**ICS PRACS**

1. **S-AES**

$ sudo apt-get install build-essential python3-dev

$ pip install pycryptodome

$ pip install pycryptodome-test-vectors

$ python3 -m Cryptodome.SelfTest

**from** Crypto.Cipher **import** AES

**from** secrets **import** token\_bytes

In [3]:

key **=** token\_bytes(16)

In [4]:

**def** encrypt(msg):

cipher **=** AES**.**new(key,AES**.**MODE\_EAX)

nonce **=** cipher**.**nonce

ciphertext,tag **=** cipher**.**encrypt\_and\_digest(msg**.**encode('ascii'))

**return** nonce,ciphertext,tag

In [5]:

**def** decrypt(nonce,ciphertext,tag):

cipher **=** AES**.**new(key,AES**.**MODE\_EAX,nonce **=** nonce)

plaintext **=** cipher**.**decrypt(ciphertext)

**try**:

cipher**.**verify(tag)

**return** plaintext**.**decode('ascii')

**except**:

**return** **False**

In [6]:

nonce,ciphertext,tag **=** encrypt(input("Enter a message: "))

plaintext **=** decrypt(nonce,ciphertext,tag)

print(f'Cipher text: {ciphertext}')

**if** **not** plaintext:

print('Message is corrupted!!!')

**else**:

print(f'Plain text: {plaintext}')

output:

Enter a message: AES Implementation using PyCryptodome

Cipher text: b'\xda\xd9b\xe3\xa6\xb7tu\x01{7\x82\x18[\r\x80F\xe9S\xfa\xae<\xf1\xebg\xc1\xf5!=\x9d\x938\xb0\xb2\x0cs\x01'

Plain text: AES Implementation using PyCryptodome

1. **S-DES**

from Crypto.Cipher import DES

from secrets import token\_bytes

key = token\_bytes(8)

def encrypt(msg):

cipher = DES.new(key, DES.MODE\_EAX)

nonce = cipher.nonce

ciphertext, tag = cipher.encrypt\_and\_digest(msg.encode('ascii'))

return nonce, ciphertext, tag

def decrypt(nonce, ciphertext, tag):

cipher = DES.new(key, DES.MODE\_EAX, nonce=nonce)

plaintext = cipher.decrypt(ciphertext)

try:

cipher.verify(tag)

return plaintext.decode('ascii')

except:

return False

nonce, ciphertext, tag = encrypt(input('Enter a message: '))

plaintext = decrypt(nonce, ciphertext, tag)

print(f'Cipher text: {ciphertext}')

if not plaintext:

print('Message is corrupted!')

else:

print(f'Plain text: {plaintext}')

1. **RSA**

**!**pip install rsa

**import** rsa

In [2]:

**def** generate\_keys():

(pubKey,privKey) **=** rsa**.**newkeys(1024)

"""A PEM file is a Base64-encoded certificate file

used to authenticate a secure website, so we will save our keys in PEM files

create a separate folder of keys in same directory as that of this code

"""

*# save public & private key in folder with write bytes mode*

**with** open('keys/pubkey.pem','wb') **as** f:

f**.**write(pubKey**.**save\_pkcs1('PEM'))

**with** open('keys/privkey.pem','wb') **as** f:

f**.**write(privKey**.**save\_pkcs1('PEM'))

In [3]:

**def** load\_keys():

*# load public and private key from folder in read bytes mode*

**with** open('keys/pubKey.pem','rb') **as** f:

pubKey **=** rsa**.**PublicKey**.**load\_pkcs1(f**.**read())

**with** open('keys/privKey.pem','rb') **as** f:

privKey **=** rsa**.**PrivateKey**.**load\_pkcs1(f**.**read())

**return** pubKey,privKey

In [4]:

**def** encrypt(msg,key):

**return** rsa**.**encrypt(msg**.**encode('ascii'),key)

In [5]:

**def** decrypt(ciphertext,key):

**try**:

**return** rsa**.**decrypt(ciphertext,key)**.**decode('ascii')

**except**:

**return** **False**

In [6]:

**def** sign\_sha1(msg,key):

""" sign with sha-256 hash function, SHA-256 is hashing algorithm

"""

**return** rsa**.**sign(msg**.**encode('ascii'),key,'SHA-256')

In [7]:

**def** verify\_sha1(msg,signature,key):

""" rsa verify will return key has

been signed using which hashing algorithm

"""

**try**:

**return** rsa**.**verify(msg**.**encode('ascii'),signature,key) **==** 'SHA-256'

**except**:

**return** **False**

In [8]:

generate\_keys()

pubKey,privKey **=** load\_keys()

message **=** input('Enter a message: ')

*# encrypt with public key*

ciphertext **=** encrypt(message,pubKey)

*# we will sign our message with our private key,*

*#so other will verify using our public key*

signature **=** sign\_sha1(message,privKey)

plaintext **=** decrypt(ciphertext,privKey)

print(f'Cipher text: {ciphertext} \n')

print(f'Signature: {signature} \n')

**if** plaintext:

print(f'Plaintext: {plaintext} \n')

**else**:

print('Could not decrypt the message \n')

**if** verify\_sha1(plaintext,signature,pubKey):

print('Signature verified!!! \n')

**else**:

print('Could not verify the message signature \n')

1. **TRIPLE DES**

from Crypto.Cipher import DES3

from Crypto.Random import get\_random\_bytes

while True:

try:

key = DES3.adjust\_key\_parity(get\_random\_bytes(24))

break

except ValueError:

pass

def encrypt(msg):

cipher = DES3.new(key, DES3.MODE\_EAX)

nonce = cipher.nonce

ciphertext = cipher.encrypt(msg.encode('ascii'))

return nonce, ciphertext

def decrypt(nonce, ciphertext):

cipher = DES3.new(key, DES3.MODE\_EAX, nonce=nonce)

plaintext = cipher.decrypt(ciphertext)

return plaintext.decode('ascii')

nonce, ciphertext = encrypt(input('Enter a message: '))

plaintext = decrypt(nonce, ciphertext)

print(f'Cipher text: {ciphertext}')

print(f'Plain text: {plaintext}')

1. **Diffie Hellman**

p **=** 27

g **=** 13

print(f'The value of p is: {p}')

print(f'The value of g is: {g}')

a **=** 7

print(f'The Private key for a is: {a}')

x **=** int(pow(g,a,p))

b **=** 2

print(f'The Private key for b is: {b}')

y **=** int(pow(g,b,p))

ka **=** int(pow(y,a,p))

kb **=** int(pow(x,b,p))

print(f'Secret key for a is: {ka}')

print(f'Secret key for b is: {kb}')

output:

The value of p is: 27

The value of g is: 13

The Private key for a is: 7

The Private key for b is: 2

Secret key for a is: 16

Secret key for b is: 16

1. **ECC**

import math

import random

def point(a, b):

if (4\*(a\*\*3) + 27\*(b\*\*2)) != 0:

x = 1

print("generating")

while True:

rhs = (x\*\*3) + (a\*x) + b

y = int(math.sqrt(rhs))

lhs = (y\*\*2)

if lhs == rhs:

return [x, y]

else:

x += 1

else:

print("Enter another coefficients.")

a = int(input("Enter the coefficient 'a' of curve: "))

b = int(input("Enter the coefficient 'b' of curve: "))

private\_A = 13

private\_B = 15

generator = point(a, b)

print("Generator point: ", generator)

m = int(input("Enter the plaintext integer: "))

public\_key\_A = [private\_A\*generator[0], private\_A\*generator[1]]

print("Public Key of A: ", public\_key\_A)

public\_key\_B = [private\_B\*generator[0], private\_B\*generator[1]]

print("Public Key of B: ", public\_key\_B)

k = random.randint(0, 10)

c1 = k \* (generator[0] + generator[1])

c2 = m + ((k\*public\_key\_B[0]) + (k\*public\_key\_B[1]))

ciphertext = [c1, c2]

print("Ciphertext: ", ciphertext)

r = private\_B\*c1

plaintext = c2 - r

print("Decrypted Plaintext: ", plaintext)

output:

Enter the coefficient 'a' of curve: 5

Enter the coefficient 'b' of curve: 7

generating

Generator point: [2, 5]

Enter the plaintext integer: 3

Public Key of A: [26, 65]

Public Key of B: [30, 75]

Ciphertext: [14, 213]

Decrypted Plaintext: 3