        Number of shifting for encrypting or decrypting a text.
    m : int
        Key for encrypting or decrypting a text.
    """

    def __init__(self, b:int, m:int, plaintext: str ="", ciphertext:str="") ->
None:
        """
        Constructor for AffineCipher class. Either plaintext or ciphertext must
be empty at
        creation.

        Parameters
        ----------
        plaintext : str, optional
            Text you want to decrypt or plaintext after encrypted.
        cipherthex : int, optional
            Key for encrypting or decrypting a text.
        b : int
            Number of shifting for encrypting or decrypting a text.
        m : int
            Key for encrypting or decrypting a text.
        """

        # Input validation.
        if (plaintext != "" and ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")
        if (plaintext == "" and ciphertext == ""):
            raise Exception("Either plaintext or ciphertext must be filled")
        if (b == None or m == None):
            raise Exception("Key must be filled!")

        self.plaintext = plaintext
        if (plaintext != ""):
            self.plaintext = self.normalizeText(plaintext)

        ciphertext = ciphertext.lower()
        self.ciphertext = ciphertext
        if (ciphertext != ""):
            self.ciphertext = self.normalizeText(ciphertext)


        b = b % 26
        self.b = b

        m = m % 26
        if (not relativePrime(m,26)):
            raise Exception("m must be relative prime with 26, eg (1, 3, 5, 7,
9, 11, 15, 17, 19, 21, 23, and 25).")
        self.m = m

    @staticmethod
    def normalizeText(text:str)-> str:
        """
        Method to normalize text by removing space and punctuation.

        Return the normalized text.

        Parameters
        ----------
        text : str
            Text you want to normalize.
        """

        # Variable declaration.
        normalizedText:str

        # Remove number, punctuation, and space.
        normalizedText = "".join(filter(str.isalpha, text)).lower()
        return normalizedText

    def encrypt(self)->str:
        """
        Method to encrypt current plaintext with current key. Modify ciphertext
attribute also

        Return the capitalized ciphertext.
        """

        # Class validation.
        if (self.plaintext == "" or self.ciphertext != ""):
            raise Exception("Plaintext must be filled and ciphertext must be
empty")

        # Variable declaration.
        ciphertext:str = ""

        # Encrypt the plaintext.
        for p in self.plaintext:
            ciphertext = ciphertext + alphabets[(alphabets.find(p)*self.m + 
self.b)%26]

        self.ciphertext = ciphertext
        return ciphertext.upper()

    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. Modify plaintext 
attribute also

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext == ""):
            raise Exception("Plaintext must be empty and ciphertext must be 
filled")

        # Variable declaration.
        plaintext:str = ""

        # Encrypt the plaintext.
        modInverse = modularInverse(self.m, 26)
        for c in self.ciphertext:
            plaintext = plaintext + alphabets[(modInverse*(alphabets.find(c)-
self.b))%26]

        self.plaintext = plaintext
        return plaintext
```

d. Hill.py (kode untuk *Hill Cipher*)

```
import random
from Utility import alphabets, modularInverse, relativePrime

class HillCipher:
    """
    A class used for representing Hill Cipher and it's component.

    Attributes.
    ----------
    plaintext : str
        Text you want to encrypt or ciphertext after decrypted. In lowercase 
format.
    ciphertext : str
        Text you want to decrypt or plaintext after encrypted. In lowercase 
format.
    m : list of int
        Key for encrypting or decrypting a text.
    """

    def __init__(self, m:list, plaintext: str ="", ciphertext:str="") -> None:
        """
        Constructor for HillCipher class. Either plaintext or ciphertext must be 
empty at
        creation.

        Parameters
        ----------
        plaintext : str, optional
            Text you want to decrypt or plaintext after encrypted.
        cipherthex : int, optional
            Key for encrypting or decrypting a text.
        m : list of int
            Key for encrypting or decrypting a text.
        """

        # Input validation.
        if (plaintext != "" and ciphertext != ""):
            raise Exception("Either plaintext or ciphertext must be empty")
        if (plaintext == "" and ciphertext == ""):
            raise Exception("Either plaintext or ciphertext must be filled")
        if (m == None):
            raise Exception("Key must be filled!")
        if (not relativePrime(self.determinantMatrix(m), 26)):
            raise Exception("matrix determinant must be relative prime with 26,
eg (1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, and 25).)")

        self.plaintext = plaintext
        if (plaintext != ""):
            self.plaintext = self.normalizeText(plaintext)

        self.ciphertext = ciphertext
        if (ciphertext != ""):
            self.ciphertext = self.normalizeText(ciphertext)

        self.m = m

    @staticmethod
    def generateBasicHillTable()->str:
        """
        Method to generate normal Hill Cipher encrypt table.

        Return the List of string representing normal ill Cipher encrypt table.
        """

        # Variable declaration.
        basicHillTable:list = []
        templist:list = []
        num:int

        # Loop to create Playfair Cipher table.
        for i in range(9):

            # Generate random int
            num = random.randint(0, 25)
            templist.append(num)

            # Add new int to table
            if (i % 3 == 2):
                basicHillTable.append(templist)
                templist = []

        return basicHillTable

    @staticmethod
    def normalizeText(text:str)-> str:
        """
        Method to normalize text by removing space and punctuation.
        Return the normalized text.

        Parameters
        ----------
        text : str
            Text you want to normalize.
        """

        # Variable declaration.
        normalizedText:str
        tail:int

        # Remove number, punctuation, and space.
        normalizedText = "".join(filter(str.isalpha, text)).lower()

        # Add dummy "x" char
        tail = len(normalizedText) % 3
        if (tail != 0):
            normalizedText += "x" * (3-tail)

        return normalizedText

    @staticmethod
    def determinantMatrix(m)->int:
        """
        Method to find matrix determinant.

        Return the determinant (mod 26).
        """

        # Variable declaration.
        det:int = 0
        minor = [[0 for i in range (3)] for j in range (3)]

        # Find minor entry matrix.
        for i in range (3):
            for j in range(3):
                minor[i][j] = (m[(i+1)%3][(j+1)%3] * m[(i+2)%3][(j+2)%3] -
m[(i+1)%3][(j+2)%3] * m[(i+2)%3][(j+1)%3])

        # Find determinant.
        for i in range (3):
            det += m[0][i]*minor[0][i]

        # Find modular inverse determinant.
        det %= 26

        return det

    def encrypt(self)->str:
        """
        Method to encrypt current plaintext with current key. Modify ciphertext
attribute also

        Return the capitalized ciphertext.
        """

        # Class validation.
        if (self.plaintext == "" or self.ciphertext != ""):
            raise Exception("Plaintext must be filled and ciphertext must be 
empty")

        # Variable declaration.
        ciphertext:str = ""

        # Encrypt the plaintext.
        for i in range (0, len(self.plaintext), 3):

            # Find value of p1 p2 p3
            p1 = alphabets.find(self.plaintext[i])
            p2 = alphabets.find(self.plaintext[i+1])
            p3 = alphabets.find(self.plaintext[i+2])

            # Find it's corresponding cipher value
            ciphertext += alphabets[((self.m[0][0]*p1 + self.m[0][1]*p2 + 
self.m[0][2]*p3)%26)]
            ciphertext += alphabets[((self.m[1][0]*p1 + self.m[1][1]*p2 + 
self.m[1][2]*p3)%26)]
            ciphertext += alphabets[((self.m[2][0]*p1 + self.m[2][1]*p2 + 
self.m[2][2]*p3)%26)]

        self.ciphertext = ciphertext
        return ciphertext.upper()

    def decrypt(self)->str:
        """
        Method to decrypt current ciphertext with current key. Modify plaintext 
attribute also

        Return the plaintext.
        """

        # Class validation.
        if (self.plaintext != "" and self.ciphertext == ""):
            raise Exception("Plaintext must be empty and ciphertext must be 
filled")

        # Variable declaration.
        plaintext:str = ""
        minor = [[0 for i in range (3)] for j in range (3)]
        mInverse = [[0 for i in range (3)] for j in range (3)]
        det:int = 0

        # Find minor entry matrix.
        for i in range (3):
            for j in range(3):
                minor[i][j] = (self.m[(i+1)%3][(j+1)%3] * 
self.m[(i+2)%3][(j+2)%3] - self.m[(i+1)%3][(j+2)%3] * self.m[(i+2)%3][(j+1)%3])

        # Find determinant.
        for i in range (3):
            det += self.m[0][i]*minor[0][i]

        # Find modular inverse determinant.
        det %= 26
        detInverse = modularInverse(det, 26)

        # Find modular inverse matrix
        for i in range (3):
            for j in range(3):
                mInverse[j][i] = ((minor[i][j]%26) * detInverse) % 26

        # Decrypt the ciphertext.
        for i in range (0, len(self.ciphertext), 3):

            # Find value of c1 c2 c3
            c1 = alphabets.find(self.ciphertext[i])
            c2 = alphabets.find(self.ciphertext[i+1])
            c3 = alphabets.find(self.ciphertext[i+2])

            # Find it's corresponding plain value
            plaintext += alphabets[((mInverse[0][0]*c1 + mInverse[0][1]*c2 +
mInverse[0][2]*c3)%26)]
            plaintext += alphabets[((mInverse[1][0]*c1 + mInverse[1][1]*c2 +
mInverse[1][2]*c3)%26)]
            plaintext += alphabets[((mInverse[2][0]*c1 + mInverse[2][1]*c2 +
mInverse[2][2]*c3)%26)]

        self.plaintext = plaintext
        return plaintext
```

e. Utility.py (kode untuk utilitas pembantu)

```
from re import I

def gcd(a:int, b:int)-> int:
    """
    Method to count the greatest common divisor of two number.

    Parameters
    ----------
    a : int
        Number you want to count the gcd.
    b : int
        Number you want to count the gcd.
    """
    a = abs(a)
    b = abs(b)
    if (b == 0):
        return a
    return gcd(b, a % b)

def relativePrime(a:int, b:int)->bool:
    """
    Method to check relative prime of two number.

    Parameters
    ----------
    a : int
        Number you want to check relative prime.
    b : int
        Number you want to check relative prime.
    """
    return gcd(a, b) == 1

def modularInverse(a:int, b:int) -> int:
    """
    Method to find modular inverse of two number.
    Return the modular inverse of two number or 0 if modular inverse not exist.

    Parameters
    ----------
    a : int
        Number you want to check modular inverse.
    b : int
        Number you want to check modular inverse.
    """
    a = a % b
    for i in range (b):
        if ((i*a) % b == 1):
            return i
    return 0

# alphabets is a string that represent all Indonesia alphabet.
alphabets:str = "abcdefghijklmnopqrstuvwxyz"
```

f. app.py (kode untuk *router* sekaligus *logic* dari *web/backend*)

```
import os
from flask import Flask, render_template, request, redirect, url_for,
send_from_directory, current_app
from werkzeug.datastructures import FileStorage

from Playfair import PlayfairCipher
from Vigenere import VigenereCipher, FullVigenereCipher, ExtendedVigenereCipher,
AutoKeyVigenereCipher
from Affine import AffineCipher
from Hill import HillCipher
from Utility import alphabets

# Flask Configuration.
app = Flask(__name__)
UPLOAD_FOLDER = './static/uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['SECRET_KEY'] = 'mysecret'

# Generate random vigenere encrypt table for handling full vigenere.
randomEncipherTable = VigenereCipher.generateRandomVigenereTable()

"""
--------------------------------------------------------------
# Default Route
--------------------------------------------------------------
"""
@app.route('/', defaults={'path': ''})
@app.route('/<path:path>')
def catch_all(path):
                return redirect(url_for('vigenere'))


"""
--------------------------------------------------------------
# Route for Vigenere Cipher
--------------------------------------------------------------
"""
# Index route.
@app.route('/vigenere-cipher')
```

```
def vigenere():
        return render_template('pages/vigenere-cipher.html', encrypt=True)

# Encrypt route.
@app.route('/vigenere-cipher/encrypt', methods=['POST', 'GET'])
def vigenereEncrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                plaintext = request.form['plaintext']
                # Catch exception when processing Vigenere Cipher.
                try:
                        # Process Encrypt Vigenere Cipher.
                        vigenere = VigenereCipher(key=key, plaintext=plaintext)
                        vigenere.encrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/vigenere-cipher.html', 
encrypt=True,
                                result_ciphertext = vigenere, form = 
request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/vigenere-cipher.html', 
encrypt=True, error = e,
                                form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('vigenere'))

# Decrypt route.
@app.route('/vigenere-cipher/decrypt', methods=['POST', 'GET'])
def vigenereDecrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                ciphertext = request.form['ciphertext']
        # Catch exception when processing Vigenere Cipher.
                try:
                        # Process Decrypt Vigenere Cipher.
                        vigenere = VigenereCipher(key=key, ciphertext=ciphertext)
                        vigenere.decrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/vigenere-cipher.html', 
encrypt=False,
                                        result_plaintext = vigenere, form = 
request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/vigenere-cipher.html', 
encrypt=False,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('vigenere'))


"""
-----------------------------------------------------------------
# Route for Auto-key Vigenere Cipher
-----------------------------------------------------------------
"""
```

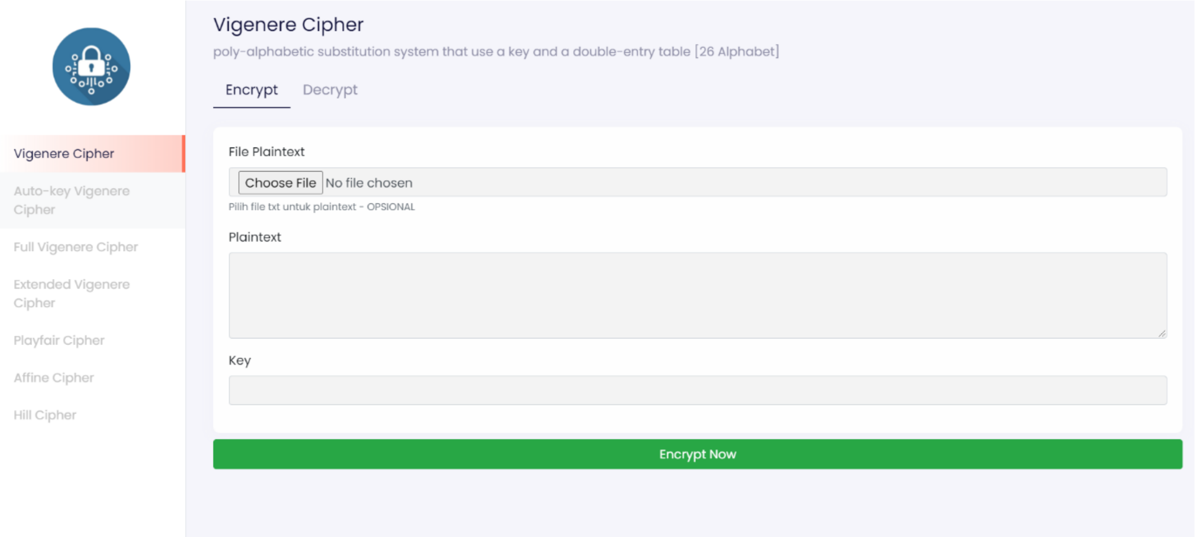
```
# Index route.
@app.route('/auto-key-vigenere-cipher')
def autoKeyVigenere():
                return render_template('pages/auto-key-vigenere-cipher.html',
encrypt=True)

# Encrypt route.
@app.route('/auto-key-vigenere-cipher/encrypt', methods=['POST', 'GET'])
def autoKeyVigenereEncrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                plaintext = request.form['plaintext']
                # Catch exception when processing Auto-key Vigenere Cipher.
                try:
                        # Process Encrypt Auto-key Vigenere Cipher.
                        autoVigenere = AutoKeyVigenereCipher(key=key,
plaintext=plaintext)
                        autoVigenere.encrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/auto-key-vigenere-
cipher.html', encrypt=True,
                                        result_ciphertext = autoVigenere, form =
request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/auto-key-vigenere-
cipher.html', encrypt=True,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('autoKeyVigenere'))

# Decrypt route.
@app.route('/auto-key-vigenere-cipher/decrypt', methods=['POST', 'GET'])
def autoKeyVigenereDecrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                ciphertext = request.form['ciphertext']
                # Catch exception when processing Auto-key Vigenere Cipher.
                try:
                        # Process Decrypt Auto-key Vigenere Cipher.
                        autoVigenere = AutoKeyVigenereCipher(key=key,
ciphertext=ciphertext)
                        autoVigenere.decrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/auto-key-vigenere-
cipher.html', encrypt=False,
                                result_plaintext = autoVigenere, form =
request.form)
                except (Exception) as e:
                        # Render error webpage.
                        return render_template('pages/auto-key-vigenere-
cipher.html', encrypt=False,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('autoKeyVigenere'))
```

```
"""
----------------------------------------------------------------
# Route for Full Vigenere Cipher
----------------------------------------------------------------
"""
# Index route.
@app.route('/full-vigenere-cipher')
def fullKeyVigenere():
        return render_template('pages/full-vigenere-cipher.html', encrypt=True,
                encryptTable=randomEncipherTable, alphabets= alphabets)

# Encrypt route.
@app.route('/full-vigenere-cipher/encrypt', methods=['POST', 'GET'])
def fullKeyVigenereEncrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                plaintext = request.form['plaintext']
                # Catch exception when processing Full-key Vigenere Cipher.
                try:
                        # Process Encrypt Full-key Vigenere Cipher.
                        fullVigenere = FullVigenereCipher(key=key,
plaintext=plaintext,
                                encryptTable=randomEncipherTable)
                        fullVigenere.encrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/full-vigenere-cipher.html',
encrypt=True,
                                result_ciphertext = fullVigenere, form =
request.form, encryptTable=randomEncipherTable
                                , alphabets= alphabets)
                except (Exception) as e:
                        # Render error webpage.
                        return render_template('pages/full-vigenere-cipher.html',
encrypt=True,
                                error = e, form = request.form,
encryptTable=randomEncipherTable, alphabets= alphabets)
        else:
                # Render default webpage.
                return redirect(url_for('fullKeyVigenere'))

# Decrypt route.
@app.route('/full-vigenere-cipher/decrypt', methods=['POST', 'GET'])
def fullKeyVigenereDecrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                ciphertext = request.form['ciphertext']
                # Catch exception when processing Full-key Vigenere Cipher.
                try:
                                # Process Decrypt Full-key Vigenere Cipher.
                        fullVigenere = FullVigenereCipher(key=key,
ciphertext=ciphertext,
                                        encryptTable=randomEncipherTable)
                        fullVigenere.decrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/full-vigenere-cipher.html',
encrypt=False,
                                        result_plaintext = fullVigenere, form =
request.form, encryptTable=randomEncipherTable
                                        , alphabets= alphabets)
                except (Exception) as e:
                        # Render error webpage.
                        return render_template('pages/full-vigenere-cipher.html',
encrypt=False,
                                error = e, form = request.form,
encryptTable=randomEncipherTable, alphabets= alphabets)
        else:
                # Render default webpage.
                        return redirect(url_for('fullKeyVigenere'))

"""
-----------------------------------------------------------------
# Route for Extended Vigenere Table
-----------------------------------------------------------------
"""
# Index route.
@app.route('/extended-vigenere-cipher')
def extendedVigenere():
        return render_template('pages/extended-vigenere-cipher.html',
encrypt=True,
                encryptTable=randomEncipherTable, alphabets= alphabets)

# Encrypt route.
@app.route('/extended-vigenere-cipher/encrypt', methods=['POST', 'GET'])
def extendedVigenereEncrypt():
        if request.method == 'POST':
                # Get the request payload.
                try:
                        # Catch exception when processing Extended Vigenere
Cipher.
                        choice = request.form["encrypt"]
                        key = request.form['key']
                        # Process Encrypt Extended Vigenere Cipher.
                        if (choice == "file"):
                                # Encrypt file value.
                                plaintext = request.form['plaintext']
                                extendedVigenere =
ExtendedVigenereCipher(key=key, plaintext=plaintext)
                                extendedVigenere.encrypt()
                                # Render successfull webpage with data.
                                return render_template('pages/extended-vigenere-
cipher.html', encrypt=True,
                                        result_ciphertext = extendedVigenere, form
= request.form)
                        else:
                                # Encrypt file byte.
                                file = request.files['file-plaintext']
                                # Save the file to local and then open it.
                                file.stream.seek(0)
                                file.save(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER'], file.filename))
                                with open(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER'], file.filename), "rb") as f:
                                        # Encrypt isi filenya dan simpen ke dalam
file di tempat yg sama kek upload.
                                        extendedVigenere =
ExtendedVigenereCipher(key=key, plaintext=bytearray(f.read()))

                                with open(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER'], file.filename), "wb") as f:
                                        f.write(extendedVigenere.encryptByte())
                                        # Ntar file hasil encrypt yang di save
namanya harus berbeda terus di download.
                                        # Kalo misal bisa langsung rewrite file
nya tanpa harus save file baru lebih bagus si. Ntar kalo gini nama filenya sama
gpp.


                                # Download The file.
                                return
send_from_directory(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER']), file.filename, as_attachment=True)
                except (Exception) as e:
                        # Render error webpage.
                        return render_template('pages/extended-vigenere-
cipher.html', encrypt=True,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('extendedVigenere'))

# Decrypt route.
@app.route('/extended-vigenere-cipher/decrypt', methods=['POST', 'GET'])
def extendedVigenereDecrypt():
        if request.method == 'POST':
                # Get the request payload.
                try:
                        # Catch exception when processing Extended Vigenere
Cipher.
                        choice = request.form["decrypt"]
                        key = request.form['key']
                        # Process Decrypt Extended Vigenere Cipher.
                        if (choice == "file"):
                                # Encrypt file value.
                                ciphertext = request.form['ciphertext']
                                extendedVigenere =
ExtendedVigenereCipher(key=key, ciphertext=ciphertext)
                                extendedVigenere.decrypt()
                                # Render successfull webpage with data.
                                return render_template('pages/extended-vigenere-
cipher.html', encrypt=False,
                                        result_plaintext = extendedVigenere, form
= request.form)
                        else:
                                # Decrypt file byte.
                                file = request.files['file-ciphertext']
                                # Save the file to local and then open it.
                                file.stream.seek(0)
                                file.save(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER'], file.filename))
                                with open(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER'], file.filename), "rb") as f:
                                        extendedVigenere =
ExtendedVigenereCipher(key=key, ciphertext=bytearray(f.read()))

                                with open(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER'], file.filename), "wb") as f:
                                        f.write(extendedVigenere.decryptByte())
                                        # Decrypt isi filenya dan simpen ke dalam
file di tempat yg sama kek upload.
                                        # Ntar file hasil decrypt yang di save
namanya harus berbeda terus di download.
```

```
                                        # Kalo misal bisa langsung rewrite file
nya tanpa harus save file baru lebih bagus si. Ntar kalo gini nama filenya sama
gpp.

                                # Download The file.
                                return
send_from_directory(os.path.join(current_app.root_path,
app.config['UPLOAD_FOLDER']), file.filename, as_attachment=True)

                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/extended-vigenere-
cipher.html', encrypt=False,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('extendedVigenere'))

"""
-----------------------------------------------------------------
# Route for Playfair Cipher
-----------------------------------------------------------------
"""
# Index route.
@app.route('/playfair-cipher')
def playfair():
        return render_template('pages/playfair-cipher.html', encrypt=True)

# Encrypt route.
@app.route('/playfair-cipher/encrypt', methods=['POST', 'GET'])
def playfairEncrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                plaintext = request.form['plaintext']
                # Catch exception when processing Playfair Cipher.
                try:
                        # Process Encrypt Playfair Cipher.
                        playfair = PlayfairCipher(key=key, plaintext=plaintext)
                        playfair.encrypt()
                        return render_template('pages/playfair-cipher.html',
encrypt=True,
                                result_ciphertext = playfair, form =
request.form, matrix = playfair.keyToMatrix())
                                # Render successfull webpage with data.
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/playfair-cipher.html',
encrypt=True,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect('pages/playfair-cipher.html', encrypt=True)

# Decrypt route.
@app.route('/playfair-cipher/decrypt', methods=['POST', 'GET'])
def playfairDecrypt():
        if request.method == 'POST':
                # Get the request payload.
                key = request.form['key']
                ciphertext = request.form['ciphertext']
                # Catch exception when processing Playfair Cipher.
                try:
                        # Process Decrypt Playfair Cipher.
                        playfair = PlayfairCipher(key=key, ciphertext=ciphertext)
                        playfair.decrypt()
                        return render_template('pages/playfair-cipher.html',
encrypt=False,
                                result_plaintext = playfair, form = request.form,
matrix = playfair.keyToMatrix())
                except (Exception) as e:
                        return render_template('pages/playfair-cipher.html',
encrypt=False,
                                error = e, form = request.form)
        else:
                        return redirect(url_for('playfair'))

"""
-----------------------------------------------------------------
# Route for Affine Cipher
-----------------------------------------------------------------
"""
# Index route.
@app.route('/affine-cipher')
def affine():
        return render_template('pages/affine-cipher.html', encrypt=True)

# Encrypt route.
@app.route('/affine-cipher/encrypt', methods=['POST', 'GET'])
def affineEncrypt():
        if request.method == 'POST':
                # Get the request payload.
                keyM = int(request.form['keyM'])
                keyB = int(request.form['keyB'])
                plaintext = request.form['plaintext']
                # Catch exception when processing Affine Cipher.
                try:
                        # Process Encrypt Affine Cipher.
                        affine = AffineCipher(b=keyB, m=keyM,
plaintext=plaintext)
                        affine.encrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/affine-cipher.html',
encrypt=True,
                                result_ciphertext = affine, form = request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/affine-cipher.html',
encrypt=True,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('affine'))

# Decrypt route.
@app.route('/affine-cipher/decrypt', methods=['POST', 'GET'])
def affineDecrypt():
        if request.method == 'POST':
                # Get the request payload.
                keyM = int(request.form['keyM'])
                keyB = int(request.form['keyB'])
                ciphertext = request.form['ciphertext']
                # Catch exception when processing Affine Cipher.
                try:
                                # Process Decrypt Affine Cipher.
                        affine = AffineCipher(b=keyB, m=keyM,
ciphertext=ciphertext)
                        affine.decrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/affine-cipher.html',
encrypt=False,
                                result_plaintext = affine, form = request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/affine-cipher.html',
encrypt=False,
                                error = e, form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('affine'))

"""
-----------------------------------------------------------------
# Route for Hill cipher
-----------------------------------------------------------------
"""
@app.route('/hill-cipher')
def hill():
        return render_template('pages/hill-cipher.html', encrypt=True)

@app.route('/hill-cipher/encrypt', methods=['POST', 'GET'])
def hillEncrypt():
        if request.method == 'POST':
                # Catch exception when processing payload or encrypting Hill
Cipher.
                try:
                        # Get request payload.
                        matrixKey=[]
                        for i in range(3):
                                matrixRow = []
                                for j in range(3):
                                        matrixCol = request.form['r-'+str(i+1)+'c-
'+str(j+1)]
                                        if (not matrixCol):
                                                raise Exception("All matrix key
must be filled")
                                        matrixRow.append(int(matrixCol))
                                matrixKey.append(matrixRow)
                        plaintext = request.form['plaintext']
                        # Process encrypting Hill Cipher.
                        hill = HillCipher(m=matrixKey ,plaintext=plaintext)
                        hill.encrypt()
                        # Render successfull webpage with data.
                        return render_template('pages/hill-cipher.html',
encrypt=True, result_ciphertext = hill,
                                form = request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/hill-cipher.html',
encrypt=True, error = e,
                                form = request.form)
        else:
                # Render default webpage.
                return redirect(url_for('hill'))

@app.route('/hill-cipher/decrypt', methods=['POST', 'GET'])
def hillDecrypt():
        if request.method == 'POST':
                # Catch error when processing payload or decrypting Hill Cipher.
                try:
                        # Get the request payload.
                        matrixKey=[]
                        for i in range(3):
                                matrixRow = []
                                for j in range(3):
                                        matrixCol = request.form['r-'+str(i+1)+'c-
'+str(j+1)]
                                        if (not matrixCol):
                                                raise Exception("All matrix key
must be filled")
                                        matrixRow.append(int(matrixCol))
                                matrixKey.append(matrixRow)
                        ciphertext = request.form['ciphertext']
                        # Process Decrypt Hill Cipher.
                        hill = HillCipher(m=matrixKey, ciphertext=ciphertext)

                        # Render successfull webpage with data.
                        return render_template('pages/hill-cipher.html',
encrypt=False, result_plaintext = hill,
                                form = request.form)
                except (Exception) as e:
                        # Rende error webpage.
                        return render_template('pages/hill-cipher.html',
encrypt=False, error = e,
                                form = request.form)
        else:
                # Render default webpage.
                        return redirect(url_for('hill'))

"""
---------------------------------------------------------------
# Flask Main Program
---------------------------------------
------------------------
"""
if __name__ == '__main__':
        app.run(debug=True,threaded=True)
```

2. Tampilan antarmuka program

a. Tampilan halaman Vigenere Cipher (Auto-key Vigenere Cipher dan Playfair Cipher mirip seperti halaman ini)



Vigenere Cipher

poly-alphabetic substitution system that use a key and a double-entry table [26 Alphabet]

File Plaintext

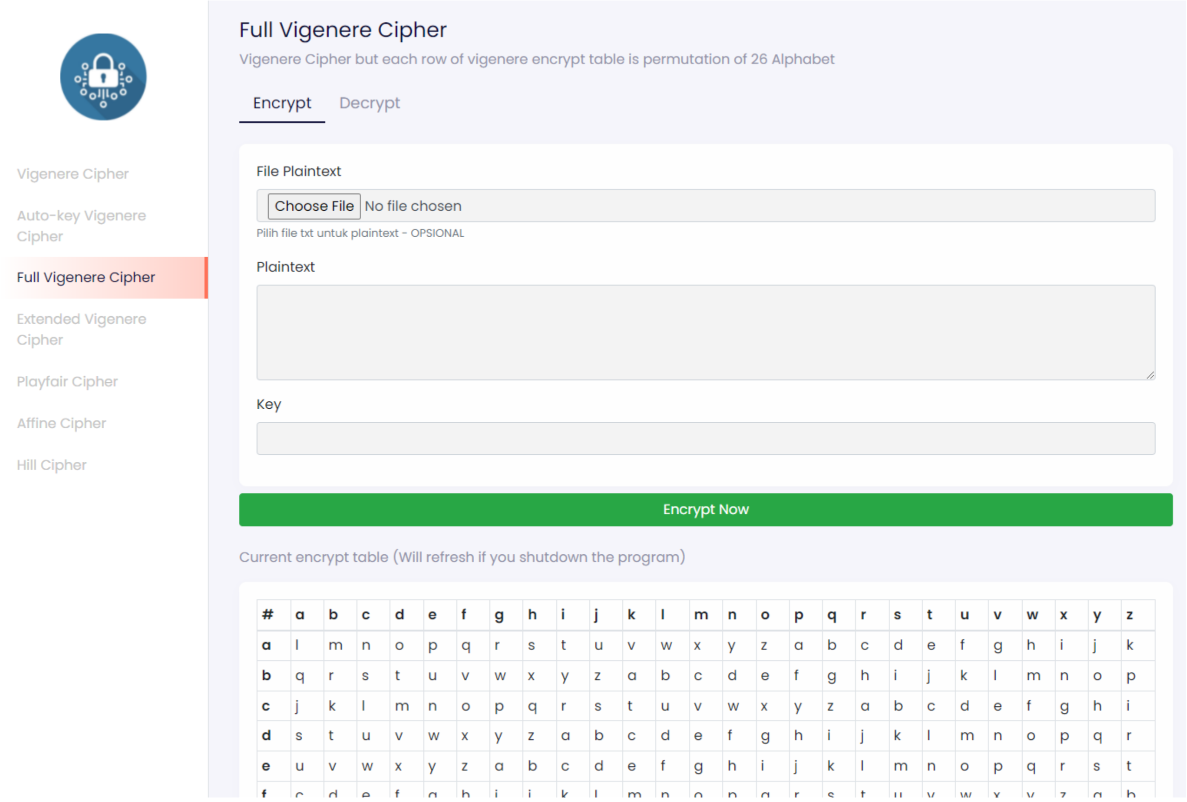
No file chosen

Pilih file txt untuk plaintext - OPSIONAL

Plaintext

Key

b. Tampilan halaman Full Vigenere Cipher



Full Vigenere Cipher

Vigenere Cipher but each row of vigenere encrypt table is permutation of 26 Alphabet

File Plaintext

No file chosen

Pilih file txt untuk plaintext - OPSIONAL

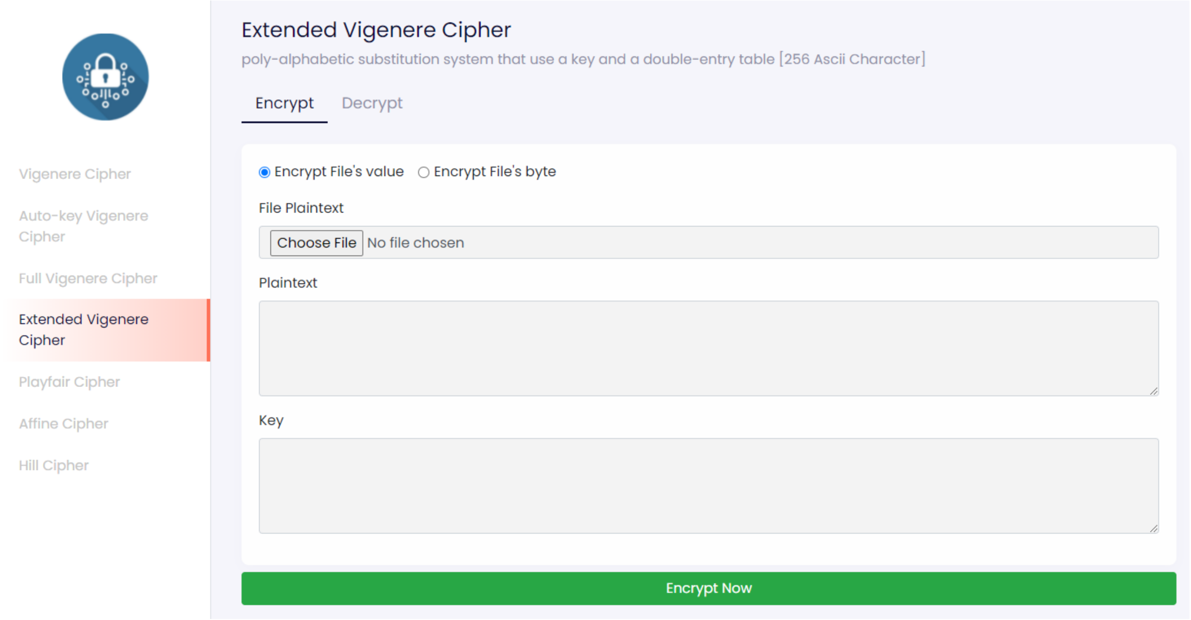
Plaintext

Key

Current encrypt table (Will refresh if you shutdown the program)

#	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
b	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
c	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
d	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
e	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
f	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b

c. Tampilan halaman Extended Vigenere Cipher



Extended Vigenere Cipher

poly-alphabetic substitution system that use a key and a double-entry table [256 Ascii Character]

● Encrypt File's value ○ Encrypt File's byte

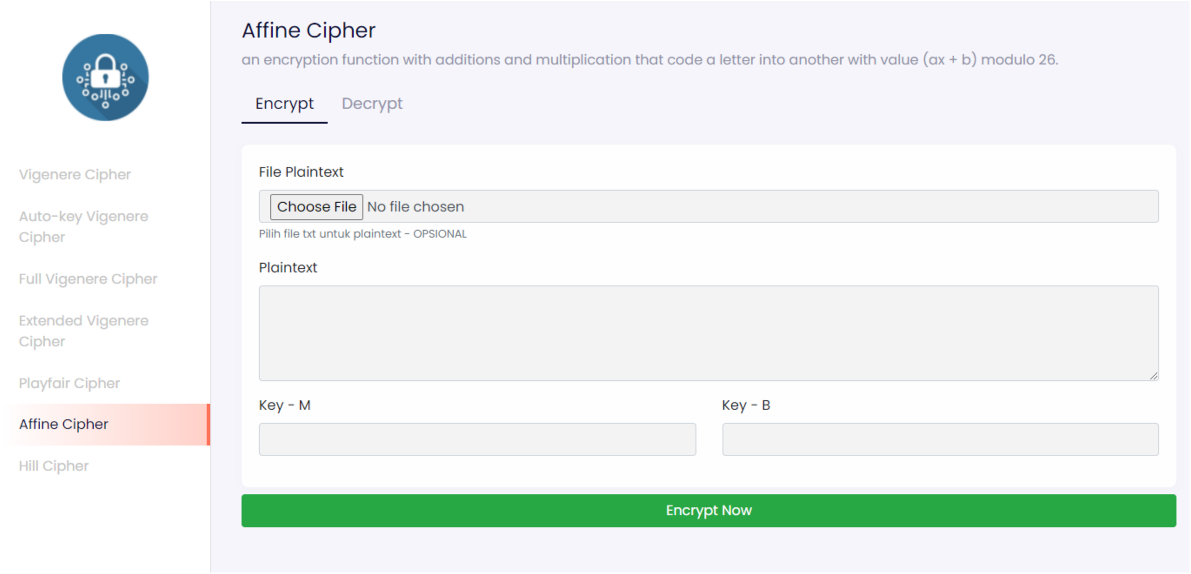
File Plaintext

No file chosen

Plaintext

Key

d. Tampilan halaman Affine Cipher



Affine Cipher

an encryption function with additions and multiplication that code a letter into another with value (ax + b) modulo 26.

File Plaintext

No file chosen

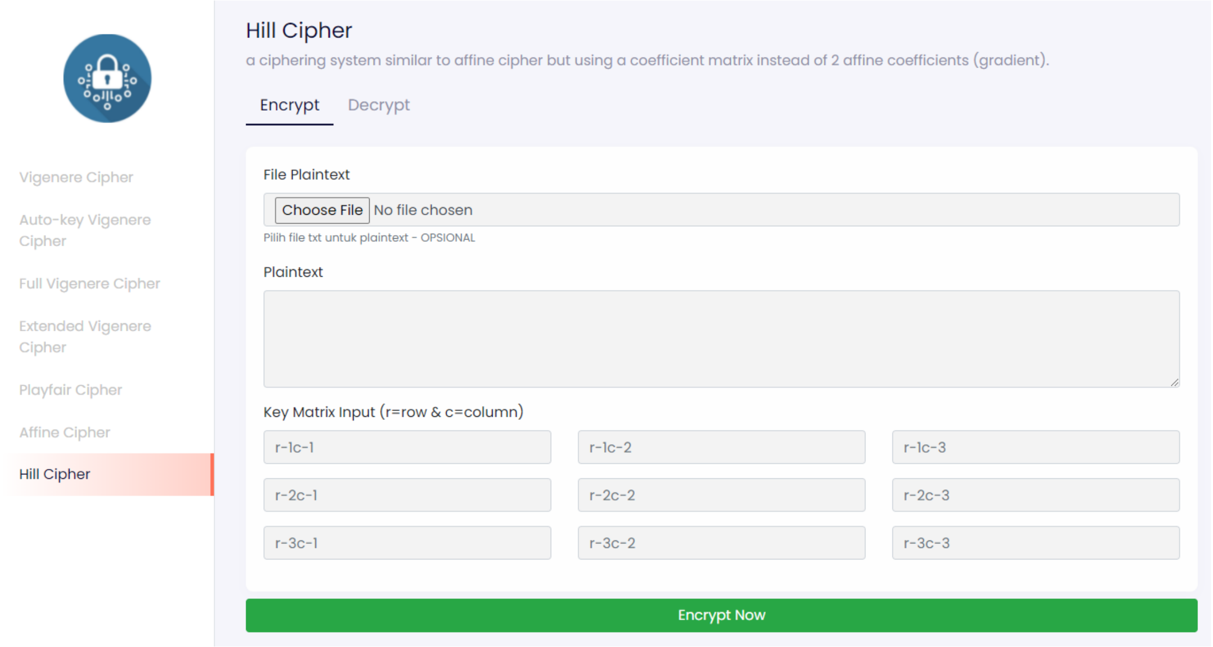
Pilih file txt untuk plaintext - OPSIONAL

Plaintext

Key - M

Key - B

e. Tampilan halaman Hill Cipher



Hill Cipher

a ciphering system similar to affine cipher but using a coefficient matrix instead of 2 affine coefficients (gradient).

File Plaintext

Pilih file txt untuk plaintext - OPSIONAL

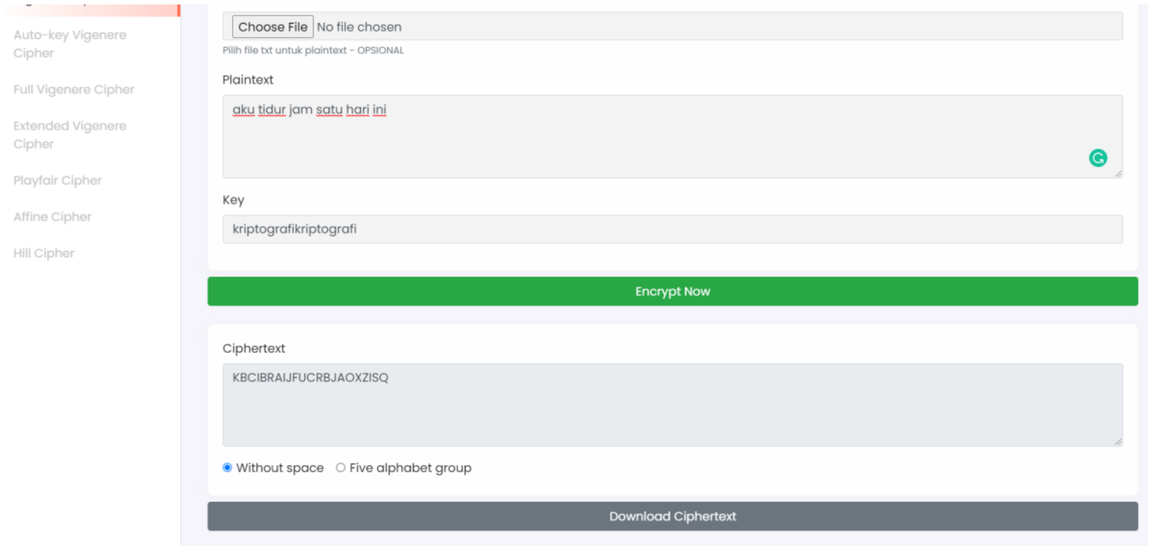
Plaintext

Key Matrix Input (r=row & c=column)

3. Contoh plainteks dan cipherteks

a. Vigenere standard

- **Encrypt:** Plainteks “aku tidur jam satu hari ini” dengan kunci “kriptografi”



Pilih file txt untuk plaintext - OPSIONAL

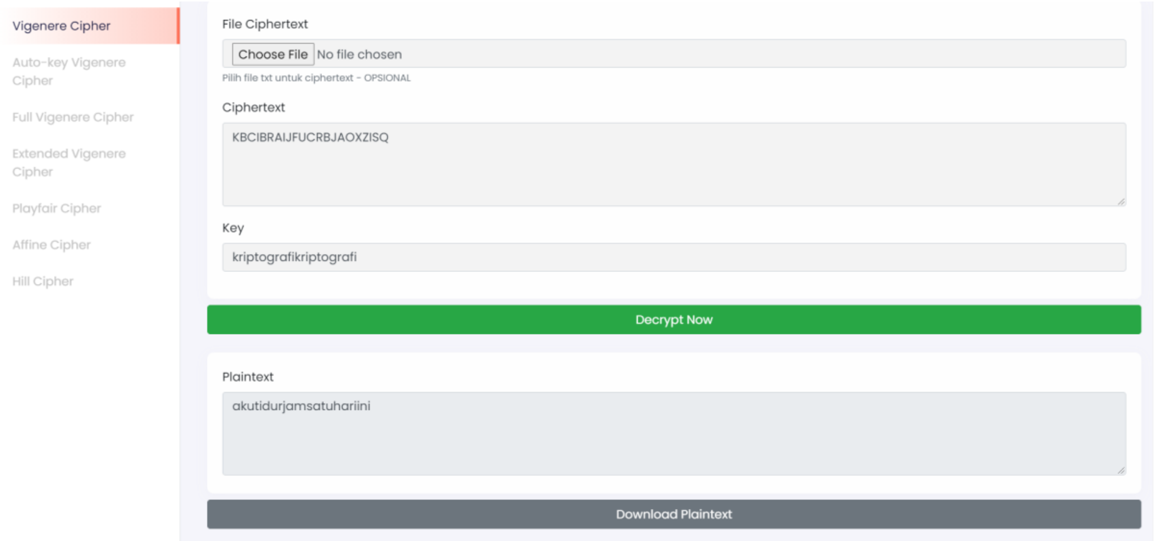
Plaintext: aku tidur jam satu hari ini

Key: kriptografikriptografi

Ciphertext: KBCIBRAIJFUCRBJAOXZISQ

● Without space ○ Five alphabet group

- **Decrypt:** Cipherteks “KBCIBRAIJFUCRBJAOXZISQ” dengan kunci “kriptografi”



File Ciphertext

No file chosen

Pilih file txt untuk ciphertext - OPSIONAL

Ciphertext

KBCIBRAIJFUCRBJAOXZISQ

Key

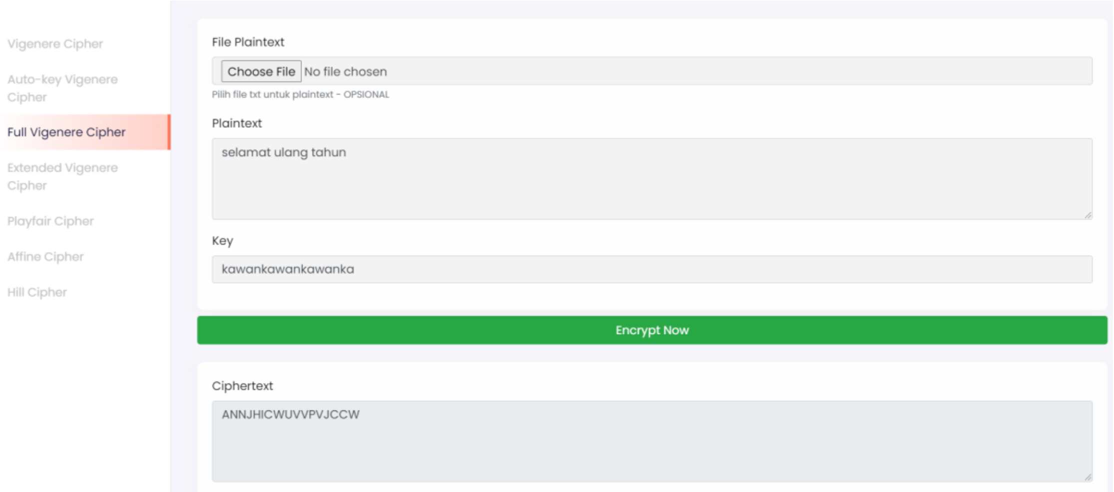
kriptografikriptografi

Plaintext

akutidurjamsatuhariini

b. Full Vigenere Cipher

- **Encrypt:** Plainteks "selamat ulang tahun" dengan kunci "kawan"



File Plaintext

No file chosen

Pilih file txt untuk plaintext - OPSIONAL

Plaintext

selamat ulang tahun

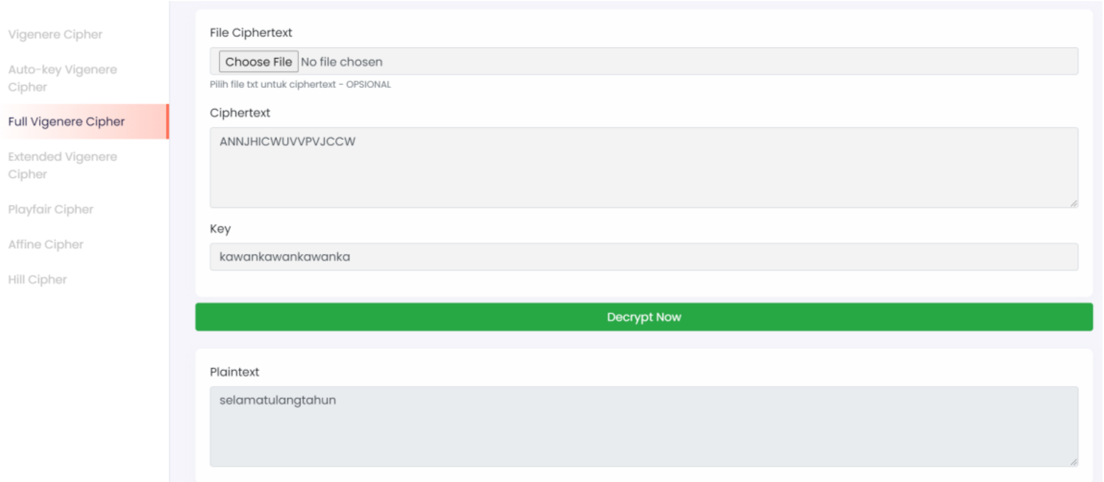
Key

kawankawankawanka

Ciphertext

ANNJHICWUVVPVJCCW

- **Decrypt:** Cipherteks "ANNJHICWUVVPVJCCW" dengan kunci "kawan"



File Ciphertext

No file chosen

Pilih file txt untuk ciphertext - OPSIONAL

Ciphertext

ANNJHICWUVVPVJCCW

Key

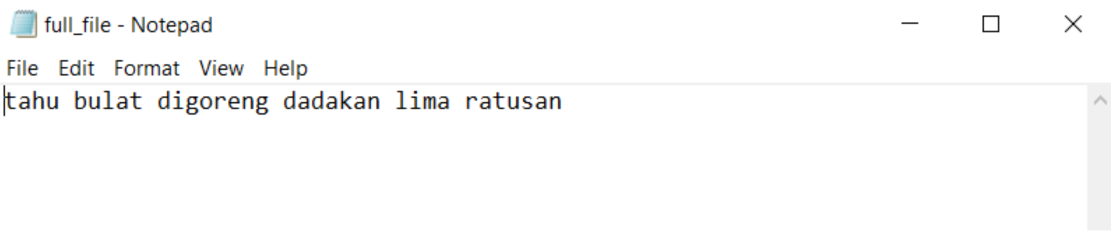
kawankawankawanka

Plaintext

selamatulangtahun

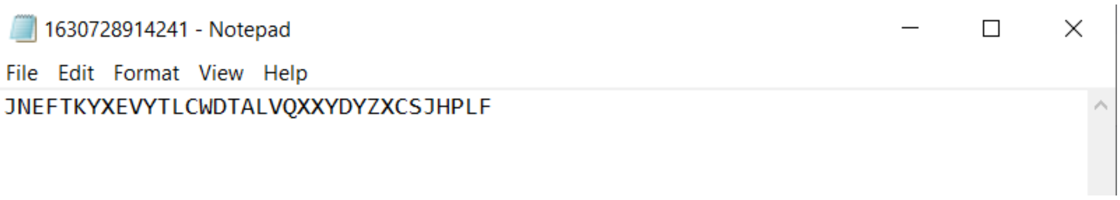
- **Encrypt (File):**

File sebelum dienkripsi:



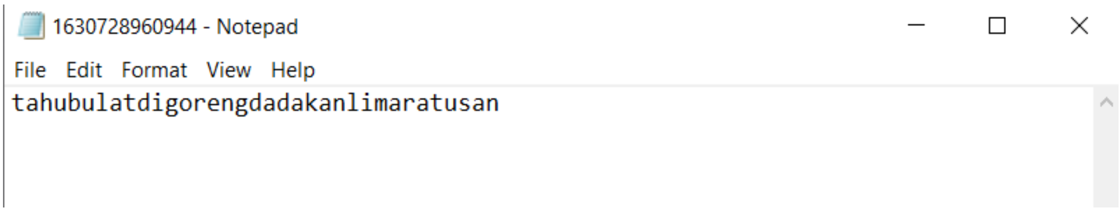
```
tahu bulat digoreng dadakan lima ratusan
```

File setelah dienkripsi dengan kunci “gurih”:



```
JNEFTKYXEVYTLCWDTALVQXXYDYZXCSJHPLF
```

- **Decrypt(File):** File setelah didekripsi kembali dengan kunci “gurih”:

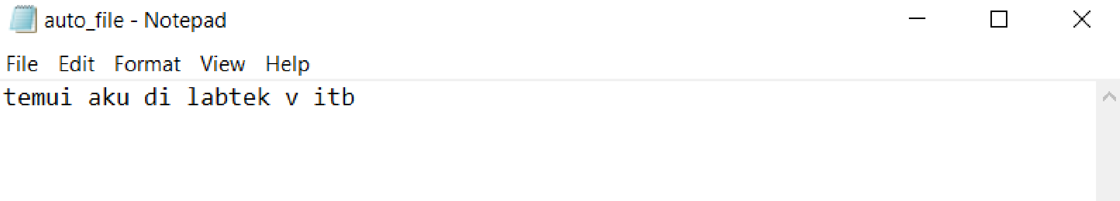


```
tahubulatdigorengdadakanlimaratusan
```

c. Auto-key Vigenere Cipher

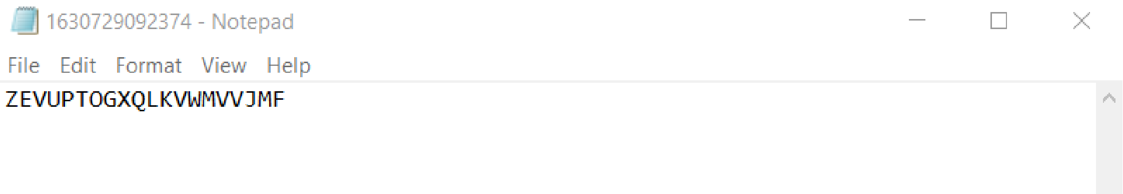
- **Encrypt (File):**

File sebelum dienkripsi:



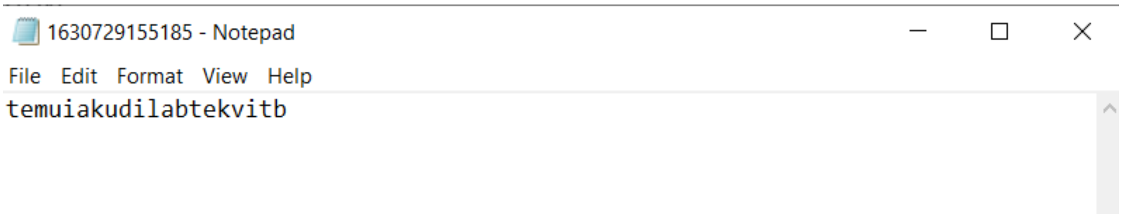
```
temui aku di labtek v itb
```

File setelah dienkripsi dengan kunci “gajah”



```
ZEVUPTOGXQLKVWMVVJMF
```

- **Decrypt(File):** File setelah didekripsi kembali dengan kunci “gajah”:

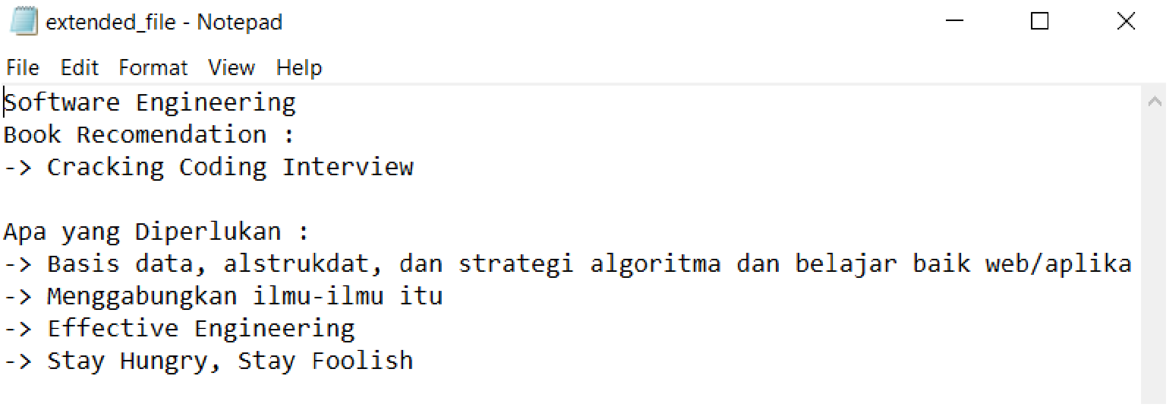


```
temuiakudilabtekvitb
```

d. Extended Vigenere Cipher

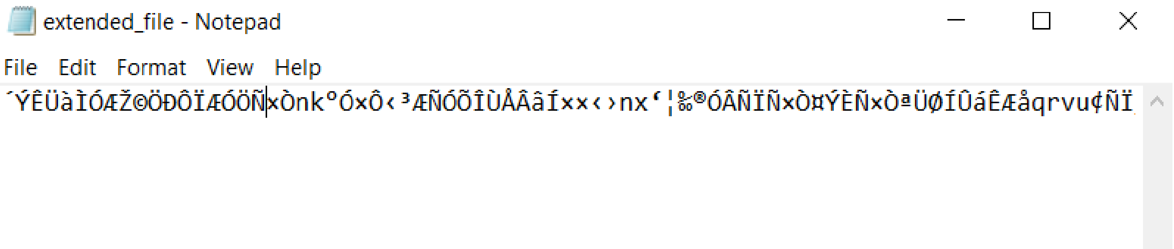
- **Encrypt (File)**

File sebelum dienkripsi:



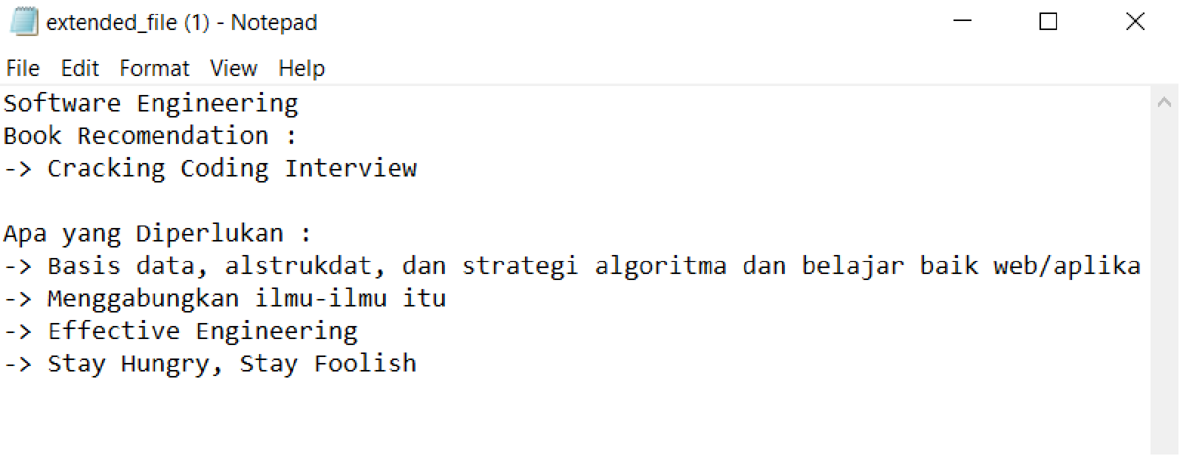
```
Software Engineering
Book Recomendation :
-> Cracking Coding Interview

Apa yang Diperlukan :
-> Basis data, alstrukdat, dan strategi algoritma dan belajar baik web/aplika
-> Menggabungkan ilmu-ilmu itu
-> Effective Engineering
-> Stay Hungry, Stay Foolish
```

File setelah dienkripsi dengan kunci “andhika”:



```
´ÝÊÜàÌÓÆŽ©ÖÐÔÏÆÓÖÑ×Ònk°Ó×Ô‹³ÆÑÓÕÎÙÅÂâÍ××‹›nx‘¦‰®ÓÂÑÏÑ×Ò¤ÝÈÑ×ÒªÜØÍÛáÊÆåqrvu¢ÑÏ
```

- **Decrypt(File):** File setelah didekripsi kembali denan kunci “andhika”:

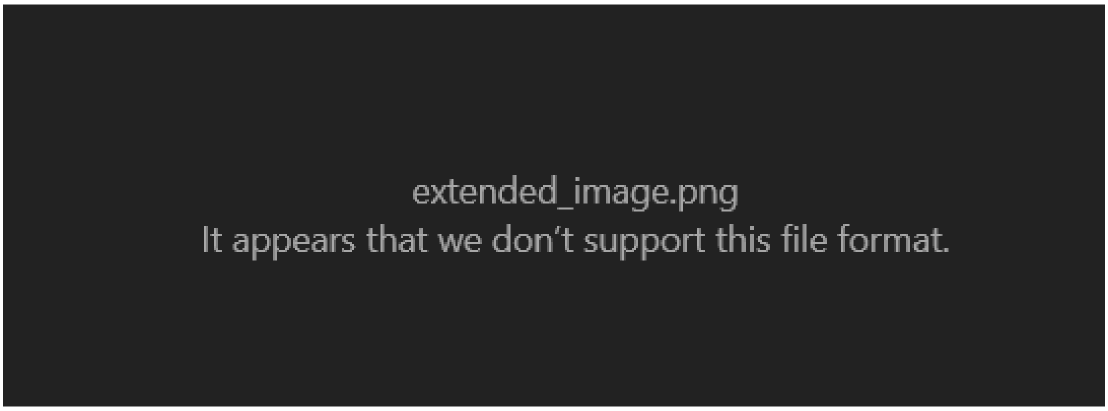


```
Software Engineering
Book Recomendation :
-> Cracking Coding Interview

Apa yang Diperlukan :
-> Basis data, alstrukdat, dan strategi algoritma dan belajar baik web/aplika
-> Menggabungkan ilmu-ilmu itu
-> Effective Engineering
-> Stay Hungry, Stay Foolish
```

- **Encrypt (Image):**

Image sebelum dienkripsi:

Image setelah dienkripsi dengan kunci “putra”:



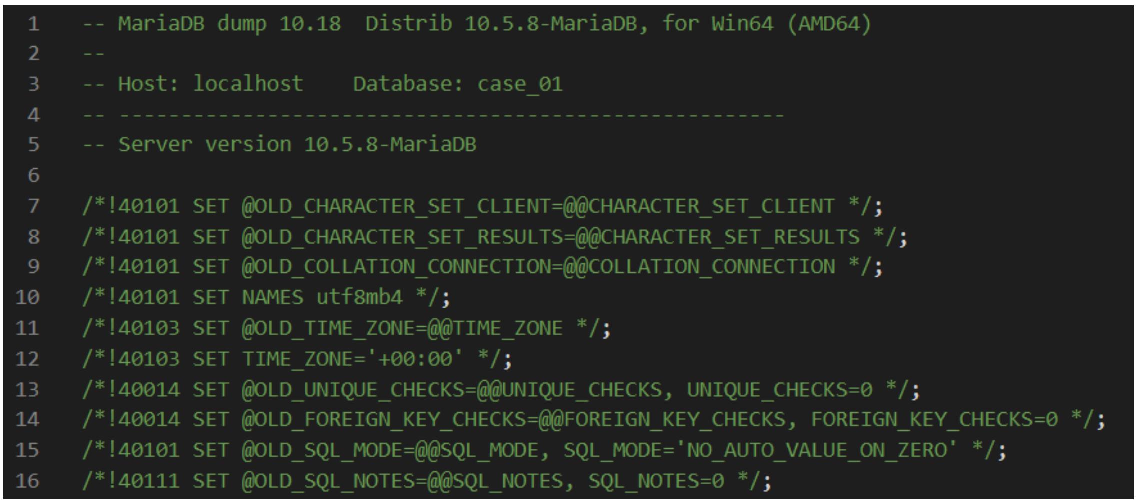
extended_image.png

It appears that we don’t support this file format.

- **Decrypt(Image):** Image setelah didekripsi kembali denan kunci “putra”:

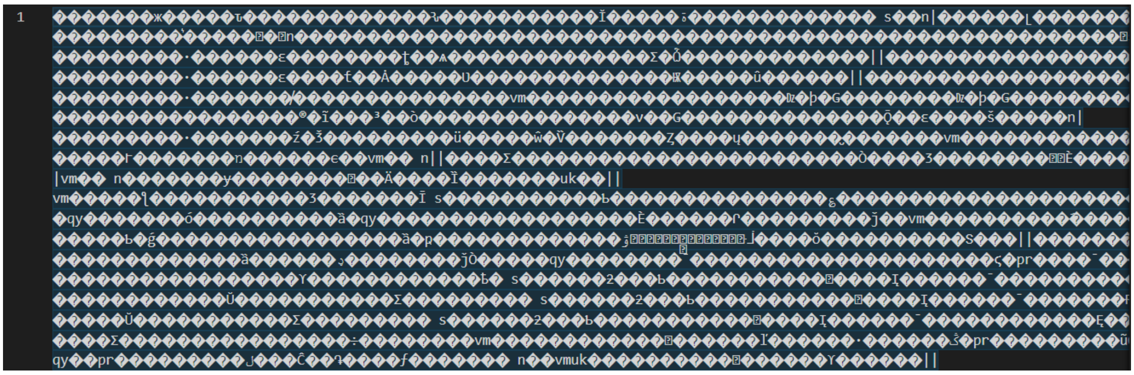
- **Encrypt (SQL):**

  SQL sebelum dienkripsi:

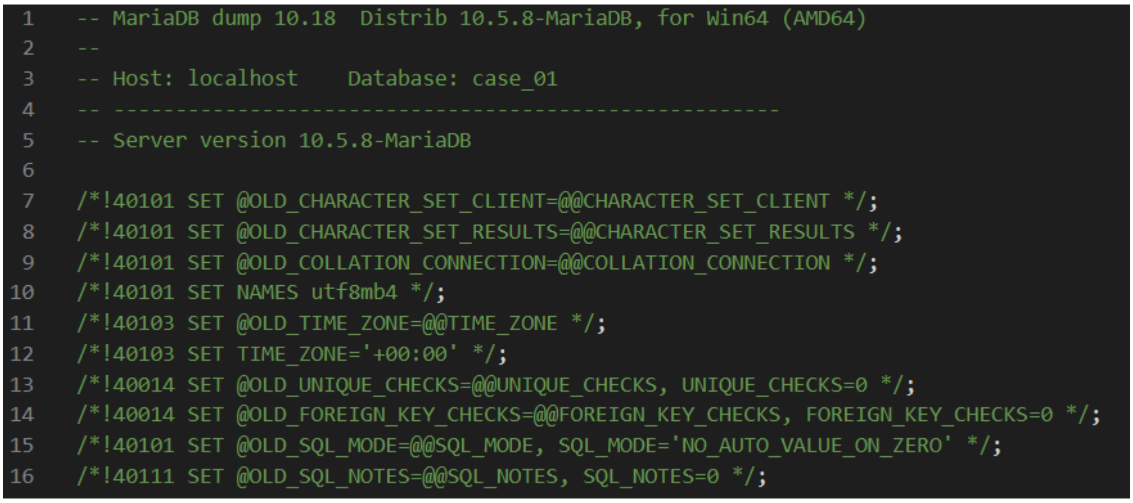


```
-- MariaDB dump 10.18  Distrib 10.5.8-MariaDB, for Win64 (AMD64)
--
-- Host: localhost    Database: case_01
-- ------------------------------------------------------
-- Server version 10.5.8-MariaDB

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
```

SQL setelah dienkripsi dengan kunci “richardo”:



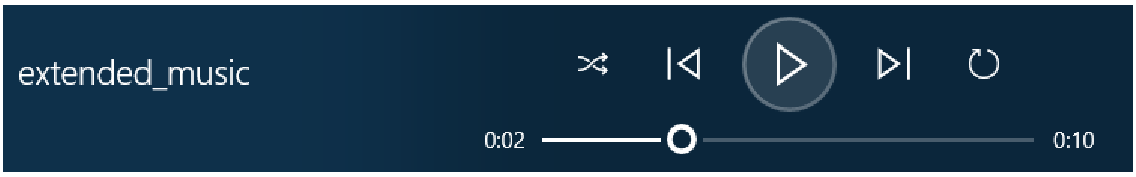
- **Decrypt(SQL):** SQL setelah didekripsi kembali denan kunci “richardo”:



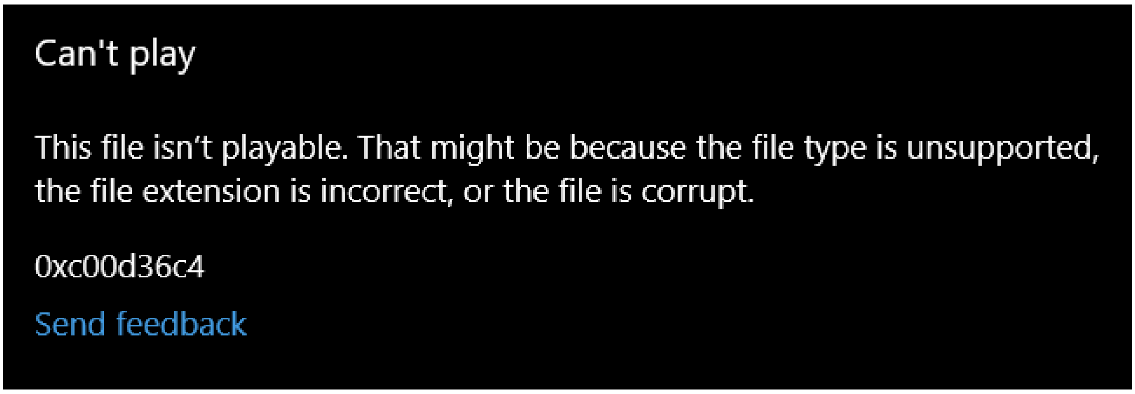
```
-- MariaDB dump 10.18  Distrib 10.5.8-MariaDB, for Win64 (AMD64)
--
-- Host: localhost    Database: case_01
-- ------------------------------------------------------
-- Server version 10.5.8-MariaDB

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
```

- **Encrypt (Music):**

  Music sebelum dienkripsi:



  Music setelah dienkripsi dengan kunci “bandung”:



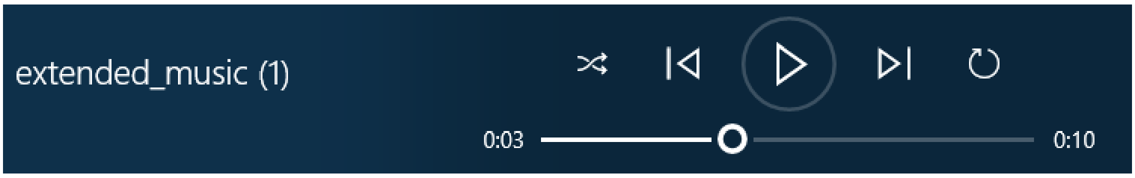
Can't play

This file isn’t playable. That might be because the file type is unsupported, the file extension is incorrect, or the file is corrupt.

0xc00d36c4

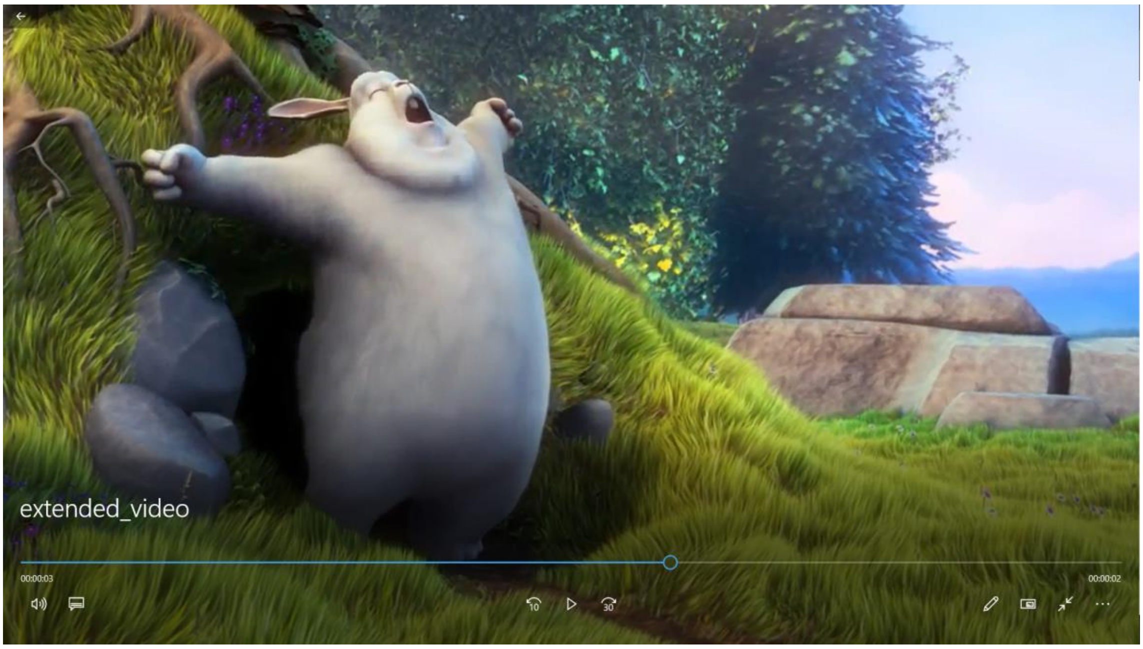
Send feedback

- **Decrypt(Music):** Music setelah didekripsi kembali denan kunci “bandung”:

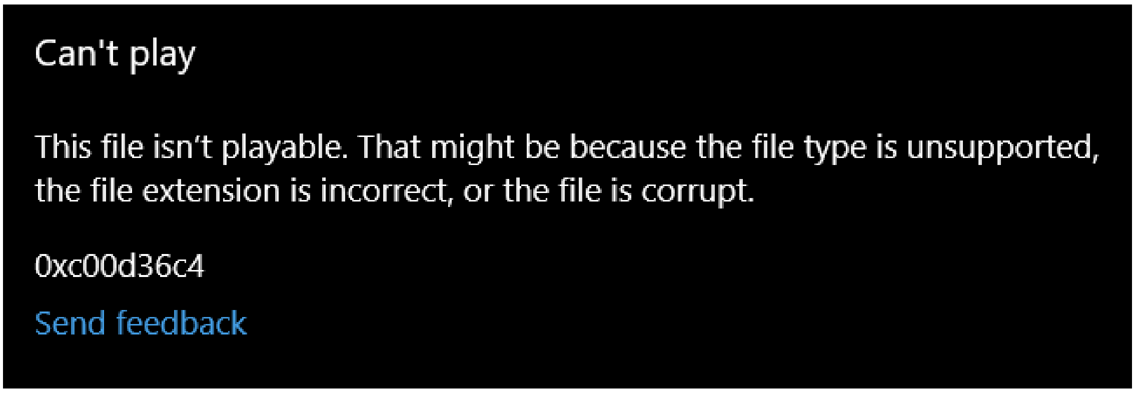


- **Encrypt (Video):**

  Video sebelum dienkripsi:



  Video setelah dienkripsi dengan kunci “teknologi”:



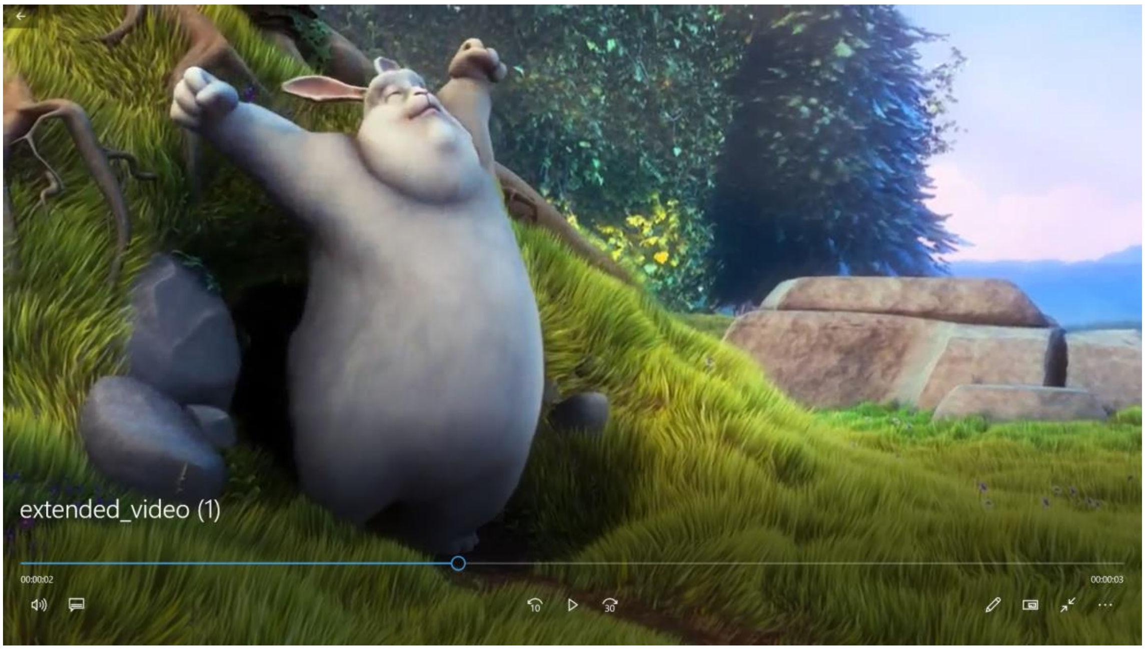
Can't play

This file isn’t playable. That might be because the file type is unsupported, the file extension is incorrect, or the file is corrupt.

0xc00d36c4

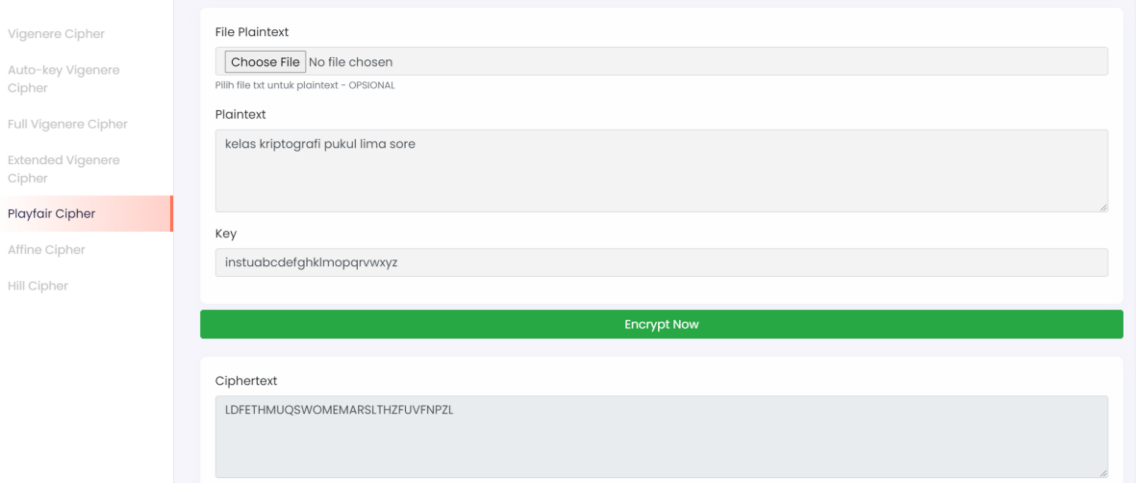
Send feedback

- **Decrypt(Video):** Video setelah didekripsi kembali denan kunci “teknologi”:



e. Playfair Cipher

- **Encrypt:** Plainteks "kelas kriptografi pukul lima sore" dengan kunci "institut"



File Plaintext

No file chosen

Pilih file txt untuk plaintext - OPSIONAL

Plaintext

kelas kriptografi pukul lima sore

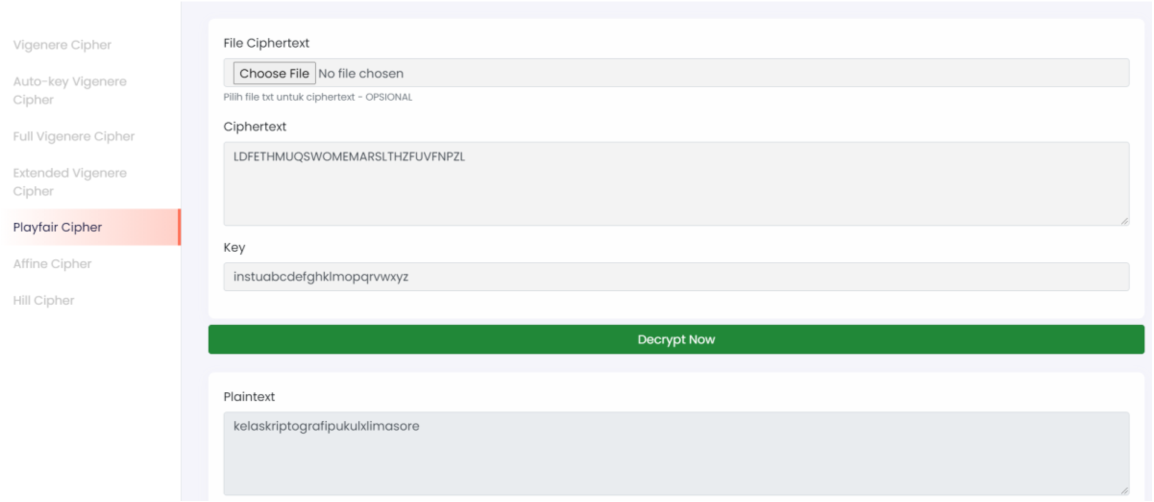
Key

instuabcdefghklmopqrvwxyz

Ciphertext

LDFETHMUQSWOMEMARSLTHZFUVFNPZL

- **Decrypt:** Cipherteks "LDFETHMUQSWOMEMARSLTHZFUVFNPZL" dengan kunci "institut"



File Ciphertext

No file chosen

Pilih file txt untuk ciphertext - OPSIONAL

Ciphertext

LDFETHMUQSWOMEMARSLTHZFUVFNPZL

Key

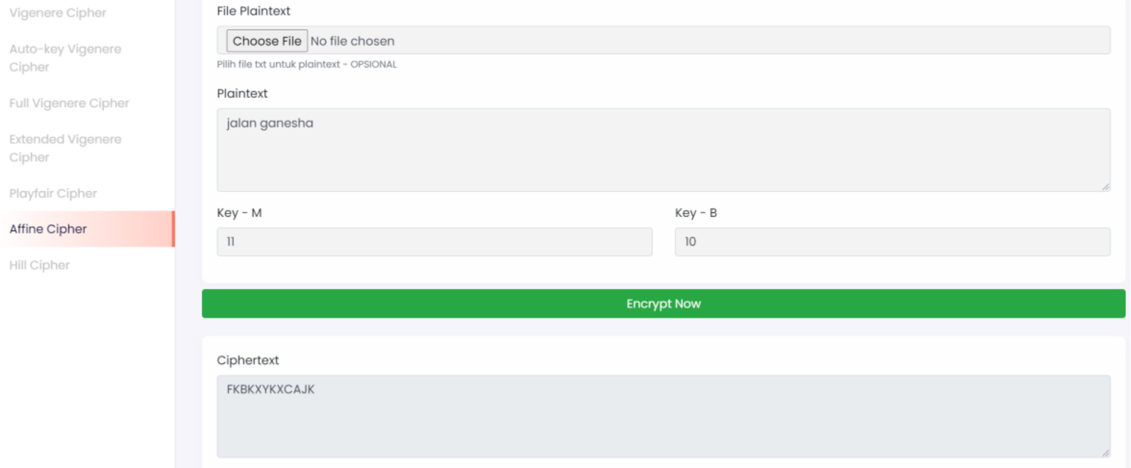
instuabcdefghklmopqrvwxyz

Plaintext

kelaskriptografipukulxlimasore

f. Affine Cipher

- **Encrypt:** Plainteks “jalan ganesha” dengan m = 11 dan b = 10



File Plaintext

No file chosen

Pilih file txt untuk plaintext - OPSIONAL

Plaintext

jalan ganesha

Key - M

11

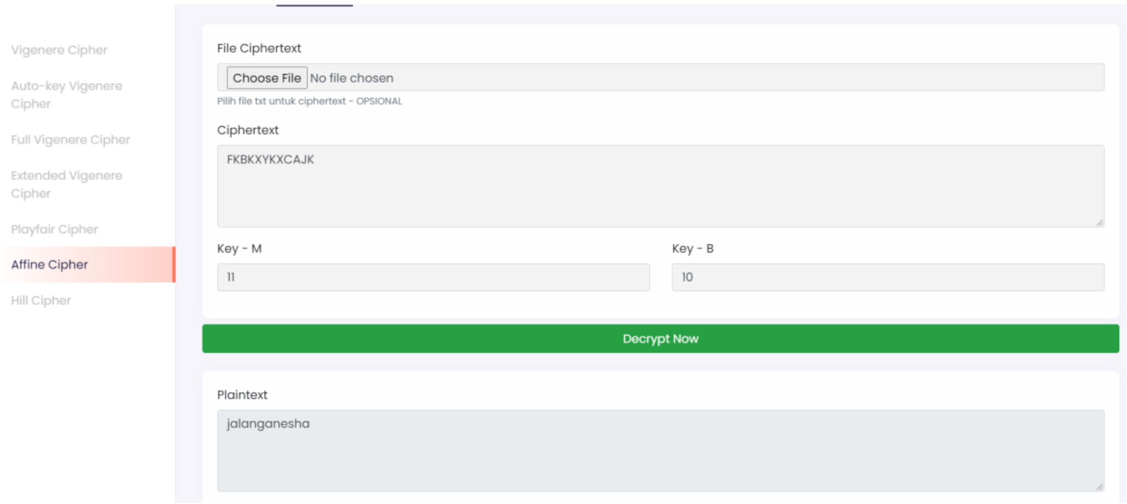
Key - B

10

Ciphertext

FKBKXYKXCAJK

- **Decrypt:** Cipherteks “FKBKXYKXCAJK” dengan m = 11 dan b = 10



File Ciphertext

No file chosen

Pilih file txt untuk ciphertext - OPSIONAL

Ciphertext

FKBKXYKXCAJK

Key - M

11

Key - B

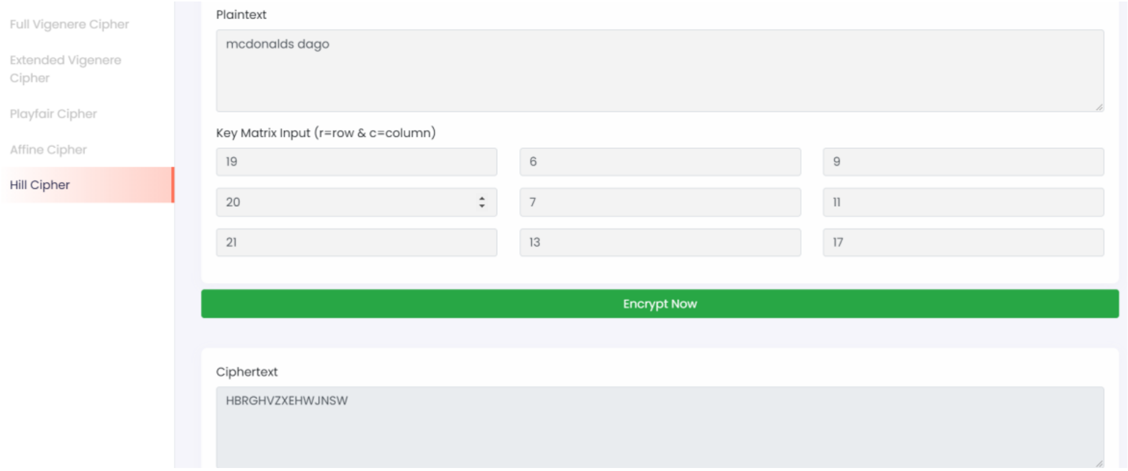
10

Plaintext

jalanganesha

g. Hill Cipher

- **Encrypt:** Plainteks “mcdonalds dago” dengan key [[19, 6, 9], [20, 7, 11], [21, 13, 17]]



Plaintext

mcdonalds dago

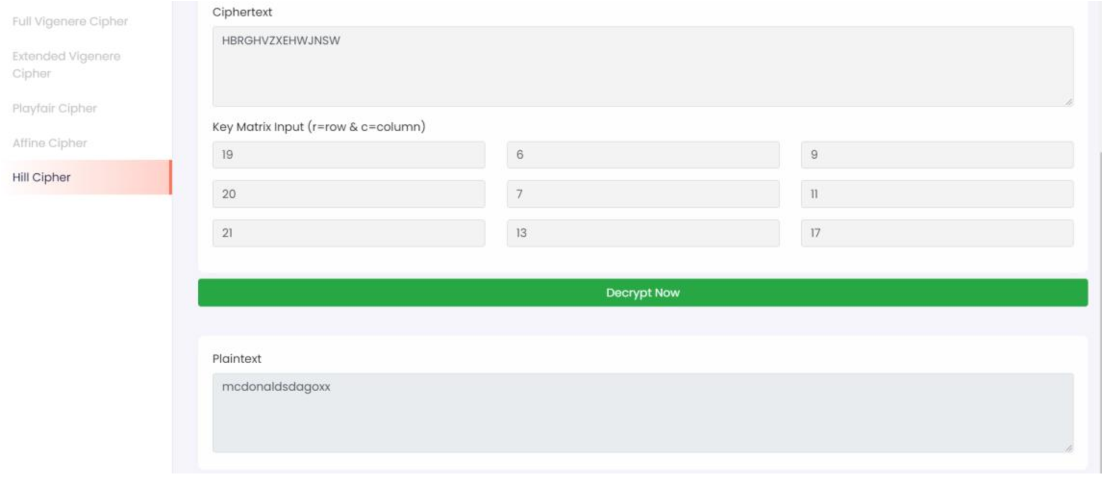
Key Matrix Input (r=row & c=column)

19	6	9
20	7	11
21	13	17

Ciphertext

HBRGHVZXEHWJNSW

- **Decrypt:** Cipherteks "HBRGHVZXEHWJNSW" dengan key [[19, 6, 9], [20, 7, 11], [21, 13, 17]]



Ciphertext

HBRGHVZXEHWJNSW

Key Matrix Input (r=row & c=column)

19	6	9
20	7	11
21	13	17

Plaintext

mcdonaldsdagoxx

4. Link github

https://github.com/AndhikaRei/Classic-Cipher.git

No	Spek	Berhasil	Kurang berhasil	Keterangan
1	Vigenere standard	√		
2	Full Vigenere Cipher	√		
3	Auto-key Vigenere Cipher	√		
4	Extended Vigenere Cipher	√		
5	Playfair Cipher	√		
6	Affine Cipher	√		
7	Bonus: Enigma cipher/Hill Cipher	√		