Diffie Hellman Code

import hashlib

import ssl

import binascii

import os

global generator

global prime

global key\_length

prime = 

#GLOBAL PRIMITIVE ROOT

generator = 2

key\_length = 600

'''

--------------------------------------------------------

\*\*\*\*\*\*\*\*\* DIFFIE HELLMAN KEY EXCHANGE PROTOCOL \*\*\*\*\*\*\*\*\*

--------------------------------------------------------

'''

def generate\_private\_key(length):

\_rand = 0

\_bytes = length // 8 + 8

#Generate a random private key such that it's less than the prime number

while (\_rand.bit\_length() < length):

#TODO: Can use Crypto library hash functions

hex\_key = binascii.b2a\_hex(os.urandom(\_bytes))

#Convert to denary format

\_rand = int(hex\_key.encode('hex'),16)

#Update object

private\_key = \_rand

return private\_key

#Public key = primitive root ^ private key % prime

def generate\_public\_key(private\_key):

public\_key = pow(generator, private\_key, prime)

return public\_key

#Secret key = public key ^ private key % q

def generate\_secret(private\_key, public\_key):

#Formula

secret = pow(long(public\_key), long(private\_key), prime)

try:

secret\_bytes = secret.to\_bytes(

shared\_secret.bit\_length() // 8 + 1, byteorder="big")

except AttributeError:

secret\_bytes = str(secret)

#Generate hash key using SHA256

key = hashlib.sha256()

key.update(bytes(secret\_bytes))

secretKey = key.hexdigest()

return secretKey

AES

import binascii

import os

import time

import base64

import hashlib

from Crypto import Random

from Crypto.Cipher import AES

from secretsharing import PlaintextToHexSecretSharer

from secretsharing import SecretSharer

BS = 16

pad = lambda s: s + (BS - len(s) % BS) \* chr(BS - len(s) % BS).encode()

unpad = lambda s: s[:-ord(s[len(s)-1:])]

#GENERATE A (random hexadecimal number)

#temporaryKey = binascii.b2a\_hex(os.urandom(16))

'''

Method:

shamirs\_split

Args:

file object

Function:

This method splits the text content of the file using shamir's secret

sharing algorithm. A list of hex codes are generated. These hex codes

are later broken into three parts again. This list of list is returns

as output

'''

def shamirs\_split(file\_object):

text = file\_object.read()

list = PlaintextToHexSecretSharer.split\_secret(text,2,2)

hexcode = SecretSharer.split\_secret(list[0][2:],2,2);

return hexcode,list[1]

'''

Method:

shamir's join

Args:

list

Function:

Converts the excrypted hexcodes into decrypted ones. Use those decrypted

hexcode to decrypt the text. Returns the text.

'''

def shamirs\_join(list,str):

temp = []

msg\_alpha = SecretSharer.recover\_secret(list[0:2])

msg\_alpha = '1-'+msg\_alpha

temp.append(msg\_alpha)

temp.append(str)

text = PlaintextToHexSecretSharer.recover\_secret(temp[0:2])

return text

'''

Method:

AES encryption

'''

def iv():

"""

The initialization vector to use for encryption or decryption.

It is ignored for MODE\_ECB and MODE\_CTR.

"""

return chr(0) \* 16

class AESCipher(object):

def \_\_init\_\_(self, key):

self.key = key

#self.key = hashlib.sha256(key.encode()).digest()

def encrypt(self, message):

"""

It is assumed that you use Python 3.0+

, so plaintext's type must be str type(== unicode).

"""

message = message.encode()

raw = pad(message)

cipher = AES.new(self.key, AES.MODE\_CBC, iv())

enc = cipher.encrypt(raw)

return base64.b64encode(enc).decode('utf-8')

def decrypt(self, enc):

enc = base64.b64decode(enc)

cipher = AES.new(self.key, AES.MODE\_CBC, iv())

dec = cipher.decrypt(enc)

return unpad(dec).decode('utf-8')