1. **Source Code:**

```
def caesar_encrypt(plaintext, shift):
    encrypted_text = ""
    for char in plaintext:
        if char.isalpha():
            if char.islower():
                encrypted_text += chr((ord(char) + shift - ord('a')) % 26 + ord('a'))
            else:
                encrypted_text += chr((ord(char) + shift - ord('A')) % 26 + ord('A'))
        else:
            encrypted_text += char
    return encrypted_text

def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            if char.islower():
                decrypted_text += chr((ord(char) - shift - ord('a')) % 26 + ord('a'))
            else:
                decrypted_text += chr((ord(char) - shift - ord('A')) % 26 + ord('A'))
        else:
            decrypted_text += char
    return decrypted_text

plaintext = str(input("enter plain text:"));
shift = int(input('enter key: '));
ciphertext = caesar_encrypt(plaintext, shift)
print(f"Caesar Cipher Encryption: {ciphertext}")
plaintext=caesar_decrypt(ciphertext,3)
print(f"Caesar Cipher Decryption: {plaintext}")
```

Input:
enter plain text:welcome
enter key: 3
Output:
Caesar Cipher Encryption: zhofrph
Caesar Cipher Decryption: welcome

2. Source Code:

```
import  random
plain_text = []
key =[]
for i in range(65, 65+26):
    plain_text.append(chr(i))
    key.append(chr(i))
message = input("Enter message: ")
random.shuffle(key)
print("Plain Text: ",plain_text)
print("Key:        ",key)
cipher = ''
#encryption
for ch in message:
    try:
        index = plain_text.index(ch.upper())
        cipher = cipher + key[index]
    except:
        cipher = cipher + ch
print("Cipher: ", cipher)
decrypted_mess = ''
for ch in cipher:
    try:
        index = key.index(ch.upper())
        decrypted_mess = decrypted_mess + plain_text[index]
    except:
        decrypted_mess = decrypted_mess + ch
print("Decrypted Message: ", decrypted_mess)
```

Input:

Enter message: sakib

Output:

Plain Text: ['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']

Key: ['I', 'H', 'C', 'R', 'Q', 'F', 'S', 'Y', 'L', 'U', 'X', 'E', 'P', 'Z', 'O', 'A', 'V', 'B', 'W', 'D', 'T', 'G', 'N', 'M', 'J', 'K']

Cipher: WIXLH

Decrypted Message: SAKIB

3. Source Code:

```
def playfair_cipher(plaintext, key, mode):
    # Define the alphabet, excluding 'j'
    alphabet = 'abcdefghiklmnopqrstuvwxyz'
    # Remove whitespace and 'j' from the key and convert to lowercase
    key = key.lower().replace(' ', '').replace('j', 'i')
    # Construct the key square
    key_square = ''
    for letter in key + alphabet:
        if letter not in key_square:
            key_square += letter
    # Split the plaintext into digraphs, padding with 'x' if necessary
    plaintext = plaintext.lower().replace(' ', '').replace('j', 'i')
    if len(plaintext) % 2 == 1:
        plaintext += 'x'
    digraphs = [plaintext[i:i+2] for i in range(0, len(plaintext), 2)]
    # Define the encryption/decryption functions
    def encrypt(digraph):
        a, b = digraph
        row_a, col_a = divmod(key_square.index(a), 5)
        row_b, col_b = divmod(key_square.index(b), 5)
        if row_a == row_b:
            col_a = (col_a + 1) % 5
            col_b = (col_b + 1) % 5
        elif col_a == col_b:
            row_a = (row_a + 1) % 5
            row_b = (row_b + 1) % 5
        else:
            col_a, col_b = col_b, col_a
        return key_square[row_a*5+col_a] + key_square[row_b*5+col_b]

    def decrypt(digraph):
        a, b = digraph
        row_a, col_a = divmod(key_square.index(a), 5)
        row_b, col_b = divmod(key_square.index(b), 5)
        if row_a == row_b:
            col_a = (col_a - 1) % 5
            col_b = (col_b - 1) % 5
        elif col_a == col_b:
            row_a = (row_a - 1) % 5
            row_b = (row_b - 1) % 5
        else:
            col_a, col_b = col_b, col_a
        return key_square[row_a*5+col_a] + key_square[row_b*5+col_b]
    # Encrypt or decrypt the plaintext
    result = ''
    for digraph in digraphs:
        if mode == 'encrypt':
            result += encrypt(digraph)
        elif mode == 'decrypt':
            result += decrypt(digraph)
```

```
    # Return the result
    return result

# Example usage
plaintext = str(input("enter plain text: "));
key = str(input("enter Key: "));
ciphertext = playfair_cipher(plaintext, key, 'encrypt')
print(ciphertext)  # outputs: "iisggymlgmsyjqu"
decrypted_text = playfair_cipher(ciphertext, key, 'decrypt')
print(decrypted_text)  # (Note: 'x' is added as padding)
```

Input:
enter plain text: mosque
enter Key: monarchy

Output
ontsml
mosque

4. **Source Code**:

```
import numpy as np
from egcd import egcd
from egcd import egcd  # pip install egcd

alphabet = "abcdefghijklmnopqrstuvwxyz"

letter_to_index = dict(zip(alphabet, range(len(alphabet))))
index_to_letter = dict(zip(range(len(alphabet)), alphabet))


def matrix_mod_inv(matrix, modulus):

    det = int(np.round(np.linalg.det(matrix)))  # Step 1)
    det_inv = egcd(det, modulus)[1] % modulus  # Step 2)
    matrix_modulus_inv = (
            det_inv * np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
    )  # Step 3)

    return matrix_modulus_inv


def encrypt(message, K):
    encrypted = ""
    message_in_numbers = []

    for letter in message:
        message_in_numbers.append(letter_to_index[letter])

    split_P = [
        message_in_numbers[i: i + int(K.shape[0])]
        for i in range(0, len(message_in_numbers), int(K.shape[0]))
    ]

    for P in split_P:
        P = np.transpose(np.asarray(P))[:, np.newaxis]

        while P.shape[0] != K.shape[0]:
            P = np.append(P, letter_to_index[" "])[:, np.newaxis]

        numbers = np.dot(K, P) % len(alphabet)
        n = numbers.shape[0]  # length of encrypted message (in numbers)

        # Map back to get encrypted text
        for idx in range(n):
            number = int(numbers[idx, 0])
            encrypted += index_to_letter[number]

    return encrypted
```

```
def decrypt(cipher, Kinv):
    decrypted = ""
    cipher_in_numbers = []

    for letter in cipher:
        cipher_in_numbers.append(letter_to_index[letter])

    split_C = [
        cipher_in_numbers[i: i + int(Kinv.shape[0])]
        for i in range(0, len(cipher_in_numbers), int(Kinv.shape[0]))
    ]

    for C in split_C:
        C = np.transpose(np.asarray(C))[:, np.newaxis]
        numbers = np.dot(Kinv, C) % len(alphabet)
        n = numbers.shape[0]

        for idx in range(n):
            number = int(numbers[idx, 0])
            decrypted += index_to_letter[number]

    return decrypted

message = str(input('Enter plain text: '))
message = message.lower()
K = np.matrix([[7, 8], [11, 11]])
# K = np.matrix([[6, 24, 1], [13,16,10], [20,17,15]]) # for length of alphabet = 26
# K = np.matrix([[3,10,20],[20,19,17], [23,78,17]]) # for length of alphabet = 27
Kinv = matrix_mod_inv(K, len(alphabet))

encrypted_message = encrypt(message, K)
decrypted_message = decrypt(encrypted_message, Kinv)

print("Original message: " + message)
print("Encrypted message: " + encrypted_message)
print("Decrypted message: " + decrypted_message)

alphabet='abcdefghijklmnopqrstuvwxyz'
letterToIndex=dict(zip(alphabet))
```

Output:

```
Enter plain text: code
Original message: code
Encrypted message: wubz
Decrypted message: code
```

5. Source code:

```
# Poly_alphabetic cipher

alphabet = "abcdefghijklmnopqrstuvwxyz".upper()
mp = dict(zip(alphabet, range(len(alphabet))))
mp2 = dict(zip(range(len(alphabet)), alphabet))


def generateKey(plainText, keyword):
    key = ''
    for i in range(len(plainText)):
        key += keyword[i % len(keyword)]
    return key


def cipherText(plainText, key):
    cipher_text = ""
    for i in range(len(plainText)):
        shift = mp[key[i].upper()] - mp['A']
        newChar = mp2[(mp[plainText[i].upper()] + shift) % 26]
        cipher_text += newChar
    return cipher_text


def decrypt(cipher_text, key):
    plainText = ''
    for i in range(len(cipher_text)):
        shift = mp[key[i].upper()] - mp['A']
        newChar = mp2[(mp[cipher_text[i].upper()] - shift + 26) % 26]
        plainText += newChar
    return plainText


plainText =str(input("input plain text:" ));
keyword = "deceptive"
key = generateKey(plainText, keyword)
cipher_text = cipherText(plainText, key)
print("Ciphertext :", cipher_text)
print("Decrypted Text :", decrypt(cipher_text, key))
```

Output:

```
input plain text:wearediscover
Ciphertext : ZICVTWQNGRZGV
Decrypted Text : WEAREDISCOVER
```

6. Source Code:

```
import random


def generate_key(length):
    key = ""
    for i in range(length):
        key += chr(random.randint(65, 90))  # ASCII codes for A-Z
    return key


def encrypt(plaintext, key):
    ciphertext = ""
    cipherCode = []
    for i in range(len(plaintext)):
        x = ord(plaintext[i]) ^ ord(key[i])
        cipherCode.append(x)
        ciphertext += chr(x % 26 + 65)
    return ciphertext, cipherCode


def decrypt(cipherCode, key):
    plaintext = ""
    for i in range(len(cipherCode)):
        x = (cipherCode[i] ^ ord(key[i]))
        plaintext += chr(x)
    return plaintext


plaintext = str(input('Enter plain text: '))
key = generate_key(len(plaintext))  # Generate a random key

ciphertext, cipherCode = encrypt(plaintext, key)
print("Ciphertext:", ciphertext)
decryptedtext = decrypt(cipherCode, key)
print("Decrypted text:", decryptedtext)
```

Output:

```
Enter plain text: son
Ciphertext: FIN
Decrypted text: son
```

7. Source code:

```
def brute_force_decrypt(ciphertext):
    for shift in range(26):
        decrypted_text = caesar_decrypt(ciphertext, shift)
        print(f"Shift {shift}: {decrypted_text}")

def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            if char.islower():
                decrypted_text += chr((ord(char) - shift - ord('a')) % 26 + ord('a'))
            else:
                decrypted_text += chr((ord(char) - shift - ord('A')) % 26 + ord('A'))
        else:
            decrypted_text += char
    return decrypted_text

ciphertext = str(input("enter cipher text :"))
print("Brute Force Decryption for Caesar Cipher:", ciphertext)
brute_force_decrypt(ciphertext)
```

Output:

```
enter cipher text :zhofrph
Brute Force Decryption for Caesar Cipher: zhofrph
Shift 0: zhofrph
Shift 1: ygneqog
Shift 2: xfmdpnf
Shift 3: welcome
Shift 4: vdkbnld
Shift 5: ucjamkc
Shift 6: tbizljb
Shift 7: sahykia
Shift 8: rzgxjhz
Shift 9: qyfwigy
Shift 10: pxevhfx
Shift 11: owdugew
Shift 12: nvctfdv
Shift 13: mubsecu
Shift 14: ltardbt
Shift 15: kszqcas
Shift 16: jrypbzr
Shift 17: iqxoayq
Shift 18: hpwnzxp
Shift 19: govmywo
Shift 20: fnulxvn
Shift 21: emtkwum
Shift 22: dlsjvtl
Shift 23: ckriusk
Shift 24: bjqhtrj
Shift 25: aipgsqi
```