**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**

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

**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**

**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**

**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: hello arumnhoi

Cipher text: b'\xf7\x8a\xbdN\x9d\x03\xc0\xb9yr\x07\x8bY\xc8'

Plain text: hello arumnhoi

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**

**import random**

**import string**

**def get\_N(p, q):**

**return p\*q**

**def phi\_of\_n(p, q):**

**return (p - 1) \* (q - 1)**

**def get\_gcd(x, y):**

**while(y):**

**x, y = y, x % y**

**return x**

**# Generate Public and private keys**

**def get\_encryption\_key(n, phi\_of\_n):**

**lst = [i for i in range(1, n+1)]**

**e\_list = []**

**for i in lst:**

**if (1 < i) and (i < phi\_of\_n):**

**gcd = get\_gcd(i, n)**

**gcd\_phi = get\_gcd(i, phi\_of\_n)**

**if (gcd == 1) and (gcd\_phi == 1):**

**e\_list.append(i)**

**if len(e\_list) == 1:**

**return e\_list[0]**

**else:**

**return e\_list[random.randint(1, len(e\_list)-1)]**

**def get\_decryption\_key(e, phi\_of\_n):**

**d\_list = []**

**for i in range(e \* 25):**

**if (e \* i) % phi\_of\_n == 1:**

**d\_list.append(i)**

**return d\_list[random.randint(1, len(d\_list) - 1)]**

**p = 11**

**q = 29**

**n = get\_N(p, q) # 319**

**phi\_func = phi\_of\_n(p, q) # 280**

**e = get\_encryption\_key(n, phi\_func)**

**d = get\_decryption\_key(e, phi\_func)**

**# to avoid key collision**

**while d == e:**

**d = get\_decryption\_key(e, phi\_func)**

**public\_key = [e, n] # [137, 319]**

**private\_key = [d, n] # [1633, 319]**

**# Encryption**

**def text\_to\_digits(PT):**

**pool = string.ascii\_letters + string.punctuation + " "**

**M = []**

**for i in PT:**

**M.append(pool.index(i))**

**return M**

**def digits\_to\_text(DT):**

**pool = string.ascii\_letters + string.punctuation + " "**

**msg = ''**

**for i in DT:**

**# print(i)**

**msg += pool[i]**

**return msg**

**def encrypt(M, public\_key):**

**return [(i \*\* public\_key[0]) % public\_key[1] for i in M]**

**message = "hello world!"**

**M = text\_to\_digits(message)**

**CT = encrypt(M, public\_key)**

**print("Cipher text: ",CT)**

**# Decryption**

**def decrypt(CT, private\_key):**

**return [((i \*\* private\_key[0]) % private\_key[1]) for i in CT]**

**DT = decrypt(CT, private\_key)**

**m = digits\_to\_text(DT)**

**print("Decrypted text: ", m)**

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