Project Outline: SecureText

Objectives:

1. Develop a text encryption and decryption application.
2. Ensure the security and integrity of textual data.
3. Provide a user-friendly interface for seamless encryption and decryption.

Tools and Technologies:

1. Programming Language: Python
2. Cryptographic Libraries: PyCryptodome
3. GUI Framework: Tkinter

Step-by-Step Methodology:

1. Setup the Environment: Install Python: Python Download Install PyCryptodome: bash Copy code pip install pycryptodome
2. Design the Application: Create a simple GUI using Tkinter. Provide input fields for plaintext and ciphertext. Provide buttons for encryption and decryption.
3. Implement Cryptographic Algorithms: Use AES for symmetric encryption. Use RSA for asymmetric encryption. Use SHA for hashing (if needed).

1. Import Libraries:

In [ ]:
```python
from Crypto.Cipher import AES
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
import base64
import tkinter as tk
from tkinter import messagebox
```

2. AES Encryption and Decryption:

In [ ]:
```python
# AES Encryption
def aes_encrypt(plaintext, key):
    cipher = AES.new(key, AES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(plaintext.encode('utf-8'))
    return base64.b64encode(nonce + ciphertext).decode('utf-8')

# AES Decryption
def aes_decrypt(ciphertext, key):
    ciphertext = base64.b64decode(ciphertext)
    nonce = ciphertext[:16]
    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext[16:]).decode('utf-8')
    return plaintext
```

3. RSA Key Generation, Encryption, and Decryption:

In [ ]:

```
# RSA Key Generation
def generate_rsa_keys():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key

# RSA Encryption
def rsa_encrypt(plaintext, public_key):
    public_key = RSA.import_key(public_key)
    ciphertext = public_key.encrypt(plaintext.encode('utf-8'), None)
    return base64.b64encode(ciphertext[0]).decode('utf-8')

# RSA Decryption
def rsa_decrypt(ciphertext, private_key):
    private_key = RSA.import_key(private_key)
    ciphertext = base64.b64decode(ciphertext)
    plaintext = private_key.decrypt(ciphertext).decode('utf-8')
    return plaintext
```

4. GUI Implementation:

In [ ]:

```
class SecureTextApp:
    def __init__(self, root):
        self.root = root
        self.root.title("SecureText Encryption")
        self.root.geometry("400x400")

        self.key_label = tk.Label(root, text="Key:")
        self.key_label.pack()
        self.key_entry = tk.Entry(root, width=50)
        self.key_entry.pack()

        self.plaintext_label = tk.Label(root, text="Plaintext:")
        self.plaintext_label.pack()
        self.plaintext_entry = tk.Entry(root, width=50)
        self.plaintext_entry.pack()

        self.ciphertext_label = tk.Label(root, text="Ciphertext:")
        self.ciphertext_label.pack()
        self.ciphertext_entry = tk.Entry(root, width=50)
        self.ciphertext_entry.pack()

        self.encrypt_button = tk.Button(root, text="Encrypt", command=self.encry
        self.encrypt_button.pack()
        self.decrypt_button = tk.Button(root, text="Decrypt", command=self.decry
        self.decrypt_button.pack()

    def encrypt_text(self):
        plaintext = self.plaintext_entry.get()
        key = self.key_entry.get().encode('utf-8')
        if len(key) < 16:
            key = key.ljust(16, b'\0')
        elif len(key) > 16:
            key = key[:16]
        ciphertext = aes_encrypt(plaintext, key)
        self.ciphertext_entry.delete(0, tk.END)
        self.ciphertext_entry.insert(0, ciphertext)

    def decrypt_text(self):
        ciphertext = self.ciphertext_entry.get()
        key = self.key_entry.get().encode('utf-8')
        if len(key) < 16:
            key = key.ljust(16, b'\0')
        elif len(key) > 16:
            key = key[:16]
        try:
            plaintext = aes_decrypt(ciphertext, key)
            self.plaintext_entry.delete(0, tk.END)
            self.plaintext_entry.insert(0, plaintext)
        except Exception as e:
            messagebox.showerror("Decryption Error", str(e))

if __name__ == "__main__":
    root = tk.Tk()
    app = SecureTextApp(root)
    root.mainloop()
```

Explanation of the Code:

1. Import Libraries:

-> The necessary libraries for cryptographic operations and GUI development are imported.

2. AES Encryption and Decryption Functions:

-> aes_encrypt function takes plaintext and a key, and returns the encrypted ciphertext.
-> aes_decrypt function takes ciphertext and a key, and returns the decrypted plaintext.

3. RSA Key Generation, Encryption, and Decryption Functions:

-> generate_rsa_keys function generates RSA public and private keys.
-> rsa_encrypt function encrypts plaintext using the public key.
-> rsa_decrypt function decrypts ciphertext using the private key.

4. GUI Implementation:

-> A Tkinter GUI is created with input fields for key, plaintext, and ciphertext.
-> Buttons for encryption and decryption are provided, which call respective functions.

Running the Project:

1. Save the provided code in a file named securetext.py.
2. Run the file using the command: python securetext.py
3. The GUI application will open, allowing you to enter a key and plaintext, and perform encryption and decryption.

Outcome:

The SecureText application will provide a user-friendly interface to encrypt and decrypt textual communications using AES encryption. This ensures the confidentiality and integrity of the data, protecting it from unauthorized access and interception.