Audio Steganography with AES Encryption for Confidential Communication

WORKING CODE

!pip install flask pycryptodome numpy scipy pyngrok

import os

# Create directories

os.makedirs('templates', exist\_ok=True)

os.makedirs('static', exist\_ok=True)

# login.html

with open('templates/login.html', 'w') as f:

f.write("""

<!DOCTYPE html>

<html>

<head>

<title>Login - Audio Steganography</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<div class="container">

<h2>Login</h2>

{% if error %}

<div class="error">{{ error }}</div>

{% endif %}

<form method="post">

<input type="text" name="username" placeholder="Username" required><br><br>

<input type="password" name="password" placeholder="Password" required><br><br>

<button type="submit">Login</button>

</form>

</div>

</body>

</html>

""")

# home.html

with open('templates/home.html', 'w') as f:

f.write("""

<!DOCTYPE html>

<html>

<head>

<title>Home - Audio Steganography</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<div class="container">

<h1>Welcome, {{ session['username'] }}</h1>

<div class="nav-links">

<a href="{{ url\_for('hide') }}">Hide Message</a> |

<a href="{{ url\_for('extract') }}">Extract Message</a> |

<a href="{{ url\_for('logout') }}">Logout</a>

</div>

</div>

</body>

</html>

""")

# hide.html

with open('templates/hide.html', 'w') as f:

f.write("""

<!DOCTYPE html>

<html>

<head>

<title>Hide Message</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<div class="container">

<h2>Hide Message</h2>

<form method="post" enctype="multipart/form-data">

<label>Upload WAV Audio:</label><br>

<input type="file" name="audio" required><br><br>

<label>Message:</label><br>

<input type="text" name="message" required><br><br>

<label>AES Key:</label><br>

<input type="text" name="key" required><br><br>

<button type="submit">Hide Message</button>

</form>

<p>{{ message }}</p>

<a href="{{ url\_for('home') }}">Back to Home</a>

</div>

</body>

</html>

""")

# extract.html

with open('templates/extract.html', 'w') as f:

f.write("""

<!DOCTYPE html>

<html>

<head>

<title>Extract Message</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<div class="container">

<h2>Extract Message</h2>

<form method="post" enctype="multipart/form-data">

<label>Upload WAV Audio:</label><br>

<input type="file" name="audio" required><br><br>

<label>AES Key:</label><br>

<input type="text" name="key" required><br><br>

<label>Message Length (in bytes):</label><br>

<input type="number" name="length" required><br><br>

<button type="submit">Extract Message</button>

</form>

<h3>Output:</h3>

<div class="output">{{ result }}</div>

<a href="{{ url\_for('home') }}">Back to Home</a>

</div>

</body>

</html>

""")

# style.css

with open('static/style.css', 'w') as f:

f.write("""

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

}

.container {

width: 400px;

margin: 50px auto;

background: white;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

h1, h2, h3 {

text-align: center;

}

form {

display: flex;

flex-direction: column;

}

input[type="text"], input[type="password"], input[type="file"], input[type="number"] {

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px;

background-color: #5cb85c;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #4cae4c;

}

.error {

color: red;

text-align: center;

}

.output {

background: #eee;

padding: 10px;

border-radius: 4px;

min-height: 50px;

}

.nav-links {

text-align: center;

margin-top: 20px;

}

.nav-links a {

text-decoration: none;

margin: 0 10px;

color: #337ab7;

}

.nav-links a:hover {

text-decoration: underline;

}

""")

import os

# Ensure static folder exists

os.makedirs('static', exist\_ok=True)

# Write the updated colorful CSS file

with open('static/style.css', 'w') as f:

f.write("""

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: linear-gradient(135deg, #74ABE2 0%, #5563DE 100%);

color: #333;

margin: 0;

padding: 0;

}

.container {

width: 90%;

max-width: 450px;

margin: 60px auto;

background: white;

padding: 30px;

border-radius: 12px;

box-shadow: 0 10px 20px rgba(0,0,0,0.2);

text-align: center;

}

h1, h2, h3 {

color: #444;

}

form {

display: flex;

flex-direction: column;

gap: 15px;

margin-top: 20px;

}

input[type="text"], input[type="password"], input[type="file"], input[type="number"] {

padding: 12px;

border: 2px solid #ccc;

border-radius: 8px;

transition: 0.3s ease;

font-size: 1rem;

}

input[type="text"]:focus, input[type="password"]:focus, input[type="file"]:focus, input[type="number"]:focus {

border-color: #5563DE;

outline: none;

}

button {

padding: 12px;

background: #5563DE;

color: white;

border: none;

border-radius: 8px;

font-size: 1rem;

cursor: pointer;

transition: background 0.3s ease;

}

button:hover {

background: #3a47b5;

}

.error {

color: red;

margin-bottom: 10px;

}

.output {

background: #f0f0f0;

padding: 12px;

border-radius: 8px;

min-height: 50px;

margin-top: 20px;

text-align: left;

word-break: break-word;

}

.nav-links {

margin-top: 25px;

}

.nav-links a {

text-decoration: none;

margin: 0 12px;

color: #5563DE;

font-weight: bold;

transition: color 0.3s ease;

}

.nav-links a:hover {

color: #3a47b5;

}

.login-container {

animation: fadeIn 1s ease;

}

.home-container {

animation: slideIn 0.8s ease;

}

form button {

font-weight: bold;

}

@keyframes fadeIn {

from { opacity: 0; transform: translateY(-20px); }

to { opacity: 1; transform: translateY(0); }

}

@keyframes slideIn {

from { opacity: 0; transform: translateX(-50px); }

to { opacity: 1; transform: translateX(0); }

}

""")

print("style.css updated!")

from flask import Flask, request, send\_file, render\_template, redirect, url\_for, session

import numpy as np

from scipy.io import wavfile

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

import io

from pyngrok import ngrok

app = Flask(\_name\_)

app.secret\_key = 'supersecretkey'

# Allow large file uploads

app.config['MAX\_CONTENT\_LENGTH'] = 500 \* 1024 \* 1024 # 500 MB limit

# Configure ngrok

ngrok.set\_auth\_token("YOUR\_NGROK\_AUTH\_TOKEN") # Replace this with your actual token

public\_url = ngrok.connect(5000)

print("Public URL:", public\_url)

# Dummy user credentials

USER\_CREDENTIALS = {

"user": "password123"

}

@app.route('/', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if USER\_CREDENTIALS.get(username) == password:

session['username'] = username

return redirect(url\_for('home'))

else:

error = "Invalid credentials"

return render\_template('login.html', error=error)

return render\_template('login.html')

@app.route('/home')

def home():

if 'username' not in session:

return redirect(url\_for('login'))

return render\_template('home.html')

@app.route('/hide', methods=['GET', 'POST'])

def hide():

if 'username' not in session:

return redirect(url\_for('login'))

message = ""

max\_message\_size = None

if request.method == 'POST':

try:

audio\_file = request.files['audio\_file']

msg = request.form['message']

key = request.form['key']

# Prepare AES key

key\_bytes = key.encode('utf-8')

if len(key\_bytes) < 16:

key\_bytes = key\_bytes.ljust(16, b'\0')

else:

key\_bytes = key\_bytes[:16]

# Encrypt message

iv = np.random.randint(0, 256, 16, dtype=np.uint8).tobytes()

cipher = AES.new(key\_bytes, AES.MODE\_CBC, iv)

ct = cipher.encrypt(pad(msg.encode('utf-8'), AES.block\_size))

encrypted\_data = iv + ct

# Read original WAV

rate, audio\_data = wavfile.read(io.BytesIO(audio\_file.read()))

if len(audio\_data.shape) > 1:

audio\_data = audio\_data[:, 0]

# Check available space

available\_bytes = len(audio\_data) // 8

required\_bytes = len(encrypted\_data)

max\_message\_size = available\_bytes - 16 # Subtract 16 bytes for IV

if required\_bytes > available\_bytes:

message = f"Audio too short! Max message size: {max\_message\_size} bytes."

return render\_template('hide.html', message=message, max\_message\_size=max\_message\_size)

# Hide bits in LSBs

flat\_audio = audio\_data.copy()

for i in range(required\_bytes \* 8):

bit = (encrypted\_data[i // 8] >> (i % 8)) & 1

flat\_audio[i] = (flat\_audio[i] & ~1) | bit

# Save to in-memory file

output = io.BytesIO()

wavfile.write(output, rate, flat\_audio.astype(audio\_data.dtype))

output.seek(0)

return send\_file(output, as\_attachment=True, download\_name="encrypted\_audio.wav", mimetype="audio/wav")

except Exception as e:

message = "Error: " + str(e)

return render\_template('hide.html', message=message, max\_message\_size=max\_message\_size)

@app.route('/extract', methods=['GET', 'POST'])

def extract():

if 'username' not in session:

return redirect(url\_for('login'))

result = ""

max\_message\_size = None

if request.method == 'POST':

try:

audio\_file = request.files['audio\_file']

key = request.form['key']

length = int(request.form['length'])

key\_bytes = key.encode('utf-8')

if len(key\_bytes) < 16:

key\_bytes = key\_bytes.ljust(16, b'\0')

else:

key\_bytes = key\_bytes[:16]

rate, audio\_data = wavfile.read(io.BytesIO(audio\_file.read()))

if len(audio\_data.shape) > 1:

audio\_data = audio\_data[:, 0]

available\_bytes = len(audio\_data) // 8

max\_message\_size = available\_bytes - 16

required\_bits = length \* 8

if required\_bits > len(audio\_data):

result = f"Error: Audio too short! Max extractable size: {max\_message\_size} bytes."

return render\_template('extract.html', result=result, max\_message\_size=max\_message\_size)

# Extract bits from LSBs

bits = []

for i in range(required\_bits):

bits.append(audio\_data[i] & 1)

encrypted\_bytes = bytearray()

for i in range(0, len(bits), 8):

byte = 0

for j in range(8):

byte |= (bits[i + j] << j)

encrypted\_bytes.append(byte)

iv = encrypted\_bytes[:16]

ct = encrypted\_bytes[16:]

cipher = AES.new(key\_bytes, AES.MODE\_CBC, iv)

decrypted = unpad(cipher.decrypt(ct), AES.block\_size).decode('utf-8')

result = decrypted

except Exception as e:

result = "Error: " + str(e)

return render\_template('extract.html', result=result, max\_message\_size=max\_message\_size)

@app.route('/logout')

def logout():

session.pop('username', None)

return redirect(url\_for('login'))

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

app.run(port=5000)