In this lab, you will create a secure registration and login system using Python. This project will help you understand encryption, hashing, and basic GUI development with Tkinter. You will work with libraries like rsa, hashlib, and base64, and build an application that securely handles user credentials. A template with the GUI has been provided along with placeholder functions.

\* You may need to make a new Replit with the “**tkinter**” template.

**Step 1: Set Up Your Environment**

* Ensure Python is installed on your system.
* Install the rsa library using **pip install rsa**.

**Step 2: Read the Template Code**

* Study the provided template code to understand the basic structure and functionalities.
* Do not modify any of the code above the line “**TODO: Implement the following functions to complete the application.**”

**Step 3: Create Backend Functions**

* Implement register\_user to:
  + Get the username and password from the input fields.
  + Hash the password using SHA-256.
  + Save the username and hashed password to a file using RSA encryption.
  + Show a message box upon successful registration or if the username exists.
* Implement login\_user to:
  + Validate the entered credentials against the stored data.
  + Show a success message and display the user list if the login is successful.
  + Show an error message if the login fails.

**Step 4: Implement File Handling Functions**

* save\_to\_file: Encrypt and save the credentials to 'accounts.txt'.
* check\_username\_exists: Check if a username already exists in the file.
* validate\_account: Validate login credentials against the stored data.
* get\_user\_list: Retrieve and decrypt all usernames from the file.

**Step 5: Encrypt and Decrypt Data**

* Use RSA encryption to encrypt credentials before saving to the file.
* Decrypt data when reading from the file for validation.
* Utilize base64 encoding for handling encrypted data.

**Step 6: Test Your Application**

* Run the application and test different scenarios:
  + Register a new user.
  + Attempt to register with an existing username.
  + Log in with correct and incorrect credentials.
  + View the list of registered usernames after successful login.
* If you finish the project early, add an administrative panel after a successful login that displays the usernames of all stored accounts and provides buttons to remove a selected account and another to change the password of that account. You can also add in a fixed number of allowed login attempts before the program forcefully closes.

**Template**

import os

import rsa

import base64

import hashlib

import tkinter as tk

import tkinter.messagebox

from tkinter import ttk

# Main application window

root = tk.Tk()

root.title("Secure Registration and Login System")

# Notebook for tabs

tab\_control = ttk.Notebook(root)

# Registration tab

reg\_tab = ttk.Frame(tab\_control)

tab\_control.add(reg\_tab, text='Register')

# Login tab

login\_tab = ttk.Frame(tab\_control)

tab\_control.add(login\_tab, text='Login')

tab\_control.pack(expand=1, fill="both")

# Registration widgets

reg\_username\_label = tk.Label(reg\_tab, text="Username")

reg\_username\_label.pack()

reg\_username\_entry = tk.Entry(reg\_tab)

reg\_username\_entry.pack()

reg\_password\_label = tk.Label(reg\_tab, text="Password")

reg\_password\_label.pack()

reg\_password\_entry = tk.Entry(reg\_tab, show="\*")

reg\_password\_entry.pack()

reg\_submit\_button = tk.Button(reg\_tab, text="Register")

reg\_submit\_button.pack()

# Login widgets

login\_username\_label = tk.Label(login\_tab, text="Username")

login\_username\_label.pack()

login\_username\_entry = tk.Entry(login\_tab)

login\_username\_entry.pack()

login\_password\_label = tk.Label(login\_tab, text="Password")

login\_password\_label.pack()

login\_password\_entry = tk.Entry(login\_tab, show="\*")

login\_password\_entry.pack()

login\_submit\_button = tk.Button(login\_tab, text="Login")

login\_submit\_button.pack()

# RSA key generation

global public\_key, private\_key

def get\_rsa\_keys():

    global public\_key, private\_key

    KEY\_SIZE = 2048

    def generate\_keys():

        global public\_key, private\_key

        print("Generating RSA keys...")

        public\_key, private\_key = rsa.newkeys(KEY\_SIZE)

        with open('public\_key.pem', mode='wb') as public\_file:

            public\_file.write(public\_key.save\_pkcs1())

        with open('private\_key.pem', mode='wb') as private\_file:

            private\_file.write(private\_key.save\_pkcs1())

        print("Keys generated!")

    if os.path.exists('public\_key.pem') and os.path.exists('private\_key.pem'):

        try:

            with open('public\_key.pem', mode='rb') as public\_file:

                public\_key = rsa.PublicKey.load\_pkcs1(public\_file.read())

            with open('private\_key.pem', mode='rb') as private\_file:

                private\_key = rsa.PrivateKey.load\_pkcs1(private\_file.read())

            print("Keys loaded successfully!")

        except:

            generate\_keys()

    else:

        generate\_keys()

    return public\_key, private\_key

def display\_user\_list(usernames):

    # Create a new window

    list\_window = tk.Toplevel()

    list\_window.title("User List")

    # Create a listbox widget

    listbox = tk.Listbox(list\_window)

    listbox.pack(fill=tk.BOTH, expand=True)

    # Insert usernames into the listbox

    for username in usernames:

        listbox.insert(tk.END, username)

    # Add a scrollbar

    scrollbar = ttk.Scrollbar(list\_window, orient='vertical', command=listbox.yview)

    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

    listbox.config(yscrollcommand=scrollbar.set)

    # Run the window's main loop

    list\_window.mainloop()

# ======================================================================================================================

"""TODO: Implement the following functions to complete the application.

You will need to:

    \* Run "pip install rsa" in the terminal to install the rsa library

    \* Use the hashlib library to hash the passwords with sha256 (using hashlib.sha256(password.encode()).hexdigest())

    \* Use the RSA keys generated above to encrypt and decrypt the credentials in a file called "accounts.txt"

        \* Use the "rsa.encrypt(text, public\_key)" function to encrypt the credentials

        \* Use the "rsa.decrypt(text, private\_key)" function to decrypt the credentials

        \* To encrypt a str, you will need to encode it first (e.g. "string".encode())

        \* To decrypt a str, you will need to decode it after decrypting (e.g. rsa.decrypt(text, private\_key).decode())

    \* Use the base64 library to encode the encrypted credentials before writing them to the file

        \* Use "base64.b64encode(text).decode()" to encode the encrypted credentials as base64

        \* Use "base64.b64decode(text)" to decode the encrypted credentials from base64

    \* Use the os library to check if the password database file exists (os.path.exists('accounts.txt'))

    \* Use tk.messagebox.showinfo(title="Title", message="Message") to pop up a message box (success)

    \* Use tk.messagebox.showerror(title="Title", message="Message") to pop up an error message box (failure)

    \* Use "reg\_username\_entry.get()" to get the text from the username entry box on the registration tab

    \* Use "reg\_password\_entry.get()" to get the text from the password entry box on the registration tab

    \* Use "login\_username\_entry.get()" to get the text from the username entry box on the login tab

    \* Use "login\_password\_entry.get()" to get the text from the password entry box on the login tab

"""

# Backend Functions

def register\_user():

    # Grab the username and password from the entry boxes, then hash the password with sha256 and call save\_to\_file

    # Pop up a messagebox if account registration is successful or if the username already exists

    ...

    pass

def login\_user():

    # Grab the username and password from the entry boxes,

    # then hash the password with sha256 and compare to the saved hash

    # Pop up a messagebox if login is successful or if the username/password is incorrect

    # If the login is successful, pop up a new window with a listbox containing all the usernames in the file

    # using the display\_user\_list function and your get\_user\_list function

    ...

    pass

def save\_to\_file(username, password\_hash):

    global public\_key, private\_key

    # Encrypt the username and password with RSA

    # Save credentials to 'accounts.txt'; create the file if it doesn't exist, otherwise append a new line to it

    # using "with open('accounts.txt', mode='a') as file:" and "file.write(str)"

    # Pop up a messagebox if account registration is successful or if the username already exists

    # Return True if registration is successful, False otherwise (username already exists)

    # Hint: write the encrypted username and password to the file as a single string, separated by a tab character

    # you'll need to encode the encrypted strings as base64 before writing them to the file

    # e.g., base64.b64encode(encrypted\_username).decode() to encode the encrypted username as base64

    ...

    return False

def check\_username\_exists(username):

    global public\_key, private\_key

    # Check if the username exists in the credentials file

    # Return True if the username exists, False otherwise (username or file don't exist)

    # Hint: you'll need to decode the encrypted username from base64 before comparing it to the username

    # using rsa.decrypt(base64.b64decode(encrypted\_username), private\_key).decode() to decrypt the username

    ...

    return False

def validate\_account(username, attempted\_hash):

    global public\_key, private\_key

    # Read and decrypt credentials from the file and compare to the attempted login

    # Return True if login is successful, False otherwise (incorrect password, or username or file don't exist)

    # Hint: you'll need to decode the encrypted username and password from base64 before comparing them (see above)

    ...

    return False

def get\_user\_list():

    global public\_key, private\_key

    # Read and decrypt credentials from the file and return a list of all the usernames

    # Return an empty list if the file doesn't exist

    ...

    return []

# Bind functions to buttons

reg\_submit\_button.config(command=register\_user)

login\_submit\_button.config(command=login\_user)

# Main loop to run the application

get\_rsa\_keys()

root.mainloop()