

Computer Security ALgorihms

Abstract

Computer security and encryption are critical components of modern information technology. Encryption is a method of converting information into a code that can only be decrypted by authorized parties, thereby protecting it from unauthorized access. Computer security involves the use of various technologies and processes to prevent unauthorized access, theft, and damage to computer systems and networks. Both encryption and computer security are essential for protecting sensitive information, such as financial data, personal information, and classified government information, from cyber threats. As technology continues to advance, the need for robust encryption and computer security measures will only increase in importance.

1 Cryptography

Cryptography is a method of using advanced mathematical principles in storing and transmitting data in a particular form so that only those whom it is intended can read and process it. Encryption is a key concept in cryptography – It is a process whereby a message is encoded in a format that cannot be read or understood by an eavesdropper. The technique is old and was first used by Caesar to encrypt his messages using Caesar cipher. A plain text from a user can be encrypted to a ciphertext, then send through a communication channel and no eavesdropper can interfere with the plain text. When it reaches the receiver end, the ciphertext is decrypted to the original plain text.

Cryptosystems

A cryptosystem is an implementation of cryptographic techniques and their accompanying infrastructure to provide information security services. A cryptosystem is also referred to as a cipher system . And is a structure or scheme consisting of a set of algorithms that converts plaintext to ciphertext to encode or decode messages securely.

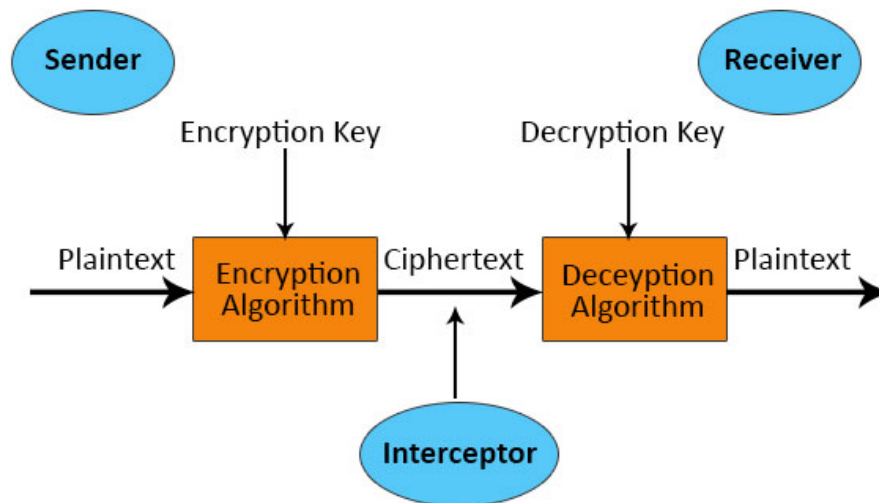


Figure 1: cryptosystem.

Components of cryptosystems

- **Plaintext:** This is the data that needs to be protected.
- **Encryption Algorithm:** This is the mathematical algorithm that takes plaintext as the input and returns ciphertext. It also produces the unique encryption key for that text.
- **Ciphertext:** This is the encrypted, or unreadable, version of the plaintext.
- **Decryption Algorithm:** This is the mathematical algorithm that takes ciphertext as the input and decodes it into plaintext. It also uses the unique decryption key for that text.
- **Encryption Key:** This is the value known to the sender that is used to compute the ciphertext for the given plaintext.
- **Decryption Key:** This is the value known to the receiver that is used to decode the given ciphertext into plaintext.

Types of Cryptosystems

- Symmetric Encryption
- Asymmetric Encryption

Symmetric Encryption

This is the simplest kind of encryption that involves only one secret key to cipher and decipher information. Symmetric encryption is an old and best-known technique. It uses a secret key that can either be a number, a word or a string of random letters. It is a blended with the plain text of a message to change the content in a particular way. The sender and the recipient should know the secret key that is used to encrypt and decrypt all the messages. Blowfish, AES, RC4, DES, RC5, and RC6 are examples of symmetric encryption. The most widely used symmetric algorithm is caesar, monoalphabetic, vigenere,

playfair,Rail Fence,Hill Cipher , Row Transposition Cipher, AES-128.

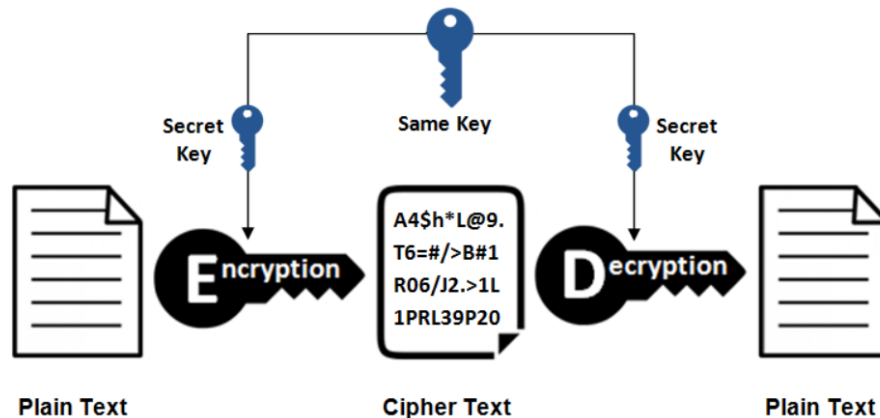


Figure 2: Symmetric Encryption Overview.

The main disadvantage of the symmetric key encryption is that all parties involved have to exchange the key used to encrypt the data before they can decrypt it.

Pros and cons of symmetric encryption

Pros:

- Easier to implement and use
- Faster than asymmetric encryption
- Less resource-intensive
- Good for handling and transferring larger amounts of data

Cons:

- Loss of a key will mean that data encrypted with it is compromised
- Key has to be shared securely with the other party

Asymmetric Encryption

Asymmetric encryption is also known as public key cryptography, which is a relatively new method, compared to symmetric encryption. Asymmetric encryption uses two keys to encrypt a plain text. Secret keys are exchanged over the Internet or a large network. It ensures that malicious persons do not misuse the keys. It is important to note that anyone with a secret key can decrypt the message and this is why asymmetric encryption uses two related keys to boosting security. A public key is made freely available to anyone who might want to send you a message. The second private key is kept a secret so that you can only know.

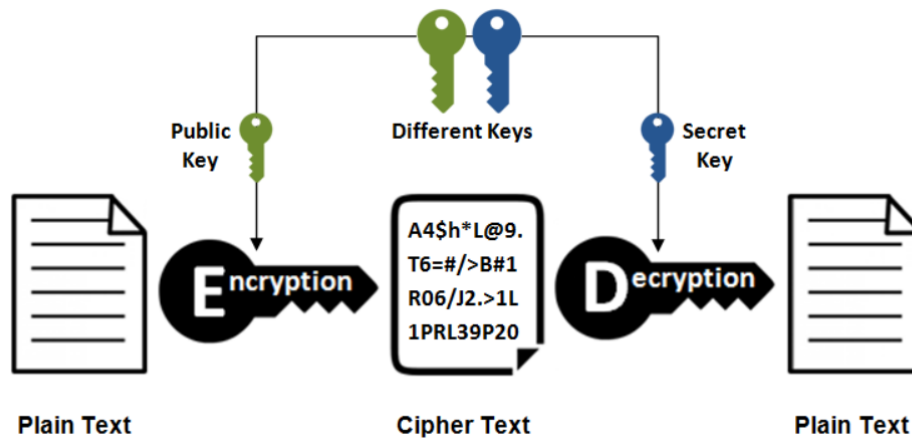


Figure 3: Asymmetric Encryption Overview.

A message that is encrypted using a public key can only be decrypted using a private key, while also, a message encrypted using a private key can be decrypted using a public key. Security of the public key is not required because it is publicly available and can be passed over the internet. Asymmetric key has a far better power in ensuring the security of information transmitted during communication.

Asymmetric encryption is mostly used in day-to-day communication channels, especially over the Internet. Popular asymmetric key encryption algorithm includes RSA, DSA, PKCS.

Pros and cons of asymmetric encryption

Pros:

- Data can only be decrypted using the private key held by the owner
- If the public key is lost or stolen, data won't be compromised
- Provides authentication and non-repudiation in addition to confidentiality

Cons:

- It's slower than symmetric encryption
- Uses more resources
- If the private key is lost, there is no way to retrieve it

Cryptanalysis

cryptanalysis involves the study of cryptographic systems and techniques with the goal of breaking them or finding weaknesses in them. Cryptanalysis involves various techniques such as analyzing ciphertext, guessing keys or algorithms, and exploiting weaknesses in cryptographic protocols. Cryptanalysis is often used to test the strength of cryptographic systems or to break the security of encrypted messages.

In summary, cryptography is the practice of creating and using techniques for secure communication, while cryptanalysis is the practice of analyzing and breaking cryptographic systems to test their strength or to gain unauthorized access to encrypted information.

2 Hashing and Encryption

Hashing and Encryption have different functions. Encryption includes encryption and decryption process while hashing is a one-way process that changes data into the message digest which is irreversible.

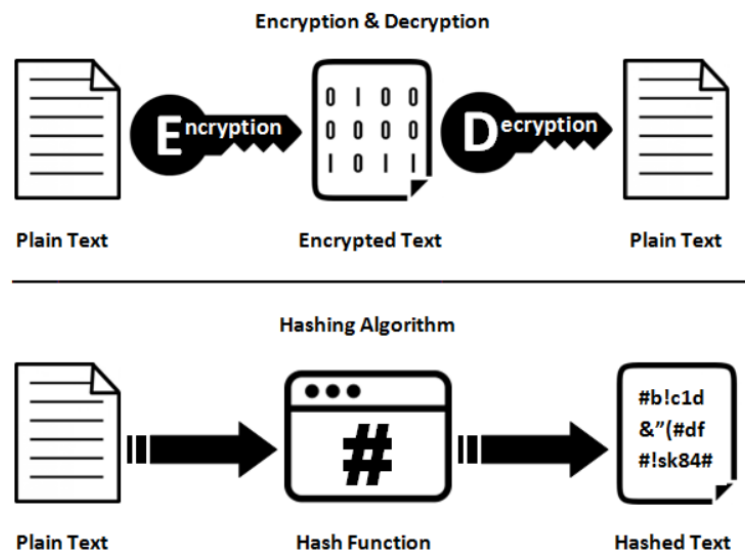


Figure 4: Hashing and Encryption Overview.

Encryption

Encryption is the process of encoding simple text and other information that can be accessed by the sole authorized entity if it has a decryption key. It is the most effective way of achieving data security in modern communication systems. In order for the receiver to read an encrypted message, it should have a security key that is used in decryption. Data that has not been encrypted is known as plain text while encrypting data is known as a cipher text. There are a number of encryption systems, where an asymmetric encryption is also known as public-key encryption, symmetric encryption and hybrid encryption are the most common.

Hashing algorithm

A hashing algorithm is a cryptographic hash function. It is a mathematical algorithm that maps data of arbitrary size to a hash of a fixed size. A hash function algorithm is designed to be a one-way function, infeasible to invert.

3 Symmetric Algorithms

Caesar(shift) Cipher:

The Caesar cipher is a type of **substitution cipher** in which each letter in the plaintext is shifted a certain number of places down the alphabet. For example, if the shift is 5, then the letter A would be replaced by F, the letter G would become E, and so on.

- Example: I want to send a message to my friend, and we agree that we will deal with Caesar to encrypt the messages between us, and since we use symmetric encryption, we have agreed that we will use "11" as a key to encrypt and decrypt the text between us.

the plain Text : eslam ahmed , and the key : 11 **the first step:** We need to put the letters of the alphabet in order and give each letter a number.

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

the second step: remove all space

the third step: We replace each letter of the plain text with the corresponding number in the alphabet as we made in the letters of in the first step alphabet.

E	S	L	A	M	A	H	M	E	D
4	18	11	0	12	0	7	12	4	3

the last step: applying the algorithm

$$E_e(m) = (m + e) \mod 26 \quad (1)$$

The frist letter (E): $E_{11}(E) = (4 + 11) \text{ mode } 26 = 15 \rightarrow P$

The second letter (S): $E_{11}(S) = (18 + 11) \text{ mode } 26 = 3 \rightarrow D$

The letter (L): $E_{11}(L) = (11 + 11) \text{ mode } 26 = 22 \rightarrow W$

The letter (A): $E_{11}(A) = (0 + 11) \text{ mode } 26 = 11 \rightarrow L$

The letter (M): $E_{11}(M) = (12 + 11) \text{ mode } 26 = 23 \rightarrow X$

The letter (A): $E_{11}(A) = (0 + 11) \text{ mode } 26 = 11 \rightarrow L$

The letter (H): $E_{11}(H) = (7 + 11) \text{ mode } 26 = 18 \rightarrow S$

The letter (M): $E_{11}(M) = (12 + 11) \text{ mode } 26 = 23 \rightarrow X$

The letter (E): $E_{11}(E) = (4 + 11) \text{ mode } 26 = 15 \rightarrow P$

The letter (d): $E_{11}(E) = (3 + 11) \text{ mode } 26 = 14 \rightarrow O$

The cipher Text : PDWLXLSXPO

Encryption code

```
def caesar_encryption(text, key):
    result = ""

    # remove all spaces
    text = text.replace(" ", "")

    # transverse the plain text
    for i in range(len(text)):
        char = text[i]
        # Encrypt uppercase characters in plain text

        if (char.isupper()):
            result += chr((ord(char) + key - 65) % 26 + 65)
        # Encrypt lowercase characters in plain text
        elif (char.islower()):
            result += chr((ord(char) + key - 97) % 26 + 97)

    return result
```

Figure 5: Function of Caesar Encryption

Decryption

Decryption is the process of converting an encrypted message back into its original plaintext form. In the context of the Caesar cipher, decryption involves shifting each letter in the encrypted message back up the alphabet by the same shift value used in the encryption process.

The cipher text: PDWLXLSXPO

$$D_e(m) = (m - e) \text{ mod } 26 \quad (2)$$

The letter (P): $D_{11}(P) = (15 - 11) \text{ mode } 26 = 4 \rightarrow E$

The letter (D): $D_{11}(D) = (3 - 11) \text{ mode } 26 = 18 \rightarrow S$

The letter (W): $D_{11}(W) = (22 - 11) \text{ mode } 26 = 11 \rightarrow L$

The letter (L): $D_{11}(L) = (11 - 11) \text{ mode } 26 = 0 \rightarrow A$

The letter (X): $D_{11}(X) = (23 + 11) \text{ mode } 26 = 12 \rightarrow M$

The letter (L): $D_{11}(L) = (11 - 11) \text{ mode } 26 = 0 \rightarrow A$

The letter (S): $D_{11}(S) = (18 - 11) \text{ mode } 26 = 7 \rightarrow H$

The letter (X): $D_{11}(X) = (23 + 11) \text{ mode } 26 = 12 \rightarrow M$

The letter (P): $D_{11}(P) = (15 - 11) \text{ mode } 26 = 4 \rightarrow E$

The letter (O): $D_{11}(O) = (14 - 11) \text{ mode } 26 = 3 \rightarrow D$

The decryption Text : ESLAMAHMED

Decryption code

```
def caesar_decrypt(Text, key):  
    result = ""  
    # transverse the plain text  
    for i in range(len(Text)):  
        char = Text[i]  
        # decrypt uppercase characters in plain text  
  
        if (char.isupper()):  
            result += chr((ord(char) - key - 65) % 26 + 65)  
        # Encrypt lowercase characters in plain text  
        elif(char.islower()):  
            result += chr((ord(char) - key - 97) % 26 + 97)  
  
    return result
```

Figure 6: Function of Caesar Decryption

Monoalphabetic Cipher:

A monoalphabetic cipher is a type of substitution cipher in which each letter in the plaintext is replaced with a fixed letter or symbol in the ciphertext. In other words, each letter in the plaintext is encrypted using the same letter or symbol in the ciphertext, regardless of its position in the message

the steps to perform the Monoalphabetic Cipher:

- 1 Choose a secret key, which is a mapping of each letter in the alphabet to a unique letter or symbol in the ciphertext. like **"Example"**. The dictionary used for encryption will be :

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

This means that the letter A in the plaintext will be replaced by the letter E in the ciphertext, the letter B will be replaced by X, and so on

- 2 Write down the plaintext message that you want to encrypt. like **ESLAMAHME**
- 3 Convert the plaintext message to uppercase or lowercase, depending on the desired case of the output. In this case, we'll use uppercase.
- 4 For each letter in the plaintext message, look up the corresponding letter or symbol in the ciphertext alphabet using the secret key.
- 5 Replace the plaintext letter with the corresponding ciphertext letter or symbol to form the encrypted message.

the letter (E) — > P

the letter (A) – > E

the letter (S) – -> R

the letter (H) – > C

the letter (L) – > H

the letter (M) – > I

the letter (A) – > E

the letter (E) – > P

the letter (M) – > I

the letter (D) – > M

The cipher text: PRHEIECIPM

Encryption Code

```
def mono_encryption(text , word):
    AL = "abcdefghijklmnopqrstuvwxyz"
    AL_list = list(AL)
    word = word+AL
    word_list = list(word)

    # make the dectionary that we will use to encryption/ decryption process
    result = []
    for i in word:
        if i not in result:
            result.append(i)

    # search for all letter in plain text in dectionary and replace it to
    corresponding character
    string= ""
    for i in range(len(text)):
        char = text[i]

        for key in AL_list:
            if (char == key):
                string += result[AL_list.index(key)]
            elif (char == " "):
                string+= " "

    return string
```

Figure 7: Function of Monoalphabetic Encryption

Decryption

To decrypt the message, the receiver would need to have the same secret key used in the encryption process ("EXAMPLE"). They would simply reverse the substitution process by looking up each letter in the ciphertext message in the dectionary and replacing it with the corresponding letter in the plaintext alphabet. For example, the letter Q in the ciphertext would be replaced by the letter R in the plaintext, the letter W would be replaced by X, and so on.

the letter (P) — > E

the letter (E) – > A

the letter (R) – -> S

the letter (C) – > H

the letter (H) – > L

the letter (I) – > M

the letter (E) – > A

the letter (P) – > E

the letter (I) – > M

the letter (M) – > D

The decryption Text : ESLAMAHMED

Decryption code

```
def mono_decryption(text , word):
    AL = "abcdefghijklmnopqrstuvwxyz"
    AL_list = list(AL)
    word = word+AL
    word_list = list(word)

    # make the dictionary that we will use to encryption/ decryption process
    result = []
    for i in dicct:
        if i not in result:
            result.append(i)

    # search for all letter in cipher text in dictionary and replace it to
    # corresponding character
    string= ""
    for i in range(len(text)):
        char = text[i]
        for key in result:
            if (char == key):
                string += AL_list[result.index(key)]
            elif (char == " "):
                string+= " "

    return string
```

Figure 8: Function of Monoalphabetic Decryption

Playfair Cipher:

The Playfair cipher is a polygraphic substitution cipher that encrypts pairs of letters instead of individual letters.

the steps to perform the Playfair cipher:

- 1- Choose a secret key (**EXAMPLE**) , which is a sequence of letters that will be used to generate **a 5x5 grid or matrix**.
The secret key should not contain any repeated letters, and any repeated letters should be removed or replaced.
- 2- Fill the remaining spaces in the grid with the letters of the alphabet, omitting any letters that are already in the key phrase
- 3- Divide the plaintext message into pairs of letters. If the message contains an odd number of letters, add a dummy letter (usually Z) to the end of the message to make it even.

For example: PlainText: "ESLAMAHMEDM"

After Split: 'ES' 'LA' 'MA' 'HM' 'ED' 'MZ'

- 4- If the two letters are the same, add a dummy letter (usually X) after the first letter and encrypt the new pair of letters.

For example: PlainText: "ESLAMMAHMED"

After Split: 'ES' 'LA' 'MX' 'MA' 'HM' 'ED'

5- Encryption :

- If the two letters appear in the same row of the grid, replace each letter with the letter to its right, wrapping around to the left end of the row if necessary. for example **"MX"** will encrypt to **"PX"**

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

- If the two letters appear in the same column of the grid, replace each letter with the letter beneath it, wrapping around to the top of the column if necessary. for example **"AR"** will encrypt to **'CW'**

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

- Otherwise, form a rectangle with the two letters and replace each letter with the letter in the same row but in the column of the other letter in the rectangle. for example **"ES"** will encrypt to **'MO'**

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

Example:

the secret Key: EXAMPLE

The plain text: ESLAMAHMED

The Matrix:

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

The plain text after splitting 'ES' 'LA' 'MA' 'HM' 'ED'

The sub text "ES"

E → M

S → O

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

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

The sub text "LA"

L → C

A → E

The sub text "MA"

M → P

A → M

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

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

The sub text "HM"

H → K

M → X

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

The sub text "ED"

E → M

D → L

The Cipher Text: MOCEPMKXML

Encryption Code

```
def Playfair_encryption(Text , keyword):

    #using the secret key to creat the Matrix 5X5 (grid)
    def matrix(keyword):
        stg = (keyword.lower() + string.ascii_lowercase).replace("j", "")
        matrix = []
        word_list = list(stg)
        result = []

        for i in word_list:
            if i not in result:
                result.append(i)

        for i in range(len(result)):
            char = result[i]
            matrix.append(char)

        matrix = np.array(matrix).reshape(5,5)
        return matrix

    # Split the plain Text to sub text with two letters and check if the plain text
    # is odd to add "z" or if there are litter duplicate in the same sub text
    def Text_new(Text):
        Text = Text.lower().replace(" ", "").replace("j", "i")
        Text_new = list(Text)
        le = len(Text)

        for i in range(le-1):
            if (i % 2 == 0) and (Text_new[i] == Text_new[i+1]):
                Text_new.insert(i+1 , "x")

        if (len(Text_new) % 2) != 0 :
            Text_new.insert(len(Text_new) , 'z')

        Text_arr = np.array(Text_new).reshape(len(Text_new)//2 , 2)

        return Text_arr
```

```

# Search about the plain text in the matrix
# and return the numbers of row and column for the letter
def Search(matr , element):
    for i in range(5):
        for j in range(5):
            if (matr[i][j] == element):
                return i , j

matrix = matrix(keyword)
Text_arr = Text_new(Text)
print(matrix , end = "\n\n")
print(Text_arr , end = "\n\n")

cipher = ""
#applying the playfair algorithm
for i in range(Text_arr.shape[0]):
    ele1 = Text_arr[i][0]
    ele2 = Text_arr[i][1]

    i1 , j1 = Search(matrix , ele1)
    i2 , j2 = Search(matrix , ele2)

    # if the two letters in the same row
    if (i1 == i2):
        cipher += matrix[i1][(j1+1)%5]
        cipher += matrix[i2][(j2+1)%5]

    #if the two letters in the same column
    elif (j1 == j2):
        cipher += matrix[(i1+1)%5][j1]
        cipher += matrix[(i2+1)%5][j2]

    #Otherwise
    else:
        cipher+= matrix[i1][j2]
        cipher += matrix[i2][j1]

return cipher

```

Figure 9: Function of Playfair Encryption

The output runing:

```

[['e' 'x' 'a' 'm' 'p']
 ['l' 'b' 'c' 'd' 'f']
 ['g' 'h' 'i' 'k' 'n']
 ['o' 'q' 'r' 's' 't']
 ['u' 'v' 'w' 'y' 'z']]

```

```

[['e' 's']
 ['l' 'a']
 ['m' 'a']
 ['h' 'm']
 ['e' 'd']]

```

Key text: EXAMPLE
Plain text: ESLAMAHMED
cipher text: MOCEPMKXML

Decryption

To decrypt the ciphertext message, the receiver must know the key phrase used to generate the grid. The decryption process involves reversing the substitution process by performing the inverse of each of the above steps.

- If the two letters appear in the same row of the grid, replace each letter with the letter to its Left, wrapping around to the left end of the row if necessary.
- If the two letters appear in the same column of the grid, replace each letter with the letter above it,
- Otherwise, form a rectangle with the two letters and replace each letter with the letter in the same row but in the column of the other letter in the rectangle.

Example:

the secret Key: **EXAMPLE**

The Cipher Text: *MOCEPMKXML*

The Matrix:

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

The plain text after splitting 'MO' 'CE' 'PM' 'KX' 'ML'

The sub text "MO"

M → E

O → S

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

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

The sub text "*CE*"

C → L

E → A

The subtext: '*PM*'

p → M

M → A

The subtext '*KX*'

k → H

X → M

The subtext *ML*

M → E

L → D

The decryption Text : "*ESLAMAHMED*"

Decryption Code:

```
def Playfair_decryption (Text , keyword):  
    #using the secret key to creat the Matrix 5X5 (grid)  
    def matrix(keyword):  
        stg = (keyword.lower() + string.ascii_lowercase).replace("j", "")  
        matrix = []  
        word_list = list(stg)  
        result = []  
  
        for i in word_list:  
            if i not in result:  
                result.append(i)  
  
        for i in range(len(result)):  
            char = result[i]  
            matrix.append(char)  
  
        matrix = np.array(matrix).reshape(5,5)  
        return matrix  
    # Split the plain Text to sub text with two letters  
    def Text_new(Text):  
        Text_new = list(Text)  
        Text_arr = np.array(Text_new).reshape(len(Text_new)//2 , 2)  
  
        return Text_arr  
    # Search about the plain text in the matrix and return the numbers of row and  
    # column for the letter  
    def Search(matr , element):  
        for i in range(5):  
            for j in range(5):  
                if (matr[i][j] == element):  
                    return i , j
```

```
matrix = matrix(keyword)  
Text_arr = Text_new(Text)  
print(matrix , end = "\n\n")  
print(Text_arr , end = "\n\n")  
plain = ""  
  
#applying the playfair algorithm (Decryption)  
for i in range(Text_arr.shape[0]):  
    ele1 = Text_arr[i][0]  
    ele2 = Text_arr[i][1]  
    i1 , j1 = Search(matrix , ele1)  
    i2 , j2 = Search(matrix , ele2)  
  
    # if the two letters in the same row  
    if (i1 == i2):  
        plain += matrix[i1][((j1+4)%5)]  
        plain += matrix[i2][(j2+4)%5]  
    #if the two letters in the same column  
    elif (j1 == j2):  
        plain += matrix[(i1+4)%5][j1]  
        plain += matrix[(i2+4)%5][j2]  
    #otherwise  
    else:  
        plain += matrix[i1][j2]  
        plain += matrix[i2][j1]  
  
return plain
```

Figure 10: Function of Playfair Decryption

```
[['e' 'x' 'a' 'm' 'p']  
['l' 'b' 'c' 'd' 'f']  
['g' 'h' 'i' 'k' 'n']  
['o' 'q' 'r' 's' 't']  
['u' 'v' 'w' 'y' 'z']]
```

The Output of running :

```
[['m' 'o']  
['c' 'e']  
['p' 'm']  
['k' 'x']  
['m' 'l']]
```

```
Key text:  EXAMPLE  
cipher text: mocepmkxml  
Plain text:  ESLAMAHMED
```

Vigenere Cipher:

The Vigenere cipher is a classical polyalphabetic cipher that uses a series of interwoven Caesar ciphers to encrypt plaintext messages. The Vigenère Cipher encrypts n alphabetic characters at a time: each plaintext element is equivalent to n alphabetic characters. Such a cipher is called polyalphabetic, A cipher is called monoalphabetic if "each alphabetic character is mapped to a unique alphabetic character".

the steps to perform the Vigenere Cipher:

- 1 Choose a secret keyword or phrase that will be used to encrypt the plaintext message.
 - 2 Write the keyword repeatedly above the plaintext message, aligning the first letter of the keyword with the first letter of the plaintext message.
 - 3 Convert each letter of the keyword and the plaintext message into their numerical equivalents, using a standard alphabetical numbering system where A=0, B=1, C=2, and so on.
 - 4 For each letter of the plaintext message, find the corresponding letter of the keyword above it.
 - 5 Add the numerical values of the plaintext and keyword letters together, using modular arithmetic to ensure that the result is within the range of the alphabetical numbering system.
 - 6 Convert the resulting numerical value back into a letter, using the same alphabetical numbering system.
 - 7 Write the encrypted letter below the plaintext letter and the corresponding keyword letter.
- Repeat steps 4-7 for each letter of the plaintext message.
 - The resulting encrypted message is the collection of the letters written below the plaintext message and the keyword.

Example:

The Secret key: "LOMA"

The plain text: "ESLAMAHMED"

Secret key	L	O	M	A	L	O	M	A	L	O
K	11	14	12	0	11	14	12	0	11	14
Plain text	E	S	L	A	M	A	H	M	E	D
M	4	18	11	0	12	0	7	12	4	3
(M+K) mod 26	15	6	23	0	23	14	19	12	15	17
The cipher text	P	G	X	A	X	O	T	M	P	R

The Cipher text: PGXAXOTMPR

Encryption Code

```
def vigenere_encryption (Text , Keyword):
    stg = list(string.ascii_lowercase)

    Keyword = list( Keyword.lower())
    Text = Text.lower()
    # to convert the secret key to list of letters positions in alphabet
    keylist= []
    for i in range(len(Keyword)) :
        for j in range(len(stg)):
            if (stg[j]== Keyword[i]):
                keylist.append(j)

    # to convert the plain Text to list of letters positions in alphabet
    Textlist = []
    for i in range(len(Text)):
        for j in range(len(stg)):
            if (Text[i] == stg[j]):
                Textlist.append(j)

    #applay vigenere encryption algorithm (m+k) %26
    result=""
    for i in range (len(Textlist)):
        result+= stg[( Textlist[i] + (keylist[(i%len(keylist))]))%26]

    print("Keylists ", keylist , end="\n\n" )
    print("Textlist ", Textlist , end= "\n\n")

    return result
```

Figure 11: Function of Vigenere encryption

The Output of running :

```
Keylists  [11, 14, 12, 0]

Textlist  [4, 18, 11, 0, 12, 0, 7, 12, 4, 3]

Key text:  LOMA
Plain text: ESLAMAHMED
cipher text: PGXAXOTMPR
```

Decryption

The resulting encrypted message is the collection of the letters written below the plaintext message and the keyword.

The secret key 'LOMA'

The cipher text: 'PGXAXOTMPR'

Secret key	L	O	M	A	L	O	M	A	L	O
K	11	14	12	0	11	14	12	0	11	14
The cipher text	P	G	X	A	X	O	T	M	P	R
M	15	6	23	0	23	14	19	12	15	17
(M - K) mod 26	4	18	11	0	12	0	7	12	4	3
Plain text (decryption)	E	S	L	A	M	A	H	M	E	D

The decryption text: 'ESLAMAHMED'

Decryption code

```
def vigenere_decryption (Text , Keyword):
    stg = list(string.ascii_lowercase)
    Text = Text.lower()
    Keyword = list( Keyword.lower())

    # to convert the secret key to list of letters positions in alphabet
    keylist= []
    for i in range(len(Keyword)):
        for j in range(len(stg)):
            if (stg[j]== Keyword[i]):
                keylist.append(j)

    # to convert the cipher Text to list of letters positions in alphabet
    Textlist = []
    for i in range(len(Text)):
        for j in range(len(stg)):
            if (Text[i] == stg[j]):
                Textlist.append(j)

    #applay vigenere decryption algorithm (m-k) %26
    result=""
    for i in range (len(Textlist)):
        result+= stg[(Textlist[i] - keylist[(i-len(keylist))%len(keylist))%26]

    print("Keylists ", keylist , end="\n\n" )
    print("Textlist ", Textlist , end= "\n\n")
    return result
```

Figure 12: Function of Vigenere decryption

The Output of running :

```
Keylists  [11, 14, 12, 0]
Textlist  [15, 6, 23, 0, 23, 14, 19, 12, 15, 17]
Key text:  LOMA
cipher text: PGXAXOTMPR
Plain text: ESLAMAHMED
```

Hill cipher:

The Hill cipher is a polygraphic substitution cipher that uses matrix operations to encrypt and decrypt plaintext messages. It is a polygraphic substitution cipher, which means that it encrypts multiple letters of the plaintext at a time.

the steps to perform the Hill Cipher:

- 1 Choose a matrix key that will be used to encrypt the plaintext message. The size of the matrix key determines the number of letters that are encrypted at a time.
- 2 Divide the plaintext message into blocks of letters that correspond to the size of the matrix key.
- 3 Convert each block of letters into a numerical matrix, using a standard alphabetical numbering system where A=0, B=1, C=2, and so on.
- 4 Multiply each matrix block by the matrix key, using matrix multiplication. The resulting matrix is the encrypted matrix.
- 5 Convert the encrypted matrix back into letters, using the same alphabetical numbering system.
- 6 Combine the encrypted letters from each block to obtain the encrypted message.

Example:

Encryption:

The Secret key: "1234"

The plain text: "ESLAMAHMED"

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

The secret key matrix :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- convert the plain text to matrix of sub text

The sub text : 'ES'

$$\begin{bmatrix} E \\ S \end{bmatrix}$$

$$\begin{bmatrix} 4 \\ 18 \end{bmatrix}$$

The sub text : 'LA'

$$\begin{bmatrix} L \\ A \end{bmatrix}$$

$$\begin{bmatrix} 11 \\ 0 \end{bmatrix}$$

The sub text : 'MA'

$$\begin{bmatrix} M \\ A \end{bmatrix}$$

$$\begin{bmatrix} 12 \\ 0 \end{bmatrix}$$

The sub text : 'HM'

$$\begin{bmatrix} H \\ M \end{bmatrix}$$

$$\begin{bmatrix} 7 \\ 12 \end{bmatrix}$$

The sub text : 'ED'

$$\begin{bmatrix} E \\ D \end{bmatrix}$$

$$\begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

1stVector

$$\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 4 \\ 18 \end{bmatrix} \right) \text{mod } 26 = \begin{bmatrix} 14 \\ 6 \end{bmatrix} = \begin{bmatrix} O \\ G \end{bmatrix}$$

2ndVector

$$\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 11 \\ 0 \end{bmatrix} \right) \text{mod } 26 = \begin{bmatrix} 11 \\ 7 \end{bmatrix} = \begin{bmatrix} L \\ H \end{bmatrix}$$

3rdVector

$$\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 12 \\ 0 \end{bmatrix} \right) \text{mod } 26 = \begin{bmatrix} 12 \\ 10 \end{bmatrix} = \begin{bmatrix} M \\ K \end{bmatrix}$$

$4^{rd}Vector$

$$\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 7 \\ 12 \end{bmatrix} \right) \bmod 26 = \begin{bmatrix} 5 \\ 17 \end{bmatrix} = \begin{bmatrix} F \\ R \end{bmatrix}$$

$5^{rd}Vector$

$$\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 4 \\ 3 \end{bmatrix} \right) \bmod 26 = \begin{bmatrix} 10 \\ 24 \end{bmatrix} = \begin{bmatrix} K \\ Y \end{bmatrix}$$

'The cipher Text' 'OGLHMKFRKY'

Encryption Code

```
def hill_encryption ( text , key):
    #func: creat secret matrix
    def matrix (key):
        sqrt = int(np.sqrt(len(key)))
        key = list(key)
        key_matrix = np.array(list(map(int, key))).reshape(sqrt, sqrt)
        return key_matrix

    # func: add padding letters if the text not the same length of the matrix key
    # splilt the Text to sub text with the length of the matrix key , creat the
    vectors of sub Text
    def Text_new(text):
        sqrt = int(np.sqrt(len(key)))
        text = list(text.lower().replace(" ", ""))
        alphabet = list(string.ascii_lowercase)
        while (len(text)%sqrt) != 0 :
            text.append("z")

        text_new = []
        for i in range(len(text)):
            for j in range(len(alphabet)):
                if text[i] == alphabet[j]:
                    text_new.append(j)

        num = len (text) // sqrt
        text = np.array(text_new).reshape(num , sqrt , 1)
        return text

    Matrix = matrix (key)
    Text = Text_new(text.lower())
    print("Matrix" );print( Matrix);print()
    print("The plain text");print(Text)

    #apply hill cipher encryption
    result = ""
    for i in range(len(Text)):
        sub = np.mod(np.dot( Matrix , Text[i]), 26)
        for j in range(len(Matrix)):
            result+= chr(int(sub[j])+97)

    return result
```

Figure 13: Function of Hill encryption

The running output:

```
the secret key: 1234
The plain text: ESLAMAHMED
Matrix
[[1 2]
 [3 4]]

The plain text
[[[ 4]
 [18]]

 [[11]
 [ 0]]

 [[12]
 [ 0]]

 [[ 7]
 [12]]

 [[ 4]
 [ 3]]]

'OGLHMKFRKY'
```

Rail fence Cipher

The Rail Fence Cipher is a transposition cipher that rearranges the plaintext by writing it in a zigzag pattern along diagonal lines. It is a transposition cipher, which means that it rearranges the order of the plaintext without changing the actual letters.

the steps to perform the Rail fence Cipher:

- 1 Choose a key that will be used to determine the number of rows used to write the plaintext in a zigzag pattern.
- 2 Write the plaintext message in a zigzag pattern along diagonal lines, starting from the top row to the bottom row and then back up to the top row again.
- 3 Read off the letters from each row in order from left to right to obtain the ciphertext message.

Exmample

'The Key number: ' 3'

"The plain Text: " 'ESLAMAHMED'

Matrix (3,10) : key -> number of rows , length of plain text -> number of columns

E				M				E	
	S		A		A		M		D
		L				H			

"the Cipher text:" *'EMESAAMDHLH'*

Encryption Code

```
def rail_fence_encrypt( text, key):
    text = list(text)
    matrix = [[""] * len(text) for rows in range(key)]
    k = [ i for i in range (key)]
    k_iv = k[1:-1]
    k_iv.reverse()
    i = 0
    lin = len(text)
    # creat matrix with letters in diagonal up to down ans so on
    # Rail fence "encryption"
    while i < lin:
        for j in k :
            if text != []:
                matrix[j][i] = text[0]
                text.pop(0)
                i+=1
        for n in k_iv:
            if text !=[]:
                matrix[n][i] = text[0]
                text.pop(0)
                i+=1
    print(np.array(matrix).reshape((key , lin)))

    #read the cipher text from the matrix
    result = ""
    for i in range(key):
        for j in range(lin):
            result+=matrix[i][j]

    return result
```

Figure 14: Function of Rail fence decryption

The Output of running :

```
[['E'  ''  ''  ''  'M'  ''  ''  ''  'E'  '']]
[[''  'S'  ''  'A'  ''  'A'  ''  'M'  ''  'D']]
[[''  ''  'L'  ''  ''  ''  ''  'H'  ''  '']]

'EMESAAMDHLH'
```

Decryption

- 1 Choose a key that will be used to determine the number of rows used to write the plaintext in a zigzag pattern.
- 2 Reverses the encryption process by writing the letters in the destined places in the matrix.
- 3 Read off the letters from left to right in the zigzag pattern to obtain the plaintext message.

Example

the key number: "3"

the cipher text: "EMESAAMD LH"

As a first step, determine the places of the letters

*				*				*	
	*		*		*		*		*
		*				*			

As a second step, replace "*" with letters row by row

E				M				E	
	S		A		A		M		D
		L				H			

last step , read the message in the zigzag pattern to obtain the plaintext message.

E				M				E	
	S		A		A		M		D
		L				H			

The decryption text: *ESLAMAHMED*

Decryption Code

```
def rail_fence_decrypt( text, key):
    text = list(text)
    matrix = [["*" * len(text) for rows in range(key)]]
    text_ = ["*" for i in range(len(text))]
    k = [ i for i in range (key)]
    k_iv = k[1:-1]
    k_iv.reverse()
    i = 0
    lin = len(text)

    #put "*" in the correct place like encryption
    while i < lin:
        for j in k :
            if text_ != []:
                matrix[j][i] = text_[0]
                text_.pop(0)
                i+=1
        for n in k_iv:
            if text_ != []:
                matrix[n][i] = text_[0]
                text_.pop(0)
                i+=1

    print(np.array(matrix).reshape((key , lin)) , end='\n''\n')

    #replace "*" for corect litter
    index = 0
    for i in range(key):
        for j in range(lin):
            if matrix[i][j] == "*":
                matrix[i][j] = text[index]
                index+=1

    print(np.array(matrix).reshape((key , lin)))
    result = ""
    l = 0
    while l < lin:
        for j in k :
            if text != [] and matrix[j][l]!="*":
                result += matrix[j][l]
                text.pop()
                l+=1
        for n in k_iv:
            if text!=[] and matrix[n][l]!="*":
                result+= matrix[n][l]
                text.pop()
                l+=1

    return result
```

Figure 15: Function of Rail fence decryption

```
[['*' '*' '*' '*' '*' '*' '*' '*' '*' '*']]
[['*' '*' '*' '*' '*' '*' '*' '*' '*' '*']]
[['*' '*' '*' '*' '*' '*' '*' '*' '*' '*']]
```

The Output of running :

```
[['E' '*' '*' '*' 'M' '*' '*' '*' 'E' '*']]
[['' 'S' '*' 'A' '*' 'A' '*' 'M' '*' 'D']]
[['' '*' 'L' '*' '*' '*' 'H' '*' '*' '*']]

'ESLAMAHMED'
```

Row Transposition Cipher

Row Transposition Cipher is a type of transposition cipher that involves rearranging the order of the characters in a message according to a specific pattern.

the steps to perform the Rail fence Cipher:

- Choose a keyword or phrase. This will be used to determine the order in which the characters in the message are rearranged.
- Write the keyword or phrase at the top of a grid, with each letter in its own column. I
- Write the message to be encrypted underneath the keyword, row by row , filling in any empty cells with a filler character (such as "Z").
- Rearrange the columns in the grid according to the alphabetical order of the letters in the keyword.
- Read off the characters in the grid column by column, starting from the leftmost column and moving to the right.

Example

the Secret key: '3214'

the plain text : 'ESLAMAHMED'

In the first step we make the matrix, the number of its columns is the same as the number of letters of the key, the number of rows is the result of dividing the length of the plain text by the key rounded up

The letters are placed in the matrix row by row

3	2	1	4
E	S	L	A
M	A	H	M
E	D	Z	Z

We arrange the columns in ascending order according to the secret key letters

1	2	3	4
L	S	E	A
H	A	M	M
Z	D	E	Z

Read off the characters in the grid column by column, starting from the leftmost column and moving to the right.

the cipher text: 'LHZSADEMEAMZ'

Encryption code

```
def row_encryption(text, key):
    text_le = float(len(text))
    text_list = list(text)
    key_list = sorted(list(key))
    col = len(key)
    row = int(math.ceil(text_le / col))

    # add the padding letter "z"
    fill_null = int((row * col) - text_le)
    text_list.extend('z' * fill_null)

    # create Matrix and insert message and padding characters row-wise
    matrix = [text_list[i: i + col]
               for i in range(0, len(text_list), col)]

    # read matrix column-wise using key
    result = ""
    index = 0
    for _ in range(col):
        curr_idx = key.index(key_list[index])
        result += ''.join([row[curr_idx]
                           for row in matrix])
        index += 1

    return result
```

Figure 16: Function of Row Transposition encryption

The Output of running :

```
plain text ESLAMAHMED
the key 3214
cipher text LHZSADEMEAMz
```


Decryption

- Obtain the keyword that was used to encrypt the message. This is necessary to determine the order in which the characters were rearranged.
- Write the groups of ciphertext in a grid, with each group of characters in its own column.
- Rearrange the columns in the grid to match the alphabetical order of the letters in the keyword.
- Read off the characters in the grid row by row, starting from the top and moving downwards.

Example

the secret key: '3214'

'the cipher text 'LHZSADEMEAMZ'

frist step , split cipher text to group based on lenght of key

3 → LHZ , 2 → SAD , 1 → EME , 4 → AMZ

3	2	1	4
L	S	E	A
H	A	M	M
Z	D	E	Z

Rearrange the columns in the matrix

1	2	3	4
E	S	L	A
M	A	H	M
E	D	Z	Z

the decryption text: 'ESLAMAHMEDZZ'

Decryption code

```
def row_decryption(text , key):

    text_len = float(len(text))
    text_list = list(text)
    col = len(key)

    row = int(math.ceil(text_len / col))
    key_list = sorted(list(key))

    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]

    result=""
    result_index = 0
    index = 0
    for _ in range(col):
        curr_idx = key.index(key_list[index])

        for j in range(row):
            dec_cipher[j][curr_idx] = text_list[result_index]
            result_index += 1
        index += 1

    try:
        result = ''.join(sum(dec_cipher, []))
    except TypeError:
        raise TypeError()

    return result
```

Figure 17: Function of Row Transposition decryption

The Output of running :

```
cipher text LHZSADEMEAMz
the key 3214
plain text ESLAMAHMEDzz
```

Program:

I've compiled all the algorithms we've discussed here into a simple program (*version* : 0.1).

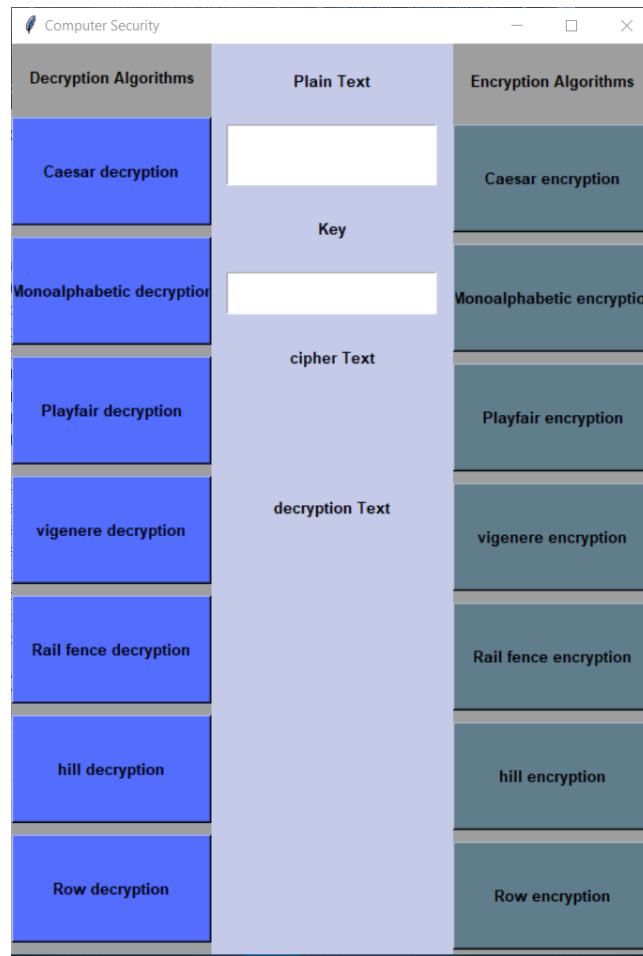
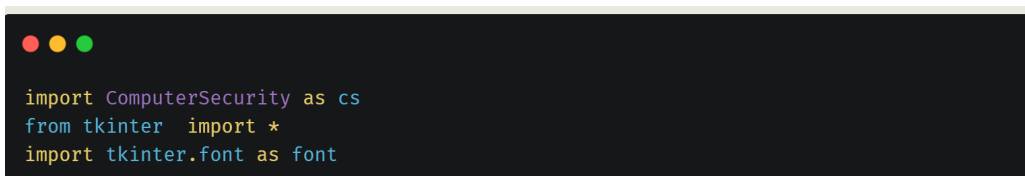


Figure 18: This is the interface of the program

This is the interface of the program:

- The left field: it has buttons for decryption algorithms.
- The middle field is represented by two text fields to obtain the plain text and the secret key - and two Labels to show the cipher text and the decryption text and an other label that shows the type of algorithm .
- The right field has buttons Specialized in encryption algorithms

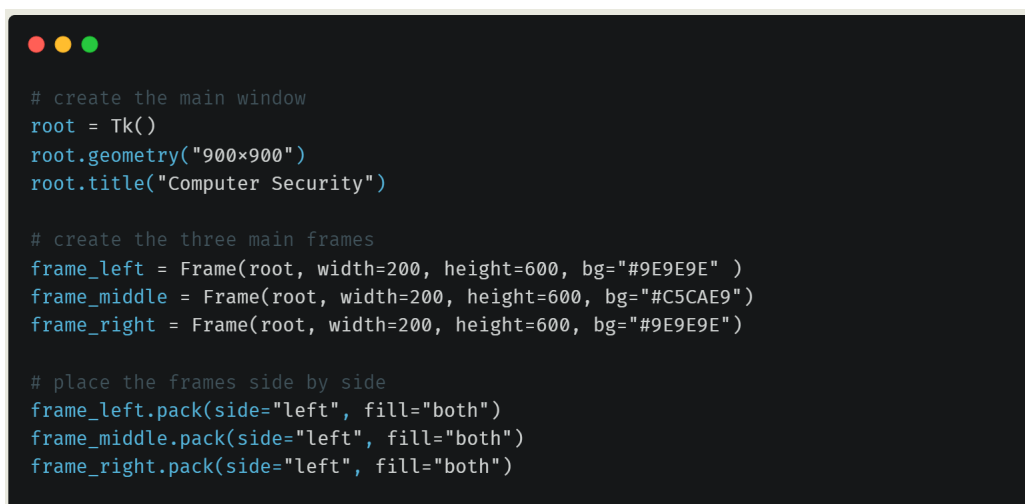
Program Code:



```
import ComputerSecurity as cs
from tkinter import *
import tkinter.font as font
```

Figure 19: program imports

- The "**ComputerSecurity**" module: It is the combined module of all the encryption and decryption functions that we discussed above such as (caesar-encryption , mono-decryption).
- "**Tkinter**" is a standard Python library for creating graphical user interfaces (GUIs).



```
# create the main window
root = Tk()
root.geometry("900x900")
root.title("Computer Security")

# create the three main frames
frame_left = Frame(root, width=200, height=600, bg="#9E9E9E" )
frame_middle = Frame(root, width=200, height=600, bg="#C5CAE9")
frame_right = Frame(root, width=200, height=600, bg="#9E9E9E")

# place the frames side by side
frame_left.pack(side="left", fill="both")
frame_middle.pack(side="left", fill="both")
frame_right.pack(side="left", fill="both")
```

Figure 20: Create a main window with three main frames

- The **Tk()** function creates the main window, and the **geometry()** function sets the size of the window. The **title()** function sets the title of the window.
- The **Frame()** function creates '*three frames*' with different background colors, widths, and heights. The '*width and height*' parameters set the dimensions of the frames, while the '*bg*' parameter sets the background color.
- The **pack()** function is used to place the frames side by side. The '*side*' parameter specifies the side of the main window where the frame will be placed, and the '*fill*' parameter specifies how the frame should fill the space allocated to it.



```

button1 = Button(frame_left,font =myFont, text="Caesar decryption" ,bg= "#536DFE" ,
command= caesar_decryption ,height= 5, width=20)

button1.pack(pady=5)

```

Figure 21: create a button widget.

- The **Button()** function is used to create the button widget, and it takes several arguments to customize its appearance and behavior.
- The *frame-left* parameter specifies the **parent frame** where the button will be placed. The *'font'* parameter specifies the font to be used for the button label. The *'text'* parameter sets the label text . The *'bg'* parameter sets the background color of the button .
- The *'command'* parameter is used to specify the function that will be called when the button is clicked.
- The *height and width* parameters set the height and width of the button in pixels.
- The **pack()** function is used to add the widget to the frame and set its position and size. The *'pady'* parameter sets the vertical padding (in pixels) around the widget to 5.



```

label1 = Label(frame_middle,font =myFont, text="Plain Text", bg="#C5CAE9" ,height=
3, width=20)

label1.pack(pady=5)

```

Figure 22: create a Label widget.

- The **Label()** function is used to create the label widget, and it takes several arguments to customize its appearance and behavior.
- The *'frame-middle'* parameter specifies the **parent frame** where the label will be placed.The *'font'* parameter specifies the font to be used for the label text. The text parameter sets the label text .The *'bg'* parameter sets the background color of the label.
- The *'height' and 'width'* parameters set the height and width of the label in pixels.
- The **pack()** function is used to add the widget to the frame and set its position and size. The *'pady'* parameter sets the vertical padding (in pixels) around the widget to 5.

```

text1 = Text(frame_middle , height= 3, font= myFont, width=25)

text1.pack(pady=5)

```

Figure 23: create a Text widget.

- The **Text()** function is used to create the text input widget, and it takes several arguments to customize its appearance and behavior.
- The '*frame-middle*' parameter specifies the **parent frame** where the text input widget will be placed. The '*font*' parameter specifies the font to be used for the text input. The '*height*' parameter sets the height of the text input widget in lines. The '*width*' parameter sets the width of the text input widget in characters.
- The **pack()** function is used to add the widget to the frame and set its position and size. The '*pady*' parameter sets the vertical padding (in pixels) around the widget to 5.

Examples of running the program:

Caesar Encryption and Decryption:

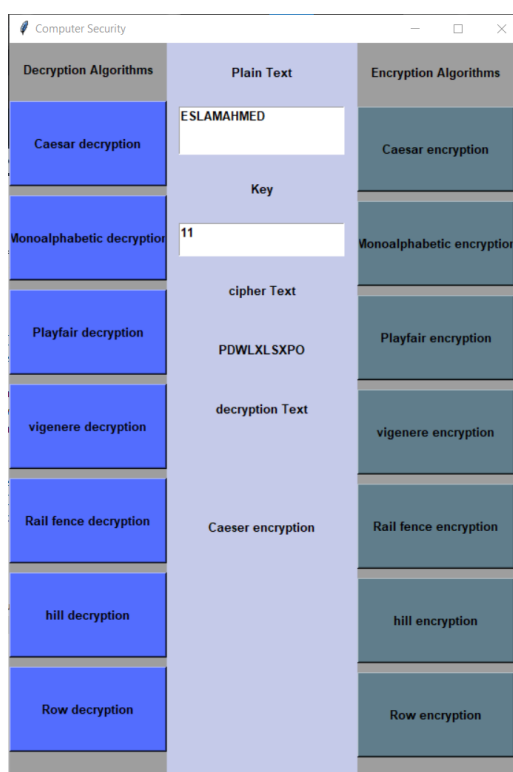


Figure 24: Caesar Encryption.

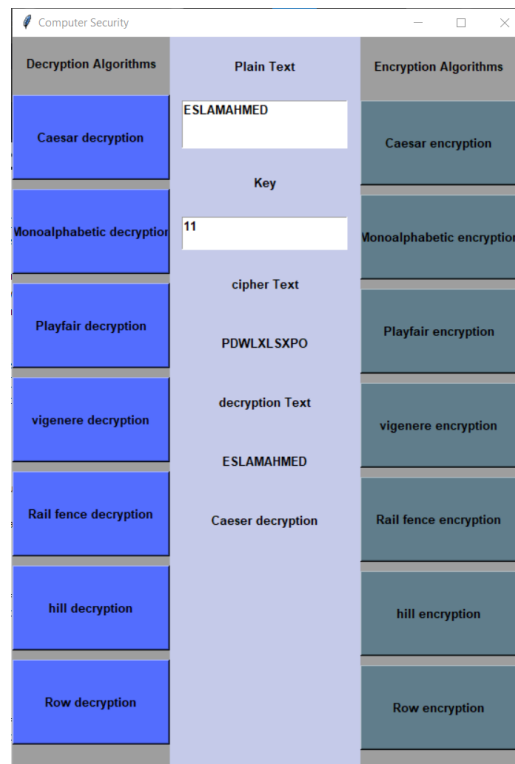


Figure 25: Caesar Dencryption.

Monoalphabetic Encryption and Decryption

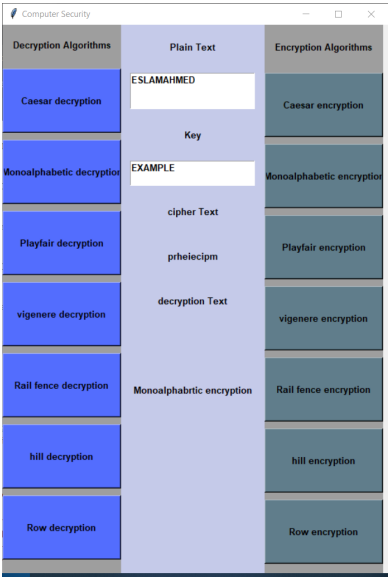


Figure 26: Monoalphabetic Encryption.

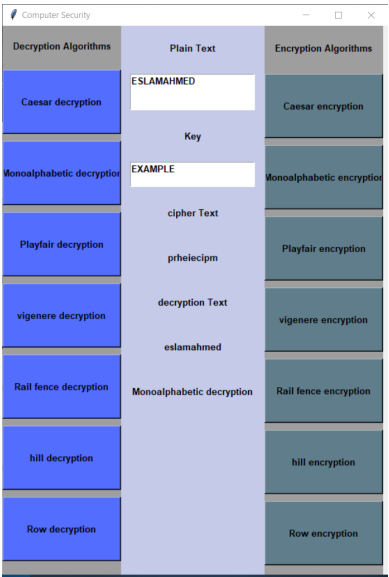


Figure 27: Monoalphabetic Decryption.

Playfair Encryption and Decryption

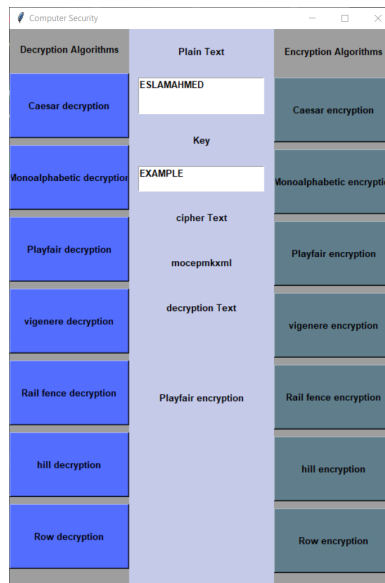


Figure 28: Playfair Encryption.

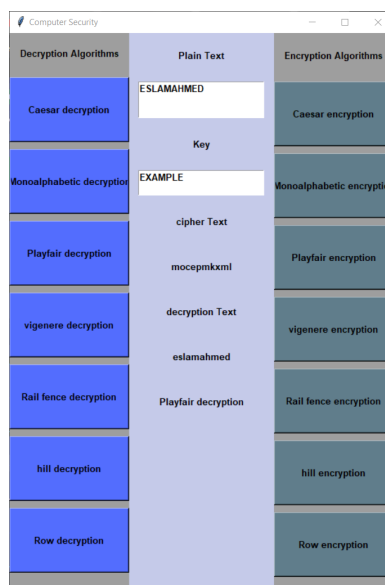


Figure 29: Playfair Decryption.

Vigenere Encryption and Decryption

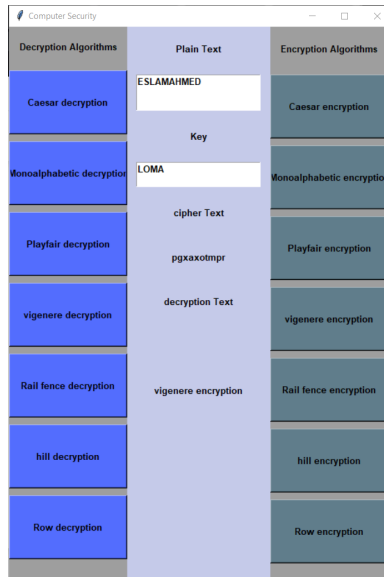


Figure 30: Vigenere Encryption.

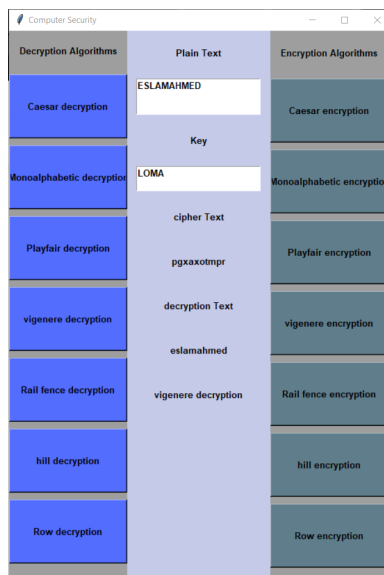


Figure 31: Vigenere Decryption.

Hill Encryption

Decryption Algorithms	Plain Text	Encryption Algorithms
Caesar decryption	ESLAMAHMED	Caesar encryption
	Key	
Monoalphabetic decryption	1234	Monoalphabetic encryption
	cipher Text	
Playfair decryption	oglhmkfrky	Playfair encryption
	decryption Text	
vigenere decryption		vigenere encryption
	hill encryption	
Rail fence decryption		Rail fence encryption
hill decryption		hill encryption
Row decryption		Row encryption

Figure 32: Hill Encryption.

Rail fence Encryption and Decryption

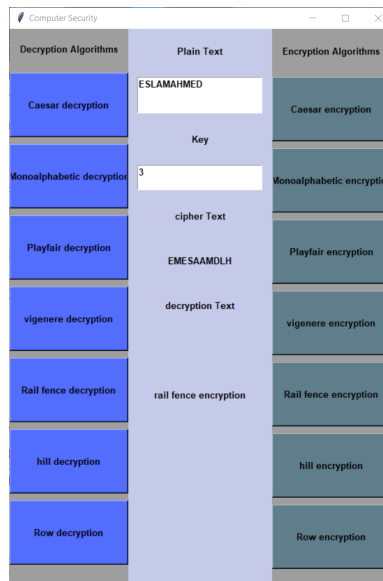


Figure 33: Rail fence Encryption.

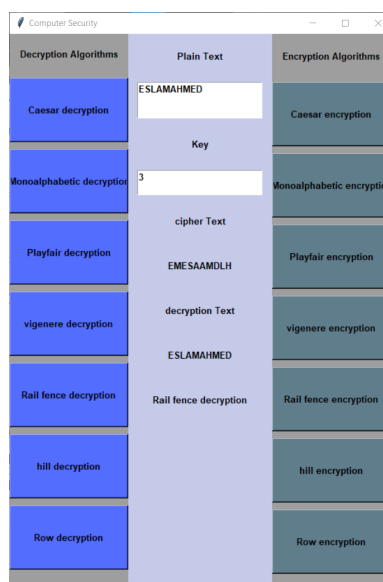


Figure 34: Rail fence Decryption.

Row transposition Encryption and Decryption

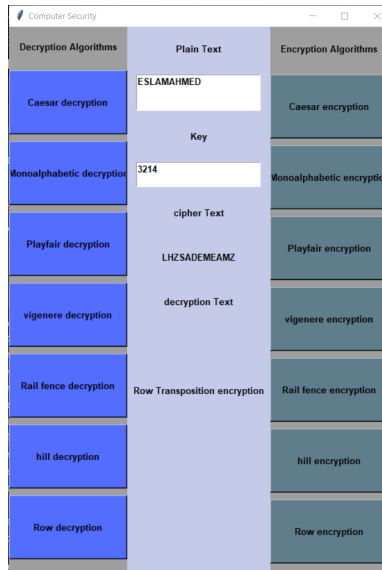


Figure 35: Row transposition Encryption.

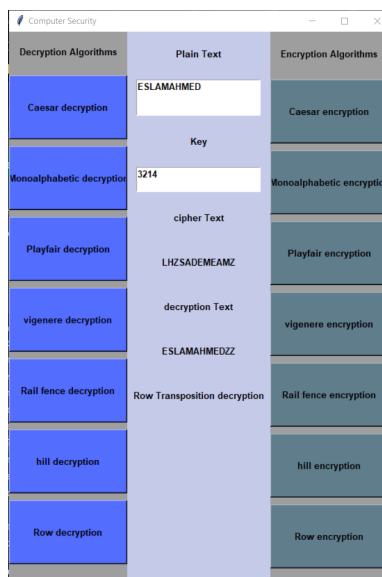


Figure 36: Row transposition Decryption.