

# **Cryptography and Network Security Lab (5CS453)**

**Name: Vishal Indradev Chauhan**

**PRN: 2020BTECS00090**

**Class: Final Year - CSE**

## **Assignment No 1**

### **Title:**

**Encryption and Decryption using Ceaser Cipher.**

### **Aim:**

**To Study and Implement Encryption and Decryption using Ceaser Cipher**

### **Theory:**

- Caesar Cipher, also known as the Shift Cipher, is one of the simplest and oldest encryption techniques used to secure information.
- It's a type of substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet.
- The number of positions a letter is shifted is determined by a key.

### **Encryption:**

**In Encryption, input is a Plain text and output is a Cipher text.**

Step 1: Choose a secret key (a positive integer).

Step 2: Take the plaintext message you want to encrypt.

Step 3: Shift each letter in the message forward in the alphabet by the key positions.

Step 4: Non-alphabetical characters remain unchanged.

Step 5: The result is the ciphertext, the encrypted message.

### **Decryption:**

**In Decryption, input is a Cipher text and output is a Plain text.**

Step 1: Have the same key used for encryption.

Step 2: Take the ciphertext (the encrypted message).

Step 3: Shift each letter in the ciphertext backward in the alphabet by the key positions.

Step 4: Non-alphabetical characters remain unchanged.

Step 5: The result is the plaintext, the original message.

## Code:

```
Assign1 > both.py
1 import string
2
3 # Function to check if a word is meaningful
4 def is_valid_word(word, dictionary):
5     return word in dictionary
6
7 def caesar_cipher_encrypt(plaintext, key):
8     ciphertext = ""
9     for c in plaintext:
10         if c.isalpha():
11             base = ord('A') if c.isupper() else ord('a')
12             encrypted_char = chr(((ord(c) - base + key) % 26) + base)
13             ciphertext += encrypted_char
14         else:
15             ciphertext += c # Preserve non-alphabetic characters as is
16     return ciphertext
17
18 def caesar_cipher_decrypt(ciphertext, key):
19     plaintext = ""
20     for c in ciphertext:
21         if c.isalpha():
22             base = ord('A') if c.isupper() else ord('a')
23             decrypted_char = chr(((ord(c) - base - key + 26) % 26) + base)
24             plaintext += decrypted_char
25         else:
26             plaintext += c # Preserve non-alphabetic characters as is
27     return plaintext
28
29 def print_all_combinations(ciphertext):
30     print("All possible plaintext combinations:")
31     for key in range(26):
32         plaintext = caesar_cipher_decrypt(ciphertext, key)
33         print(f"Key {key}: {plaintext}")
```

```
Assign1 > both.py > {} string
34
35 if __name__ == "__main__":
36     valid_words = ["hello", "world", "example", "brijesh", "is", "a", "good", "boy", "cipher", "decrypt", "meaningful", "pl
37
38     while True:
39         print("Menu:")
40         print("1. Encrypt")
41         print("2. Decrypt and Filter Meaningful Words")
42         print("3. Print All Possible Plaintext Combinations")
43         print("4. Exit")
44         choice = input("Enter your choice (1/2/3/4): ")
45
46         if choice == "1":
47             input_text = input("Enter the plaintext: ")
48             key = int(input("Enter the key (a positive integer): "))
49             ciphertext = caesar_cipher_encrypt(input_text, key)
50             print(f"Ciphertext: {ciphertext}")
51
52         elif choice == "2":
53             ciphertext = input("Enter the ciphertext: ")
54             key = int(input("Enter the key (a positive integer): "))
55             decrypted_text = caesar_cipher_decrypt(ciphertext, key)
56             print(f"Decrypted Text: {decrypted_text}")
57
58             meaningful_text = " ".join(word for word in decrypted_text.split() if is_valid_word(word, valid_words))
59             print(f"Meaningful Text: {meaningful_text}")
60
61         elif choice == "3":
62             ciphertext = input("Enter the ciphertext: ")
63             print_all_combinations(ciphertext)
64
65         elif choice == "4":
66             print("Exiting the program.")
67             break
```

## Output:

```
Menu:
1. Encrypt Plain Text
2. Decrypt Cipher Text
3. Show All Combinations
4. Show Meaningful Decryptions
5. Exit
Enter your choice: 1
Enter plain text: brijesh
Enter key: 3
Encrypted Text: eulmhvk

Menu:
1. Encrypt Plain Text
2. Decrypt Cipher Text
3. Show All Combinations
4. Show Meaningful Decryptions
5. Exit
Enter your choice: 2
Enter cipher text: eulmhvk
Enter key: 3
Decrypted Text: brijesh

Menu:
1. Encrypt Plain Text
2. Decrypt Cipher Text
3. Show All Combinations
4. Show Meaningful Decryptions
5. Exit
Enter your choice: 5
Exiting...
```

## Assignment No 2

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan** - **PRN: 2020BTECS00090** - **Class: Final Year - CSE**

#### **Title:**

**Encryption and Decryption using Transposition Cipher Technique.**

#### **Aim:**

**To Study and Implement Encryption and Decryption using Rail Fence Transposition Cipher Technique and Columnar Transposition Cipher Technique**

#### **Theory:**

##### **1. Rail Fence Transposition Cipher Technique**

□

The Rail Fence Transposition Cipher, also known as the Zigzag Cipher, is a simple columnar transposition cipher technique.

It involves arranging the plaintext characters in a zigzag pattern across multiple rows, known as "rails," and then reading them off row by row to create the encrypted message.

While this cipher is easy to understand and implement, it lacks strong security and is mainly used for educational purposes or simple puzzles.

#### **Encryption:**

- Choose the number of rails (rows) for the zigzag pattern.
- Write the message diagonally across the rails, moving up and down.
- Read the characters row by row to form the encrypted message.

#### **Decryption:**

- Create the zigzag pattern with the chosen number of rails.
- Leave blank spaces in the pattern for characters to be placed.
- Fill in the blanks with the encrypted characters, row by row.
- Read the characters diagonally to retrieve the original message.

#### **Advantages:**


- Easy to understand and implement.
- Provides basic encryption and breaks up character repetition.

#### **Disadvantages:**

- Not secure against modern cryptanalysis.

- Security depends on the number of rails, making it less practical for strong encryption.

## CODE:

Assign2 >  reilFence\_P.py > ...

```
3 class Solution:
4     def convert(self, s, n):
5         if n == 1:
6             return s
7
8         result = []
9         length = len(s)
10        k = 2 * (n - 1)
11
12        for i in range(n):
13            j = i
14            while j < length:
15                result.append(s[j])
16                if i != 0 and i != n - 1:
17                    k1 = k - (2 * i)
18                    k2 = j + k1
19                    if k2 < length:
20                        result.append(s[k2])
21                j += k
22        return ''.join(result)
23
24    def reverse_convert(self, s, n):
25        if n == 1:
26            return s
27
28        original_string = [''] * len(s)
29        length = len(s)
30        k = 2 * (n - 1)
31        idx = 0
32
33        for i in range(n):
34            j = i
35            while j < length:
36                original_string[j] = s[idx]
```

```

Assign2 > reilFence_P.py > ...
34         j = 1
35         while j < length:
36             original_string[j] = s[idx]
37             idx += 1
38             if i != 0 and i != n - 1:
39                 k1 = k - (2 * i)
40                 k2 = j + k1
41                 if k2 < length:
42                     original_string[k2] = s[idx]
43                     idx += 1
44             j += k
45
46         return ''.join(original_string)
47
48
49     # Taking input from the user
50     input_str = input("Enter the input string: ")
51     input_n = int(input("Enter the value of n: "))
52
53     solution = Solution()
54     converted_string = solution.convert(input_str, input_n)
55     print("Converted string:", converted_string)
56
57     reverse_input_n = int(input("Enter the value of n for reverse conversion: "))
58     reverse_plain_text = solution.reverse_convert(converted_string, reverse_input_n)
59     print("Reverse converted plain text:", reverse_plain_text)

```

## OUTPUT:

```

PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\reilFence_P.py"
Enter the input string: hey .. have a great day!
Enter the value of n: 3
Converted string: h.aae de .hv ra aly egty
Enter the value of n for reverse conversion: 3
Reverse converted plain text: hey .. have a great day!
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\reilFence_P.py"
Enter the input string: helloo.. yeahh, wish you the same!
Enter the value of n: 3
Converted string: ho hw eelo.yah ihyutesm!l.e,soha
Enter the value of n for reverse conversion: 3
Reverse converted plain text: helloo.. yeahh, wish you the same!
PS C:\Users\ANUSHKA\Documents\BTech\CNS>

```



## 2. Columnar Transposition Cipher Technique

□

The Columnar Transposition Cipher is a more advanced transposition cipher technique that involves reordering the characters of a message based on a chosen keyword or key phrase.

It provides a higher level of security compared to simpler ciphers like the Rail Fence Cipher. Here's how the Columnar Transposition Cipher works:

### Encryption:

- Choose a keyword or key phrase. The unique characters of the keyword determine the order of columns in the transposition grid.
- Write the message row by row into a grid, using the keyword to determine the order of columns.
- Read the characters column by column to obtain the encrypted message.

### Decryption:

- Use the keyword to determine the order of columns in the transposition grid.
- Write the encrypted message into the grid column by column.
- Read the characters row by row to retrieve the original plaintext.

### Advantages:

- Offers stronger security compared to simpler ciphers.
- Security depends on the length and uniqueness of the keyword.

### Disadvantages:

- Can be vulnerable to attacks if the keyword is short or easily guessed.
- May require additional padding characters for messages that don't fit evenly into the grid.

## CODE:

```
Assign2 > columnar.py > encryptMessage
1  import math
2
3  # Encryption
4  def encryptMessage(msg,key):
5      cipher = ""
6
7      k_indx = 0
8
9      msg_len = float(len(msg))
10     msg_lst = list(msg)
11     key_lst = sorted(list(key))
12
13     # calculate column of the matrix
14     col = len(key)
15
16     # calculate maximum row of the matrix
17     row = int(math.ceil(msg_len / col))
18
19     # add the padding character '_' in empty
20     # the empty cell of the matrix
21     fill_null = int((row * col) - msg_len)
22     msg_lst.extend('_' * fill_null)
23
24     # create Matrix and insert message and
25     # padding characters row-wise
26     matrix = [msg_lst[i: i + col]
27               for i in range(0, len(msg_lst), col)]
28
29     # read matrix column-wise using key
30     for _ in range(col):
31         curr_idx = key.index(key_lst[k_indx])
32         cipher += ''.join([row[curr_idx]
33                           for row in matrix])
34         k_indx += 1
```

Assign2 > columnar.py > encryptMessage

```
35
36     return cipher
37
38 # Decryption
39 def decryptMessage(cipher,key):
40     msg = ""
41
42     # track key indices
43     k_indx = 0
44
45     # track msg indices
46     msg_indx = 0
47     msg_len = float(len(cipher))
48     msg_lst = list(cipher)
49
50     # calculate column of the matrix
51     col = len(key)
52
53     # calculate maximum row of the matrix
54     row = int(math.ceil(msg_len / col))
55
56     # convert key into list and sort
57     # alphabetically so we can access
58     # each character by its alphabetical position.
59     key_lst = sorted(list(key))
60
61     # create an empty matrix to
62     # store deciphered message
63     dec_cipher = []
64     for _ in range(row):
65         dec_cipher += [[None] * col]
```

```

Assign2 > columnar.py > decryptMessage
67     for _ in range(col):
68         curr_idx = key.index(key_lst[k_idx])
69
70         for j in range(row):
71             dec_cipher[j][curr_idx] = msg_lst[msg_idx]
72             msg_idx += 1
73             k_idx += 1
74
75     # convert decrypted msg matrix into a string
76     try:
77         msg = ''.join(sum(dec_cipher, []))
78     except TypeError:
79         raise TypeError("This program cannot",
80                          "handle repeating words.")
81
82     null_count = msg.count('_')
83
84     if null_count > 0:
85         return msg[: -null_count]
86     return msg
87
88
89 def main():
90     msg = input("Enter the message to encrypt: ")
91     key = input("Enter the encryption key (permutation of numbers, e.g., '3124'): ")
92
93     cipher = encryptMessage(msg, key)
94     print("Encrypted message:", cipher)
95
96     decrypted_message = decryptMessage(cipher, key)
97     print("Decrypted message:", decrypted_message)
98
99 if __name__ == "__main__":
100     main()

```

## OUTPUT:

```

PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\columnar.py"
Enter the message to encrypt: hey there.. how was your day?
Enter the encryption key (permutation of numbers, e.g., '3124'): 4213
Encrypted message: yh.oaad_etehtwy ?ye.wsua_h r ry
Decrypted message: hey there.. how was your day?
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\columnar.py"
Enter the message to encrypt: yeahh.. it was a great day:)
Enter the encryption key (permutation of numbers, e.g., '3124'): 4123
Encrypted message: e.tshtya. r :h waed)yhia aa
Decrypted message: yeahh.. it was a great day:)
PS C:\Users\ANUSHKA\Documents\BTech\CNS>

```

## Assignment No 3

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan** - **PRN: 2020BTECS00090** **Class: Final Year - CSE**

#### **Title:**

**Encryption and Decryption using Playfair Cipher Technique.**

#### **Aim:**

**To Study and Implement Encryption and Decryption using Playfair Cipher Technique.**

#### **Theory:**

##### **Playfair Cipher Technique**

□

The Playfair Cipher Technique is a substitution cipher that encrypts pairs of characters (digraphs) from the plaintext using a 5x5 key square matrix.

The matrix is constructed from a keyword, with duplicate letters removed and the keyword letters placed at the beginning.

Encryption involves applying rules based on the positions of the letters within the key square.

If the letters are in the same row, column, or form a rectangle, they are replaced by specific neighboring letters.

#### **Encryption:**

- Divide the plaintext into digraphs.
- Apply rules based on the positions of letters in the key square to replace each digraph.

#### **Decryption:**

- Divide the ciphertext into digraphs.
- Apply the rules in reverse to each digraph to retrieve the original plaintext.

#### **Advantages:**

- Enhanced security due to digraphs and key square usage.
- Reduces susceptibility to frequency analysis.
- Key square generation is straightforward using a keyword.

#### **Disadvantages:**

- Complexity increases with handling various cases (same row, column, rectangle).
- Security depends on the keyword and arrangement of the key square.

## CODE:

```
Assign3 > Playfair.py > Diagraph
1  def toLowerCase(text):
2      return text.lower()
3
4  def removeSpaces(text):
5      newText = ""
6      for i in text:
7          if i == " ":
8              continue
9          else:
10             newText = newText + i
11     return newText
12
13 def Diagraph(text):
14     Diagraph = []
15     group = 0
16     for i in range(2, len(text), 2):
17         Diagraph.append([text[group:i]])
18         group = i
19     Diagraph.append(text[group:])
20     return Diagraph
21
22 def FillerLetter(text):
23     k = len(text)
24     if k % 2 == 0:
25         for i in range(0, k, 2):
26             if text[i] == text[i+1]:
27                 new_word = text[0:i+1] + str('x') + text[i+1:]
28                 new_word = FillerLetter(new_word)
29                 break
30         else:
31             new_word = text
```

Assign3 > Playfair.py > Diagram

```
32     else:
33         for i in range(0, k-1, 2):
34             if text[i] == text[i+1]:
35                 new_word = text[0:i+1] + str('x') + text[i+1:]
36                 new_word = FillerLetter(new_word)
37                 break
38             else:
39                 new_word = text
40     return new_word
41
42
43 list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',
44         'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
45
46 # Function to generate the 5x5 key square matrix
47
48
49 def generateKeyTable(word, list1):
50     key_letters = []
51     for i in word:
52         if i not in key_letters:
53             key_letters.append(i)
54
55     compElements = []
56     for i in key_letters:
57         if i not in compElements:
58             compElements.append(i)
59     for i in list1:
60         if i not in compElements:
61             compElements.append(i)
```

Assign3 > Playfair.py > ...

```
62
63     matrix = []
64     while compElements != []:
65         matrix.append(compElements[:5])
66         compElements = compElements[5:]
67
68     return matrix
69
70 def printMatrix(matrix):
71     for row in matrix:
72         print(" ".join(row))
73     print()
74
75 def search(mat, element):
76     for i in range(5):
77         for j in range(5):
78             if(mat[i][j] == element):
79                 return i, j
80
81 def encrypt_RowRule(matr, e1r, e1c, e2r, e2c):
82     char1 = ''
83     if e1c == 4:
84         char1 = matr[e1r][0]
85     else:
86         char1 = matr[e1r][e1c+1]
87
88     char2 = ''
89     if e2c == 4:
90         char2 = matr[e2r][0]
91     else:
92         char2 = matr[e2r][e2c+1]
93
94     return char1, char2
```



Assign3 > Playfair.py > encrypt\_RectangleRule

```
96
97 def encrypt_ColumnRule(matr, e1r, e1c, e2r, e2c):
98     char1 = ''
99     if e1r == 4:
100         char1 = matr[0][e1c]
101     else:
102         char1 = matr[e1r+1][e1c]
103
104     char2 = ''
105     if e2r == 4:
106         char2 = matr[0][e2c]
107     else:
108         char2 = matr[e2r+1][e2c]
109
110     return char1, char2
111
112 def encrypt_RectangleRule(matr, e1r, e1c, e2r, e2c):
113     char1 = ''
114     char1 = matr[e1r][e2c]
115
116     char2 = ''
117     char2 = matr[e2r][e1c]
118
119     return char1, char2
120
121 def encryptByPlayfairCipher(Matrix, plainList):
122     CipherText = []
123     for i in range(0, len(plainList)):
124         c1 = 0
125         c2 = 0
126         ele1_x, ele1_y = search(Matrix, plainList[i][0])
127         ele2_x, ele2_y = search(Matrix, plainList[i][1])
```

```

Assign3 > Playfair.py > main
129     if ele1_x == ele2_x:
130         c1, c2 = encrypt_RowRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
131         # Get 2 letter cipherText
132     elif ele1_y == ele2_y:
133         c1, c2 = encrypt_ColumnRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
134     else:
135         c1, c2 = encrypt_RectangleRule(
136             Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
137
138     cipher = c1 + c2
139     CipherText.append(cipher)
140     return CipherText
141
142 def main():
143     text_Plain = input("Enter the message to encrypt: ")
144     text_Plain = removeSpaces(toLowerCase(text_Plain))
145     PlainTextList = Diagraph(FillerLetter(text_Plain))
146     if len(PlainTextList[-1]) != 2:
147         PlainTextList[-1] = PlainTextList[-1]+'z'
148
149     key = input("Enter the encryption key: ")
150     key = toLowerCase(key)
151     Matrix = generateKeyTable(key, list1)
152     print("\nMatrix:")
153     printMatrix(Matrix)
154     print("Plain Text:", text_Plain)
155     CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)
156     CipherText = ""
157     for i in CipherList:
158         CipherText += i
159     print("CipherText:", CipherText)
160
161 if __name__ == "__main__":
162     main()

```

## OUTPUT:

```

PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Playfair.py"
Enter the message to encrypt: My name is Anushka
Enter the encryption key: anushit

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

Plain Text: mynameisanushka
CipherText: qvunvmcanushslhv
PS C:\Users\ANUSHKA\Documents\BTech\CNS> 

```



## Assignment No 4

### Cryptography and Network Security Lab (5CS453)

Name: Vishal Chauhan - PRN: 2020BTECS00090 Class: Final Year - CSE

#### Problem Statement:

##### Vigenère Cipher Technique

□

The Vigenère Cipher Technique is a polyalphabetic substitution cipher that adds an extra layer of complexity to encryption by using a keyword or key phrase to determine the shifts applied to the plaintext letters.

Unlike monoalphabetic ciphers, where each letter is replaced with a fixed substitution, the Vigenère Cipher employs multiple alphabets with different shifts, making it more secure against frequency analysis.

#### Key Setup:

- Choose a keyword or key phrase.
- Replicate the keyword to match the length of the plaintext, repeating it as needed.
- Convert the keyword letters to their corresponding numerical values (A=0, B=1, ..., Z=25).

#### Encryption:

- Divide the plaintext into individual letters and convert them to numerical values.
- For each letter, determine the shift value using the corresponding keyword letter.
- Shift the plaintext letter by the calculated shift value (mod 26).
- Convert the shifted numerical value back to a letter to create the ciphertext.

#### Decryption:

- Divide the ciphertext into individual letters and convert them to numerical values.
- For each letter, determine the shift value using the corresponding keyword letter.
- Reverse the shift (subtract the shift value, mod 26).
- Convert the shifted numerical value back to a letter to retrieve the original plaintext.

#### Advantages:

- Stronger security due to polyalphabetic nature and keyword-driven shifts.
- Reduces susceptibility to frequency analysis.
- Key space increases with keyword length, enhancing security.

#### Disadvantages:

- Vulnerable to key length repetition (Kasiski examination) for shorter keywords.

- Security can weaken if the keyword is short or predictable.

## CODE:

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

def cipherText(string, key):
    cipher_text = []
    for i in range(len(string)):
        x = (ord(string[i]) + ord(key[i])) % 26
        x += ord('A')
        cipher_text.append(chr(x))
    return("".join(cipher_text))

def originalText(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("".join(orig_text))

def main():
    string = input("Enter the plaintext: ").upper()
    keyword = input("Enter the keyword: ").upper()
    key = generateKey(string, keyword)
    cipher_text = cipherText(string, key)
    print("Ciphertext:", cipher_text)
    print("Original/Decrypted Text:",
          originalText(cipher_text, key))

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

## OUTPUT:

```
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\vigenereCipher.py"
Enter the plaintext: anushka
Enter the keyword: jadhav
Ciphertext: JNXZHFJ
Original/Decrypted Text: ANUSHKA
PS C:\Users\ANUSHKA\Documents\BTech\CNS> 
```

## Assignment No 5

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan**

**PRN: 2020BTECS00090**

**Class: Final Year - CSE**

#### Problem Statement:

##### Data Encryption Standard (DES)

□

- **Data Encryption Standard (DES)** is one of the earliest and most widely used encryption standards for securing digital data.
- Developed in the early 1970s by IBM, it was adopted as a federal standard in the United States for the protection of sensitive data.
- It's a symmetric-key block cipher, meaning the same key is used for both encryption and decryption, and it operates on fixed-size data blocks.
- DES uses a 56-bit encryption key (originally 64 bits with 8 bits used for parity). The key length is relatively short by modern standards.
- DES uses a Feistel network structure, dividing data into halves and applying multiple rounds of operations with the encryption key.
- However, due to its vulnerability to brute-force attacks and advances in computing power, DES is now considered obsolete and has been largely replaced by more secure encryption algorithms like AES (Advanced Encryption Standard).

#### Code:

```
from Crypto.Cipher import DES
from secrets import token_bytes

key = token_bytes(8)

def encrypt(msg):
    cipher = DES.new(key, DES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(msg.encode('ascii'))
    return nonce, ciphertext, tag

def decrypt(nonce, ciphertext, tag):
    cipher = DES.new(key, DES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)

    try:
        cipher.verify(tag)
        return plaintext.decode('ascii')
    except:
        return False
```

```

print('\n*** Data Encryption Standard Algorithm ***')
nonce, ciphertext, tag = encrypt(input('Enter Plain Text:'))
plaintext = decrypt(nonce,ciphertext,tag)

print(f'Cipher Text is: {ciphertext}')

if not plaintext:
    print('Message is Corrupted!')
else:
    print(f'Plain Text is: {plaintext}')

```

### Output:

```

*** Data Encryption Standard Algorithm ***
Enter Plain Text:Hey there! Anushka here..
Cipher Text is: b'j\xab)eZ~\x94\x01X\xdb\t\x8f\x13A\x10w\xc2\xd7wD\xb2\xe0\x10\xdc\xcb'
Plain Text is: Hey there! Anushka here..
PS C:\Users\ANUSHKA\Documents\BTech\CNS\Assign5>

```



## Assignment No 6

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan** - **PRN: 2020BTECS00090** **Class: Final Year - CSE**

#### **Problem Statement:**

#### **Advanced Encryption Standard :**

**AES (Advanced Encryption Standard)** has several advantages that make it a widely adopted and respected encryption algorithm:

**Security:** AES is highly secure and has withstood extensive cryptanalysis since its inception. Its resistance to various cryptographic attacks has been thoroughly evaluated, making it a trusted choice for data encryption.

**Versatility:** AES can be used in various encryption modes, making it suitable for different applications and scenarios. Whether you need to encrypt data at rest or in transit, AES can be adapted to the specific requirements.

**Key Length Options:** AES supports key lengths of 128, 192, and 256 bits, allowing users to choose the level of security that suits their needs. Longer key lengths provide stronger encryption.

**Efficiency:** AES is designed for efficiency and is optimized for both hardware and software implementations. It operates relatively quickly and requires fewer computational resources compared to some other encryption algorithms.

```
File Edit Selection View Go Run Terminal Help
Welcome aes.py x
CNS LAB > Assignment6 > aes.py > ...
1 from Crypto.Cipher import AES
2 import binascii, os
3
4 def encrypt_AES_GCM(msg, secretKey):
5     aesCipher = AES.new(secretKey, AES.MODE_GCM)
6     ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
7     return (ciphertext, aesCipher.nonce, authTag)
8
9 def decrypt_AES_GCM(encryptedMsg, secretKey):
10    (ciphertext, nonce, authTag) = encryptedMsg
11    aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
12    plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
13    return plaintext
14
15 secretKey = os.urandom(32) # 256-bit random encryption key
16 print("Encryption key:", binascii.hexlify(secretKey))
17
18 msg = input("Enter msg for Encryption : ").encode('utf-8')
19 #msg = b'Message for AES-256-GCM + Scrypt encryption'
20 encryptedMsg = encrypt_AES_GCM(msg, secretKey)
21 print("encryptedMsg", {
22     'ciphertext': binascii.hexlify(encryptedMsg[0]),
23     'aesIV': binascii.hexlify(encryptedMsg[1]),
24     'authTag': binascii.hexlify(encryptedMsg[2])
25 })
26
27 decryptedMsg = decrypt_AES_GCM(encryptedMsg, secretKey)
28 print("decryptedMsg", decryptedMsg)
29
```

```
(kali@kali)~[/Desktop]
$ python AES.py
Encryption key: b'd31bac6d065237201ae110c0e9f3ab2855e1aaa3993bcf28737cfc24ec69b8be'
Enter msg for Encryption : brijesh
encryptedMsg {'ciphertext': b'd36c5289a60b36', 'aesIV': b'a6f84b918f240c5d77041add41bcb3e0', 'authTag': b'd000e26f3b1b6fb333b29ae101588599'}
decryptedMsg b'brijesh'
```

## Assignment No 7

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan** - **PRN: 2020BTECS00090** **Class: Final Year - CSE**

#### Problem Statement:

RSA works by generating a pair of keys: a public key and a private key. The public key is used to encrypt messages, and the private key is used to decrypt messages. Only the person with the private key can decrypt messages that have been encrypted with the public key.

RSA is a very secure algorithm, and it is used in a variety of applications, such as HTTPS, SFTP, and PGP.

Here is a brief overview of how RSA works:

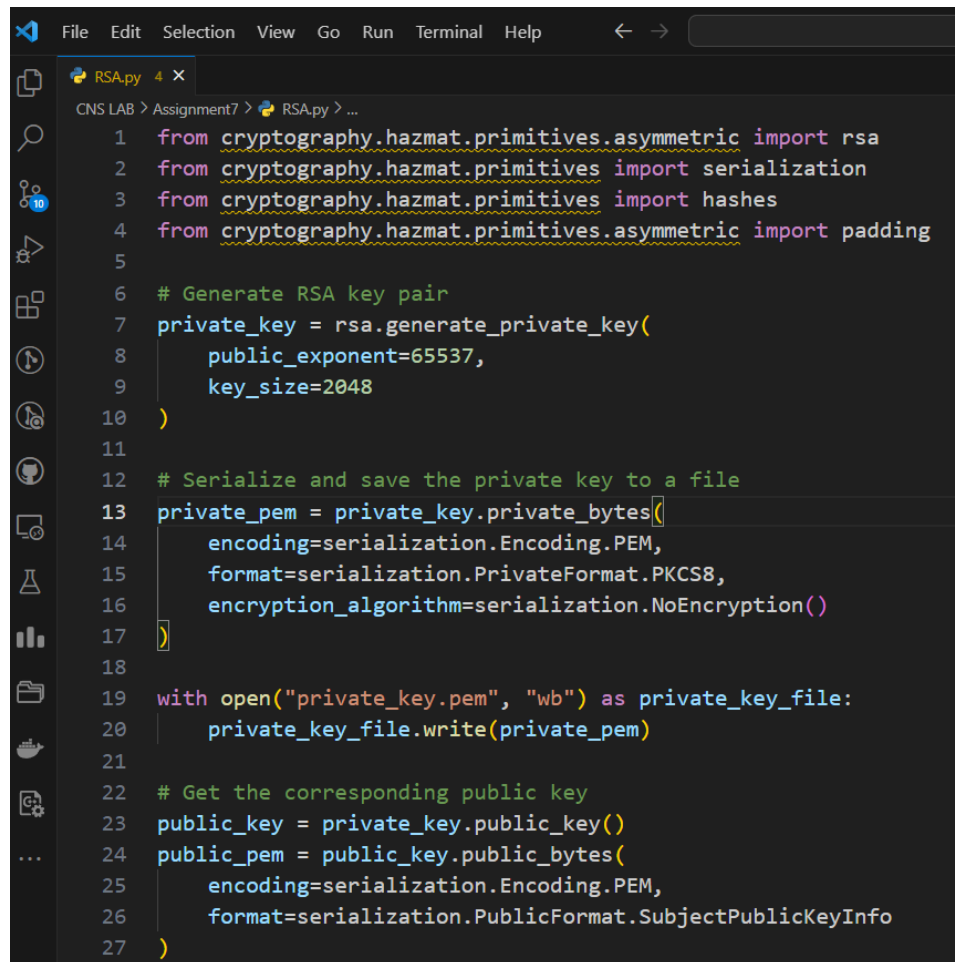
1. The sender of a message generates a random number called the session key.
2. The sender encrypts the message with the session key using the recipient's public key.
3. The sender sends the encrypted message and the session key to the recipient.
4. The recipient decrypts the message with the session key using their private key.

Once the recipient has decrypted the message, they can read it.

RSA is a very versatile algorithm, and it can be used for a variety of purposes, such as:

- Encrypting messages
- Generating digital signatures
- Authenticating users
- Exchanging encryption keys

## RSA Implementation :

A screenshot of a code editor window titled 'RSA.py' with a dark theme. The editor shows Python code for RSA key generation and serialization. The code is numbered from 1 to 27. The imports are from 'cryptography.hazmat.primitives' for 'rsa', 'serialization', 'hashes', and 'padding'. The code generates a private key with a public exponent of 65537 and a key size of 2048. It then serializes the private key to a PEM file named 'private\_key.pem'. Finally, it generates the corresponding public key and serializes it to a PEM file named 'public\_key.pem'.

```
1 from cryptography.hazmat.primitives.asymmetric import rsa
2 from cryptography.hazmat.primitives import serialization
3 from cryptography.hazmat.primitives import hashes
4 from cryptography.hazmat.primitives.asymmetric import padding
5
6 # Generate RSA key pair
7 private_key = rsa.generate_private_key(
8     public_exponent=65537,
9     key_size=2048
10 )
11
12 # Serialize and save the private key to a file
13 private_pem = private_key.private_bytes(
14     encoding=serialization.Encoding.PEM,
15     format=serialization.PrivateFormat.PKCS8,
16     encryption_algorithm=serialization.NoEncryption()
17 )
18
19 with open("private_key.pem", "wb") as private_key_file:
20     private_key_file.write(private_pem)
21
22 # Get the corresponding public key
23 public_key = private_key.public_key()
24 public_pem = public_key.public_bytes(
25     encoding=serialization.Encoding.PEM,
26     format=serialization.PublicFormat.SubjectPublicKeyInfo
27 )
```

```

# Serialize and save the public key to a file
with open("public_key.pem", "wb") as public_key_file:
    public_key_file.write(public_pem)

# Encrypt a message with the public key
message = input("Enter msg for encryption : ").encode('utf-8')

ciphertext = public_key.encrypt(
    message,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

print("Encrypted message:", ciphertext.hex())

# Decrypt the message with the private key
decrypted_message = private_key.decrypt(
    ciphertext,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

print("Decrypted message:", decrypted_message.decode('utf-8'))

```

```

kali@kali: ~/Desktop
$ python rsa_implementation.py
Enter msg for encryption : brijiash
Encrypted message: 5207f86fd39a0f465915280c759bfc000a3e61872e83148ef949f55fc97832c094699430019a357a7e0732aa663e1a9ddb53e0e93ad4e6c997a2b5e302c65dcdcc922bdc2710
a6f285dd42f575b044ff6eb5c5dcfc227473169f202edc97f1a2e7b3f6c7d9c3febff782cd47f0eb5858827774124b56dd6f7ed0570e0ae4a4b833e708df3ca78c64337027e6f3c0d7c52ca5084fd7811
f651d99144b8a991f984ce398ce557a265d4cb1336b97c1500a0a8e57245
Decrypted message: brijiash

```

## Assignment No 8

### Cryptography and Network Security Lab (5CS453)

Name: Vishal Chauhan - PRN: 2020BTECS00090 Class: Final Year - CSE

#### Problem Statement: Diffie-Hellman

Diffie-Hellman key exchange is a cryptographic protocol that allows two parties to establish a shared secret key over an insecure channel. The protocol works by using a publicly known prime number and a privately generated random number to generate a secret key that is only known to the two parties.

The following is a brief overview of how Diffie-Hellman key exchange works:

1. The two parties agree on a publicly known prime number
2.  $p$  and a generator  $g$ .
3. Each party generates a private random number
4.  $a$  and  $b$ .
5. Each party calculates a public value
6.  $A$  and  $B$  using the following formulas:

$$A = g^a \text{ mod } p$$

$$B = g^b \text{ mod } p$$

4. Each party exchanges their public value with the other party.
5. Each party calculates the shared secret key
6.  $K$  using the following formula:

$$K = B^a \text{ mod } p$$

The shared secret key

$K$  can then be used to encrypt and decrypt messages, or to generate digital signatures.

## Defi-Helman :

```
1 from cryptography.hazmat.primitives.asymmetric import dh
2
3 # Generate a Diffie-Hellman parameters object with a larger key size (e.g., 2048 bits)
4 parameters = dh.generate_parameters(generator=2, key_size=2048)
5
6 # Generate private and public keys for Alice and Bob
7 alice_private_key = parameters.generate_private_key()
8 alice_public_key = alice_private_key.public_key()
9
10 bob_private_key = parameters.generate_private_key()
11 bob_public_key = bob_private_key.public_key()
12
13 # Calculate the shared secret key on both sides
14 alice_shared_key = alice_private_key.exchange(bob_public_key)
15 bob_shared_key = bob_private_key.exchange(alice_public_key)
16
17 # The shared secrets should be the same
18 print("Shared Secret (Alice):", int.from_bytes(alice_shared_key, "big"))
19 print("Shared Secret (Bob):", int.from_bytes(bob_shared_key, "big"))
20
```

```

(kali@kali) ~/Desktop
$ python defi_hel.py
Shared Secret (Alice): 8858350558961266433993442948548311260705674877034687676553970619613806246157719835050025556249412205176172607185520867357734107839072958
58865107939464529314652543801703126522988349682869267723950367938100500500945327220298813321575516241355996774119645510303220115450273166117683951438473931811
89011622294754353580599197669198022765445333858293198795959793261749157154415330929168394524335546497710825723862017412680418503512580132039650632568854392502
Shared Secret (Bob): 8858350558961266433993442948548311260705674877034687676553970619613806246157719835050025556249412205176172607185520867357734107839072958
88651079394645293146525438017031265229883496828692677239503679381005005009453272202988133215755162413559967741196455103032201154502731661176839514384739318114
91162229475435358059919766919802276544533385829319879595979326174915715441533092916839452433554649771082572386201741268041850351258013203965063256885439250214
```

## Assignment No 9

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan** - **PRN: 2020BTECS00090** **Class: Final Year - CSE**

#### Problem Statement:

Euclidean division and Extended Euclidean division algorithm

The EEA is a recursive algorithm that finds the greatest common divisor (GCD) of two integers,  $a$  and  $b$ , and also computes the Bezout coefficients,  $x$  and  $y$ , such that the following equation holds:

$$ax + by = \gcd(a, b)$$

The EEA works by repeatedly calling itself with the smaller number and the remainder of the larger number divided by the smaller number. The recursion terminates when the smaller number is 0, and the GCD is the larger number. The Bezout coefficients are computed as follows:

- If  $b$  is 0, then  $x$  is 1 and  $y$  is 0.
- Otherwise,  $x$  is the Bezout coefficient for  $b$  and  $a \% b$ , and  $y$  is the Bezout coefficient for  $a \% b$  and  $b$ .

The following is a step-by-step explanation of the EEA code snippet:

1. The `if b == 0:` statement checks if the smaller number,  $b$ , is 0. If it is, then the GCD is the larger number,  $a$ , and the Bezout coefficients are 1 and 0.
2. Otherwise, the recursive call `gcd, x, y = extended_euclidean(b, a % b)` is made. This call computes the GCD of  $b$  and  $a \% b$ , as well as the Bezout coefficients for  $b$  and  $a \% b$ .
3. The GCD,  $x$ , and  $y$  are then updated using the following formulas:

$$\text{gcd} = \text{gcd}(b, a \% b)$$

$$y = x - (a // b) * y$$

$$x = y$$

The updated values of  $\text{gcd}$ ,  $x$ , and  $y$  are then returned from the function.



The while loop at the end of the code snippet allows the user to perform multiple calculations without having to restart the program. The user is prompted to enter two integers, a and b, and the EEA is called to compute the GCD and Bezout coefficients. The results are then printed to the console. The user is then asked if they want to perform another calculation. If they do, the loop repeats. Otherwise, the program terminates.

Here is an example of how to use the code snippet:

Enter the first integer (a): 10

Enter the second integer (b): 15

GCD of 10 and 15 is 5

x = 2, y = -1

```
1 def euclidean_algorithm(a, b):
2     while b:
3         a, b = b, a % b
4     return a
5
6 def extended_euclidean_algorithm(a, b):
7     if a == 0:
8         return (b, 0, 1)
9     else:
10         gcd, x, y = extended_euclidean_algorithm(b % a, a)
11         return (gcd, y - (b // a) * x, x)
12
13 # Get user input for the two numbers
14 a = int(input("Enter the first number: "))
15 b = int(input("Enter the second number: "))
16
17 # Calculate the GCD using the Euclidean Algorithm
18 gcd = euclidean_algorithm(a, b)
19 print(f"GCD of {a} and {b} is {gcd}")
20
21 # Calculate the GCD, x, and y using the Extended Euclidean Algorithm
22 gcd, x, y = extended_euclidean_algorithm(a, b)
23 print(f"GCD of {a} and {b} is {gcd}")
24 print(f"x: {x}, y: {y}")
25
```

```
(kali@kali)-[~/Desktop]
$ python eucl.py
Enter the first number: 45
Enter the second number: 18
GCD of 45 and 18 is 9
GCD of 45 and 18 is 9
x: 1, y: -2
```

## Assignment No 10

### Cryptography and Network Security Lab (5CS453)

**Name: Vishal Chauhan** - **PRN: 2020BTECS00090** **Class: Final Year - CSE**

#### Problem Statement:

Chinese Remainder theorem :

The Chinese Remainder Theorem (CRT) is a mathematical theorem that states that if two or more integers have different prime factorizations, then there is a unique integer that is congruent to each of the original integers modulo their respective factors. The CRT is useful for solving a variety of problems in number theory, cryptography, and computer science.

One way to think about the CRT is to imagine that you have a number of different clocks, each with a different number of hours on its face. If you start each clock at the same time, they will all eventually reach the same time again, but they will all do so at different intervals. The CRT tells us that there is a unique time at which all of the clocks will be in sync, and that this time can be calculated using the CRT.

The CRT can be used to solve a variety of problems, including:

- Finding the modular inverse of an integer
- Computing the GCD of two integers
- Solving Diophantine equations
- Finding the least common multiple of two integers
- Decrypting messages encrypted using RSA

Here is an example of how the CRT can be used to decrypt a message encrypted using RSA:

1. The sender encrypts the message using their public key.
2. The receiver decrypts the message using their private key.
3. The receiver uses the CRT to recombine the decrypted pieces of the message into the original message.

The CRT is a powerful tool for solving a variety of problems in number theory, cryptography, and computer science. It is used in many popular cryptographic libraries, such as OpenSSL and Cryptography.

# How to use the CRT to decrypt a message encrypted using RSA

To use the CRT to decrypt a message encrypted using RSA, the receiver must know the following:

- Their private key
- The modulus, which is the product of the two large prime numbers that were used to generate the public and private keys

The receiver then follows these steps:

1. Splits the encrypted message into two parts, each of which is a number modulo one of the large prime numbers
2. Decrypts each part of the message using their private key
3. Uses the CRT to recombine the decrypted pieces of the message into the original message

```
def extended_gcd(a, b):  
    """Computes the greatest common divisor of two integers a and b, and  
    also computes the Bezout coefficients, x and y, such that the following  
equation holds:  
  
    ax + by = gcd(a, b)  
    """  
  
    if b == 0:  
        return a, 1, 0  
    else:  
        gcd, x, y = extended_gcd(b, a % b)  
        return gcd, y, x - (a // b) * y
```

```

def modular_inverse(a, m):
    """Computes the modular inverse of a modulo m.

    Args:
        a: An integer.
        m: An integer.

    Returns:
        The modular inverse of a modulo m, or None if no modular inverse exists.
    """

    gcd, x, y = extended_gcd(a, m)
    if gcd != 1:
        return None
    else:
        return x % m


def chinese_remainder_theorem(a, m):
    """Solves the system of linear congruences  $x \equiv a[i] \pmod{m[i]}$  using the CRT.

    Args:
        a: A list of integers.
        m: A list of pairwise coprime integers.

    Returns:
        An integer x that satisfies all of the congruences.
    """

```

```

"""

# Check that the lengths of a and m are the same.
assert len(a) == len(m)

# Calculate the product of all moduli.
M = 1

for mi in m:
    M *= mi

# Calculate x using the CRT.
x = 0

for i in range(len(a)):
    mi = m[i]

    Mi = M // mi

    xi = modular_inverse(Mi, mi)

    x += a[i] * Mi * xi

return x % M

# Example usage:
a = [1, 2, 3]
m = [3, 4, 5]

x = chinese_remainder_theorem(a, m)

```

```
print("Solution x:", x)
```