# Practical No:1 Roll No:59

1. **Write a program to implement symmetric encryption using Ceaser Cipher algorithm.**

**Code:-**

```
def encrypt(text,s):
    result=""
    for i in range(len(text)):
        char=text[i]

        #Encrypt uppercase characters in plain text
        if(char==" "):
            result+=" "
        else:
            if(char.isupper()):
                #chr() funtion return character that
represents specified unicode
                #ord() function return the number
                result+=chr((ord(char) + s - 65) % 26 + 65)
            else:
                result+=chr((ord(char) + s - 97) % 26 + 97)
    return result

text="Hello 1"
s=4
print("plain text:"+text)
print("Shift pattern : "+str(s))
print("cipher "+encrypt(text,s))
```

**Output:-**

```
PS D:\Finolex\SEM 3\BlockChain> python .\ceaser_cipher.py
plain text:Hello 1
Shift pattern : 4
cipher Lipps i
PS D:\Finolex\SEM 3\BlockChain>
```

2. **Write a program to implement asymmetric encryption using RSA algorithm. Generate both the keys public key and private key and store it in file. Also encrypt and decrypt the message using keys.**

**Code:-**

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from binascii import hexlify

message=b"Private And Public Key Encrytion"

private_key=RSA.generate(1024)

public_key=private_key.publickey()

print(type(private_key),type(public_key))

private_pem=private_key.export_key().decode()
public_pem=public_key.export_key().decode()

print(type(private_pem),type(public_pem))

with open('private_pem.pem','w') as pr:
    pr.write(private_pem)

with open('public_pem.pem','w') as pu:
    pu.write(public_pem)

pr_key=RSA.import_key(open('private_pem.pem','r').read())
pu_key=RSA.import_key(open('public_pem.pem','r').read())

print(type(pr_key),type(pu_key))

cipher=PKCS1_OAEP.new(key=pu_key)
cipher_text=cipher.encrypt(message)

print(cipher_text)

decrypt=PKCS1_OAEP.new(key=pr_key)

decrypted_message=decrypt.decrypt(cipher_text)
print(decrypted_message)
```

**Output:-**

```
PS D:\MCA\sem3\Blockchain\Practical\Practical1> python RSA.py
<class 'Crypto.PublicKey.RSA.RsaKey'> <class 'Crypto.PublicKey.RSA.RsaKey'>
<class 'str'> <class 'str'>
<class 'Crypto.PublicKey.RSA.RsaKey'> <class 'Crypto.PublicKey.RSA.RsaKey'>
Cipher Text: b'g\xcc#\xfe\xaf\xbc\x89\xa0\x8c\xf0\xfa\xe9\xad\x8fN\xac\x86}\xef;\xd3<\xa98\x1c\xac\x1e\xc9\x180\xf6\xd5\xad\x0c\xa0{q\xf6\xd9H\x0c\xa5\x7f\x
18^\xb92\x97\xba\x80P\xfb\x9e\xf6\xe4\xca\xf9K$"oA\xd0\xfb\xd6\xa95\xac\xa3c\xbc\x0c\xa1Y\x9c\xael\x91\xcc\x9c\x0b\x06=p|\xe0\x8b\xc5-\xe5\xe3\x8c\xaa?\xfb\
xef\xde\xfb\x8c\xa5\xa2\xc7\n5\xe8\xc1#\xbeN\xf4\x12\xba\x14\x94^\x150H\x1fgDr\x15\xccMy5\x86'
Decrypted Message: b'Private And Public Key Encrytion'
PS D:\MCA\sem3\Blockchain\Practical\Practical1>
```

3. **Write a program to demonstrate the use of Hash Functions (SHA-256).**

**Code:-**

```
import hashlib
string ="hello how are you?"
encoded=string.encode()
result=hashlib.sha256(encoded)
print("String: ",end="")
print(string)
print("Hash value: ",end="")
print(result)
print("Hexadecima equivalent ",result.hexdigest())
print("Digest size: ",end="")
print(result.digest_size)
print("Block size: ",end="")
print(result.block_size)
```

**Output:-**

```
[Running] python -u "d:\Finolex\SEM 3\BlockChain\tempCodeRunnerFile.py"
String: hello how are you?
Hash value: <sha256 _hashlib.HASH object @ 0x000002D6730A3CD0>
Hexadecima equivalent  2a02da097cb6c1c39ba9fa7b01673eb69b20cdddb6b4ad54fc7fc315055ba714
Digest size: 32
Block size: 64
```

# Practical No:1 Roll No:59

4. **Write a program to demonstrate Merkle Tree.**

**Code:-**

```
var merkle=require('merkle')
var str='Fred, Bret, Bill, Bob, Alice, Trent';
var arr=str.split(',');
console.log("Input:\t\t",arr);
var tree=merkle('sha1').sync(arr);
console.log("Root hash:\t",tree.root());
console.log("Tree depth:\t",tree.depth());
console.log("Tree levels:\t",tree.levels());
console.log("Tree nodes:\t",tree.nodes());

var i;
for (i=0;i<tree.levels();i++){
    console.log("\nLevels ",i);
    console.log(tree.levels(i));
}
```

**Output:-**

```
D:\Finolex\SEM 3\BlockChain>node Merkle.js
Input:          [ 'Fred', ' Bret', ' Bill', ' Bob', ' Alice', ' Trent' ]
Root hash:      CA2DA9FB28EED6DDBF324396FAC88F12D8013986
Tree depth:     3
Tree levels:    4
Tree nodes:     6

Levels  0
4

Levels  1
4

Levels  2
4

Levels  3
4

D:\Finolex\SEM 3\BlockChain>
```