Practical No. 01

Exercise

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

```
def encrypt(text,s):
  result = ""
  for i in range (len(text)):
    char = text[i]
    if(char==" "):
      result += " ";
    else:
      if(char.isupper()):
        result += chr((ord(char) + s - 65) % 26 + 65)
      else:
        result += chr((ord(char) + s - 97) % 25 + 97)
    return result
print("CEASER CIPER DEMO")
text = input("Enter text to encrypt: ")
s = 4
print("Plain Text: " + text)
print("Shift patter: " + str(s))
print("Cipher: " + encrypt(text,s))
```

Output:-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\SEM3\Blockchain\pract1> python ceaser_cipher.py
Ceaser cipher demo
Enter text to encrypt : hello
Plain text: hello
Shift pattern : 4
Cipher: lipps
PS D:\SEM3\Blockchain\pract1>
```

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.

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

message = b"Public and Private Keys encryption"
#generating private key
private_key = RSA.generate(1024)
#generating public key
public_key = private_key.publickey()
print(type(private_key), type(public_key))
#converting rsa object to string
private_pem = private_key.export_key().decode()
public_pem = public_key.export_key().decode()
print(type(private_pem), type(public_pem))
#writing down the private and public key to 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)

#importing keys from files, converting it into rsakey object
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))
#instantiating PKCS1
cipher = PKCS1_OAEP.new(key=pu_key)
#encrypting the message with PKCS
cipher_text = cipher.encrypt(message)

print(cipher_text)
#instantiating PKCS1
decrypt = PKCS1_OAEP.new(key=pr_key)
#decrypting the message with PKCS
decrypted_message = decrypt.decrypt(cipher_text)
print(decrypted_message)
```

Output:-

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL                    powershell

PS E:\Blockchain\Practicalno1> 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'>
b"G\xfa\xb3\xee\xec\xb9B\x96?i\xdee{\xfd\x07p6vd\xfa43\xa6\xe0\xa3L\x95\xa6\x95'\xd4\x14\x92N\xbcr\xe2\xe2\x93
\x8f\xb8\x8c\xf6l\x81&\x9e\x9e\xd1(\xc2\xe4\x9c\xde\xa0~\xden\n]\xda*\x1e\x14\x0e\xafQ\x8dCQ\x0e\x93\xe9m\xc3E
\x93\xe9m\xc3E\xf3\xf23\xdb\xe5\xf7\x95\xec<\xc9\xe5\x93D\x0eX\xc1cyHC\xb8\xdfj\xd7\xa0h\x02N\xa5\t\\k\xda\xc1
\xb9k\x9c\xb2\x18\x94M\xb37k\x96\x92\xd6d?\x14Q\x9e"
b'Public and Private Keys encryption'
PS E:\Blockchain\Practicalno1>
```

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

```
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("Hexadecimal equivalent: ", result.hexdigest())

print("Digest Size: ", end="")

print(result.digest_size)

print("Block Size : ", end="")

print(result.block_size)
```

Output:-

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    Code

[Running] python -u "e:\Blockchain\Practicalno1\SHA256demo.py"
String: hello how are you?
Hash Value: <sha256 _hashlib.HASH object @ 0x00871F50>
Hexadecimal equivalent:  2a02da097cb6c1c39ba9fa7b01673eb69b20cdddb6b4ad54fc7fc315055ba714
Digest Size: 32
Block Size : 64

[Done] exited with code=0 in 0.3 seconds
```

4. Write a program to demonstrate Merkle Tree.

```
var merkle = require('merkle')

var str = 'Pooja, Archana, Abhishek, Sumit, Sourabh';

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("\nLevel ", i);

   console.log(tree.level(i));

}
```

Output:-

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
[Running] node "e:\Blockchain\Practicalno1\MerkleTree.js"
Input:          [ 'Pooja', ' Archana', ' Abhishek', ' Sumit', ' Sourabh' ]
Root hash:      01E5814C91594BD03934D1F9038F6F8DDFE1F3B3
Tree depth:     3
Tree levels:    4
Tree nodes:     6

Level  0
[ '01E5814C91594BD03934D1F9038F6F8DDFE1F3B3' ]

Level  1
[
  '874ADABFDA79745FC3A6E576EBDF25A081A106E6',
  '45B54305FFD1CC5845522BFAA4CFAA277AC3A289'
]

Level  2
[
  '99D97A578AD03B2507EEB027C7F02CE85AFEFD65',
  '1AF34A6004C05993FE32EB7A0204D955B8442963',
  '45B54305FFD1CC5845522BFAA4CFAA277AC3A289'
]

Level  3
[
  'BBFA68F15818AE1222DEFCDE840E4ADFCA6B5B18',
  'CAFC5385A8107A29C1B3A78F50DED2D51794DCF2',
  '230DEA35F99BA8C5E876691283 2E56EB08DD2FFC',
  '984D99B6D3C5956FC3D2C6DCB438769B183B03B0',
  '45B54305FFD1CC5845522BFAA4CFAA277AC3A289'
]

[Done] exited with code=0 in 0.468 seconds
```