**Python Implementation:**

**RSA Encryption and Decryption**

**REPORT**

**Priaranjan V**

**CS22B1044**

**Implementation**

The RSA encryption algorithm is based on modular arithmetic and number theory. The implementation consists of the following key steps:

* Random selection of two distinct prime numbers p and q
* Computation of n = p \* q and Euler's totient ϕ(n) = (p-1)\*(q-1)
* Selection of public exponent e such that gcd(e, ϕ(n)) = 1
* Computation of private exponent d such that e\*d ≡ 1 (mod ϕ(n))

Encryption and decryption follow the standard RSA equations:

* Ciphertext = M^e mod n
* Plaintext = C^d mod n

The key functions implemented include:

* GCD and Extended Euclidean Algorithm for computing modular inverse
* Modular exponentiation
* Character-wise encryption and decryption

**Code**

import random

import tkinter as tk

from tkinter import messagebox, scrolledtext

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

def xgcd(a, b):

x, old\_x = 0, 1

y, old\_y = 1, 0

while b != 0:

quotient = a // b

a, b = b, a - quotient \* b

old\_x, x = x, old\_x - quotient \* x

old\_y, y = y, old\_y - quotient \* y

return a, old\_x, old\_y

def mod\_pow(base, exponent, mod):

result = 1

base %= mod

while exponent > 0:

if exponent % 2 == 1:

result = (result \* base) % mod

exponent //= 2

base = (base \* base) % mod

return result

def choose\_keys():

with open('primes-to-100k.txt', 'r') as f:

primes = list(map(int, f.read().splitlines()))

filtered\_primes = [p for p in primes if p >= 17]

while True:

p = random.choice(filtered\_primes)

q = random.choice(filtered\_primes)

if p != q:

n = p \* q

if n >= 256:

break

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

while True:

e = random.randrange(2, phi)

if gcd(e, phi) == 1:

break

\_, d, \_ = xgcd(e, phi)

if d < 0:

d += phi

return p, q, n, phi, e, d

def encrypt\_char(char, e, n):

return mod\_pow(ord(char), e, n)

def decrypt\_char(cipher, d, n):

return chr(mod\_pow(cipher, d, n))

class RSAApp:

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

self.root = root

self.root.title("RSA Encrypt/Decrypt")

self.p, self.q, self.n, self.phi, self.e, self.d = choose\_keys()

# Title

tk.Label(root, text="RSA Encryption and Decryption", font=('Arial', 14, 'bold')).pack(pady=10)

# Show key info

tk.Button(root, text="Show Key Details", command=self.show\_keys).pack()

# Input message

tk.Label(root, text="Enter message to encrypt:").pack()

self.message\_entry = tk.Entry(root, width=50)

self.message\_entry.pack(pady=5)

# Encrypt button

tk.Button(root, text="Encrypt", command=self.encrypt\_message).pack(pady=5)

# Encrypted output

tk.Label(root, text="Encrypted message (as list of integers):").pack()

self.encrypted\_box = scrolledtext.ScrolledText(root, height=4, width=60)

self.encrypted\_box.pack(pady=5)

# Decrypt button

tk.Button(root, text="Decrypt", command=self.decrypt\_message).pack(pady=5)

# Decrypted output

tk.Label(root, text="Decrypted message:").pack()

self.decrypted\_label = tk.Label(root, text="", font=('Arial', 12, 'bold'))

self.decrypted\_label.pack(pady=10)

def encrypt\_message(self):

msg = self.message\_entry.get()

if not msg:

messagebox.showwarning("Input Error", "Please enter a message to encrypt.")

return

self.encrypted = [encrypt\_char(c, self.e, self.n) for c in msg]

self.encrypted\_box.delete("1.0", tk.END)

self.encrypted\_box.insert(tk.END, str(self.encrypted))

def decrypt\_message(self):

if not hasattr(self, 'encrypted') or not self.encrypted:

messagebox.showerror("Decryption Error", "No encrypted message found.")

return

decrypted = ''.join([decrypt\_char(val, self.d, self.n) for val in self.encrypted])

self.decrypted\_label.config(text=decrypted)

def show\_keys(self):

key\_info = (

f"p = {self.p}\n"

f"q = {self.q}\n"

f"n = {self.n}\n"

f"phi = {self.phi}\n"

f"e = {self.e}\n"

f"d = {self.d}"

)

messagebox.showinfo("RSA Key Details", key\_info)

# Run the app

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

root = tk.Tk()

app = RSAApp(root)

root.mainloop()

**Results**

The implementation was tested with example plaintexts. The ciphertext obtained through encryption and the restored plaintext after decryption matched correctly, validating the accuracy of the RSA system.



