



MADRAS INSTITUTE OF TECHNOLOGY
ANNA UNIVERSITY



DEPARTMENT OF INFORMATION TECHNOLOGY
IT5711 – MOBILE AND SECURITY LABORATORY

RECORD

REGISTER NUMBER : 2020506055
NAME : MONISH KUMAR A
SEMESTER : 7

DEPARTMENT OF INFORMATION TECHNOLOGY

ANNA UNIVERSITY, MIT CAMPUS

CHROMEPET, CHENNAI – 600 044

BONAFIDE CERTIFICATE

Certified that the bonafide record of the practical work done by
Mr/~~Ms.~~ Monish Kumar A, Register Number: 2020506055 of **Seventh** Semester **B.Tech**
Information Technology during the academic period from **June 2023 to December 2023** in
the IT5711 – Mobile and Security Laboratory.

Date:

Course Instructor: Dr. J. Dhalia Sweetlin

Examiner:

TABLE OF CONTENTS

S. No	DATE	TITLE	PAGE NO	SIGN
1.	01/08/2023	A) Caesar Cipher	5	
	08/08/2023	B) Affine Cipher	8	
	08/08/2023	C) Hill Cipher	12	
	08/08/2023	D) Transposition Cipher	17	
2.	08/08/2023	Cryptanalytic Attacks	20	
3.	22/08/2023	Symmetric Encryption - DES	24	
	29/08/2023	Symmetric Encryption - AES	30	
4.	19/09/2023	Asymmetric Encryption / PKCs - RSA	36	
	10/10/2023	Asymmetric Encryption / PKCs – Diffie Hellman	40	
	10/10/2023	Asymmetric Encryption / PKCs – ElGamal Cryptosystem	42	
5.	17/10/2023	Hashing algorithms – MD5 and SHA	46	
6.	03/10/2023	Event handling and push notifications in Android applications	50	
7.	20/10/2023	Animations and graphical primitives in Android applications	54	
8.	20/10/2023	Location based services in Android	60	

Aim:

To implement Caesar Cipher and to encrypt and decrypt plaintext messages using them.

Algorithm:

- Choose a Shift Value: Pick a number, known as the "shift" or "key." This determines how much each letter in the message will be moved in the alphabet.
- Prepare Your Message: Take the message you want to protect. Remove any spaces and punctuation, as the Caesar cipher works with letters only.
- Encrypt the Message: Go through each letter in the message and shift it according to the chosen key. To encrypt a letter, add the shift value to its position in the alphabet. If it goes beyond 'Z,' wrap around to 'A.'
- Finish Encrypting: Continue this process for all the letters in the message until you've encrypted the entire text.
- Encrypted Message (Ciphertext): The result is your encrypted message, also called the ciphertext. You can send this to someone, but they must know the same shift value to decrypt it.
- Decrypting the Message: To decrypt the message, the recipient uses the same shift value but in reverse. They subtract the shift from each letter in the ciphertext to reveal the original message.

Source Code:**I) caesar.py**

```
alphabets = "abcdefghijklmnopqrstuvwxyz"
def encrypt(plaintext: str, key: int) -> str:
    cipher = ""
    for char in plaintext:
        cipher += alphabets[(alphabets.find(char)+key)%26]
    return cipher
def decrypt(ciphertext: str, key: int) -> str:
    plain = ""
    for char in ciphertext:
        plain += alphabets[(alphabets.index(char)-key)%26]
    return plain
```

II) client.py

```
import socket
import sys
import caesar
HOST = "127.0.0.1"
PORT = 9090
key: int = 0
def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    clientSocket.connect((HOST, PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST, PORT))
def handleConnection():
    while True:
        serverCipher = clientSocket.recv(1024).decode("utf-8").strip()
        serverMessage = caesar.decrypt(serverCipher, key)
```

```

        print("[SERVER]: {}".format(serverMessage))
        print("[SERVER RAW]: {}".format(serverCipher))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = caesar.encrypt(clientMessage,key) + '\n'
        clientSocket.send(clientCipher.encode("utf-8"))
        if clientMessage.lower() == "exit":
            break
if __name__ == "__main__":
    key = int(input("[CLIENT]> Enter key:"))
    if len(sys.argv) == 2:
        HOST = sys.argv[1]
        connectServer()
        handleConnection()
        clientSocket.close()
        print("[INFO]: Connection to server closed and exiting...")

```

III) server.py

```

import socket
import caesar
HOST = "127.0.0.1"
PORT = 9090
key:int = 0
def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))
def handleConnections() -> None:
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from
    {}:{}".format(clientAddress[0],clientAddress[1]))
    try:
        while True:
            userInput = input("[SERVER]> Enter message:")
            cipherInput = caesar.encrypt(userInput,key) + "\n"
            clientSocket.send(cipherInput.encode("utf-8"))
            clientCipher = clientSocket.recv(1024).decode("utf-8").strip()
            clientMessage = caesar.decrypt(clientCipher,key)
            print("[CLIENT RAW]: {}".format(clientCipher))
            print("[CLIENT]: {}".format(clientMessage))
            if clientMessage.lower() == "exit":
                break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")
    finally:
        print("[INFO]: Closing server")
        clientSocket.close()
        serverSocket.close()
if __name__ == "__main__":
    key = int(input("[SERVER]> Enter key:"))
    initServer()
    handleConnections()

```

Output:

I) Server Side:

```
...ster VII/Cryptography and Security /P/1. Caesar Cipher 09:32:29
└─> python server.py
[SERVER]> Enter key:5
[SERVER]: Server started at 127.0.0.1:9090 and listening
[INFO]: Connection established from 127.0.0.1:38428
[SERVER]> Enter message:hellothere
[CLIENT RAW]: mjqqtgfhp
[CLIENT]: helloback
[SERVER]> Enter message:yes
[CLIENT RAW]: jcnv
[CLIENT]: exit
[INFO]: Closing server
```

II) Client Side:

```
...ster VII/Cryptography and Security /P/1. Caesar Cipher 09:32:33
└─> python client.py
[CLIENT]> Enter key:5
[INFO]: Connected to server at 127.0.0.1:9090
[SERVER]: hellothere
[SERVER RAW]: mjqqtymjwj
[CLIENT]> Enter Message:helloback
[SERVER]: yes
[SERVER RAW]: djx
[CLIENT]> Enter Message:exit
[INFO]: Connection to server closed and exiting...
```

Result:

Hence, Caesar cipher was implemented successfully and tested over a client-server program.

Aim:

To implement Affine Cipher and to encrypt and decrypt plaintext messages using them.

Algorithm:

- Choose two key values, "a" and "b." Ensure that "a" is relatively prime to 26 (no common factors except 1).
- Take the plaintext message for encryption or the ciphertext for decryption.
- Convert each letter in the message to its numerical equivalent (A=0, B=1, ..., Z=25).
- Encryption: Apply the encryption formula:
$$\text{Ciphertext (C)} = (a * \text{Plaintext} + b) \bmod 26.$$
- Decryption: Apply the decryption formula:
$$\text{Plaintext (P)} = (a^{-1} * (C - b)) \bmod 26.$$
- In both cases, "a" is the first key value, "Plaintext" is the numerical value of the letter (or "C" for decryption), and "b" is the second key value.
- Record the resulting ciphertext (for encryption) or plaintext (for decryption) values for each letter.
- Convert the numerical values back to their corresponding letters to get the final message.

Source Code:**I) affine.py**

```
alphabets = "abcdefghijklmnopqrstuvwxyz"
```

```
def encrypt(plaintext: str, a:int, b:int) -> str:
    cipher = ""
    for char in plaintext:
        index = (alphabets.find(char)*a) + b
        index = index % 26
        cipher += alphabets[index]
    return cipher
```

```
def decrypt(ciphertext:str, a:int, b:int) -> str:
    aInverse = pow(a,-1,26)
    plain = ""
    for char in ciphertext:
        index = (alphabets.find(char) - b) * aInverse
        index = index % 26
        plain += alphabets[index]
    return plain
```

II) client.py

```
import socket
import sys
import affine
from math import gcd
```

```
HOST = "127.0.0.1"
PORT = 9090
a:int = 0
b:int = 0
```

```

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    clientSocket.connect((HOST,PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST,PORT))

def handleConnection():
    while True:
        serverCipher = clientSocket.recv(1024).decode("utf-8").strip()
        serverMessage = affine.decrypt(serverCipher,a,b)
        print("[SERVER]: {}".format(serverMessage))
        print("[SERVER RAW]: {}".format(serverCipher))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = affine.encrypt(clientMessage,a,b) + '\n'
        clientSocket.send(clientCipher.encode("utf-8"))
        if clientMessage.lower() == "exit":
            break

if __name__ == "__main__":
    while True:
        a = int(input("[CLIENT]> Enter key (a):"))
        if gcd(a,26) == 1:
            break
        b = int(input("[CLIENT]> Enter key (b):"))
        if len(sys.argv) == 2:
            HOST = sys.argv[1]
        connectServer()
        handleConnection()
        clientSocket.close()
        print("[INFO]: Connection to server closed and exiting...")

```

III) server.py

```

import socket
import affine
from math import gcd

HOST = "127.0.0.1"
PORT = 9090
a:int = 0
b:int = 0

def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))

def handleConnections() -> None:
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from

```



```

{:}:".format(clientAddress[0],clientAddress[1]))
try:
    while True:
        userInput = input("[SERVER]> Enter message:")
        cipherInput = affine.encrypt(userInput,a,b) + "\n"
        clientSocket.send(cipherInput.encode("utf-8"))
        clientCipher = clientSocket.recv(1024).decode("utf-8").strip()
        clientMessage = affine.decrypt(clientCipher,a,b)
        print("[CLIENT RAW]: {}".format(clientCipher))
        print("[CLIENT]: {}".format(clientMessage))
        if clientMessage.lower() == "exit":
            break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")
    finally:
        print("[INFO]: Closing server")
        clientSocket.close()
        serverSocket.close()

if __name__ == "__main__":
    while True:
        a = int(input("[CLIENT]> Enter key (a):"))
        if gcd(a,26) == 1:
            break
    b = int(input("[SERVER]> Enter key (b):"))
    initServer()
    handleConnections()

```

Output:

I) Server Side:

```
...ster VII/Cryptography and Security /P/2. Affine Cipher 09:45:26
└─> python server.py
[CLIENT]> Enter key (a):5
[SERVER]> Enter key (b):8
[SERVER]: Server started at 127.0.0.1:9090 and listening
[INFO]: Connection established from 127.0.0.1:56366
[SERVER]> Enter message:hellothere
[CLIENT RAW]: rcllanisg
[CLIENT]: helloback
[SERVER]> Enter message:goodday
[CLIENT RAW]: ctwz
[CLIENT]: exit
[INFO]: Closing server
```

II) Client Side:

```
...ster VII/Cryptography and Security /P/2. Affine Cipher 09:45:24
└─> python client.py
[CLIENT]> Enter key (a):5
[CLIENT]> Enter key (b):8
[INFO]: Connected to server at 127.0.0.1:9090
[SERVER]: hellothere
[SERVER RAW]: rcllazrcpc
[CLIENT]> Enter Message:helloback
[SERVER]: goodday
[SERVER RAW]: maaxxiy
[CLIENT]> Enter Message:exit
[INFO]: Connection to server closed and exiting...
```

Result:

Hence, Affine cipher was implemented successfully and tested over a client-server program.

Aim:

To implement Hill Cipher and to encrypt and decrypt plaintext messages using them

Algorithm:**Encryption:**

- Choose a square matrix of size $n \times n$ as the key.
- Ensure the key is invertible. Its determinant should be relatively prime to 26.
- Take the plaintext and divide it into n -letter blocks.
- Convert each letter to its numerical equivalent.
- Multiply the key matrix with the numerical values of the plaintext block.
- Apply modulo 26 to each element of the result.
- Convert numerical values back to letters.
- Repeat for all blocks to encrypt the entire message.

Decryption:

- Take the ciphertext and divide it into n -letter blocks.
- Convert each letter to its numerical equivalent.
- Calculate the inverse of the key matrix used for encryption.
- Multiply the key matrix inverse with the numerical values of the ciphertext block.
- Apply modulo 26 to each element of the result.
- Convert numerical values back to letters.
- Repeat for all blocks to decrypt the entire message.

Source Code:**I) hill.py**

```
def getMatrixMinor(m: list, i: int, j: int) -> list:
    return [row[:j] + row[j+1:] for row in (m[:i]+m[i+1:])]

def getMatrixDeterminant(m: list) -> int:
    if len(m) == 1:
        return m[0][0]
    if len(m) == 2:
        return m[0][0]*m[1][1]-m[0][1]*m[1][0]
    determinant = 0
    for c in range(len(m)):
        determinant += ((-1)**c)*m[0][c]*getMatrixDeterminant(getMatrixMinor(m,0,c))
    return determinant

def getModInverse(matrix: list, p: int) -> list:
    mLen = len(matrix)
    inverse = []
    det = getMatrixDeterminant(matrix)
    modInv = pow(det,-1,p)
    for _ in range(mLen):
        inverse.append([0 for _ in range(mLen)])
    for i in range(mLen):
        for j in range(mLen):
            inverse[j][i] = (((-1)**(i+j))*getMatrixDeterminant(getMatrixMinor(matrix,i,j))) % p
            inverse[j][i] = (inverse[j][i] * modInv) % p
```

```

return inverse

def getMatrixProduct(mat1: list, mat2: list) -> list:
    product = []
    for _ in range(len(mat1)):
        product.append([0 for _ in range(len(mat2[0]))])
    for i in range(len(mat1)):
        for j in range(len(mat2[0])):
            for k in range(len(mat2)):
                product[i][j] += mat1[i][k] * mat2[k][j]
    return product

def encrypt(plaintext:str, key: list) -> str:
    pLen = len(plaintext)
    keyOrder = len(key)
    plainMatrix = []
    row = []
    it = 0
    while pLen % keyOrder != 0:
        plaintext += 'x'
        pLen = len(plaintext)
    for char in plaintext:
        row.append(ord(char) - 97)
        it += 1
        if it % keyOrder == 0:
            plainMatrix.append(row)
            row = []
    cipherMatrix = getMatrixProduct(plainMatrix, key)
    cipher = ""
    for row in cipherMatrix:
        for char in row:
            cipher += chr((char % 26) + 97)
    return cipher

def decrypt(ciphertext:str, key: list) -> str:
    cipherMatrix = []
    keyOrder = len(key)
    row = []
    it = 0
    for char in ciphertext:
        row.append(ord(char) - 97)
        it += 1
        if it % keyOrder == 0:
            cipherMatrix.append(row)
            row = []
    keyInverse = getModInverse(key, 26)
    plainMatrix = getMatrixProduct(cipherMatrix, keyInverse)
    plain = ""
    for row in plainMatrix:
        for char in row:
            plain += chr((char % 26) + 97)
    return plain

```

II) client.py

```
import socket
import sys
import hill

HOST = "127.0.0.1"
PORT = 9090
key = []

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    clientSocket.connect((HOST,PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST,PORT))

def handleConnection():
    while True:
        serverCipher = clientSocket.recv(1024).decode("utf-8").strip()
        serverMessage = hill.decrypt(serverCipher,key)
        print("[SERVER]: {}".format(serverMessage))
        print("[SERVER RAW]: {}".format(serverCipher))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = hill.encrypt(clientMessage,key) + '\n'
        clientSocket.send(clientCipher.encode("utf-8"))
        if clientMessage.lower() == "exit":
            break

if __name__ == "__main__":
    while True:
        n = int(input("[SERVER]> Enter key order:"))
        for i in range(n):
            row = []
            for j in range(n):
                temp = int(input("[SERVER]: Enter key value at {},{}:".format(i+1,j+1)))
                row.append(temp)
            key.append(row)
        if hill.getMatrixDeterminant(key) != 0:
            break
    if len(sys.argv) == 2:
        HOST = sys.argv[1]
    connectServer()
    handleConnection()
    clientSocket.close()
    print("[INFO]: Connection to server closed and exiting...")
```

III) server.py

```
import socket
import hill

HOST = "127.0.0.1"
PORT = 9090
key = []
```

```

def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))

def handleConnections() -> None:
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from
    {}:{}".format(clientAddress[0],clientAddress[1]))
    try:
        while True:
            userInput = input("[SERVER]> Enter message:")
            cipherInput = hill.encrypt(userInput,key) + "\n"
            clientSocket.send(cipherInput.encode("utf-8"))
            clientCipher = clientSocket.recv(1024).decode("utf-8").strip()
            clientMessage = hill.decrypt(clientCipher,key)
            print("[CLIENT RAW]: {}".format(clientCipher))
            print("[CLIENT]: {}".format(clientMessage))
            if clientMessage.lower() == "exit":
                break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")
    finally:
        print("[INFO]: Closing server")
        clientSocket.close()
        serverSocket.close()

if __name__ == "__main__":
    while True:
        n = int(input("[SERVER]> Enter key order:"))
        for i in range(n):
            row = []
            for j in range(n):
                temp = int(input("[SERVER]: Enter key value at {},{}:".format(i+1,j+1)))
                row.append(temp)
            key.append(row)
        if hill.getMatrixDeterminant(key) != 0:
            break
    initServer()
    handleConnections()

```

Output:

I) Server Side:

```
...mester VII/Cryptography and Security /P/3. Hill Cipher 09:57:01
└─> python server.py
[SERVER]> Enter key order:2
[SERVER]: Enter key value at 1,1:5
[SERVER]: Enter key value at 1,2:8
[SERVER]: Enter key value at 2,1:17
[SERVER]: Enter key value at 2,2:3
[SERVER]: Server started at 127.0.0.1:9090 and listening
[INFO]: Connection established from 127.0.0.1:47736
[SERVER]> Enter message:hellothere
[CLIENT RAW]: zqirjligzt
[CLIENT]: hellobackx
[SERVER]> Enter message:good
[CLIENT RAW]: vxzr
[CLIENT]: exit
[INFO]: Closing server
```

II)Client Side:

```
...mester VII/Cryptography and Security /P/3. Hill Cipher 09:56:57
└─> python client.py
[SERVER]> Enter key order:2
[SERVER]: Enter key value at 1,1:5
[SERVER]: Enter key value at 1,2:8
[SERVER]: Enter key value at 2,1:17
[SERVER]: Enter key value at 2,2:3
[INFO]: Connected to server at 127.0.0.1:9090
[SERVER]: hellothere
[SERVER RAW]: zqirdnzqxs
[CLIENT]> Enter Message:helloback
[SERVER]: good
[SERVER RAW]: imrr
[CLIENT]> Enter Message:exit
[INFO]: Connection to server closed and exiting...
```

Result:

Hence, Hill cipher was implemented successfully and tested over a client-server program.

Aim:

To implement Column Transposition Cipher and to encrypt and decrypt plaintext messages using them

Algorithm:

- Choose a key, which determines the order in which the letters are rearranged.
- Take the plaintext message.
- Remove spaces and punctuation, if needed.
- Rearrange the letters of the message based on the chosen key.
- The rearranged message is the ciphertext.
- Take the ciphertext message.
- Rearrange the letters back to their original order based on the key. The restored message is the plaintext.

Source Code:**I) transposition.py**

```
def encrypt(plaintext:str, key: list) -> str:
    plainMatrix = []
    row = []
    kLen = len(key)
    cipher = ""
    for char in plaintext:
        row.append(char)
        if len(row) % kLen == 0:
            plainMatrix.append(row)
            row = []
    if row != []:
        while len(row) % kLen != 0:
            row.append('x')
        plainMatrix.append(row)
    for col in key:
        for r in plainMatrix:
            cipher += r[col-1]
    return cipher

def decrypt(ciphertext:str, key: list) -> str:
    kLen = len(key)
    plain = ""
    ptr, iter = 0, 0
    cipherMatrix = []
    cLen = len(ciphertext)//kLen
    for _ in range(cLen):
        cipherMatrix.append(['a' for _ in key])
    for char in ciphertext:
        cipherMatrix[iter % cLen][key[ptr]-1] = char
        iter += 1
        if iter % cLen == 0:
            ptr += 1
    for row in cipherMatrix:
        plain += ".join(row)
    return plain
```


II) client.py

```
import socket
import sys
import transposition

HOST = "127.0.0.1"
PORT = 9090
key = []

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    clientSocket.connect((HOST,PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST,PORT))

def handleConnection():
    while True:
        serverCipher = clientSocket.recv(1024).decode("utf-8").strip()
        serverMessage = transposition.decrypt(serverCipher,key)
        print("[SERVER]: {}".format(serverMessage))
        print("[SERVER RAW]: {}".format(serverCipher))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = transposition.encrypt(clientMessage,key) + '\n'
        clientSocket.send(clientCipher.encode("utf-8"))
        if clientMessage.lower() == "exit":
            break

if __name__ == "__main__":
    key = [int(i) for i in input("[CLIENT]> Enter key:").split()]
    if len(sys.argv) == 2:
        HOST = sys.argv[1]
    connectServer()
    handleConnection()
    clientSocket.close()
    print("[INFO]: Connection to server closed and exiting...")
```

III) server.py

```
import socket
import transposition

HOST = "127.0.0.1"
PORT = 9090
key = []

def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))

def handleConnections() -> None:
```

```

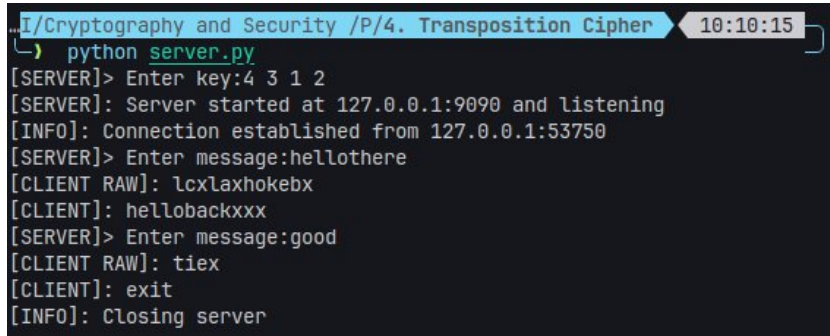
clientSocket, clientAddress = serverSocket.accept()
print("[INFO]: Connection established from
{},{ {}".format(clientAddress[0],clientAddress[1]))
try:
    while True:
        userInput = input("[SERVER]> Enter message:")
        cipherInput = transposition.encrypt(userInput,key) + "\n"
        clientSocket.send(cipherInput.encode("utf-8"))
        clientCipher = clientSocket.recv(1024).decode("utf-8").strip()
        clientMessage = transposition.decrypt(clientCipher,key)
        print("[CLIENT RAW]: {}".format(clientCipher))
        print("[CLIENT]: {}".format(clientMessage))
        if clientMessage.lower() == "exit":
            break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")
    finally:
        print("[INFO]: Closing server")
        clientSocket.close()
        serverSocket.close()

if __name__ == "__main__":
    key = [int(i) for i in input("[SERVER]> Enter key:").split()]
    initServer()
    handleConnections()

```

Output:

I) Server Side:

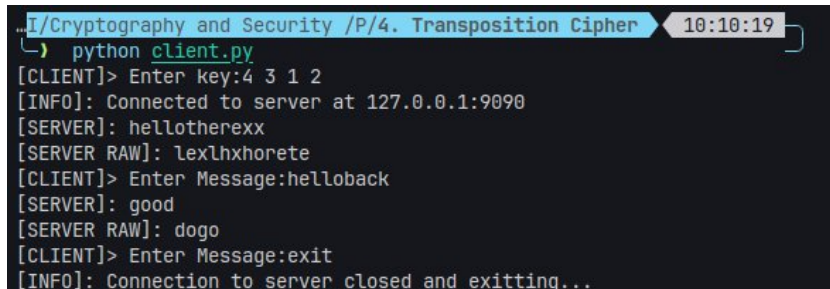


```

I/Cryptography and Security /P/4. Transposition Cipher 10:10:15
python server.py
[SERVER]> Enter key:4 3 1 2
[SERVER]: Server started at 127.0.0.1:9090 and listening
[INFO]: Connection established from 127.0.0.1:53750
[SERVER]> Enter message:hellothere
[CLIENT RAW]: lcxlxhokebx
[CLIENT]: hellobackxxx
[SERVER]> Enter message:good
[CLIENT RAW]: tiex
[CLIENT]: exit
[INFO]: Closing server

```

II) Client Side:



```

I/Cryptography and Security /P/4. Transposition Cipher 10:10:19
python client.py
[CLIENT]> Enter Key:4 3 1 2
[INFO]: Connected to server at 127.0.0.1:9090
[SERVER]: hellotherexx
[SERVER RAW]: lexlxhorete
[CLIENT]> Enter Message:helloback
[SERVER]: good
[SERVER RAW]: dogo
[CLIENT]> Enter Message:exit
[INFO]: Connection to server closed and exiting...

```

Result:

Hence, Column Transposition cipher was implemented successfully and tested over a client-server program.

Aim:

To employ attacks on Caesar, Affine and Hill ciphers to find keys used to cipher the plain text.

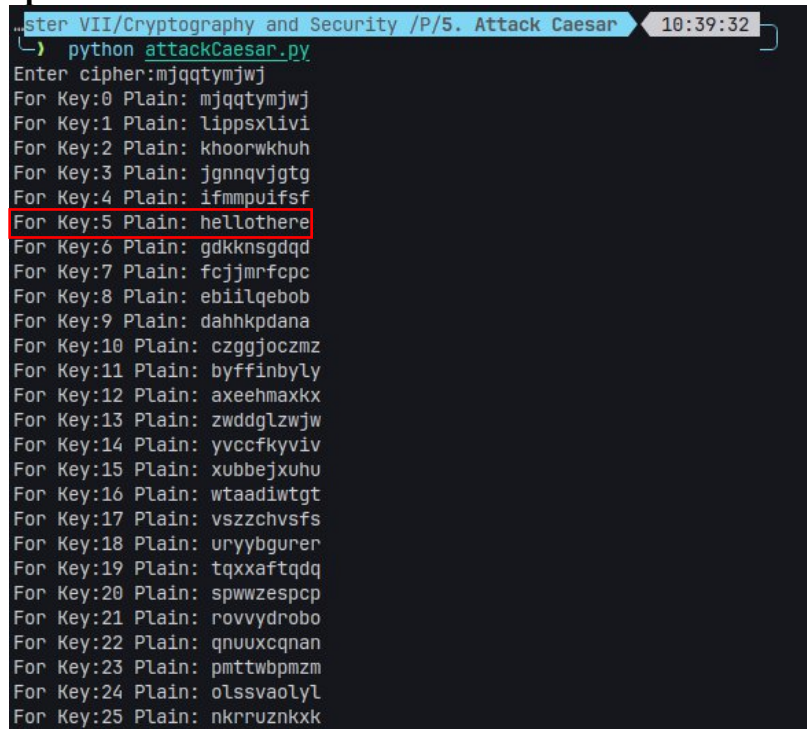
A) Caesar Cipher:**Algorithm:**

- Brute force attack on Caesar cipher can be mounted by generating possible plain text for all 25 keys from cipher text and manually check for readable plaintext.

Source Code:

```
alphabets = "abcdefghijklmnopqrstuvwxyz"
```

```
if __name__ == "__main__":
    cipher = input("Enter cipher:")
    for key in range(26):
        attackPlain = ""
        for char in cipher:
            attackPlain += chr((((ord(char)-97)-key)%26)+97)
        print("For Key:{ } Plain: { }".format(key,attackPlain))
```

Output:


```
ster VII/Cryptography and Security /P/5. Attack Caesar 10:39:32
python attackCaesar.py
Enter cipher:mjqqtymjwj
For Key:0 Plain: mjqqtymjwj
For Key:1 Plain: lippsxlivi
For Key:2 Plain: khoorwkhuh
For Key:3 Plain: jgnnqvjgtg
For Key:4 Plain: ifmmpuifsf
For Key:5 Plain: hellothere
For Key:6 Plain: gdkknsqddq
For Key:7 Plain: fcjjmrfcpc
For Key:8 Plain: ebiilqebob
For Key:9 Plain: dahhkpdana
For Key:10 Plain: czggjoczmz
For Key:11 Plain: byffinbyly
For Key:12 Plain: axeehmaxkx
For Key:13 Plain: zwddglzjw
For Key:14 Plain: yvccfkyviv
For Key:15 Plain: xubbejxuhu
For Key:16 Plain: wtaadiwtgt
For Key:17 Plain: vszzchvsfs
For Key:18 Plain: uryybgurer
For Key:19 Plain: tqxxaftqdq
For Key:20 Plain: spwwzespcp
For Key:21 Plain: rovydrobo
For Key:22 Plain: qnuuxcqnan
For Key:23 Plain: pmttwbpmzm
For Key:24 Plain: olssvaolyl
For Key:25 Plain: nkrruznkxk
```

B) Affine Cipher:**Algorithm:**

- Brute force attack on Affine cipher can be mounted by generating all possible plaintexts for possible keys in $a = \{1,3,5,7,9,11,15,17,19,21,23,25\}$ and $b = [0,25]$
- If Cipher text and corresponding plaintext are known, an attack to find the key can be mounted formulating two linear equation in two variables and solving for a and b .

Source Code:

```
possible_a = [1,3,5,7,9,11,15,17,19,21,23,25]
if __name__ == "__main__":
    print("Enter method of attack:\n1. Brute force\n2. Known plain and cipher")
```

```

choice = int(input("Enter choice:"))
try:
    if choice == 1:
        cipher = input("Enter cipher:")
        for a in possible_a:
            for b in range(26):
                plain = ""
                aInv = pow(a,-1,26)
                for char in cipher:
                    plain += chr((((ord(char)-97-b)*aInv)%26)+97)
                print("For a={ } b={ } plaintext:{ }".format(a,b,plain))
            if input("Press enter to continue or enter q to quit:") == "q":
                break
    elif choice == 2:
        cipher = input("Enter cipher:")
        plain = input("Enter corresponding plaintext:")
        det = (ord(plain[0]) - ord(plain[1])) % 26
        detInv = pow(det,-1,26)
        a = (detInv * (ord(cipher[0]) - ord(cipher[1]))) % 26
        b = (detInv * (((ord(plain[0])-97) * (ord(cipher[1])-97)) - ((ord(cipher[0])-97) * (ord(plain[1])-97)))) % 26
        print("Keys are a:{ } b:{ }".format(a,b))
    else:
        print("Enter right option")
except ValueError:
    print("A modular inverse cannot be computed. Try with another set of inputs")

```

Output:

```

Semester VII/Cryptography and Security /P/7. Attack Affine 10:47:52
python attackAffine.py
Enter method of attack:
1. Brute force
2. Known plain and cipher
Enter choice:1
Enter cipher:rcllazrcpc
For a=5 b=0 plaintext:tqxxaftqdd
For a=5 b=1 plaintext:yvccfkyviv
For a=5 b=2 plaintext:dahhkpdana
For a=5 b=3 plaintext:ifmmpuifsf
For a=5 b=4 plaintext:nkrruznkxx
For a=5 b=5 plaintext:spwzespccp
For a=5 b=6 plaintext:xubbejxuhu
For a=5 b=7 plaintext:czggjocmz
For a=5 b=8 plaintext:hellothere
For a=5 b=9 plaintext:mjqqtymjwj
For a=5 b=10 plaintext:rovvydrobo
For a=5 b=11 plaintext:wtaadiwtgt
For a=5 b=12 plaintext:byffinbyly
For a=5 b=13 plaintext:gdkknsqdqd
For a=5 b=14 plaintext:lippsxlivi
For a=5 b=15 plaintext:qnuvxcqnan
For a=5 b=16 plaintext:vszzchvsfs
For a=5 b=17 plaintext:axeehmaxkx
For a=5 b=18 plaintext:fcjjmrfcpc
For a=5 b=19 plaintext:khoorwkhuu
For a=5 b=20 plaintext:pmttwbpmzm
For a=5 b=21 plaintext:uryybgurer
For a=5 b=22 plaintext:zwdgdglzwjw
For a=5 b=23 plaintext:ebiilqebob
For a=5 b=24 plaintext:jgnnqvjgtg
For a=5 b=25 plaintext:olssvaolyl
Press enter to continue or enter q to quit:q

```

```
..I/Cryptography and Security /P/7. Attack Affine 1m 32s 10:49:34
python attackAffine.py
Enter method of attack:
1. Brute force
2. Known plain and cipher
Enter choice:2
Enter cipher:rcllazrope
Enter corresponding plaintext:hellothere
Keys are a:5 b:8
```

C) Hill Cipher:

Algorithm:

- If cipher text and corresponding plain text are known, then an attack on hill cipher can be mounted by finding the inverse of plain text matrix and multiplying with cipher text matrix. It should be noted that the length of known text must be atleast equal to the size of key and plain text matrix must be invertible.

Source Code:

```
def getMatrixMinor(m: list,i: int,j: int) -> list:
    return [row[:j] + row[j+1:] for row in (m[:i]+m[i+1:])]

def getMatrixDeterminant(m: list) -> int:
    if len(m) == 1:
        return m[0][0]
    if len(m) == 2:
        return m[0][0]*m[1][1]-m[0][1]*m[1][0]
    determinant = 0
    for c in range(len(m)):
        determinant += ((-1)**c)*m[0][c]*getMatrixDeterminant(getMatrixMinor(m,0,c))
    return determinant

def getModInverse(matrix: list,p: int) -> list:
    mLen = len(matrix)
    inverse = []
    det = getMatrixDeterminant(matrix)
    modInv = pow(det,-1,p)
    for _ in range(mLen):
        inverse.append([0 for _ in range(mLen)])
    for i in range(mLen):
        for j in range(mLen):
            inverse[j][i] = (((-1)**(i+j))*getMatrixDeterminant(getMatrixMinor(matrix,i,j))) % p
            inverse[j][i] = (inverse[j][i] * modInv) % p
    return inverse

def getMatrixProduct(mat1: list, mat2: list) -> list:
    product = []
    for _ in range(len(mat1)):
        product.append([0 for _ in range(len(mat2[0]))])
    for i in range(len(mat1)):
        for j in range(len(mat2[0])):
            for k in range(len(mat2)):
                product[i][j] += mat1[i][k] * mat2[k][j]
    return product

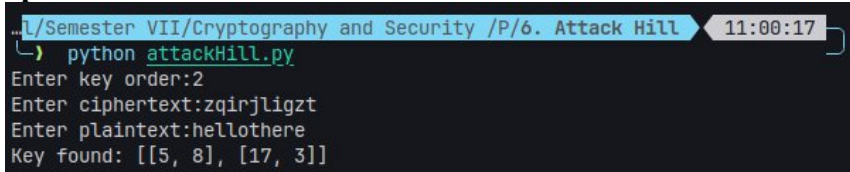
if __name__ == "__main__":
    keyOrder = int(input("Enter key order:"))
```

```

cipher = input("Enter ciphertext:")
plain = input("Enter plaintext:")
plainMatrix = []
cipherMatrix = []
row = []
colSize = 0
it = 0
for char in plain:
    row.append(ord(char) - 97)
    it += 1
    if it % keyOrder == 0:
        plainMatrix.append(row)
        row = []
        colSize += 1
    if colSize == keyOrder:
        break
colSize = 0
for char in cipher:
    row.append(ord(char) - 97)
    it += 1
    if it % keyOrder == 0:
        cipherMatrix.append(row)
        row = []
        colSize += 1
    if colSize == keyOrder:
        break
try:
    plainInverse = getModInverse(plainMatrix,26)
    key = getMatrixProduct(plainInverse,cipherMatrix)
    for i in range(len(key)):
        for j in range(len(key[0])):
            key[i][j] %= 26
    print("Key found: {}".format(key))
except ValueError:
    print("The given plaintext has no inverse... Try with some other combination")

```

Output:



```

C:/Semester VII/Cryptography and Security /P/6. Attack Hill 11:00:17
> python attackHill.py
Enter key order:2
Enter ciphertext:zqirjligzt
Enter plaintext:hellothere
Key found: [[5, 8], [17, 3]]

```

Result:

Hence, cryptanalytic attacks were successfully mounted on Caesar, Affine and Hill cipher and their keys were found using the attacks.

AIM:

To implement symmetric encryption algorithm DES, and encrypt plain text and decrypt cipher text using them.

ALGORITHM:**Key Generation:**

- Generate a 56-bit secret key.
- Expand the key to 64 bits and generate 16 subkeys, each 48 bits long.

Encryption:

Initial Permutation (IP): Permute the plaintext using the initial permutation table.

Feistel Network (16 Rounds):

- Split the 64-bit data block into two 32-bit halves: L0 and R0.
- For each round (i = 1 to 16):
- Calculate the new left half (Li) as the previous right half (Ri-1).
- Calculate the new right half (Ri) as the previous left half (Li-1) XOR f(Ri-1, Ki) where Ki is the subkey for round i.
- Swap the final left and right halves after 16 rounds: L16 and R16.

Final Permutation (FP): Permute the data block using the final permutation table.

Output: The 64-bit permuted block is the ciphertext.

Decryption:

Initial Permutation (IP): Permute the ciphertext using the initial permutation table.

Feistel Network (16 Rounds, in reverse order):

- Split the 64-bit ciphertext block into two 32-bit halves: L16 and R16 (initially swapped from the encryption process).
- For each round (i = 16 to 1, in reverse order):
- Calculate the new right half (Ri-1) as the previous left half (Li).
- Calculate the new left half (Li-1) as the previous right half (Ri) XOR f(Li, Ki) where Ki is the subkey for round i.

Final Permutation (FP): Permute the data block using the final permutation table.

Output: The 64-bit permuted block is the plaintext.

SOURCE CODE:**I) DES.py**

```
#Initial permut matrix for the datas
```

```
PI = [...]
```

```
#Initial permut made on the key
```

```
CP_1 = [...]
```

```
#Permut applied on shifted key to get Ki+1
```

```
CP_2 = [...]
```

```
#Expand matrix to get a 48bits matrix of datas to apply the xor with Ki
```

```
E = [...]
```

```
#SBOX
```

```
S_BOX = [...]
```

```
#Permut made after each SBox substitution for each round
```

```
P = [...]
```

```
#Final permut for datas after the 16 rounds
```

```
IPInv = [...]
```

```
SHIFT = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,1]
```

```
def toBitArray(text):
```

```
    array = list()
```

```
    for char in text:
```

```
        binval = toBinary(char, 8)
```

```
        array.extend([int(x) for x in list(binval)])
```

```
    return array
```

```
def bitArrToString(array):
```

```
    res = ".join([chr(int(y,2)) for y in [".join([str(x) for x in _bytes]) for _bytes in nsplit(array,8)])]
```

```
    return res
```

```
def toBinary(val, bitsize):
```

```
    binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]
```

```
    if len(binval) > bitsize:
```

```
        raise "binary value larger than the expected size"
```

```
    while len(binval) < bitsize:
```

```
        binval = "0"+binval
```

```
    return binval
```

```
def nsplit(s, n):
```

```
    return [s[k:k+n] for k in range(0, len(s), n)]
```

```
ENCRYPT=1
```

```
DECRYPT=0
```

```
class des():
```

```
    def __init__(self):
```

```
        self.password = None
```

```
        self.text = None
```

```
        self.keys = list()
```

```
    def run(self, key, text, action=ENCRYPT, padding=False):
```

```
        if len(key) < 8:
```

```
            raise "Key Should be 8 bytes long"
```

```
        elif len(key) > 8:
```

```
            key = key[:8]
```

```
        self.password = key
```

```
        self.text = text
```

```
        if padding and action==ENCRYPT:
```

```
            self.addPadding()
```

```
        elif len(self.text) % 8 != 0:
```



```

        raise "Data size should be multiple of 8"

    self.generatekeys()
    text_blocks = nsplit(self.text, 8)
    result = list()
    for block in text_blocks:
        block = toBitArray(block)
        block = self.permut(block,PI)
        g, d = nsplit(block, 32)
        tmp = None
        for i in range(16):
            d_e = self.expand(d, E)
            if action == ENCRYPT:
                tmp = self.xor(self.keys[i], d_e)
            else:
                tmp = self.xor(self.keys[15-i], d_e)
            tmp = self.substitute(tmp)
            tmp = self.permut(tmp, P)
            tmp = self.xor(g, tmp)
            g = d
            d = tmp
        result += self.permut(d+g, IPInv)
    final_res = bitArrToString(result)
    if padding and action==DECRYPT:
        return self.removePadding(final_res)
    else:
        return final_res

def substitute(self, d_e):
    subblocks = nsplit(d_e, 6)
    result = list()
    for i in range(len(subblocks)):
        block = subblocks[i]
        row = int(str(block[0])+str(block[5]),2)
        column = int("".join([str(x) for x in block[1:][::-1]]),2)
        val = S_BOX[i][row][column]
        bin = toBinary(val, 4)
        result += [int(x) for x in bin]
    return result

def permut(self, block, table):
    return [block[x-1] for x in table]

def expand(self, block, table):
    return [block[x-1] for x in table]

def xor(self, t1, t2):
    return [x^y for x,y in zip(t1,t2)]

def generatekeys(self):
    self.keys = []
    key = toBitArray(self.password)

```

```

key = self.permut(key, CP_1)
g, d = nsplit(key, 28)
for i in range(16):
    g, d = self.shift(g, d, SHIFT[i])
    tmp = g + d
    self.keys.append(self.permut(tmp, CP_2))

def shift(self, g, d, n):
    return g[n:] + g[:n], d[n:] + d[:n]

def addPadding(self):
    pad_len = 8 - (len(self.text) % 8)
    self.text += pad_len * chr(pad_len)

def removePadding(self, data):
    pad_len = ord(data[-1])
    return data[:-pad_len]

def encrypt(self, key, text, padding=False):
    return self.run(key, text, ENCRYPT, padding)

def decrypt(self, key, text, padding=False):
    return self.run(key, text, DECRYPT, padding)

```

II) client.py

```

import socket
import sys
import DES

HOST = "127.0.0.1"
PORT = 9090
key = ""
desObj = DES.des()

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    clientSocket.connect((HOST, PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST, PORT))

def handleConnection():
    while True:
        serverCipher = clientSocket.recv(1024).decode("utf-8").strip()
        serverMessage = desObj.decrypt(key, serverCipher)
        print("[SERVER]: {}".format(serverMessage))
        print("[SERVER RAW]: {}".format(serverCipher.encode("utf-8").hex()))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = desObj.encrypt(key, clientMessage) + '\n'
        clientSocket.send(clientCipher.encode("utf-8"))
        if clientMessage.lower() == "exit":
            break

```

```

if __name__ == "__main__":
    key = input("[CLIENT]> Enter key:")
    if len(sys.argv) == 2:
        HOST = sys.argv[1]
    connectServer()
    handleConnection()
    clientSocket.close()
    print("[INFO]: Connection to server closed and exiting...")

III) server.py
import socket
import DES
HOST = "127.0.0.1"
PORT = 9090
key = ""
desObj = DES.des()

def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))

def handleConnections() -> None:
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from {}:{}".format(clientAddress[0],clientAddress[1]))
    try:
        while True:
            userInput = input("[SERVER]> Enter message:")
            cipherInput = desObj.encrypt(key,userInput) + "\n"
            clientSocket.send(cipherInput.encode("utf-8"))
            clientCipher = clientSocket.recv(1024).decode("utf-8").strip()
            clientMessage = desObj.decrypt(key,clientCipher)
            print("[CLIENT RAW]: {}".format(clientCipher.encode("utf-8").hex()))
            print("[CLIENT]: {}".format(clientMessage))
            if clientMessage.lower() == "exit":
                break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")
    finally:
        print("[INFO]: Closing server")
        clientSocket.close()
        serverSocket.close()

if __name__ == "__main__":
    key = input("[SERVER]> Enter key:")
    initServer()
    handleConnections()

```

OUTPUT:

I) Server Side:

```
ol/Semester VII/Cryptography and Security /P/8. DES 11:55:06
python server.py
[SERVER]> Enter key:xsecretx
[SERVER]: Server started at 127.0.0.1:9090 and listening
[INFO]: Connection established from 127.0.0.1:60612
[SERVER]> Enter message:hellotherexxxxxx
[CLIENT RAW]: c2bc5b62c2b7c3b3c289c2ac52
[CLIENT]: hiyaback
[SERVER]> Enter message:^C[INFO]: Keyboard interrupt recieved
[INFO]: Closing server
```

II) Client Side:

```
Cryptography and Security /P/8. DES X INT 23s 11:36:08
python client.py
[CLIENT]> Enter key:xsecretx
[INFO]: Connected to server at 127.0.0.1:9090
[SERVER]: hellotherexxxxxx
[SERVER RAW]: 140b2954c3b5c28bc3a53b5bc3bdc2ab441802c2b5c2a9
[CLIENT]> Enter Message:hiyaback
```

RESULT:

Hence, Symmetric encryption algorithm DES was implemented and tested with client server program successfully.

AIM:

To implement symmetric encryption algorithm DES, and encrypt plain text and decrypt cipher text using them.

ALGORITHM:**Key Expansion:**

- Generate a key schedule from the original encryption key. The key schedule contains round keys for each round of encryption and decryption.

Encryption:

Initial Round:

- XOR the input data with the first round key.
- Main Rounds (10/12/14 rounds depending on key size):

For each round:

- Substitute Bytes: Apply the S-Box substitution to each byte in the state.
- Shift Rows: Perform a cyclic shift on the rows of the state.
- Mix Columns: Mix the columns of the state to provide diffusion.
- Add Round Key: XOR the state with the round key from the key schedule.

Final Round: Substitute Bytes, Shift Rows, Add Round Key.

Output: The final state after the last round is the ciphertext.

Decryption:

Initial Round:

- Add Round Key (use the last round key from the key schedule).
- Main Rounds (10/12/14 rounds depending on key size):

For each round:

- Inverse Shift Rows: Perform the inverse cyclic shift on the rows of the state.
- Inverse Substitute Bytes: Apply the inverse S-Box substitution to each byte in the state.
- Add Round Key: XOR the state with the round key from the key schedule.

Final Round: Inverse Shift Rows, Inverse Substitute Bytes, Add Round Key (use the first round key from the key schedule)

Output: The final state after the last round is the plaintext.

SOURCE CODE:**I) AES.py**

```
# S-Box values for substitution layer
```

```
Sbox = ( ... )
```

```
# Inverse Substitution layer values
```

```
InvSbox = ( ... )
```

```
xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)
```

```
# Rcon values in Key schedule
```

```
Rcon = ( ... )
```

```
def toMatrix(text):
```

```
    matrix = []
```

```
    for i in range(16):
```

```
        byte = (text >> (8 * (15 - i))) & 0xFF
```

```
        if i % 4 == 0:
```

```

        matrix.append([byte])
    else:
        matrix[i // 4].append(byte)
    return matrix

def matrixToText(matrix):
    text = 0
    for i in range(4):
        for j in range(4):
            text |= (matrix[i][j] << (120 - 8 * (4 * i + j)))
    return text

def intToStr(val: int) -> str:
    return bytes.fromhex(str(hex(val)[2:])).decode()

def strToInt(txt: str) -> int:
    return int(txt.encode().hex(),16)

class AES:
    def __init__(self, master_key):
        self.setKey(master_key)

    def setKey(self, master_key):
        self.round_keys = toMatrix(master_key)
        for i in range(4, 4 * 11):
            self.round_keys.append([])
            if i % 4 == 0:
                byte = self.round_keys[i - 4][0] \
                    ^ Sbox[self.round_keys[i - 1][1]] \
                    ^ Rcon[i // 4]
                self.round_keys[i].append(byte)
                for j in range(1, 4):
                    byte = self.round_keys[i - 4][j] \
                        ^ Sbox[self.round_keys[i - 1][(j + 1) % 4]]
                    self.round_keys[i].append(byte)
            else:
                for j in range(4):
                    byte = self.round_keys[i - 4][j] \
                        ^ self.round_keys[i - 1][j]
                    self.round_keys[i].append(byte)

    def encrypt(self, plaintext):
        self.plain_state = toMatrix(plaintext)
        self.addRoundKey(self.plain_state, self.round_keys[:4])
        for i in range(1, 10):
            self.roundEncrypt(self.plain_state, self.round_keys[4 * i : 4 * (i + 1)])
        self.subBytes(self.plain_state)
        self.shiftRows(self.plain_state)
        self.addRoundKey(self.plain_state, self.round_keys[40:])
        return matrixToText(self.plain_state)

```

```

def decrypt(self, ciphertext):
    self.cipher_state = toMatrix(ciphertext)
    self.addRoundKey(self.cipher_state, self.round_keys[40:])
    self.invShiftRows(self.cipher_state)
    self.invSubBytes(self.cipher_state)
    for i in range(9, 0, -1):
        self.roundDecrypt(self.cipher_state, self.round_keys[4 * i : 4 * (i + 1)])
    self.addRoundKey(self.cipher_state, self.round_keys[:4])
    return matrixToText(self.cipher_state)

def addRoundKey(self, s, k):
    for i in range(4):
        for j in range(4):
            s[i][j] ^= k[i][j]

def roundEncrypt(self, state_matrix, key_matrix):
    self.subBytes(state_matrix)
    self.shiftRows(state_matrix)
    self.mixColumns(state_matrix)
    self.addRoundKey(state_matrix, key_matrix)

def roundDecrypt(self, state_matrix, key_matrix):
    self.addRoundKey(state_matrix, key_matrix)
    self.invMixColumns(state_matrix)
    self.invShiftRows(state_matrix)
    self.invSubBytes(state_matrix)

def subBytes(self, s):
    for i in range(4):
        for j in range(4):
            s[i][j] = Sbox[s[i][j]]

def invSubBytes(self, s):
    for i in range(4):
        for j in range(4):
            s[i][j] = InvSbox[s[i][j]]

def shiftRows(self, s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]

def invShiftRows(self, s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]

def mixSingleColumn(self, a):
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])

```

```

a[2] ^= t ^ xtime(a[2] ^ a[3])
a[3] ^= t ^ xtime(a[3] ^ u)

```

```

def mixColumns(self, s):
    for i in range(4):
        self.mixSingleColumn(s[i])

```

```

def invMixColumns(self, s):
    for i in range(4):
        u = xtime(xtime(s[i][0] ^ s[i][2]))
        v = xtime(xtime(s[i][1] ^ s[i][3]))
        s[i][0] ^= u
        s[i][1] ^= v
        s[i][2] ^= u
        s[i][3] ^= v
    self.mixColumns(s)

```

II) client.py

```

import socket
import sys
import AES

```

```

HOST = "127.0.0.1"
PORT = 9090
key = ""
aesObj = object()

```

```

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    clientSocket.connect((HOST, PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST, PORT))

```

```

def handleConnection():
    while True:
        serverCipher = int(clientSocket.recv(1024).decode("utf-8").strip())
        serverMessage = aesObj.decrypt(serverCipher)
        print("[SERVER]: {}".format(AES.intToStr(serverMessage)))
        print("[SERVER RAW]: {}".format(hex(serverCipher)))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = str(aesObj.encrypt(clientMessage)) + '\n'
        clientSocket.send(clientCipher.encode("utf-8"))
        if clientMessage.lower() == "exit":
            break

```

```

if __name__ == "__main__":
    key = input("[CLIENT]> Enter key:")
    aesObj = AES.AES(int(key.encode('utf-8').hex(), 16))
    if len(sys.argv) == 2:
        HOST = sys.argv[1]
    connectServer()
    handleConnection()

```



```
clientSocket.close()
print("[INFO]: Connection to server closed and exiting...")
```

III) server.py

```
import socket
import AES
```

```
HOST = "127.0.0.1"
PORT = 9090
key = ""
aesObj = object()
```

```
def initServer() -> None:
```

```
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))
```

```
def handleConnections() -> None:
```

```
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from {}:{}".format(clientAddress[0],clientAddress[1]))
    try:
        while True:
            userInput = input("[SERVER]> Enter message:")
            cipherInput = str(aesObj.encrypt(AES.strToInt(userInput))) + "\n"
            clientSocket.send(cipherInput.encode("utf-8"))
            clientCipher = int(clientSocket.recv(1024).decode("utf-8").strip())
            clientMessage = aesObj.decrypt(clientCipher)
            print("[CLIENT RAW]: {}".format(hex(clientCipher)))
            print("[CLIENT]: {}".format(AES.intToStr(clientMessage)))
            if clientMessage.lower() == "exit":
                break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")
    finally:
        print("[INFO]: Closing server")
        clientSocket.close()
        serverSocket.close()
```

```
if __name__ == "__main__":
```

```
    key = input("[SERVER]> Enter key:")
    aesObj = AES.AES(int(key.encode('utf-8').hex(),16))
    initServer()
    handleConnections()
```

OUTPUT:

I) Server Side:

```
ter VII/Cryptography and Security /P/9. AES 24s 14:04:12
python server.py
[SERVER]> Enter key:qwertyuiopasdfg
[SERVER]: Server started at 127.0.0.1:9090 and listening
[INFO]: Connection established from 127.0.0.1:60214
[SERVER]> Enter message:hellotherexxxxxx
[CLIENT RAW]: 0x7abb9814649284011e029ef7122ba6f5
[CLIENT]: hellobackfriendx
[SERVER]> Enter message:
```

II) Client Side:

```
ter VII/Cryptography and Security /P/9. AES 15s 14:04:12
python client.py
[CLIENT]> Enter key:qwertyuiopasdfg
[INFO]: Connected to server at 127.0.0.1:9090
[SERVER]: hellotherexxxxxx
[SERVER RAW]: 0x69e6f080d2fd06a8c2284e6a5d02dd3f
[CLIENT]> Enter Message:hellobackfriendx
```

RESULT:

Hence, Symmetric encryption algorithm AES were implemented and tested with client server program successfully.

AIM:

To study and implement Asymmetric Encryption algorithms such as RSA and encrypt and decrypt plaintext using it.

Algorithm:**Key Generation:**

- Choose two large prime numbers, p and q .
- Compute the modulus, $n = p * q$.
- Compute the totient, $\phi(n) = (p - 1) * (q - 1)$.
- Choose a public exponent, typically e , such that $1 < e < \phi(n)$, and $\gcd(e, \phi(n)) = 1$.
- Compute the private exponent, d , as the modular multiplicative inverse of e modulo $\phi(n)$.

Encryption:

- Represent the plaintext message as an integer, M .
- Compute the ciphertext, C , as $C \equiv M^e \pmod{n}$

Decryption:

- Given the ciphertext, C , compute the plaintext, M , as $M \equiv C^d \pmod{n}$.

Source Code:**I) RSA.py**

```

from math import gcd
from random import randint
primeNumbers = [ 53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149]
publicKey = 0
privateKey = 0
conPublicKey = 0
N = 0
conN = 0
def getPrivateKey() -> None:
    return privateKey

def getRandomPrime() -> int:
    return primeNumbers[randint(0,len(primeNumbers)-1)]

def setConPublicKey(key: int, N:int) -> None:
    global conPublicKey,conN
    conPublicKey = key
    conN = N

def getPublicKey() -> (int,int):
    return (publicKey,N)

def generateKeys() -> None:
    global publicKey,privateKey,N
    prime1 = getRandomPrime()
    prime2 = getRandomPrime()
    N = prime1 * prime2
    chi = (prime1 - 1) * (prime2 - 1)

```

```

while True:
    publicKey = randint(0,chi)
    if gcd(publicKey,chi) == 1:
        break
privateKey = pow(publicKey,-1,chi)

def encrypt(plain: str) -> list:
    cipherList = []
    for char in plain:
        cipherList.append((ord(char) ** conPublicKey) % conN)
    return cipherList

def decrypt(cipher: []) -> str:
    plain = ""
    for val in cipher:
        plain += chr((val ** privateKey) % N)
    return plain

```

II) client.py

```

import socket
import pickle
import RSA

HOST = "127.0.0.1"
PORT = 9090
key = []

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    clientSocket.connect((HOST,PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST,PORT))
    RSA.generateKeys()
    print("[INFO]: Keys generated. Public Key: {}, Private: {}".format(RSA.getPublicKey(),RSA.getPrivateKey()))

def handleConnection():
    serverPubKey = int(clientSocket.recv(1024).decode("utf-8").strip())
    serverN = int(clientSocket.recv(1024).decode("utf-8").strip())
    RSA.setConPublicKey(serverPubKey,serverN)
    publicKey, N = RSA.getPublicKey()
    clientSocket.send("{}\n".format(publicKey).encode("utf-8"))
    clientSocket.send("{}\n".format(N).encode("utf-8"))
    while True:
        serverCipher = clientSocket.recv(1024)
        print("[SERVER RAW]: {}".format(serverCipher))
        serverMessage = RSA.decrypt(pickle.loads(serverCipher.strip()))
        print("[SERVER]: {}".format(serverMessage))
        clientMessage = input("[CLIENT]> Enter Message:")
        clientCipher = RSA.encrypt(clientMessage)
        clientSocket.send(pickle.dumps(clientCipher) + '\n'.encode("utf-8"))

```

```

        if clientMessage.lower() == "exit":
            break

if __name__ == "__main__":
    connectServer()
    handleConnection()
    clientSocket.close()
    print("[INFO]: Connection to server closed and exiting...")
III) server.py
import socket
import RSA
import pickle

HOST = "127.0.0.1"
PORT = 9090
key = []

def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))
    RSA.generateKeys()
    print("[SERVER]: Keys generated. Public Key: {}, Private:
    {}".format(RSA.getPublicKey(),RSA.getPrivateKey()))

def handleConnections() -> None:
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from {}:{}".format(clientAddress[0],clientAddress[1]))
    publicKey, N = RSA.getPublicKey()
    clientSocket.send("{}\n".format(publicKey).encode("utf-8"))
    clientSocket.send("{}\n".format(N).encode("utf-8"))
    clientPubKey = int(clientSocket.recv(1024).decode("utf-8").strip())
    clientN = int(clientSocket.recv(1024).decode("utf-8").strip())
    RSA.setConPublicKey(clientPubKey,clientN)
    print("[INFO]: Key exchanges successful")
    try:
        while True:
            userInput = input("[SERVER]> Enter message:")
            cipherInput = RSA.encrypt(userInput)
            cipherBytes = pickle.dumps(cipherInput)
            clientSocket.send(cipherBytes + '\n'.encode("utf-8"))
            clientCipher = clientSocket.recv(1024)
            print("[CLIENT RAW]: {}".format(clientCipher))
            clientMessage = RSA.decrypt(pickle.loads(clientCipher.strip()))
            print("[CLIENT]: {}".format(clientMessage))
            if clientMessage.lower() == "exit":
                break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")

```

```
finally:
    print("[INFO]: Closing server")
    clientSocket.close()
    serverSocket.close()
```

```
if __name__ == "__main__":
    initServer()
    handleConnections()
```

Output:

I) Server Side:

```

ter VII/Cryptography and Security /P/10. RSA 5s 14:21:31
python server.py
[SERVER]: Server started at 127.0.0.1:9090 and listening
[SERVER]: Keys generated. Public Key: (13365, 15481), Private: 669
[INFO]: Connection established from 127.0.0.1:41852
[INFO]: Key exchanges successful
[SERVER]> Enter message:Hellothere
[CLIENT RAW]: b'\x80\x04\x95\x1d\x00\x00\x00\x00\x00\x00\x00\x00\x94(M
\x8eM0\x03M\xc5M\xd6\x16MM\x0cM\xd6\x16M\xa5$M\x90\x0fe.\n'
[CLIENT]: Hiyaback
[SERVER]> Enter message:Goodday
[CLIENT RAW]: b'\x80\x04\x95\x11\x00\x00\x00\x00\x00\x00\x00\x94(M
\x8f9M'\x0eM0\x03ML3e.\n'
[CLIENT]: exit
[INFO]: Closing server

```

II) Client Side:

```
[L/Semester VII/Cryptography and Security /P/10. RSA 14:21:31]
python client.py
[INFO]: Connected to server at 127.0.0.1:9090
[INFO]: Keys generated. Public Key: (4365, 10057), Private: 709
[SERVER RAW]: b'\x80\x04\x95#\x00\x00\x00\x00\x00\x00\x00]\x94(Mj\x
16M\xc5\x13M\xeexce\x06MS\tM\xb M\xdc\x05M\xc5\x13M\x13\x16M
xc5\x13e.\n'
[SERVER]: Hellothere
[CLIENT]> Enter Message:Hiyaback
[SERVER RAW]: b'\x80\x04\x95\x1a\x00\x00\x00\x00\x00\x00\x00]\x94(M
\x9f\x16MS\tMS\tM8\x13M\x14\x07M\xe8$e.\n'
[SERVER]: Goodday
[CLIENT]> Enter Message:exit
[INFO]: Connection to server closed and exiting...
```

RESULT:

Hence, Asymmetric encryption algorithm RSA was implemented to enforce public key cryptosystem over client-server program.

AIM:

To study and implement Diffie Hellman Key exchange algorithm and generate a common key with a public cryptosystem.

Algorithm:

- Agree on public parameters: a prime number "p" and a base "g."
- Both parties choose private keys, "a" and "b."
- Compute and exchange public keys:
 - Party A sends " $A = g^a \text{ mod } p$ " to Party B.
 - Party B sends " $B = g^b \text{ mod } p$ " to Party A.
- Calculate the shared secret:
 - Party A computes " $s = B^a \text{ mod } p$."
 - Party B computes " $s = A^b \text{ mod } p$."
- The shared secret "s" is now established and can be used for secure communication.

Source Code:**I) diffie.py**

```
from random import randint
```

```
alpha = 151
```

```
prime = 199
```

```
def pickPrivateKey() -> int:  
    return randint(1,199)
```

II) client.py

```
import socket
```

```
import diffie
```

```
HOST = "127.0.0.1"
```

```
PORT = 8080
```

```
if __name__ == "__main__":  
    clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)  
    clientSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)  
    clientSocket.connect((HOST,PORT))  
    print("[CLIENT]: Connect to server successfully")  
    serverPubPart = int(clientSocket.recv(1024).decode("utf-8").strip())  
    privateKey = diffie.pickPrivateKey()  
    publicPart = diffie.alpha ** privateKey % diffie.prime  
    clientSocket.send("{}\n".format(publicPart).encode("utf-8"))  
    genKey = serverPubPart ** privateKey % diffie.prime  
    print("[CLIENT]: Generated Key: {}".format(genKey))  
    print("[INFO]: Key generated successfully. Closing connections")  
    clientSocket.close()
```

III) server.py

```
import socket
import diffie

HOST = "127.0.0.1"
PORT = 8080

if __name__ == "__main__":
    serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    serverSocket.bind((HOST, PORT))
    serverSocket.listen()
    print("[SERVER]: Start server and listening")
    clientSocket, clientAddress = serverSocket.accept()
    print("[SERVER]: Connection Established from {}".format(clientAddress))
    privateKey = diffie.pickPrivateKey()
    publicPart = diffie.alpha ** privateKey % diffie.prime
    clientSocket.send("{}\n".format(publicPart).encode("utf-8"))
    clientPubPart = int(clientSocket.recv(1024).decode("utf-8").strip())
    genKey = clientPubPart ** privateKey % diffie.prime
    print("[INFO]: The generated key is: {}".format(genKey))
    print("[INFO]: Key generated successfully. Closing server")
    clientSocket.close()
    serverSocket.close()
```

Output:

I) Server Side:

```
..VII/Cryptography and Security /P/11. Diffie Hellman 14:31:56
└─> python server.py
[SERVER]: Start server and listening
[SERVER]: Connection Established from ('127.0.0.1', 44832)
[INFO]: The generated key is: 98
[INFO]: Key generated successfully. Closing server
```

II) Client Side:

```
..VII/Cryptography and Security /P/11. Diffie Hellman 14:32:01
└─> python client.py
[CLIENT]: Connect to server successfully
[CLIENT]: Generated Key: 98
[INFO]: Key generated successfully. Closing connections
```

RESULT:

Hence, Diffie Hellam algorithm was implemented and a common key was successfully generated between a client and server.

AIM:

To study and implement Asymmetric Encryption algorithms such as RSA and encrypt and decrypt plaintext using it.

ALGORITHM:**Key Setup:**

- Select a large prime number, "p," and a primitive root modulo "p," denoted as "g."
- Each user generates a private key, "x," which is a random integer such that $1 < x < p - 1$.
- Compute the public key, "y," as $y = g^x \text{ mod } p$.

Encryption:

- To encrypt a plaintext message, "M," the sender selects a random integer, "k," such that $1 < k < p - 1$, and computes the ciphertext as follows:
 - Compute "c1" as $c1 = g^k \text{ mod } p$.
 - Compute "c2" as $c2 = M * (y^k) \text{ mod } p$.

Output: The ciphertext consists of (c1, c2) and can be sent to the recipient.

Decryption:

To decrypt the ciphertext (c1, c2), the recipient uses their private key, "x," to compute the plaintext as follows:

- Compute "s" as $s = (c1^x)^{-1} \text{ mod } p$.
- Retrieve the plaintext "M" as $M = c2 * s \text{ mod } p$.

SOURCE CODE:**I) elgamal.py**

```
from math import gcd
```

```
from random import randint
```

```
alpha = 151
```

```
prime = 199
```

```
conPublicKey = 0
```

```
selfPrivateKey = 0
```

```
selfPublickey = 0
```

```
def pickPrivateKey() -> int:
```

```
    return randint(1,prime)
```

```
def genKeys() -> None:
```

```
    global selfPrivateKey,selfPublickey
```

```
    if selfPrivateKey == 0:
```

```
        selfPrivateKey = pickPrivateKey()
```

```
        selfPublickey = (alpha ** selfPrivateKey) % prime
```

```
    else:
```

```
        return
```

```
def getPublicKey() -> int:
```

```
    return selfPublickey
```

```

def setConPublicKey(key: int) -> None:
    global conPublicKey
    conPublicKey = key

def encrypt(plain: str) -> ():
    randK = randint(1,prime)
    cipher = []
    K = (conPublicKey ** randK) % prime
    c1 = (alpha ** randK) % prime
    for char in plain:
        cipher.append((K * ord(char)) % prime)
    return (c1,cipher)

def decrypt(c1:int, cipher: []) -> str:
    plain = ""
    K = (c1 ** selfPrivateKey) % prime
    KInv = pow(K,-1,prime)
    for val in cipher:
        plain += chr((val * KInv) % prime)
    return plain

```

II) client.py

```

import socket
import pickle
import elgamal

HOST = "127.0.0.1"
PORT = 9090

def connectServer():
    global clientSocket
    clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    clientSocket.connect((HOST,PORT))
    print("[INFO]: Connected to server at {}:{}".format(HOST,PORT))
    elgamal.genKeys()
    print("[INFO]: Keys generated. Public Key: {}".format(elgamal.getPublicKey()))

def handleConnection():
    serverPubKey = int(clientSocket.recv(1024).decode("utf-8").strip())
    elgamal.setConPublicKey(serverPubKey)
    publicKey = elgamal.getPublicKey()
    clientSocket.send("{}\n".format(publicKey).encode("utf-8"))
    while True:
        serverC1 = int(clientSocket.recv(1024).decode("utf-8").strip())
        serverCipher = clientSocket.recv(1024)
        print("[SERVER RAW]: {}".format(serverCipher))
        serverMessage = elgamal.decrypt(serverC1,pickle.loads(serverCipher.strip()))
        print("[SERVER]: {}".format(serverMessage))
        clientMessage = input("[CLIENT]> Enter Message:")
        c1, clientCipher = elgamal.encrypt(clientMessage)
        clientSocket.send("{}\n".format(c1).encode("utf-8"))

```

```

clientSocket.send(pickle.dumps(clientCipher) + '\n'.encode("utf-8"))
if clientMessage.lower() == "exit":
    break

```

```

if __name__ == "__main__":
    connectServer()
    handleConnection()
    clientSocket.close()
    print("[INFO]: Connection to server closed and exiting...")

```

III) server.py

```

import socket
import elgamal
import pickle

```

```

HOST = "127.0.0.1"
PORT = 9090

```

```

def initServer() -> None:
    global serverSocket
    serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    serverSocket.bind((HOST,PORT))
    serverSocket.listen()
    print("[SERVER]: Server started at {}:{} and listening".format(HOST,PORT))
    elgamal.genKeys()
    print("[SERVER]: Keys generated. Public Key: {}".format(elgamal.getPublicKey()))

```

```

def handleConnections() -> None:
    clientSocket, clientAddress = serverSocket.accept()
    print("[INFO]: Connection established from {}:{}".format(clientAddress[0],clientAddress[1]))
    publicKey = elgamal.getPublicKey()
    clientSocket.send("{}\n".format(publicKey).encode("utf-8"))
    clientPubKey = int(clientSocket.recv(1024).decode("utf-8").strip())
    elgamal.setConPublicKey(clientPubKey)
    print("[INFO]: Key exchanges successful")
    try:
        while True:
            userInput = input("[SERVER]> Enter message:")
            c1, cipherInput = elgamal.encrypt(userInput)
            cipherBytes = pickle.dumps(cipherInput)
            clientSocket.send("{}\n".format(c1).encode("utf-8"))
            clientSocket.send(cipherBytes + '\n'.encode("utf-8"))
            clientC1 = int(clientSocket.recv(1024).decode("utf-8").strip())
            clientCipher = clientSocket.recv(1024)
            print("[CLIENT RAW]: {}".format(clientCipher))
            clientMessage = elgamal.decrypt(clientC1,pickle.loads(clientCipher.strip()))
            print("[CLIENT]: {}".format(clientMessage))
            if clientMessage.lower() == "exit":
                break
    except KeyboardInterrupt:
        print("[INFO]: Keyboard interrupt recieved")

```

```
print("[INFO]: Closing server")
clientSocket.close()
serverSocket.close()
```

```
if __name__ == "__main__":
    initServer()
    handleConnections()
```

I) Server Side:

```
[~]# python server.py  
[SERVER]: Server started at 127.0.0.1:9090 and listening  
[SERVER]: Keys generated. Public Key: 66  
[INFO]: Connection established from 127.0.0.1:40658  
[INFO]: Key exchanges successful  
[SERVER]> Enter message:hellothere  
[CLIENT RAW]: b'\x80\x04\x95\x17\x00\x00\x00\x00\x00\x00\x94(K\n\\\n8cKqk\xb0K\b0K\x04KVVKM_K\xa7e.\n'  
[CLIENT]: helloback  
[SERVER]> Enter message:goodday  
[CLIENT RAW]: b'\x80\x04\x95\r\x00\x00\x00\x00\x00\x00\x94(KMK\n\\xbeK\x11K3e.\n'  
[CLIENT]: exit  
[INFO]: Closing server
```

```

C:\Users\user> python cryptography /P/12. ElGamal PKC
[INFO]: Connected to server at 127.0.0.1:9090
[INFO]: Keys generated. Public Key: 86
[SERVER RAW]: b'\x80\x04\x95\x19\x00\x00\x00\x00\x00\x00\x94(K
^K\x4KCKCK\xc4K\rK^K\x4K-K\x4e.\n'
[SERVER]: hellothere
[CLIENT]> Enter Message:helloback
[SERVER RAW]: b'\x80\x04\x95\x13\x00\x00\x00\x00\x00\x00\x94(K
\x97K\x9bK\x9bK2K2K\x94K\xa0e.\n'
[SERVER]: goodday
[CLIENT]> Enter Message:exit
[INFO]: Connection to server closed and exiting...

```

Hence, ElGamal Cryptosystem was implemented to enforce asymmetric encryption over client-server program.

AIM:

To study and implement hashing algorithms such as Secure Hashing Algorithm(sha512) and Message Digest v5 (md5).

I) MD5:**Algorithm:**

Padding:

- Begin with the original message, which is a sequence of bits.
- Append a '1' bit to the end of the message.
- Append enough '0' bits to make the total length of the message a multiple of 512 bits (64 bytes).

Length Appending:

- Append a 64-bit representation of the original message length (in bits) to the end of the padded message.

Initialize Variables:

- Initialize a 128-bit buffer to store the intermediate hash values.
- Initialize four 32-bit words (A, B, C, D) with fixed constants.

Process Message in 512-bit Blocks: Break the message into 512-bit blocks.

For each block:

- Create a 64-entry table using bitwise functions, and apply these functions in multiple rounds to shuffle bits and modify the buffer's values.
- Update the buffer's values based on the current 512-bit block.

Final Result: The final 128-bit buffer contains the MD5 hash value of the original message.

Source Code:

```
import math
# Constants for rotation in each round
rotate_by = [ ... ]
constants = [int(abs(math.sin(i+1)) * 4294967296) & 0xFFFFFFFF for i in range(64)]
init_MDBuffer = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]

def pad(msg):
    msg_len_in_bits = (8*len(msg)) & 0xfffffffffffff
    msg.append(0x80)
    while len(msg)%64 != 56:
        msg.append(0)
    msg += msg_len_in_bits.to_bytes(8, byteorder='little')
    return msg

def leftRotate(x, amount):
    x &= 0xFFFFFFFF
    return (x << amount | x >> (32-amount)) & 0xFFFFFFFF

def processMessage(msg):
    init_temp = init_MDBuffer[:]
    for offset in range(0, len(msg), 64):
        A, B, C, D = init_temp
```

```

block = msg[offset : offset+64]
for i in range(64):
    if i < 16:
        func = lambda b, c, d: (b & c) | (~b & d)
        index_func = lambda i: i
    elif i >= 16 and i < 32:
        func = lambda b, c, d: (d & b) | (~d & c)
        index_func = lambda i: (5*i + 1)%16
    elif i >= 32 and i < 48:
        func = lambda b, c, d: b ^ c ^ d
        index_func = lambda i: (3*i + 5)%16
    elif i >= 48 and i < 64:
        func = lambda b, c, d: c ^ (b | ~d)
        index_func = lambda i: (7*i)%16
    F = func(B, C, D)
    G = index_func(i)
    to_rotate = A + F + constants[i] + int.from_bytes(block[4*G : 4*G + 4],
byteorder='little')
    newB = (B + leftRotate(to_rotate, rotate_by[i])) & 0xFFFFFFFF
    A, B, C, D = D, newB, B, C
for i, val in enumerate([A, B, C, D]):
    init_temp[i] += val
    init_temp[i] &= 0xFFFFFFFF
return sum(buffer_content<<(32*i) for i, buffer_content in enumerate(init_temp))

def mdToHex(digest):
    raw = digest.to_bytes(16, byteorder='little')
    return '{:032x}'.format(int.from_bytes(raw, byteorder='big'))

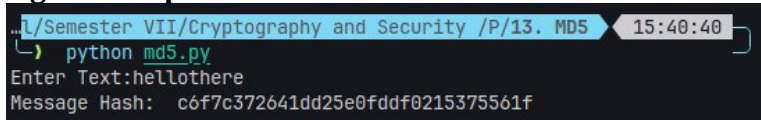
def md5(msg):
    msg = bytearray(msg, 'ascii')
    msg = pad(msg)
    processed_msg = processMessage(msg)
    message_hash = mdToHex(processed_msg)
    print("Message Hash: ", message_hash)

if __name__ == '__main__':
    message = input("Enter Text:")
    md5(message)

```

Output:

I) Program Output:

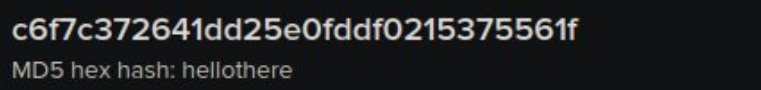


```

L/Semester VII/Cryptography and Security /P/13. MD5 15:40:40
python md5.py
Enter Text:hellothere
Message Hash: c6f7c372641dd25e0fddf0215375561f

```

II) Verification:



```

c6f7c372641dd25e0fddf0215375561f
MD5 hex hash: hellothere

```

II) SHA512:

Algorithm:

- Message Preprocessing: Pad the message and append its length to make it a multiple of 1024 bits.
- Initialize Constants and Initial Hash Values: Set predefined constants and initial hash values.
- Message Block Processing: Divide the padded message into 1024-bit blocks.
- Main Loop: For each block, perform 80 rounds of operations on temporary variables, updating the hash values.
- Final Hash Value: Concatenate the eight 64-bit hash values to form the SHA-512 hash.

Source Code:

```
import binascii
import struct
# Initial Buffer values a,b,c,d,e,f,g,h
initial_hash = ( ... )
# Round constants for all 80 rounds
round_constants = ( ... )

def rightRotate(n: int, bits: int) -> int:
    return (n >> bits) | (n << (64 - bits)) & 0xFFFFFFFFFFFFFFFF

def sha512(message: str) -> str:
    if type(message) is not str:
        raise TypeError('Given message should be a string.')
    message_array = bytearray(message, encoding='utf-8')
    mdi = len(message_array) % 128
    padding_len = 119 - mdi if mdi < 112 else 247 - mdi
    ending = struct.pack('!Q', len(message_array) << 3)
    message_array.append(0x80)
    message_array.extend([0] * padding_len)
    message_array.extend(bytearray(ending))
    sha512_hash = list(initial_hash)
    for chunk_start in range(0, len(message_array), 128):
        chunk = message_array[chunk_start:chunk_start + 128]
        w = [0] * 80
        w[0:16] = struct.unpack('!16Q', chunk)
        for i in range(16, 80):
            s0 = (
                rightRotate(w[i - 15], 1) ^
                rightRotate(w[i - 15], 8) ^
                (w[i - 15] >> 7)
            )
            s1 = (
                rightRotate(w[i - 2], 19) ^
                rightRotate(w[i - 2], 61) ^
                (w[i - 2] >> 6)
            )
            w[i] = (w[i - 16] + s0 + w[i - 7] + s1) & 0xFFFFFFFFFFFFFFFF
        a, b, c, d, e, f, g, h = sha512_hash
    for i in range(80):
        sum1 = (
            rightRotate(e, 14) ^
```

```

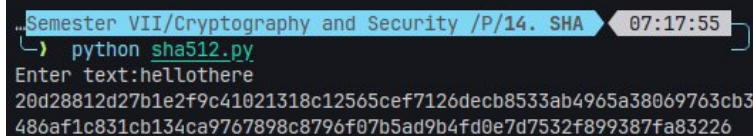
        rightRotate(e, 18) ^
        rightRotate(e, 41)
    )
    ch = (e & f) ^ (~e & g)
    temp1 = h + sum1 + ch + round_constants[i] + w[i]
    sum0 = (
        rightRotate(a, 28) ^
        rightRotate(a, 34) ^
        rightRotate(a, 39)
    )
    maj = (a & b) ^ (a & c) ^ (b & c)
    temp2 = sum0 + maj
    h = g
    g = f
    f = e
    e = (d + temp1) & 0xFFFFFFFFFFFFFFFF
    d = c
    c = b
    b = a
    a = (temp1 + temp2) & 0xFFFFFFFFFFFFFFFF
    sha512_hash = [
        (x + y) & 0xFFFFFFFFFFFFFFFF
        for x, y in zip(sha512_hash, (a, b, c, d, e, f, g, h))
    ]
    return binascii.hexlify(
        b''.join(struct.pack('!Q', element) for element in sha512_hash),
    ).decode('utf-8')

if __name__ == "__main__":
    msg = input("Enter text:")
    print(sha512(msg))

```

Output:

I) Program Output:



```

Semester VII/Cryptography and Security /P/14. SHA 07:17:55
python sha512.py
Enter text:hellothere
20d28812d27b1e2f9c41021318c12565cef7126decb8533ab4965a38069763cb3
486af1c831cb134ca9767898c8796f07b5ad9b4fd0e7d7532f899387fa83226

```

II) Verification:

SHA-512 OUTPUT:

```

20d28812d27b1e2f9c41021318c12565cef7126decb853
3ab4965a38069763cb3486af1c831cb134ca9767898c87
96f07b5ad9b4fd0e7d7532f899387fa83226

```

RESULT:

Hence, Hashing algorithms such as md5 and sha512 were implemented and hashes generated were verified successfully.

Ex. No: 6

Date: 03/10/2023

EVENT HANDLING AND PUSH NOTIFICATIONS IN ANDROID APPLICATIONS

AIM:

To study event handling and push notifications in Android applications and implement them using Android studio.

ALGORITHM:

Event Handling:

- A button is created in the app's layout XML file.
- The button is found by its ID using findViewById.
- An OnClickListener is set for the button.
- In the onClick method, the action to be performed when the button is clicked is defined.

Push Notifications:

- The logic to handle push notifications in the app is implemented, typically in a service or a BroadcastReceiver.
- The messaging service's SDK is used to send push notifications to the device using the app's registration token.
- When the button is clicked (as defined in the event handling step), the push notification is triggered by sending a message to the app using the messaging service.
- The notification when clicked triggers an event that opens chrome.

SOURCE CODE:

I) activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imgGFG"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="70dp"
        android:src="@drawable/baseline_notifications_24" />

    <EditText
        android:id="@+id/et1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imgGFG"
        android:layout_marginLeft="30dp"
        android:layout_marginRight="30dp"
        android:background="#d7ffd9"
        android:hint="Enter The Title"
```

```

        android:padding="10dp"
        android:textSize="20sp"
        android:textStyle="bold" />

<EditText
    android:id="@+id/et2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/et1"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="30dp"
    android:layout_marginRight="30dp"
    android:background="#d7ffd9"
    android:hint="Enter The Text"
    android:padding="10dp"
    android:textSize="20sp"
    android:textStyle="bold" />

<Button
    android:id="@+id/btn1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/et2"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="30dp"
    android:layout_marginRight="30dp"
    android:background="#0F9D58"
    android:text="send notification"
    android:textColor="#000000"
    android:textSize="20sp"
    android:textStyle="bold" />
</RelativeLayout>

```

II) MainActivity.kt:

```

package com.laboratory.pushnotifications
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import android.graphics.BitmapFactory
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat

class MainActivity : AppCompatActivity() {
    val CHANNEL_ID = "SecLabExp"

```

```

val CHANNEL_NAME = "NotificationChannel"
val CHANNEL_DESCRIPTION = "Channel for passing notifications"
val link1 = "https://google.com/"
lateinit var et1: EditText
lateinit var et2: EditText
lateinit var btn1: Button
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    et1 = findViewById(R.id.et1)
    et2 = findViewById(R.id.et2)
    btn1 = findViewById(R.id.btn1)
    btn1.setOnClickListener {
        val imgBitmap = BitmapFactory.decodeResource(resources,
R.drawable.baseline_notifications_24)
        val intent1 = openerIntent(link1)
        val pendingIntent1 = PendingIntent.getActivity(this, 5, intent1,
PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE)
        notificationChannel()
        val nBuilder = NotificationCompat.Builder(this, CHANNEL_ID)
            .setContentTitle(et1.text.toString())
            .setContentText(et2.text.toString())
            .setSmallIcon(R.drawable.baseline_notifications_24)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent1)
            .setAutoCancel(true)
            .setLargeIcon(imgBitmap)
            .setStyle(NotificationCompat.BigPictureStyle()
                .bigPicture(imgBitmap)
                .bigLargeIcon(null))
            .build()
        val nManager = NotificationManagerCompat.from(this)
        nManager.notify(1, nBuilder)
    }
}

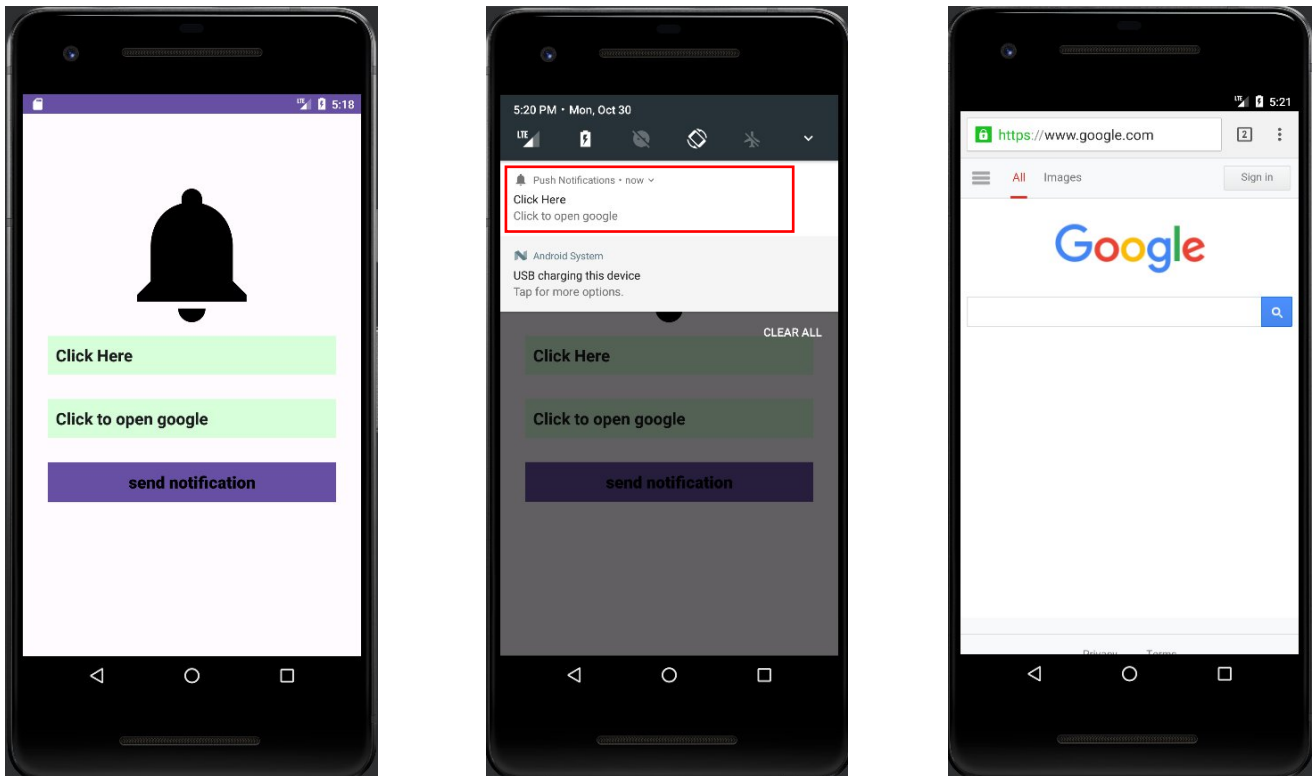
private fun notificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val channel = NotificationChannel(CHANNEL_ID, CHANNEL_NAME,
NotificationManager.IMPORTANCE_DEFAULT).apply {
            description = CHANNEL_DESCRIPTION
        }
        val notificationManager: NotificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        notificationManager.createNotificationChannel(channel)
    }
}

private fun openerIntent(link: String): Intent {
    val intent = Intent()
    intent.action = Intent.ACTION_VIEW
    intent.data = Uri.parse(link)
}

```

```
        return intent
    }
}
```

OUTPUT:



RESULT:

Hence, Event handling and push notifications were successfully implemented in Android studio and test using Android Virtual Device.

Ex. No: 7

Date: 20/10/2023

ANIMATIONS AND GRAPHICAL PRIMITIVES IN ANDROID APPLICATIONS

AIM:

To implement graphical primitives in Android applications and perform animations in android application.

ALGORITHM:

Graphical Primitives:

- Set Up Paint Objects: Create shapePaint and textPaint objects to control how shapes and text are drawn.
- Define textPaint Properties: Set the font size for text using textPaint.textSize.
- Override onDraw Method: Clear the canvas with a white background.
- Define Shapes and Colors: Create shapes using Path objects and assign colors using Paint objects.
- Draw the Shapes: Iterate through the list of shapes and draw them on the canvas using canvas.drawPath.

Animations:

Activity Setup:

- The MainActivity class is an AppCompatActivity.
- Various buttons (Blink, Rotate, Fade, Move, Slide, Zoom, Stop) and an ImageView are declared as class members.
- In the onCreate method, the layout is set, and the buttons and the ImageView are initialized by finding their views in the layout.

Button Click Listeners:

- Each button (blinkButton, rotateButton, fadeButton, moveButton, slideButton, zoomButton) has an OnClickListener attached to it.
- When a button is clicked, the associated animation is loaded using AnimationUtils.loadAnimation, and the animation is applied to the ImageView using imageView.startAnimation.

Stop Button:

- The "Stop" button has an OnClickListener that clears any ongoing animation on the ImageView by calling imageView.clearAnimation.

I) Graphical Primitives:

Source Code:

I) activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

</androidx.constraintlayout.widget.ConstraintLayout>
```

II) MainActivity.kt:

```
package com.laboratory.graphicalprimitives
```

```
import android.app.Activity
import android.content.Context
import android.graphics.*
import android.os.Bundle
import android.view.View
```

```
class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val customDrawingView = CustomDrawingView(this)
        setContentView(customDrawingView)
    }
}
```

```
class CustomDrawingView(context: Context) : View(context) {
    private val shapePaint = Paint()
    private val textPaint = Paint()
    init {
        shapePaint.isAntiAlias = true
        shapePaint.style = Paint.Style.FILL
        textPaint.isAntiAlias = true
        textPaint.color = Color.BLACK
        textPaint.textSize = 60f
    }
    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        canvas.drawColor(Color.WHITE)
        val screenWidth = width.toFloat()
        val screenHeight = height.toFloat()
        val shapes = mutableListOf<Pair<Path, Paint>>()
        val rectanglePath = Path()
        rectanglePath.addRect(0f, 0f, screenWidth, screenHeight, Path.Direction.CW)
        shapes.add(Pair(rectanglePath, Paint().apply { color = Color.GREEN }))
        val trianglePath = Path()
        trianglePath.moveTo(screenWidth / 2, 0f)
        trianglePath.lineTo(0f, screenHeight)
        trianglePath.lineTo(screenWidth, screenHeight)
        trianglePath.close()
        shapes.add(Pair(trianglePath, Paint().apply { color = Color.RED }))
        for ((path, paint) in shapes) {
            canvas.drawPath(path, paint)
        }
        canvas.drawText("Hello there!!!", 360f, 700f, textPaint)
    }
}
```

Output:



II) Animations:

Source Code:

I) activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <ImageView
        android:id="@+id/imageview"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:contentDescription="@string/app_name"
        android:src="@drawable/baseline_stop_24" />
    <LinearLayout
        android:id="@+id/linear1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/imageview"
        android:layout_marginTop="128dp"
        android:orientation="horizontal"
        android:weightSum="3">
        <Button
            android:id="@+id/ButtonBlink"
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
```

```

        android:padding="3dp"
        android:text="@string/blink"
        android:textColor="@color/white" />
<Button
    android:id="@+id/ButtonRotate"
    style="@style/TextAppearance.AppCompat.Widget.Button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:layout_weight="1"
    android:padding="3dp"
    android:text="@string/clockwise"
    android:textColor="@color/white" />
<Button
    android:id="@+id/ButtonFade"
    style="@style/TextAppearance.AppCompat.Widget.Button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:layout_weight="1"
    android:padding="3dp"
    android:text="@string/fade"
    android:textColor="@color/white" />
</LinearLayout>
<LinearLayout
    android:id="@+id/linear2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/linear1"
    android:layout_marginTop="30dp"
    android:orientation="horizontal"
    android:weightSum="3">
    <Button
        android:id="@+id/ButtonMove"
        style="@style/TextAppearance.AppCompat.Widget.Button"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:layout_weight="1"
        android:padding="3dp"
        android:text="@string/move"
        android:textColor="@color/white" />
    <Button
        android:id="@+id/ButtonSlide"
        style="@style/TextAppearance.AppCompat.Widget.Button"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:layout_weight="1"
        android:padding="3dp"
        android:text="@string/slide"
        android:textColor="@color/white" />
    <Button

```



```

        android:id="@+id/ButtonZoom"
        style="@style/TextAppearance.AppCompat.Widget.Button"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:layout_weight="1"
        android:padding="3dp"
        android:text="@string/zoom"
        android:textColor="@color/white" />
</LinearLayout>
<Button
    android:id="@+id/ButtonStop"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/linear2"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="30dp"
    android:layout_marginRight="30dp"
    android:text="@string/stop_animation" />
</RelativeLayout>
II) MainActivity.kt:
package com.laboratory.animations
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.view.animation.AnimationUtils
import android.widget.Button
import android.widget.ImageView
class MainActivity : AppCompatActivity() {
    private lateinit var imageView: ImageView
    private lateinit var blinkButton: Button
    private lateinit var rotateButton: Button
    private lateinit var fadeButton: Button
    private lateinit var moveButton: Button
    private lateinit var slideButton: Button
    private lateinit var zoomButton: Button
    private lateinit var stopButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        imageView = findViewById(R.id.imageView)
        blinkButton = findViewById(R.id.ButtonBlink)
        rotateButton = findViewById(R.id.ButtonRotate)
        fadeButton = findViewById(R.id.ButtonFade)
        moveButton = findViewById(R.id.ButtonMove)
        slideButton = findViewById(R.id.ButtonSlide)
        zoomButton = findViewById(R.id.ButtonZoom)
        stopButton = findViewById(R.id.ButtonStop)
        blinkButton.setOnClickListener(View.OnClickListener {
            imageView.startAnimation(AnimationUtils.loadAnimation(applicationContext, R.anim.blink))
        })
        rotateButton.setOnClickListener(View.OnClickListener {

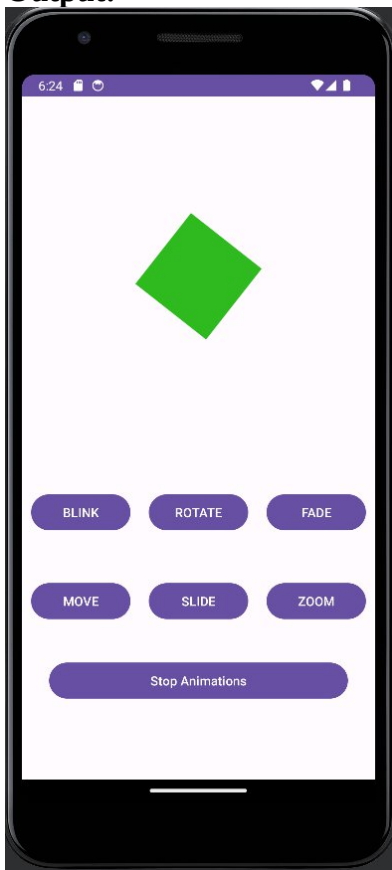
```

```

        imageView.startAnimation(AnimationUtils.loadAnimation(applicationContext,
R.anim.rotate))
    })
    fadeButton.setOnClickListener(View.OnClickListener {
        imageView.startAnimation(AnimationUtils.loadAnimation(applicationContext, R.anim.fade))
    })
    moveButton.setOnClickListener(View.OnClickListener {
        imageView.startAnimation(AnimationUtils.loadAnimation(applicationContext,
R.anim.move))
    })
    slideButton.setOnClickListener(View.OnClickListener {
        imageView.startAnimation(AnimationUtils.loadAnimation(applicationContext, R.anim.slide))
    })
    zoomButton.setOnClickListener(View.OnClickListener {
        imageView.startAnimation(AnimationUtils.loadAnimation(applicationContext,
R.anim.zoom))
    })
    stopButton.setOnClickListener(View.OnClickListener {
        imageView.clearAnimation()
    })
}
}

```

Output:



RESULT:

Hence, Graphical primitives were successfully drawn and basic animations were implemented in Android Studio and tested using AVD

AIM:

To create an Android application that makes of the location services in the Android API.

ALGORITHM:

Activity Setup:

- The MainActivity class is an AppCompatActivity.
- An FusedLocationProviderClient, a TextView (tvLocation), and a "Refresh" Button (btnRefresh) are declared as class members.

Layout Initialization: In the onCreate method, the layout is set, and the views are initialized by finding their views in the layout.

Location Permission Check:

- The checkPermission function is used to check for location permission.
- It checks if either ACCESS_FINE_LOCATION or ACCESS_COARSE_LOCATION permission is granted. If not, it requests permission from the user.

Location Retrieval:

- The getLocation function is used to retrieve the user's location using the FusedLocationProviderClient.
- It attempts to retrieve the last known location.
- If successful, it displays the latitude and longitude in the TextView (tvLocation).
- If there's a failure in retrieving the location, it displays an error message.

Button Click Listener: The "Refresh" button (btnRefresh) has an OnClickListener that triggers the getLocation function when clicked.

SOURCE CODE:**I) activity_main.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="68dp"
        android:text="Your Location is"
        android:textSize="20sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btnRefresh"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_marginBottom="340dp"
        android:text="Refresh Location"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

<TextView
    android:id="@+id/tvLocation"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="536dp"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    tools:text="23.32451423 \n 23.2342343" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

II) MainActivity.kt:

```
package com.laboratory.locationapp
```

```

import android.content.pm.PackageManager
import android.location.Location
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import androidx.annotation.RequiresPermission
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationServices

```

```

class MainActivity : AppCompatActivity() {

    private lateinit var tvLocation: TextView
    private lateinit var fusedLocation: FusedLocationProviderClient
    private lateinit var btnRefresh : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        tvLocation = findViewById(R.id.tvLocation)
        btnRefresh = findViewById(R.id.btnRefresh)
        fusedLocation = LocationServices.getFusedLocationProviderClient(this)
        if(checkPermission()) {
            updateLocation()
        }
        btnRefresh.setOnClickListener {
            updateLocation()
        }
    }
}

```

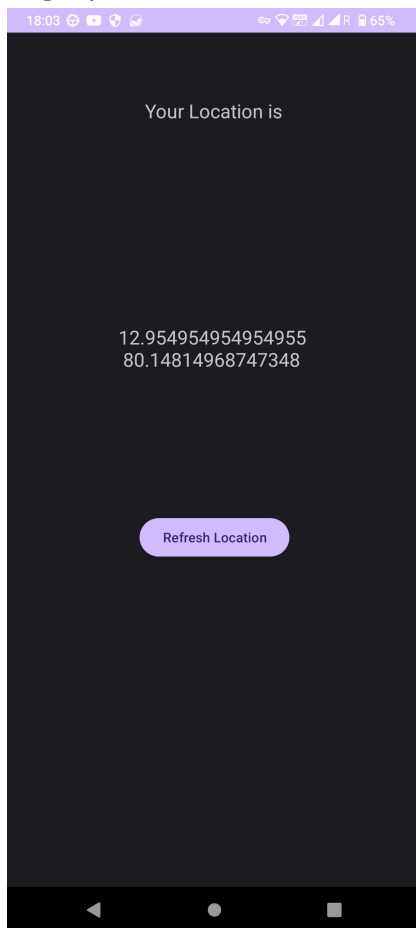
```

    @RequiresPermission(
        anyOf=[android.Manifest.permission.ACCESS_COARSE_LOCATION
        android.Manifest.permission.ACCESS_FINE_LOCATION] )
    fun checkPermission(): Boolean {
        if(ContextCompat.checkSelfPermission(this
        android.Manifest.permission.ACCESS_FINE_LOCATION) ==
        PackageManager.PERMISSION_GRANTED ||
        ContextCompat.checkSelfPermission(this,android.Manifest.permission.ACCESS_COARSE_LOCATION)
        == PackageManager.PERMISSION_GRANTED) {
            return true
        }
        ActivityCompat.requestPermissions(
            this,
            arrayOf(
                android.Manifest.permission.ACCESS_FINE_LOCATION,
                android.Manifest.permission.ACCESS_COARSE_LOCATION
            ),
            1
        )
        return ContextCompat.checkSelfPermission(this,
        android.Manifest.permission.ACCESS_FINE_LOCATION) ==
        PackageManager.PERMISSION_GRANTED ||
        ContextCompat.checkSelfPermission(this,android.Manifest.permission.ACCESS_COARSE_LOCATI
        ON) == PackageManager.PERMISSION_GRANTED
    }

    @RequiresPermission(
        anyOf = [android.Manifest.permission.ACCESS_COARSE_LOCATION ,
        android.Manifest.permission.ACCESS_FINE_LOCATION]
    )
    fun updateLocation() {
        fusedLocation.lastLocation.addOnSuccessListener {
            location: Location? -> location?.let {
                val lat = location.latitude
                val long = location.longitude
                tvLocation.text = "%s \n %s".format(lat,long)
            }
        }.addOnFailureListener {
            tvLocation.text = "Failed to retrieve location"
        }
    }
}

```

OUTPUT:



RESULT:

Hence, an Android application using location service was created to retrieve and display current user location successfully.