**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**AAT REPORT**

**on**

**RSA-AES Hybrid Cryptography**

**(20CS6PCCNS)**

***Submitted by***

**S Skanda (1BM19CS137)**

**SaiPraveen Marni(1BM19CS138)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**Under the Guidance of**

**Prof. Lohith J J**

**Assistant Professor, BMSCE**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **Topic** | **Page No.** |
| **1** | **Advanced Encryption Standard (AES)** | **3** |
| **2** | **RSA** | **4** |
| **3** | HYBRID Cryptosystem | **5** |
| **4** | IMPLEMENTATION | **7** |
| **5** | Output | **14** |
| **6** | Conclusion | **15** |

**Advanced Encryption Standard (AES)**

**Advanced Encryption Standard (AES)**was introduced by NIST in 2001 is a **symmetric block cipher** which overcomes the key size weakness of DES. AES comes with the variable key sizes i.e. 128-bit key, 192-bit key and a 256-bit key. AES does not follow the Feistel structure

**Drawbacks or disadvantages of AES**

➨It uses too simple algebraic structure.

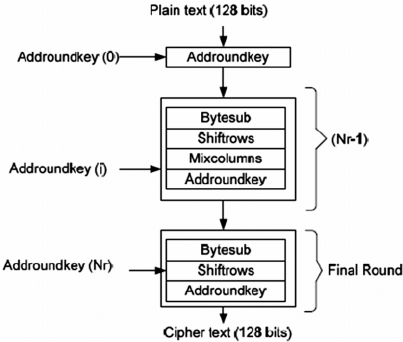
➨Every block is always encrypted in the same way.

➨Hard to implement with software.

➨AES in counter mode is complex to implement in software taking both performance and security into considerations.

The key used in AES if not employed properly it can cause a cryptanalytic attack. Therefore, key scheduling should be done carefully.

although symmetric encryption is extremely fast in terms of computation, it has weakness in key distribution. In order to implement symmetric encryption on public communication, the secret key must be shared securely and secretly with an authorized communicating party which is the main difficulty in this approach



**RSA**

RSA stands for Rivest, Shamir, Adleman. These are the creators of the RSA Algorithm. It is a public-key encryption technique used for secure data transmission especially over the internet. Transmitting confidential and sensitive data over the internet through this technology is safe due to its standard encryption method. It was developed by scientist Rivest, Shamir, and Adleman at RSA Data Security Inc. in 1978. In this algorithm, a code is added to the normal message for security purposes. The algorithm is based on the factorization of large number. Large numbers cannot be easily factorized, so breaking into the message for intruders is difficult.

**Disadvantages of RSA**

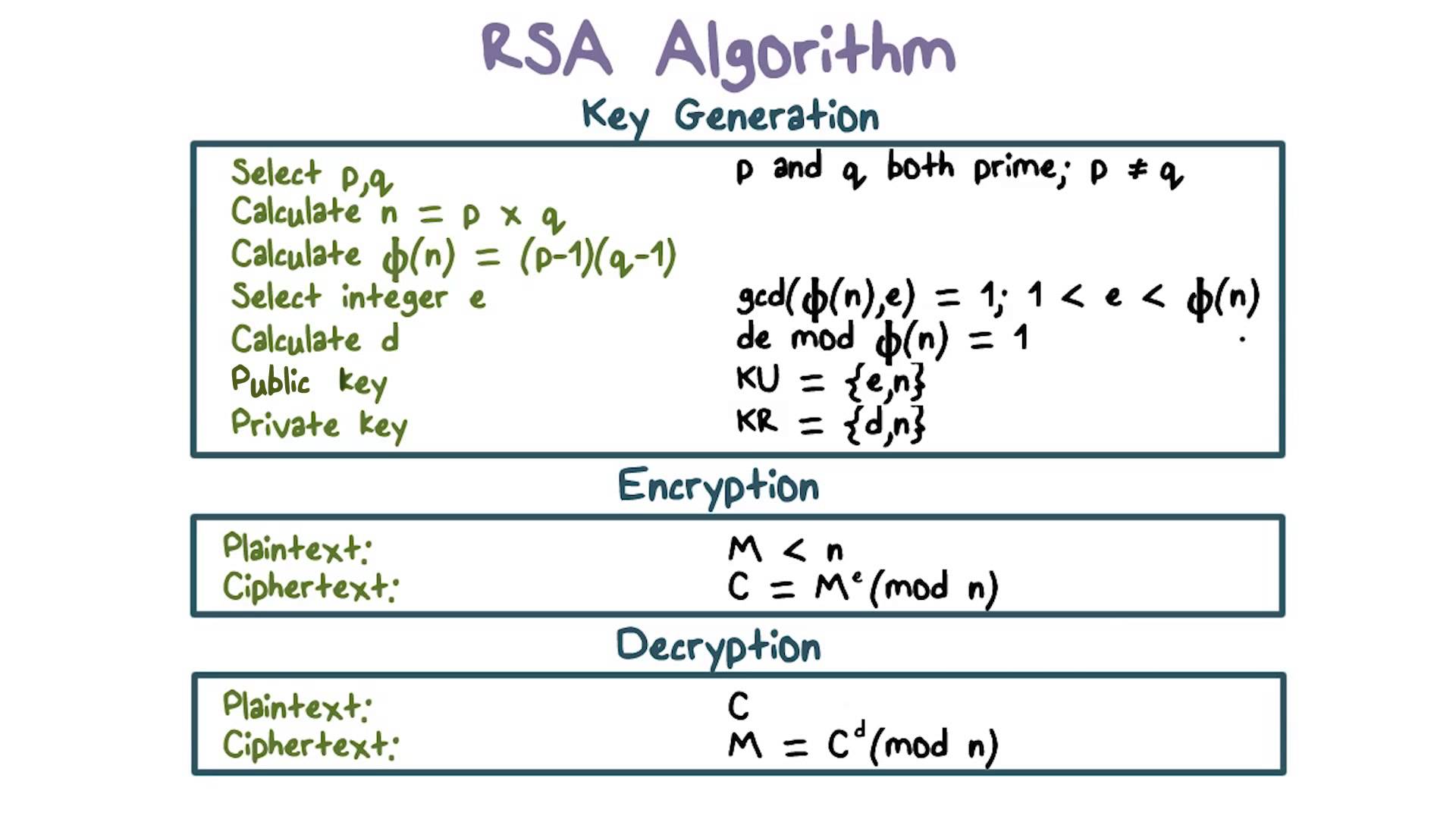
It has slow data transfer rate due to large numbers involved.

It requires third party to verify the reliability of public keys sometimes.

High processing is required at receiver’s end for decryption.

RSA is not meant for encrypting large plaintext because:

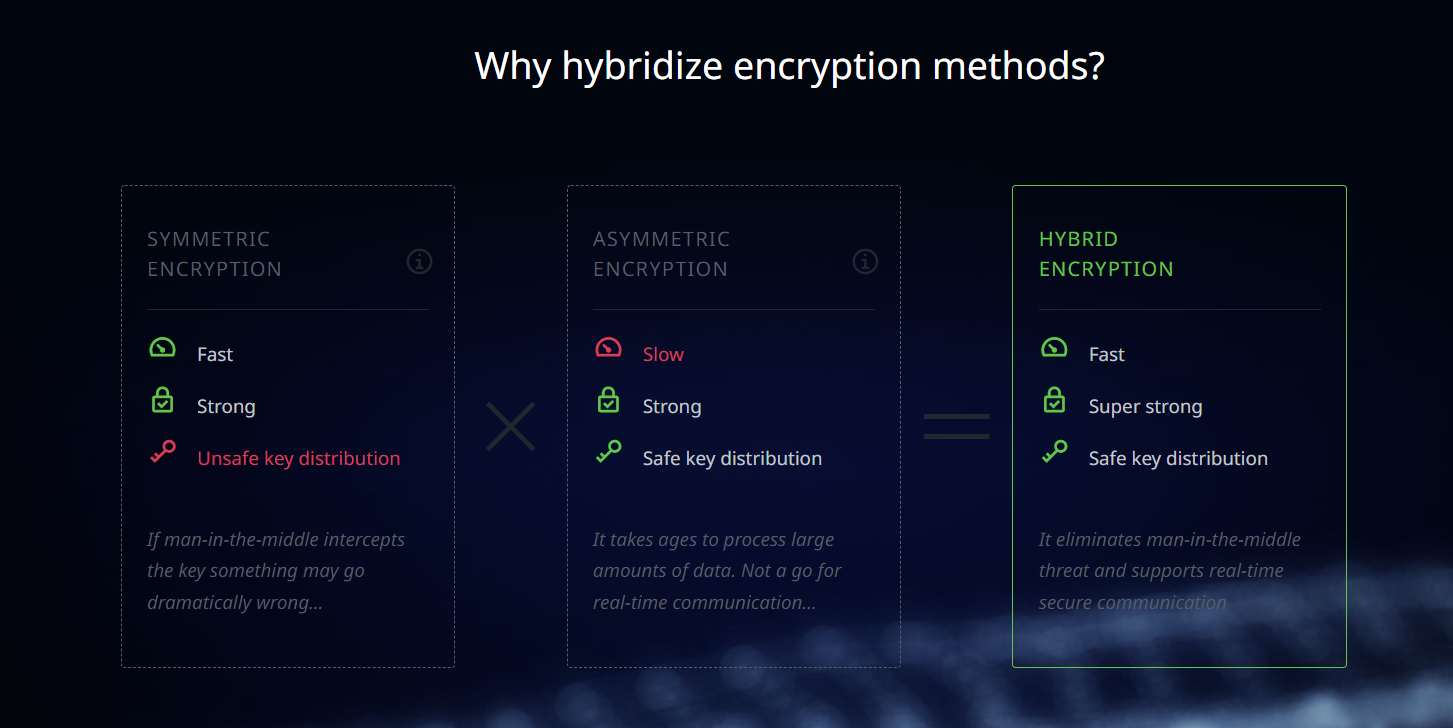
* It is slow.
* Padding makes the ciphertext blocks much longer.



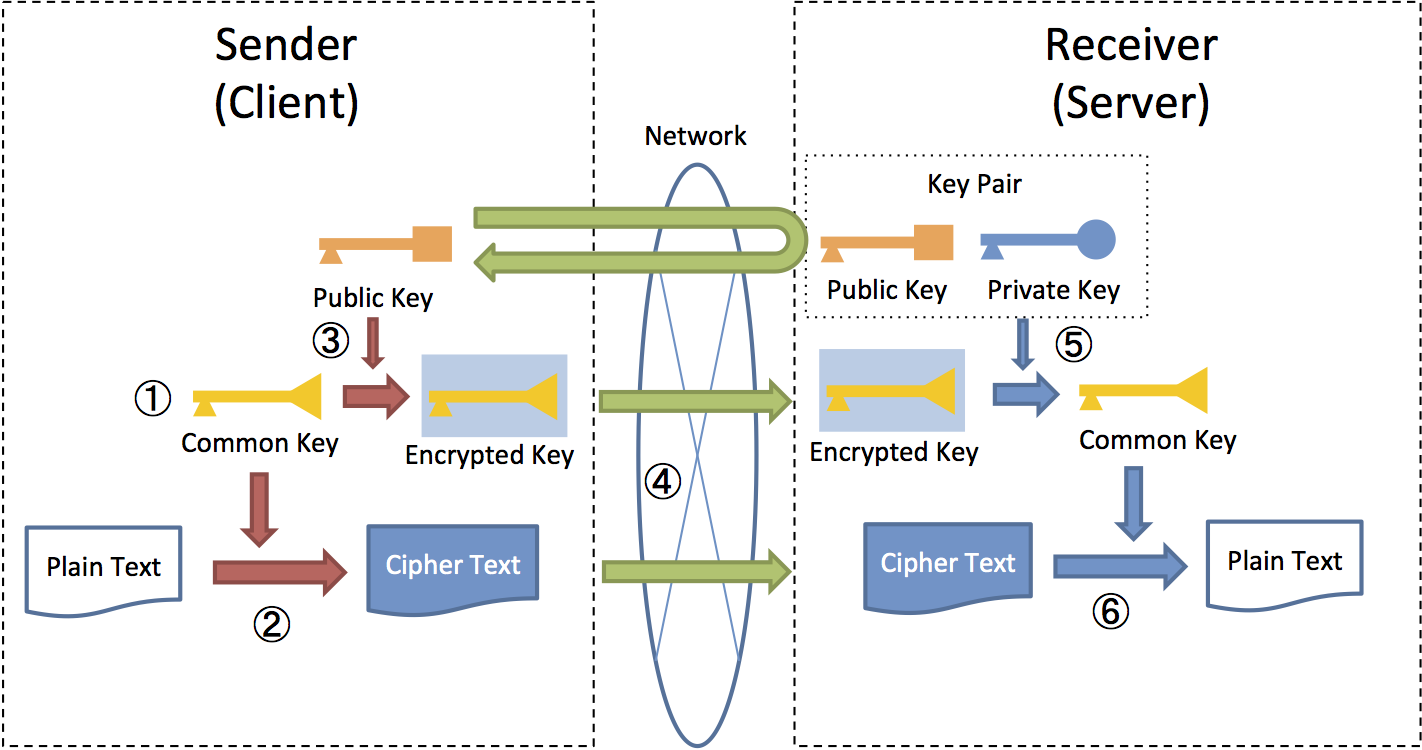
**HYBRID Cryptosystem**

**What Does Hybrid Encryption Mean?**

Hybrid encryption is an approach to encoding and decoding data that blends the speed and convenience of a public asymmetric encryption scheme with the effectiveness of a private symmetric encryption scheme. However, a hybrid approach can be accompanied by an increased risk of implementation flaws that can negate the encryption scheme's usefulness.



**How it Works?**



**Uses of hybrid cryptosystem**

## PGP

## Pretty Good Privacy (PGP) is an encryption program that provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.

## In PGP, the public key technique is used *only* to provide authentication and to securely transport the symmetric keys for a block cipher.

**Tunnelling systems**

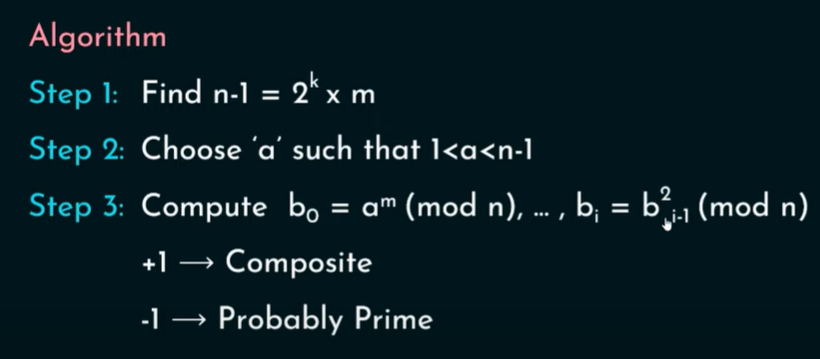
There are a number of network protocols which create secure encrypted tunnels between two endpoints, IPSec, SSL and SSH. Details vary, but in principle they all work the same way:

provide an authentication mechanism, usually some public key system but in some cases just a shared secret

Other examples are encrypted storage in cloud, streaming platforms etc.

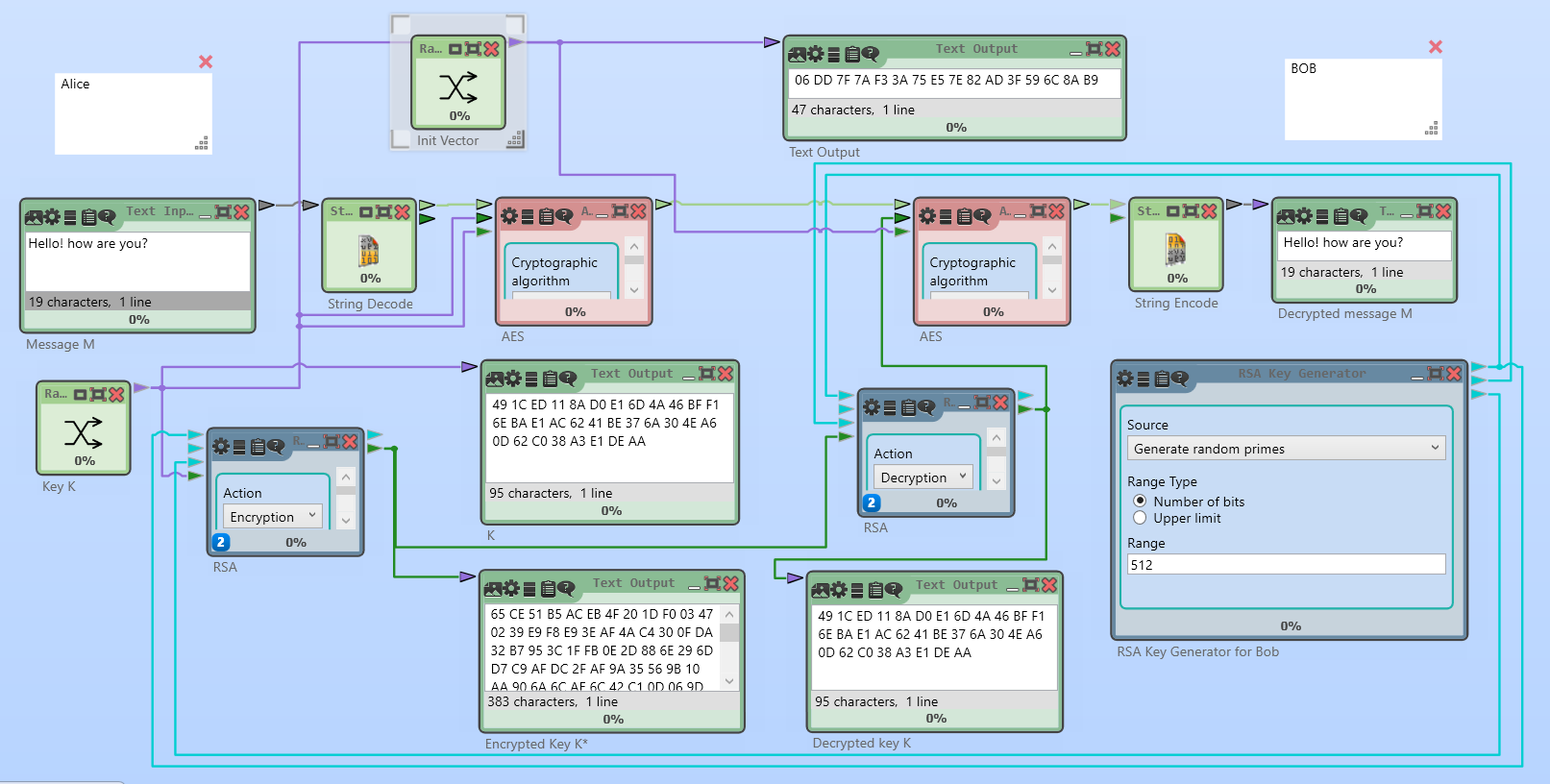
**Miller-Rabin Primality Test:**

Miller Rabin is a fast approach to test primality of the large numbers.



**IMPLEMENTATION**

**Implementation in Cryptool**



**What is an initialization vector?**

An initialization vector (IV) is an arbitrary number that can be used along with a secret key for data encryption. This number, also called a [nonce](https://www.techtarget.com/searchsecurity/definition/nonce), is employed only one time in any session.

**Code implementation**

import random

import sys

import secrets

from Crypto.Cipher import AES

from Crypto import Random

def gcd(a, b):

'''Euclid's algorithm '''

while b != 0:

temp=a % b

a=b

b=temp

return a

def multiplicativeInverse(a, b):

"""Euclid's extended algorithm"""

x = 0

y = 1

lx = 1

ly = 0

oa = a

ob = b

while b != 0:

q = a // b

(a, b) = (b, a % b)

(x, lx) = ((lx - (q \* x)), x)

(y, ly) = ((ly - (q \* y)), y)

if lx < 0:

lx += ob

if ly < 0:

ly += oa

return lx

def generatePrime(keysize):

while True:

num = random.randrange(2\*\*(keysize-1), 2\*\*(keysize))

if isPrime(num):

return num

def isPrime(num):

if (num < 2):

return False # 0, 1, and negative numbers are not prime

lowPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,

97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191,

193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293,

307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419,

421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541,

547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653,

659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787,

797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919,

929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]

if num in lowPrimes:

return True

for prime in lowPrimes:

if (num % prime == 0):

return False

return millerRabin(num)

def millerRabin(n, k = 7):

if n < 6:

return [False, False, True, True, False, True][n]

elif n & 1 == 0:

return False

else:

s, d = 0, n - 1

while d & 1 == 0:

s, d = s + 1, d >> 1

for a in random.sample(range(2, min(n - 2, sys.maxsize)), min(n - 4, k)):

x = pow(a, d, n)

if x != 1 and x + 1 != n:

for r in range(1, s):

x = pow(x, 2, n)

if x == 1:

return False

elif x == n - 1:

a = 0

break

if a:

return False

return True

def KeyGeneration(size=8):

#1)Generate 2 large random primes p,q (same size)

p=generatePrime(size)

q=generatePrime(size)

if not (isPrime(p) and isPrime(q)):

raise ValueError('Both numbers must be prime.')

elif p == q:

raise ValueError('p and q cannot be equal')

#2)compute n=pq and phi=(p-1)(q-1)

n = p \* q

phi = (p-1) \* (q-1)

#3) select random integer "e" (1<e<phi) such that gcd(e,phi)=1

e = random.randrange(1, phi)

g = gcd(e, phi)

while g != 1:

e = random.randrange(1, phi)

g = gcd(e, phi)

#4)Use Extended Euclid's Algorithm to compute another unique integer "d" (1<d<phi) such that e.d≡1(mod phi)

d = multiplicativeInverse(e, phi)

#5)Return public and private keys

#Public key is (e, n) and private key is (d, n)

return ((n, e), (d, n))

def encrypt(pk, plaintext):

#1) obtain (n,e)

n, e = pk

#2)message space [0,n-1]

#3)compute c=m^e(mod n)

c = [(ord(char) \*\* e) % n for char in plaintext]

print(c)

#4) send "C" to the other party

return c

def decrypt(pk, ciphertext):

d, n = pk

#5)m=c^d (mod n)

m = [chr((char \*\* d) % n) for char in ciphertext]

return m

def encryptAES(cipherAESe,plainText):

return cipherAESe.encrypt(plainText.encode("utf-8"))

def decryptAES(cipherAESd,cipherText):

dec= cipherAESd.decrypt(cipherText).decode('utf-8')

return dec

def main():

#To encrypt a message addressed to Alice in a hybrid crypto-system, Bob does the following:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Welcome to the hybrid cryptographic scheme demostration...")

print("We're going to encrypt and decrypt a message using AES and RSA")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#To encrypt a message addressed to Alice in a hybrid crypto-system, Bob does the following:

#1. Obtains Alice’s public key.

print("Genering RSA public and Privite keys......")

pub,pri=KeyGeneration()

#2. Generates a fresh symmetric key for the data encapsulation scheme.

print("Genering AES symmetric key......")

key = secrets.token\_hex(16)

print("AES Symmetric Key: ")

print(key)

KeyAES=key.encode('utf-8')

#3. Encrypts the message under the data encapsulation scheme, using the symmetric key just generated.

plainText = input("Enter the message: ")

cipherAESe = AES.new(KeyAES,AES.MODE\_GCM)

nonce = cipherAESe.nonce

print("Encrypting the message with AES......")

cipherText=encryptAES(cipherAESe,plainText)

print("AES cypher text: ")

print(cipherText)

#4. Encrypt the symmetric key under the key encapsulation scheme, using Alice’s public key.

cipherKey=encrypt(pub,key)

print("Encrypting the AES symmetric key with RSA......")

print("Encryted AES symmetric key")

print(cipherKey)

#5. Send both of these encryptions to Alice.

#Sending.........

#To decrypt this hybrid cipher-text, Alice does the following:

#1. Uses her private key to decrypt the symmetric key contained in the key encapsulation segment.

decriptedKey=''.join(decrypt(pri,cipherKey))

print("Decrypting the AES Symmetric Key...")

print("AES Symmetric Key:")

print(decriptedKey)

#2. Uses this symmetric key to decrypt the message contained in the data encapsulation segment.

decriptedKey=decriptedKey.encode('utf-8')

cipherAESd = AES.new(decriptedKey, AES.MODE\_GCM, nonce=nonce)

decrypted=decryptAES(cipherAESd,cipherText)

print("Decrypting the message using the AES symmetric key.....")

print("decrypted message: ")

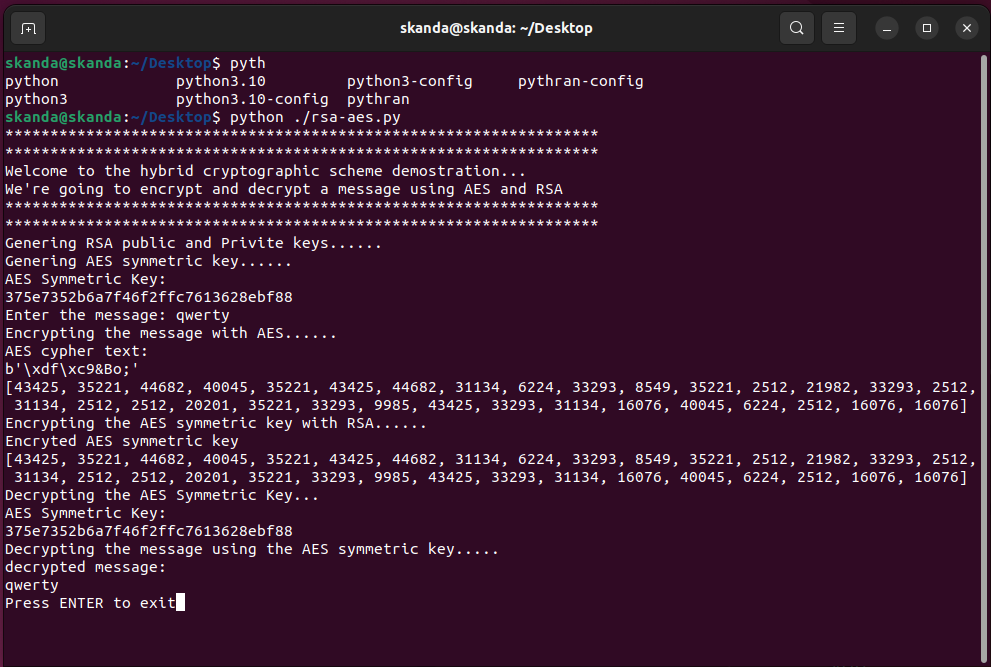
print(decrypted)

input('Press ENTER to exit')

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT**



**Conclusion**

RSA-AES hybrid algorithm takes the best properties of both RSA and AES and creates a very secure algorithm for both key sharing and data sharing. Also, it is faster than many other hybrid algorithms.

