

Podpis Cyfrowy

Jakub Jarczak 152072

10.06.2024

Opis działania programu

Program przedstawiony w kodzie jest aplikacją GUI (Graficzny Interfejs Użytkownika) stworzoną przy użyciu biblioteki **tkinter**, która umożliwia użytkownikowi generowanie i weryfikowanie podpisów cyfrowych plików. Do tworzenia podpisów program wykorzystuje generator liczb losowych z histogramów szarych obrazów. Główne funkcje programu obejmują:

1. **Generowanie kluczy RSA** - generowanie kluczy prywatnych i publicznych do podpisywania plików.
2. **Podpisywanie plików** - generowanie podpisu cyfrowego dla wybranych plików przy użyciu klucza prywatnego.
3. **Weryfikacja podpisu** - sprawdzanie ważności podpisu cyfrowego wybranego pliku przy użyciu klucza publicznego.

Użyte biblioteki

- **tkinter**: do tworzenia GUI.
- **cryptography**: do operacji kryptograficznych, takich jak generowanie kluczy RSA, podpisywanie danych oraz weryfikacja podpisów.
- **os**: do operacji na systemie plików.
- **matlab.engine**: do generowania ziarna przy użyciu skryptu MATLAB.
- **random**: do ustawiania ziarna dla generacji kluczy.

Najważniejsze funkcje w kodzie i ich opisy:

Generowanie kluczy RSA - `generate_keys_with_seed`

Funkcja ta generuje parę kluczy RSA (klucz prywatny i publiczny) na podstawie zadanego ziarna (**seed**). Klucze są następnie zapisywane w plikach *private_key.pem* oraz *public_key.pem*.

Wczytywanie kluczy - `load_keys`

Funkcja wczytuje klucze RSA z plików, jeśli istnieją, lub generuje nowe klucze przy użyciu funkcji *generate_seed_with_matlab* i *generate_keys_with_seed*, jeśli pliki kluczy nie są dostępne.

Generowanie ziarna przy użyciu MATLAB - generate_seed_with_matlab

Funkcja ta uruchamia skrypt MATLAB generujący ziarno na podstawie zadanego folderu i zwraca wynik jako liczbę całkowitą.

Dodawanie plików do listy - add_files

Funkcja ta pozwala użytkownikowi wybrać pliki do podpisania i dodaje je do listy wyświetlanej w interfejsie.

Usuwanie wybranych plików z listy - remove_selected_files

Funkcja ta usuwa wybrane pliki z listy plików do podpisania.

Podpisywanie plików - sign_files

Funkcja ta podpisuje wybrane pliki przy użyciu klucza prywatnego. Wygenerowany podpis jest zapisywany w pliku .sig w formacie szesnastkowym.

Weryfikacja podpisu pliku - verify_file_signature

Funkcja ta weryfikuje podpis pliku przy użyciu klucza publicznego. Jeśli podpis jest poprawny, wyświetla stosowną informację, w przeciwnym razie wyświetla błąd.

GUI Setup

Interfejs graficzny aplikacji jest stworzony przy użyciu biblioteki **tkinter**. Składa się z dwóch zakładerek:

1. **Sign File** - do podpisywania plików.
2. **Verify Signature** - do weryfikacji podpisów.

Każda z zakładerek zawiera odpowiednie przyciski i pola do wprowadzania danych, takie jak:

- **Przycisk dodawania plików** - umożliwia użytkownikowi wybór plików do podpisania.
- **Przycisk usuwania plików** - pozwala na usunięcie wybranych plików z listy.
- **Przycisk podpisywania plików** - rozpoczyna proces podpisywania wybranych plików.
- **Pole tekstowe do wyświetlania podpisu** - pokazuje wygenerowany podpis.
- **Przycisk wyboru pliku do weryfikacji** - umożliwia wybór pliku do sprawdzenia podpisu.
- **Pole tekstowe do wyświetlania podpisu weryfikowanego** - pokazuje podpis wybranego pliku.

Kod źródłowy:

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog, messagebox, scrolledtext, Listbox, MULTIPLE
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import padding, rsa
from cryptography.hazmat.backends import default_backend
import os
import matlab.engine
import random

# Function to generate seed using MATLAB script
def generate_seed_with_matlab(folder_path):
    eng = matlab.engine.start_matlab()
    seed = eng.generate_seed(folder_path)
    eng.quit()
    return int(seed)

# Function to generate a new pair of RSA keys (private and public) using a seed
def generate_keys_with_seed(seed):
    random.seed(seed)
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()

    # Save the private key
    with open("private_key.pem", "wb") as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))

    # Save the public key
    with open("public_key.pem", "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

    return private_key, public_key
```

```

# Function to load RSA keys from files or generate new ones using MATLAB
def load_keys():
    if os.path.exists("private_key.pem") and os.path.exists("public_key.pem"):
        with open("private_key.pem", "rb") as f:
            private_key = serialization.load_pem_private_key(
                f.read(),
                password=None,
                backend=default_backend()
            )
        with open("public_key.pem", "rb") as f:
            public_key = serialization.load_pem_public_key(
                f.read(),
                backend=default_backend()
            )
        return private_key, public_key
    else:
        # If keys do not exist, generate new keys using MATLAB
        seed = generate_seed_with_matlab('daisy') # Change 'daisy' to your folder path
        return generate_keys_with_seed(seed)

# Function to add selected files to the listbox
def add_files():
    file_paths = filedialog.askopenfilenames() # Allow multiple file selection
    for file_path in file_paths:
        file_listbox.insert(tk.END, file_path)

# Function to remove selected files from the listbox
def remove_selected_files():
    selected_indices = file_listbox.curselection()
    for index in reversed(selected_indices):
        file_listbox.delete(index)

# Function to sign selected files with the private key
def sign_files(private_key):
    selected_files = file_listbox.get(0, tk.END)
    if not selected_files:
        messagebox.showerror("Error", "No files selected.")
        return

    for file_path in selected_files:
        with open(file_path, 'rb') as f:
            file_data = f.read()

            digest = hashes.Hash(hashes.SHA3_256())
            digest.update(file_data)
            hash_data = digest.finalize()

```

```

signature = private_key.sign(
    hash_data,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA3_256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA3_256()
)

signature_hex = signature.hex()
signature_filename = file_path + '.sig'
with open(signature_filename, 'w') as f: # Save as hex
    f.write(signature_hex)

signature_info.set("Signature saved as: " + signature_filename)
signature_display.delete('1.0', tk.END)
signature_display.insert(tk.END, signature_hex)

messagebox.showinfo("Success", "Files signed successfully.")

# Function to verify the signature of any file with the public key
def verify_file_signature(public_key):
    file_path = filedialog.askopenfilename() # Allow file selection
    if not file_path:
        return

    signature_path = file_path + '.sig'
    if not os.path.exists(signature_path):
        messagebox.showerror("Error", "Signature file not found.")
        verification_info.set("Signature file not found.")
        return

    with open(file_path, 'rb') as f:
        file_data = f.read()

    digest = hashes.Hash(hashes.SHA3_256())
    digest.update(file_data)
    hash_data = digest.finalize()

    with open(signature_path, 'r') as f:
        signature_hex = f.read()

    verify_signature_display.delete('1.0', tk.END)
    verify_signature_display.insert(tk.END, signature_hex)

```

```

try:
    public_key.verify(
        signature,
        hash_data,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA3_256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA3_256()
    )
    messagebox.showinfo("Verification", "Signature is valid.")
    verification_info.set("Signature is valid.")
except Exception as e:
    messagebox.showerror("Verification", "Signature is invalid.")
    verification_info.set("Signature is invalid.")

# GUI Setup
root = tk.Tk()
root.title("Digital Signature Tool")

private_key, public_key = load_keys()

tab_control = ttk.Notebook(root)
tab1 = ttk.Frame(tab_control)
tab2 = ttk.Frame(tab_control)

tab_control.add(tab1, text='Sign File')
tab_control.add(tab2, text='Verify Signature')
tab_control.pack(expand=1, fill='both')

# Tab 1: Sign File
sign_instructions = tk.Label(tab1, text="Select files to sign:")
sign_instructions.pack(pady=10)

add_files_button = tk.Button(tab1, text="Add Files", command=add_files)
add_files_button.pack(pady=5)

file_listbox = Listbox(tab1, selectmode=MULTIPLE)
file_listbox.pack(pady=5, fill=tk.BOTH, expand=True)

remove_files_button = tk.Button(tab1, text="Remove Selected Files", command=remove_selected_files)
remove_files_button.pack(pady=5)

```

```

sign_files_button = tk.Button(tab1, text="Sign Files", command=lambda: sign_files(private_key))
sign_files_button.pack(pady=5)

signature_info = tk.StringVar()
signature_info_label = tk.Label(tab1, textvariable=signature_info)
signature_info_label.pack(pady=10)

key_preview_label = tk.Label(tab1, text="Podgląd Klucza:")
key_preview_label.pack(pady=5)

signature_display = scrolledtext.ScrolledText(tab1, height=10, width=50)
signature_display.pack(pady=10)

# Tab 2: Verify Signature
verify_instructions = tk.Label(tab2, text="Select a file to verify:")
verify_instructions.pack(pady=10)

select_verify_file_button = tk.Button(tab2, text="Select File", command=lambda: verify_file_signature(public_key))
select_verify_file_button.pack(pady=5)

verification_info = tk.StringVar()
verification_info_label = tk.Label(tab2, textvariable=verification_info)
verification_info_label.pack(pady=10)

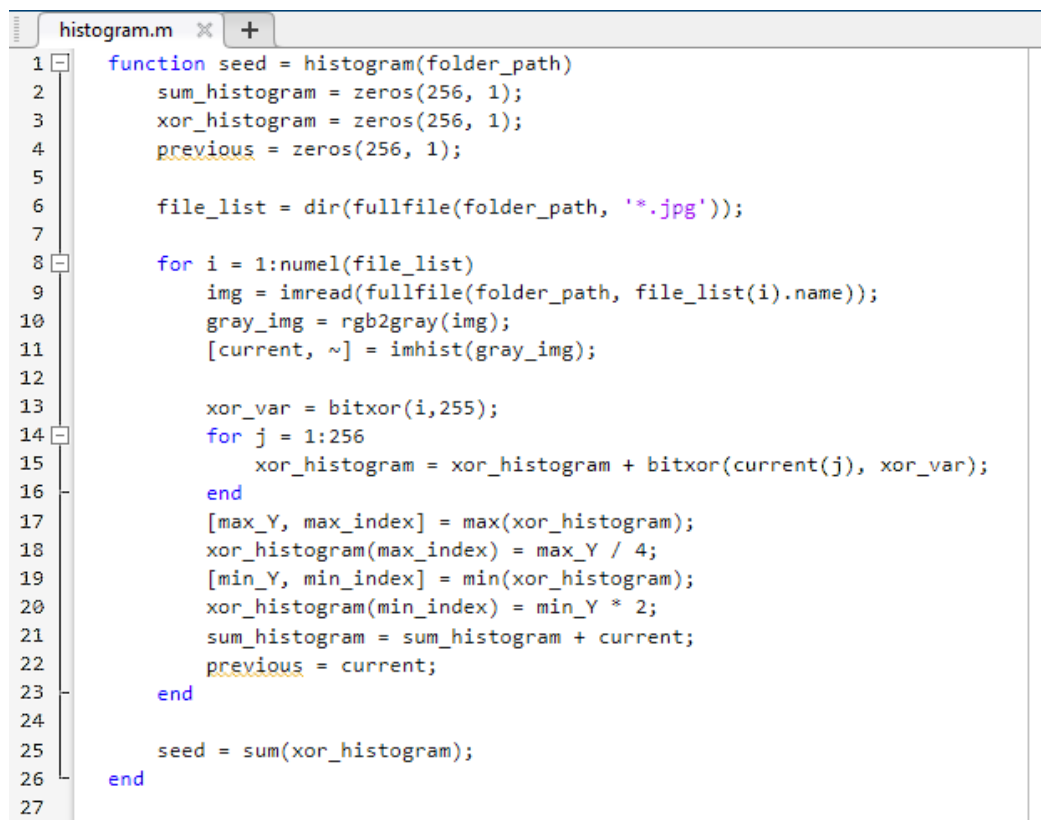
signature_preview_label = tk.Label(tab2, text="Podgląd Podpisu:")
signature_preview_label.pack(pady=5)

verify_signature_display = scrolledtext.ScrolledText(tab2, height=10, width=50)
verify_signature_display.pack(pady=10)

root.mainloop()

```

Generator TRNG w MATLAB:



```

1 function seed = histogram(folder_path)
2     sum_histogram = zeros(256, 1);
3     xor_histogram = zeros(256, 1);
4     previous = zeros(256, 1);
5
6     file_list = dir(fullfile(folder_path, '*.jpg'));
7
8     for i = 1:numel(file_list)
9         img = imread(fullfile(folder_path, file_list(i).name));
10        gray_img = rgb2gray(img);
11        [current, ~] = imhist(gray_img);
12
13        xor_var = bitxor(i, 255);
14        for j = 1:256
15            xor_histogram = xor_histogram + bitxor(current(j), xor_var);
16        end
17        [max_Y, max_index] = max(xor_histogram);
18        xor_histogram(max_index) = max_Y / 4;
19        [min_Y, min_index] = min(xor_histogram);
20        xor_histogram(min_index) = min_Y * 2;
21        sum_histogram = sum_histogram + current;
22        previous = current;
23    end
24
25    seed = sum(xor_histogram);
26 end
27

```