Field of computer Science & Engineering

Discrete Mathematics

GALALA UNIVERSITY
Powered by Arizona State University®

Project Title *(English)*	**RSA Using Chinese Random Theory**
Team	Abdelrahman Atef Saad Osheba 222100946 Ahmed Ashraf Mohammed Abo-el-ela 222100032 Ahmed Alaa eldin Moustafa 222100123 Adham Mohammed el-saied El-wakel 222100195 Youssef Hussieny 222101943 Youssef Gamal 222101929 Youssef Essam 222101958 Aser Mohammed Ali 222102487 Abdelrahman Gamal 222100933
Field	Computer science and Engineering
Program	Artificial Intelligence Science, Biomedical Informatics, Computer Science
Instructor	**Dr. Mohamed Abd Elaziz**
This part for instructor: Notice	**Degree**
Date of Submission	

Field of computer Science & Engineering

Discrete Mathematics

Contents

Abstract:

China carries the name of the world's oldest reminder issues. The Chinese Remainder Theorem served as the foundation for a lot of sciences including astronomy, architecture, commerce, and calendar computing difficulties. The theorem has sophisticated applications nowadays in numerous mathematical fields. and wide-ranging uses in coding, cryptography, and computers. A great illustration of how mathematics, which first appeared in the third century AC, has evolved and is still useful in the modern era is the Chinese Remainder Theorem.

And we will discuss it is application The RSA Cryptosystem

The RSA cryptosystem is commonly used to authenticate digital signatures and protect data such as customer information and transaction data. It can be used to create key exchanges that establish a secure, encrypted communication channel. It has numerous applications in banking, telecommunications and ecommerce

RSA encryption algorithm steps :

->Step 1 (Key Generating Process)

->Step 2 (Encryption Process)

->Step 3 (Decryption Process)

Introduction and related work:

The RSA cryptosystem, named after its inventors Rivest, Shamir, and Adleman, is a foundational public-key cryptographic system widely used for secure data transmission. Introduced in 1978, RSA is based on the mathematical complexity of factoring large integers, a problem that remains computationally infeasible to solve efficiently with current technology. This system provides both encryption and digital signature functionalities, ensuring confidentiality, data integrity, and authentication in various applications such as secure communications, digital certificates, and electronic commerce.

Public-Key Cryptography

RSA's introduction marked a significant advancement in cryptographic techniques, moving away from symmetric-key cryptography, which requires the secure exchange of a shared secret key. Public-key cryptography, also known as asymmetric cryptography, involves a pair of keys: a public key, which is openly distributed, and a private key, which remains confidential. This paradigm shift allows secure communication without the need for a prior shared secret, simplifying key management in large-scale networks.

Mathematical Foundations

The security of RSA relies on the difficulty of the integer factorization problem. Given a large composite number, the task of determining its prime factors is computationally intensive. RSA leverages this by generating keys through the multiplication of two large prime numbers. The private key, derived from these primes, remains secure as long as the factorization of the composite number remains infeasible. Research into integer factorization algorithms, such as the quadratic sieve and the general number field sieve, continues to explore potential vulnerabilities in RSA's security assumptions.

Objectives:

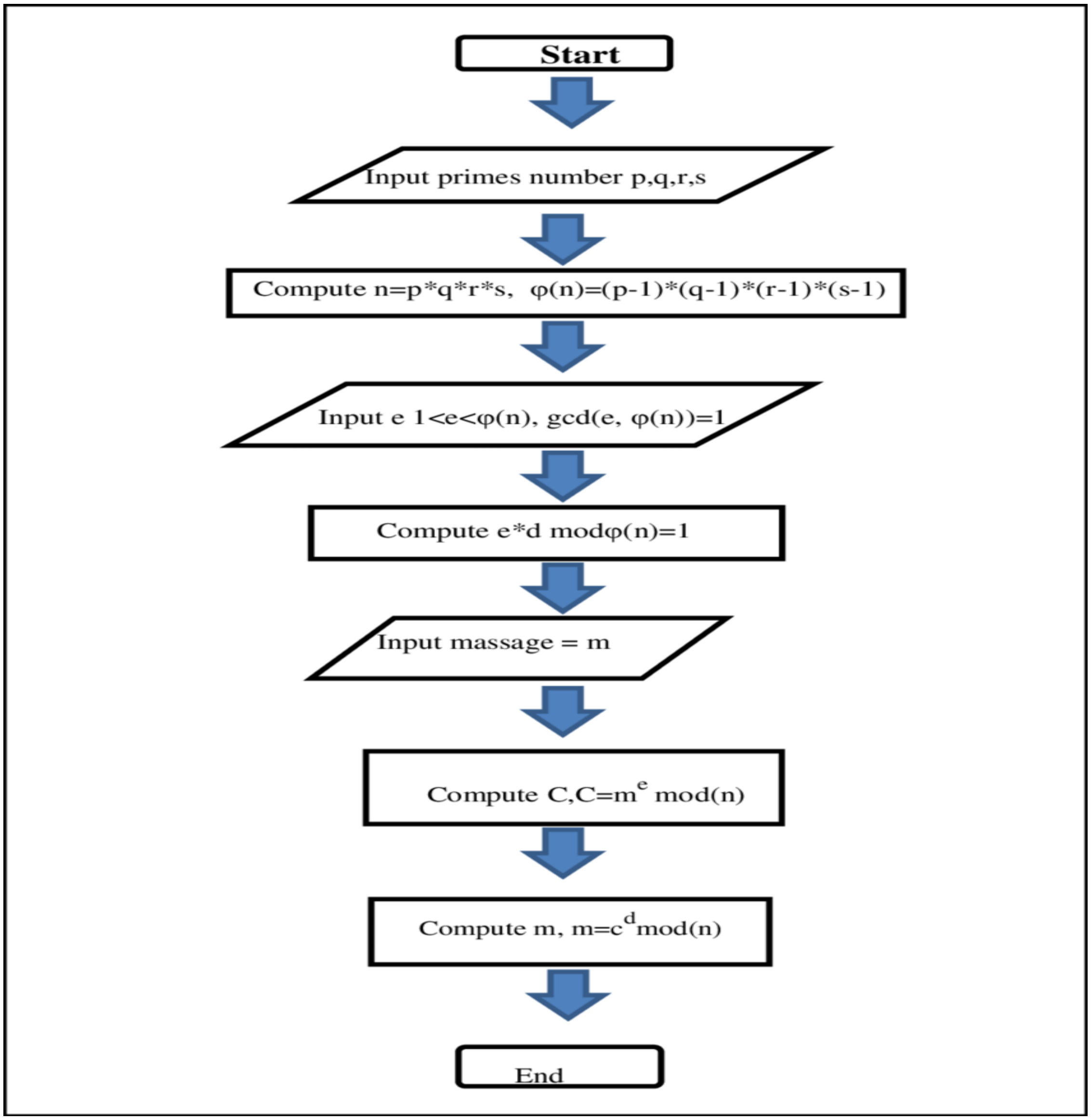
To Understand the Mathematical Foundations: Delve into the mathematical principles underlying RSA, including the concepts of prime factorization, modular arithmetic, and number theory, to understand why RSA is considered secure and how it operates.

To Explore Key Generation, Encryption, and Decryption Processes: Investigate the detailed processes involved in generating RSA keys, and the mechanisms of encryption and decryption, to comprehend the practical implementation and use of RSA in securing data.

To Evaluate Security Aspects: Analyze the security strengths and vulnerabilities of RSA, focusing on the computational difficulty of factoring large integers and the potential threats posed by advances in factorization algorithms and quantum computing.

System Analysis and Design:

RSA Process



Implementation and Outputs:

Implementation

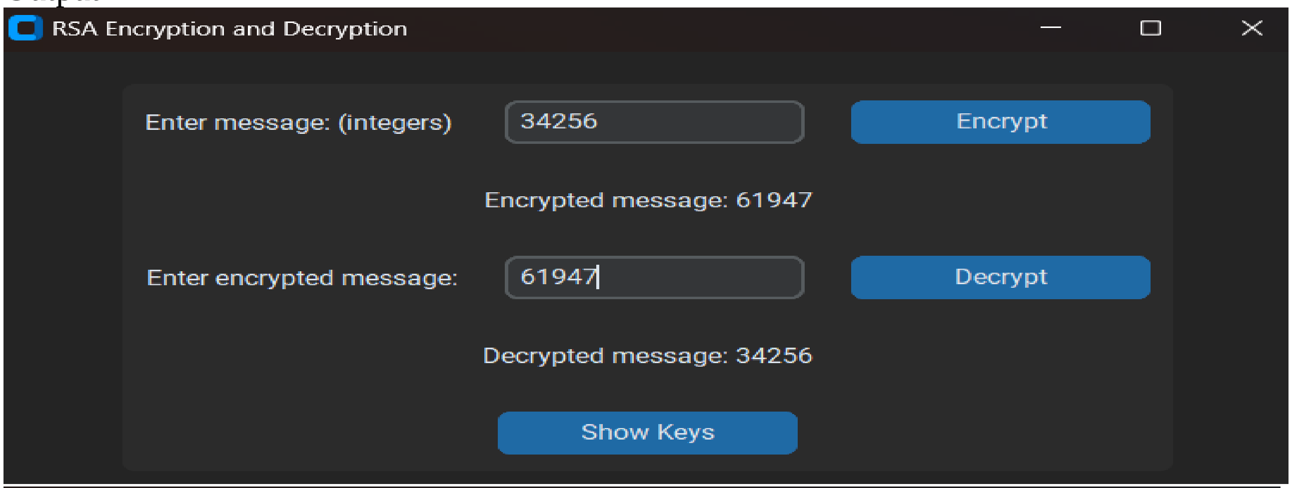
```
import customtkinter as ctk
from tkinter import messagebox
import random
import matplotlib.pyplot as plt

def mod_exp(base, exp, mod):
    result = 1
    base = base % mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        exp = exp >> 1
        base = (base * base) % mod
    return result

def is_prime(n, k=5):
    if n <= 1 or n == 4:
        return False
    if n <= 3:
        return True

    d = n - 1
    while d % 2 == 0:
        d //= 2

    for _ in range(k):
        a = 2 + random.randint(0, n - 4)
        x = mod_exp(a, d, n)
        if x == 1 or x == n - 1:
            continue
        prime = False
        while d != n - 1:
            x = (x * x) % n
            d *= 2
            if x == 1:
                return False
            if x == n - 1:
                prime = True
                break
        if not prime:
            return False
    return True

# Function to compute the greatest common divisor
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Function to find the modular inverse
```

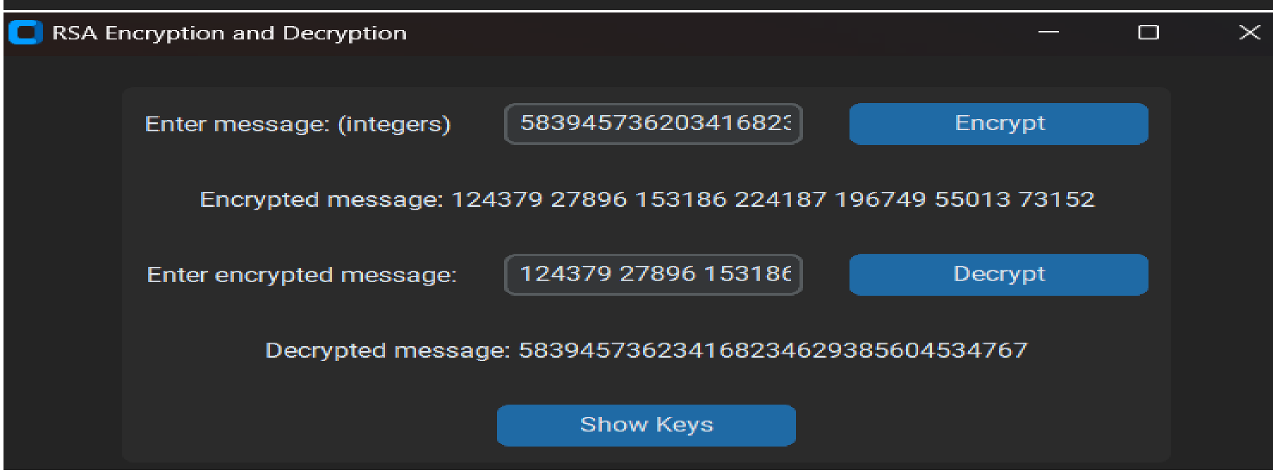
```python
def mod_inverse(e, phi):
    m0, x0, x1 = phi, 0, 1
    while e > 1:
        q = e // phi
        phi, e = e % phi, phi
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

# Function to generate a prime number
def generate_prime(bit_length):
    while True:
        prime = random.getrandbits(bit_length)
        if is_prime(prime):
            return prime

# Function to encrypt a message
def encrypt(message, e, n):
    return mod_exp(message, e, n)

# Function to decrypt a message
def decrypt(encrypted_message, d, n):
    return mod_exp(encrypted_message, d, n)

# Function to handle encryption
def encrypt_message():
    try:
        message = entry_message.get()
        message_chunks = [int(message[i:i+chunk_size]) for i in range(0,
len(message), chunk_size)]
        encrypted_chunks = [encrypt(chunk, e, n) for chunk in message_chunks]
        encrypted_message = ' '.join(map(str, encrypted_chunks))
        label_encrypted.configure(text=f"Encrypted message: {encrypted_message}")
        encrypted_map[encrypted_message] = message
    except ValueError:
        messagebox.showerror("Error", "Please enter a valid integer message")

# Function to handle decryption
def decrypt_message():
    try:
        encrypted_message = entry_encrypted.get()
        encrypted_chunks = list(map(int, encrypted_message.split()))
        decrypted_chunks = [decrypt(chunk, d, n) for chunk in encrypted_chunks]
        decrypted_message = ''.join(map(str, decrypted_chunks))
        label_decrypted.configure(text=f"Decrypted message: {decrypted_message}")
    except ValueError:
        messagebox.showerror("Error", "Please enter a valid integer encrypted
message")

# Function to plot RSA analysis
def plot_rsa_analysis(p, q, e, d, n):
    plt.figure(figsize=(10, 6))

    # Plotting prime numbers p and q
    plt.subplot(2, 2, 1)
```

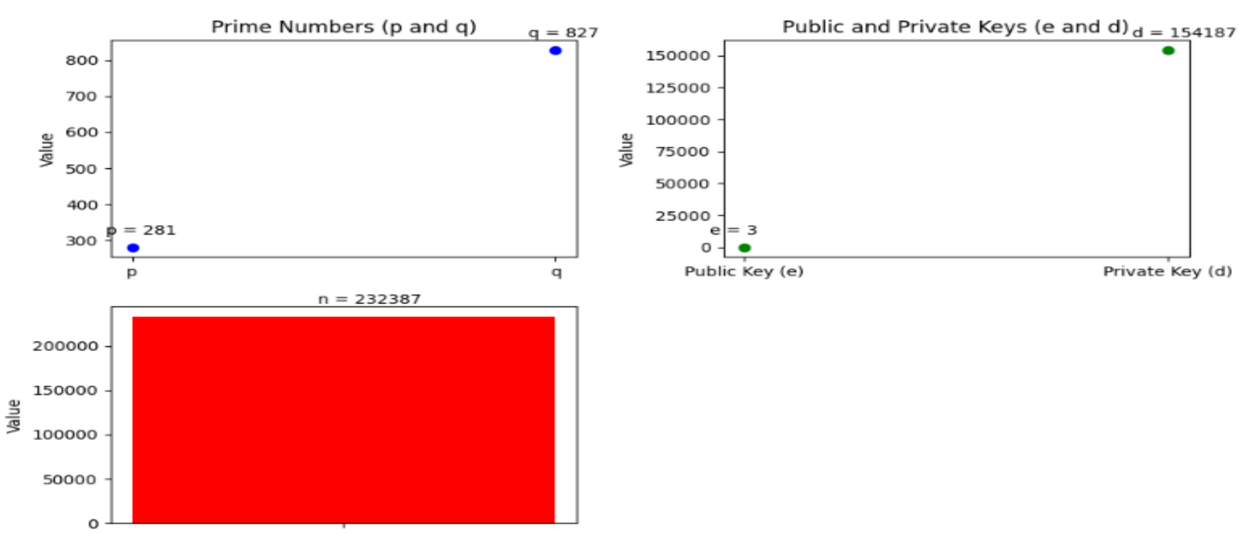
```
    plt.title('Prime Numbers (p and q)')
    plt.scatter(['p', 'q'], [p, q], color='blue')
    plt.ylabel('Value')
    for i, txt in enumerate([f"p = {p}", f"q = {q}"]):
        plt.annotate(txt, (['p', 'q'][i], [p, q][i]), xytext=(-15, 10), textcoords='offset points')

    # Plotting public and private keys (e and d)
    plt.subplot(2, 2, 2)
    plt.title('Public and Private Keys (e and d)')
    plt.scatter(['Public Key (e)', 'Private Key (d)'], [e, d], color='green')
    plt.ylabel('Value')
    for i, txt in enumerate([f"e = {e}", f"d = {d}"]):
        plt.annotate(txt, (['Public Key (e)', 'Private Key (d)'][i], [e, d][i]), xytext=(-20, 10), textcoords='offset points')

    # Plotting modulus n
    plt.subplot(2, 2, 3)
    plt.bar(['n'], [n], color='red')
    plt.ylabel('Value')
    plt.annotate(f"n = {n}", ('n', n), xytext=(-15, 10), textcoords='offset points')

    plt.tight_layout()
    plt.show()

# Function to handle the plotting button click
def show_keys():
    plot_rsa_analysis(p, q, e, d, n)

# Seed the random number generator
random.seed()

# Generate RSA keys
bit_length = 10  # Adjust the bit length for prime generation as needed

p = generate_prime(bit_length)
q = generate_prime(bit_length)
n = p * q
phi = (p - 1) * (q - 1)
e = 2
while gcd(e, phi) != 1:
    e += 1
d = mod_inverse(e, phi)

# Determine the chunk size
chunk_size = len(str(n)) - 1

# Dictionary to store encrypted messages and their original values
encrypted_map = {}

# Initialize the application
app = ctk.CTk()
```

```
# Set the theme (optional)
ctk.set_appearance_mode("dark")  # Modes: "System" (default), "Dark", "Light"
ctk.set_default_color_theme("blue")  # Themes: "blue" (default), "green", "dark-blue"

app.title("RSA Encryption and Decryption")

frame = ctk.CTkFrame(master=app)
frame.pack(pady=20, padx=60, fill="both", expand=True)

label_message = ctk.CTkLabel(master=frame, text="Enter message: (integers) ")
label_message.grid(row=0, column=0, pady=10, padx=10)
entry_message = ctk.CTkEntry(master=frame)
entry_message.grid(row=0, column=1, pady=10, padx=10)
button_encrypt = ctk.CTkButton(master=frame, text="Encrypt", command=encrypt_message)
button_encrypt.grid(row=0, column=2, pady=10, padx=10)
label_encrypted = ctk.CTkLabel(master=frame, text="")
label_encrypted.grid(row=1, column=0, columnspan=3, pady=10, padx=10)

label_encrypted_entry = ctk.CTkLabel(master=frame, text="Enter encrypted message:")
label_encrypted_entry.grid(row=2, column=0, pady=10, padx=10)
entry_encrypted = ctk.CTkEntry(master=frame)
entry_encrypted.grid(row=2, column=1, pady=10, padx=10)
button_decrypt = ctk.CTkButton(master=frame, text="Decrypt", command=decrypt_message)
button_decrypt.grid(row=2, column=2, pady=10, padx=10)
label_decrypted = ctk.CTkLabel(master=frame, text="")
label_decrypted.grid(row=3, column=0, columnspan=3, pady=10, padx=10)

button_show_keys = ctk.CTkButton(master=frame, text="Show Keys", command=show_keys)
button_show_keys.grid(row=4, column=0, columnspan=3, pady=10, padx=10)

app.mainloop()
```

Output


RSA Encryption and Decryption
Enter message: (integers)
34256
Encrypt
Encrypted message: 61947
Enter encrypted message:
61947
Decrypt
Decrypted message: 34256
Show Keys


RSA Encryption and Decryption
Enter message: (integers)
58394573620341682
Encrypt
Encrypted message: 124379 27896 153186 224187 196749 55013 73152
Enter encrypted message:
124379 27896 15318
Decrypt
Decrypted message: 583945736234168234629385604534767
Show Keys


Prime Numbers (p and q)
q = 827
p = 281
Value
800
700
600
500
400
300
p
q
Public and Private Keys (e and d)
d = 154187
e = 3
150000
125000
100000
75000
50000
25000
0
Public Key (e)
Private Key (d)
n = 232387
200000
150000
100000
50000
0
n

Conclusion:

The RSA cryptosystem has been a cornerstone of modern cryptographic practices, enabling secure digital communication and transactions. Despite its robustness, the evolving landscape of cryptographic research continues to explore new methods and technologies to enhance security and efficiency. Understanding RSA's principles and related advancements remains crucial for developing and maintaining secure systems in the digital age.

References:

R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.

D. J. Bernstein, "Introduction to post-quantum cryptography," in *Springer eBooks*, 2009, pp. 1–14. doi: 10.1007/978-3-540-88702-7_1.

C. S. Jackson, "The Chinese remainder theorem," *OPUS Open Portal to University Scholarship*. https://opus.govst.edu/capstones_math/2/

"Handbook of Applied Cryptography." https://cacr.uwaterloo.ca/hac/