A) Caeser Cipher

```
def encypt_func(txt, s):
  result = ""


  for i in range(len(txt)):
    char = txt[i]

    if (char.isupper()):
      result += chr((ord(char) + s - 64) % 26 + 65)
    else:
      result += chr((ord(char) + s - 96) % 26 + 97)
  return result
txt = "CEASER CIPHER EXAMPLE"
s = 4

print("Plain txt : " + txt)
print("Shift pattern : " + str(s))
print("Cipher: " + encypt_func(txt, s))
```

2)Reil Fence

```
def encryptRailFence(text, key):

	rail = [['\n' for i in range(len(text))]

				for j in range(key)]

	dir_down = False

	row, col = 0, 0

	for i in range(len(text)):

		if (row == 0) or (row == key - 1):

			dir_down = not dir_down

		rail[row][col] = text[i]

		col += 1

		if dir_down:

			row += 1
```

```
        else:
            row -= 1
    result = []
    for i in range(key):
        for j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return("" . join(result))
def decryptRailFence(cipher, key):
    rail = [['\n' for i in range(len(cipher))]
                for j in range(key)]
    dir_down = None
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False
        rail[row][col] = '*'
        col += 1
        if dir_down:
            row += 1
        else:
            row -= 1
    index = 0
    for i in range(key):
        for j in range(len(cipher)):
```

```
                if ((rail[i][j] == '*') and
                (index < len(cipher))):
                    rail[i][j] = cipher[index]
                    index += 1
        result = []
        row, col = 0, 0
        for i in range(len(cipher)):
            if row == 0:
                dir_down = True
            if row == key-1:
                dir_down = False
            if (rail[row][col] != '*'):
                result.append(rail[row][col])
                col += 1
            if dir_down:
                row += 1
            else:
                row -= 1
        return("".join(result))
if __name__ == "__main__":
        print(encryptRailFence("attack at once", 2))
        print(encryptRailFence("GeeksforGeeks ", 3))
        print(encryptRailFence("defend the east wall", 3))
        print(decryptRailFence("GsGsekfrek eoe", 3))
        print(decryptRailFence("atc toctaka ne", 2))
        print(decryptRailFence("dnhaweedtees alf tl", 3))
```

3)DES Algorithm:

```
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes


def initialize_des(key):
    cipher = DES.new(key, DES.MODE_ECB)
    return cipher

def pad_data(data):
    padding = 8 - (len(data) % 8)
    data += bytes([padding] * padding)
    return data


# Remove padding from decrypted data
def unpad_data(data):
    padding = data[-1]
    return data[:-padding]


# Encrypt data using DES
def encrypt_data(data, cipher):
    data = pad_data(data)
    ciphertext = cipher.encrypt(data)
    return ciphertext


# Decrypt data using DES
def decrypt_data(ciphertext, cipher):
    decrypted_data = cipher.decrypt(ciphertext)
    decrypted_data = unpad_data(decrypted_data)
    return decrypted_data


# Example usage
if __name__ == "__main__":
    key = get_random_bytes(8)  # Generate a random 8-byte key
```

```
    data_to_encrypt = b'ABISHEK'

    cipher = initialize_des(key)

    encrypted_data = encrypt_data(data_to_encrypt, cipher)
    decrypted_data = decrypt_data(encrypted_data, cipher)

    print(f"Original Data: {data_to_encrypt}")
    print(f"Encrypted Data: {encrypted_data}")
    print(f"Decrypted Data: {decrypted_data.decode('utf-8')}")
```

4)RSA

```
def gcd(e, z):
    if e == 0:
        return z
    else:
        return gcd(z % e, e)

def mod_inverse(a, m):
    return pow(a, -1, m)

# The number to be encrypted and decrypted
msg = 12
p = 3
q = 11
n = p * q
z = (p - 1) * (q - 1)
print("the value of z =", z)

e = 2
while e < z:
    # e is for the public key exponent
    if gcd(e, z) == 1:
        break
    e += 1
print("the value of e =", e)

d = mod_inverse(e, z)  # Calculate the private key d
```

```
print("the value of d =", d)

c = (msg ** e) % n
print("Encrypted message is:", c)

N = n
C = int(c)
msgback = (C ** d) % N
print("Decrypted message is:", msgback)
```

or

```
import random
import math

def gcd(a, b):
    return math.gcd(a, b)


# Function to calculate the modular multiplicative inverse
def mod_inverse(a, m):
    return pow(a, -1, m)


# Function to generate RSA key pair
def generate_key_pair(bits):
    p = 11
    q = 17
    n = p * q
    phi = (p - 1) * (q - 1)

    while True:
        e = random.randrange(2, phi)
        if gcd(e, phi) == 1:
            break

    d = mod_inverse(e, phi)
    return ((n, e), (n, d))


# Function to encrypt a message
```

```
def encrypt(public_key, plaintext):
    n, e = public_key
    encrypted = [pow(ord(char), e, n) for char in plaintext]
    return encrypted


# Function to decrypt a message
def decrypt(private_key, ciphertext):
    n, d = private_key
    decrypted = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(decrypted)


# Example usage
if __name__ == "__main__":
    # Generate a key pair with 2048 bits
    public_key, private_key = generate_key_pair(2048)

    # Message to encrypt
    message = "Hello, RSA!"

    # Encrypt the message using the public key
    encrypted_message = encrypt(public_key, message)

    # Decrypt the message using the private key
    decrypted_message = decrypt(private_key, encrypted_message)

    print(f"Original message: {message}")
    print(f"Encrypted message: {encrypted_message}")
    print(f"Decrypted message: {decrypted_message}")
```

5)Diffe hellman

```
p = 23
g = 5

# Alice's private key
a = 3

# Bob's private key
```

```
b = 5

# Calculate Alice's public key
A = (g ** a) % p

# Calculate Bob's public key
B = (g ** b) % p

# Exchange public keys (A and B) over the insecure channel

# Calculate the shared secret key for Alice
shared_secret_key_a = (B ** a) % p

# Calculate the shared secret key for Bob
shared_secret_key_b = (A ** b) % p

# Both Alice and Bob now have the same shared secret key

print("Prime (p):", p)
print("Primitive Root (g):", g)
print("Alice's private key (a):", a)
print("Bob's private key (b):", b)
print("Alice's public key (A):", A)
print("Bob's public key (B):", B)
print("Shared secret key (Alice):", shared_secret_key_a)
print("Shared secret key (Bob):", shared_secret_key_b)
```

6) Sha Hashing Algorithm

```
import hashlib

input_str = "ABISHEK"  # Input string

try:
    sha1 = hashlib.sha1()
    sha1.update(input_str.encode('utf-8'))
    hash_result = sha1.hexdigest()

    print("Input:", input_str)
    print("SHA-1 Hash:", hash_result)
```

```
except Exception as e:
    print(e)
```

7)md5:

```
import hashlib

# Input string
input_str = "Hello, MD5!"

# Create an MD5 hash object
md5_hash = hashlib.md5()

# Update the hash object with the bytes of the input string
md5_hash.update(input_str.encode('utf-8'))

# Get the hexadecimal representation of the MD5 hash
md5_result = md5_hash.hexdigest()

print("Input:", input_str)
print("MD5 Hash:", md5_result)
```

8)DSS:

```
import hashlib
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256


key = DSA.generate(1024)


private_key = key
public_key = key.publickey()

message = input("Enter the message to be signed: ").encode('utf-8')

hash_obj = SHA256.new(message)
signer = DSS.new(private_key, 'fips-186-3')
```

```python
signature = signer.sign(hash_obj)

hash_obj = SHA256.new(message)
verifier = DSS.new(public_key, 'fips-186-3')
try:
    verifier.verify(hash_obj, signature)
    print("The signature is verified.")
except ValueError:
    print("The signature is not verified.")
```