Ex1:- Implementation of Caeser Cipher 08/01/25

Aim: TO implement a Caeser cipher, a type of substitution cipher, which replaces each letter in a message with another letter based on a fixed shift value.

Algorithm:-

```
1) Define a function which takes plain text as input. and n(shift pattern).
2). in the function initialize an empty string to store encrypted text.
3) ~~Inte~~ Iterate over plain text using Its Index.
4) check character type:
      if ch is a space, append a space
      else if ch is an uppercase letter:
      convert ch to encrypted character using
      encrypted-char = ((ord(ch)+n-65) % 26)+65. and append it.
      else if ch is a lowercase letter:
     then encrypted-char = ((ord (ch)+n-97) % 26)+97. and append it.
5) Return encrypted text. and print the result.
```

Example:-

Given, plain text: "HELLO EVERYONE" and n=1:
loop through each character in plain text.
'H' is uppercase, so encrypt it to 'I'.
'E' is uppercase, so encrypt it to 'F'.
'L' is uppercase, so encrypt it to 'M'.

program:-

```
def encrypt_text(plaintext,n):
    ans = ""
    for i in range(len(plaintext)):
        ch= plaintext [i]
        if ch == " ":
            ans += " "
        elif (ch.isUpper()):
            ans += chr((ord(ch)+n-65) %26+65)
        else:
            ans+= chr((ord(ch)+n-97) %26+97)

    return ans
plaintext = "HELLO EVERYONE"
n=1
print("plain text is:"+plaintext)
print ("shift pattern is:"+ str(n))
print ("cipher Text is: " + encrypt_text(plaintext,n))
```

Output:-

Text:- HELLO EVERYONE

Shift:- 1

- H -> I
- E -> F
- L -> M
- L -> M
- O -> P.
- space remains as is.
- E -> F
- V -> W
- E -> F
- R -> S
- Y -> Z
- O -> P
- N -> O
- E -> F.

Cipher text is :- IFMMP FWFSZPOF.

ult:-

Hence, Implementation of Caeser cipher is successful.

Exp 2:- Basic Monoalphabetic Cipher. 08/01/25

Aim:- To implement a basic monoalphabetic cipher, a type of substitution cipher, which replaces each letter in a message with another letter based on a fixed shift value.

Algorithm:-

1) Define generate_cipher_key Function:
    - input: shift (Integer).
    - initialize alphabet string 'abcdefghijklmnopqrstuvwxyz'
    - create shifted alphabets
    - Create dictionary key by mapping alphabet string to shifted alphabets
    - Return key.

2). Define encrypt Function:
    - input: message (string), key (dictionary)
    - Initialize an empty-String and loop through. each character
    - if char is alphabetic:
      if char is lowercase, append key [char] to encrypted_message.
      if char is uppercase, append key [char.lower()].upper() to encrypted_message.
    - Else, append char to encrypted_message. and return it.

3) Define decrypt Function:
    - input: cipher text (String), key (dictionary)
    - create reverse_key and use it in place of key as in encrypt function.
    - Initialize. an empty string decrypted_message.
    - after looping through each character return the string.

4). Main function:
    - prompt user to input Shift value and. Choice between encryption and. decryption (e or d).
    - if the user chooses 'e', prompt for plain-text. and call encrypt function.
    - if the user chooses 'd', prompt for cipher- text and call decrypt. function.
    - if user inputs an invalid choice, print an error message.

program:-

```
def generate_cipher_key(shift):
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    return dict(zip(alphabet, alphabet[shift:] + alphabet[:shift]))
def encrypt_decrypt (message, key, reverse = False):
    if reverse : key = {v:k for k,v in key.items()}
    return ''.join (key[char.lower()].upper() if char.isupper() else
          key[char] if char.isalpha() else char for char in message)

def main():
    shift = int(input ("Enter Shift value:"))
    key = generate_cipher_key(shift)
    choice = input ("Encrypt or decrypt ? (e/d):").lower()
    text = input ("Enter Message:")
    print ("Result:", encrypt_decrypt(text, key, choice))
if __name__ == "__main__":
    main().
```

Output:-

```
Enter shift value: 2
Encrypt or decrypt : e
Enter Message: HELLO
Result: JGNNO.
```

Result:-

Hence, Implementation of Monoalphabetic cipher has been executed successfully

Ex3:- Message Authentication code. 29/01/25.

Aim:- To calculate the messages digest of a text using the SHA-1 algorithm and thereby verifying data integrity.

Algorithm :-

1) Import Hashlib module to use various SHA hash functions.

2) Compute SHA256 Hash:
   - Initialize a string str
   - Encode the string to convert it to bytes.
   - pass the encoded String to hashlib.sha256()
   - Get the hexadecimal representation.
   - print the message.

3) Compute SHA.384 Hash.
   - follow the similar process as SHA256
   - but pass the encoded string to hashlib.sha384()

4) Compute SHA224 Hash.
   - similar process as SHA256, pass encoded string to hashlib.sha224().
   - get hexadecimal representation using 'result.hexdigest()'.

5) Encode and hash using SHA1:
   - Reinitialize the string: str= "GeeksforGeeks"
   - Encode the String : encoded_str= str.encode()
   - Hash the encoded string using SHA1: result= hashlib.sha1 (encoded str).
   - print the hexadecimal equivalent.

program :-

```
import hashlib.
str= "GeeksforGeeks"
encoded_str =  str.encode()
hashes= {.
        "SHA256" : hashlib.sha256(encoded_str).hexdigest(),
        "SHA384": hashlib.sha384 (encoded_str).hexdigest(),
        "SHA224": hashlib. sha224 (encoded_str). hexdigest(),
"SHA512": hashlib.sha512(encoded_str).hexdigest(),
"SHA1" : hashlib.sha1(encoded_str).hexdigest().

}

for algo, hex_val in hashes.items():
    print(f"The hexadecimal equivalent of {algo} is {hex_val}")
```

Output:-

```
The hexadecimal equivalent. of SHA256 is: 2cf24dba5-----
The hexadecimal equivalent of SHA384 is: 3d363ff89----.
The hexadecimal equivalent of. SHA224 is: 680c7037----.
The hexadecimal equivalent of SHA512 is: cf5b16a77 ---
The hexadecimal equivalent of SHA1  is: f7177178C----.
```

Result:-

Hence, Implementation of Message Authentication Code using Shalib libraby has Completed Successfully.

29/01/25

Ex4 :- Data Encryption Standard.

Aim:- . TO implement a symmetric-key block cipher algorithm known as. Data Encryption Standard(DES).

Algorithm:-

1) Hexadecimal to Binary. Conversion (hex2bin):
- initialize a dictionary that maps each hex digit to 4-bit binary.
- initialize an empty string for the binary result.

2) Binary to Hexadecimal Conversion (bin2hex):
- initialize. a dictionary that maps each 4-bit binary string to its hex
- initialize an empty String for. the ·hex value.

3) Binary to Decimal Conversion (bin2dec):
- initialize the decimal result to 0.
- for each bit multiply the bit by 2i2'21 and add to decimal result.

4) Decimal to Binary Conversion (dec2bin):
- Convert the decimal num to its binary representation using. python's bin function and remove the "0b" prefix.
- pad zeroes if the length of binary result is not a multiple of 4.

5) permute Function (permute.):
- initialize an empty string for permutation result.
- for each pos in permutation array append the bit from input string at the given position to. the result.

6) Left shift Function (shift_left):
- perform a left circular · shift on the input ·string.

7) XOR function (xor):
- initialize. an empty String for the XOR result.
- for each bit, append the result of XOR operation to ·corresponding bits to the result.

8) Encryption Function (encrypt):

- Convert the. plaintext from hex2bin.
- perform initial permutation using initial-perm table.
- split the text into left & right halves.
- For each of 16 rounds:
  - expand the right half from 38 to 48 bits
  - XOR the right with round key
  - substitute the result using the S-boxes.
  - perform a permutation.
  - XOR the result with left half.
  - swap the left & right halver, except in , final round.
- combine the final left & right halves.
- perform a final permutation.
- return the result as binary.

9) Key Generation:-

- Convert the key from hex to bin.
- perform parity drop to get a 56-bit key.
- split the key into left & right.
  - perform left shifts on both halves
  - Combine the left & right halves.
  - Compress the key from 56 to 48 bits.

10) Main process:-

- Define the plaintext and key in hexadecimal format.
- Generate the round keys.
- perform encryption using the generated round keys.
- Reverse the round keys for decryption.
- perform decryption using the reversed round keys,

program

```
from itertools import permutations.
IP = [58,50, . _ _ _ _ 15,7]
FP = [40, 8, - - - - 57,25]
def permute (block, table):
    return ''.join ([block[x-1] for x in table])
```

```
def des_encrypt (block, key):
        black = permute (block, IP)
        L,R = block[:32] , block[32:]
        for i in range (16):
                L,R = R, ''.join([str (int (L[x]) ^ int (R[x])) for x in range (32)])
        block = permute(L+R, FP)
        return block .

plaintext = "0123456789ABCDEF"
key = "133457799BBCDFF1"
ciphertext = des_encrypt (plaintext, key)
print ("Ciphertext:" , cipher text),
```

Output :-

```
Ciphertext: C6A54A0C842FA4F2.
```

Result :-

Hence, Implementation of Data Encryption Standard (DES) has Completed successfully.

Ex 5:- Advanced Encryption Standard. 29/01/25.

Aim:-
To understand the need of highly secured symmetric encryption algorithm known as Advanced Encryption Standard (AES).

Algorithm :-

1. Import Libraries:
   - AES for AES encryption / decryption.
   - get random key
   - pad & unpad to ensure data is not invalid.

2. Encrypt function:
   - creates a new AES cipher object in CBC mode.
   - pads the data
   - Encrypt the padded data.

3. Decrypt process:
   - creates a new AES cipher object with the same IV.
   - Decrypts the ciphertext.
   - unpads & decodes the decrypted data.

Input :-
Input Message is "This is a secret message".

Output:-
The output of code will consist of ciphertext and the decrypted. message.

Program

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes.
from    "  .  ".,"  import padding.
from    "  .  ".backends import default_backend.
import os
 key = os.urandom(32)
 iv = os.urandom(16).
```

```
def encrypt_aes (message, key, vi):
    padder. = padding. PKcs7(128). padder().
    padded_data = padder. update (message.encode() ) + padder. finalize().
    backend= default_backend()).
    encryptor= cipher.encryptor()
    return encryptor.update (padded_data) + encryptor. finalize()

message= "This is a secrect message."
ciphertext = encrypt_aes (message, key, iv)
print (" Ciphertext:", Ciphertext.hex())
print ("key: ", key.hex())
print ("I V:" , iv. hex())
```

Output:-

```
Ciphertext: 4a5f9c8d3b. - - .
key :  a1b2c3d4 - — -
IV :   1a2b3c4d - - -
```

Result:-

Hence, Implementation of Advanced Encryption Standard (AES) has been completed successfully.

05/02/24

EX.NO:6 Asymmetric key Encryption

Aim:
TO implement the popular asymmetric key algorithm .Rivest, Shamir, Adleman (RSA).

Algorithm:-

1) Input:
- Two prime numbers p and q.
- A plaintext message.
- calculate n:

  $n = p*q$

  For $p=53$ and $q=59$, $n = 53*59 = 3127$.

2) Calculate the Totient:-
- $t = (p-1)*(q-1)$, for $p=53, q=59$, $t = 52*58 = 3016$.
- Select the public keye: Iterate from 2 to t, find the smallest integer e such that gcd(e,t)==1.

3) Select the public keye:
- Find the smallest integer e such that gcd(e,t)==1

4) Select the private key d:
- initialize J=0, Increment J in a loop until (J*e)%t==1.
- set d=J. Find d such that (d*e)%t==1, we get d=2011.

5) Encrypt the message:-
Calculate the ciphertext ct=(message **e)%n. => 1394.

6) Decrypt the message:-.
decrypted message mes=(ct**d)%n.
.) (1394**2011)%3127=89.

Program:-

```
import math
p=53
q=59
n=p*q.
print(f"n = {p} * {q} = {n}")
t = (p-1) * (q-1)
print(f"Totient t = ({p}-1) *({q}-1) = {t}")
```

```
e=2
while math.gcd(e,t) !=1;
        e+=1
print(f"Public key e = {e}").

d=1
while (d*e)%t !=1:
        d+=1
print(f"Private key d = {d}")

message=89
ct = (message **e)%n
print(f"Encrypted message ct= ({message} ** {e})%. {n}= {ct}")

decrypted_message= (ct **d)%n.
print(f"Decrypted message = ({ct} ** {d})%. {n} = {decrypted_message.}")
```

Output:-

```
n= 53*59 = 3127
Totient t= 3016.
public key e = 3
private key d = 2011
Encrypted message ct = (89 ** 3)%. 3127 = 1394
Decrypted message     = (1394 ** 2011) %. 3127 = 89.
```

Result:- Thus, Asymmetric key Encryption has been Implemented successfully.