

Dokumentasi Aplikasi Flask untuk Sistem Pengiriman Pesan Terenkripsi dengan RSA

Demo aplikasi: <https://youtu.be/xoUMgiOU03c>

Mochamad Zidan Hadipratama - 5027221052

Marselinus Krisnawan Riandika - 5027221056

Aplikasi ini adalah sistem pengiriman pesan berbasis web yang menggunakan **Flask** sebagai framework backend. Aplikasi ini memungkinkan pengguna untuk mendaftar, login, mengirim pesan terenkripsi menggunakan algoritma **RSA**, serta melihat dan mendekripsi pesan yang diterima. Sistem juga menerapkan enkripsi pada kunci privat dan publik setiap pengguna untuk menjamin kerahasiaan pesan.

1. Struktur Folder dan File

```
.
├── app.py
├── data
│   ├── creds.txt
│   ├── keys
│   └── messages.txt
├── static
│   └── style.css
└── templates
    ├── base.html
    ├── dashboard.html
    ├── home.html
    ├── inbox.html
    ├── kirim_pesan.html
    ├── login.html
    └── register.html
```

Penjelasan:

1. **app.py**: File utama yang mengandung logika aplikasi dan route Flask.
2. **data/**: Menyimpan file-file terkait data aplikasi:
 - a. **creds.txt**: Menyimpan username dan password yang telah di-hash menggunakan algoritma **SHA-256**.
 - b. **keys/**: Direktori tempat menyimpan kunci privat dan publik dari setiap pengguna.
 - c. **messages.txt**: Menyimpan pesan-pesan yang telah terenkripsi.
3. **static/**: Folder yang menyimpan file CSS atau file statis lainnya.

- a. **style.css**: Mengatur tampilan frontend aplikasi.
 - 4. **templates/**: Folder berisi template HTML untuk tampilan halaman web.
 - a. **base.html**: Template dasar yang digunakan oleh halaman lainnya.
 - b. **dashboard.html**: Halaman dashboard setelah pengguna login.
 - c. **home.html**: Halaman beranda.
 - d. **inbox.html**: Halaman inbox untuk melihat pesan yang diterima.
 - e. **irim_pesan.html**: Halaman untuk mengirim pesan kepada pengguna lain.
 - f. **login.html**: Halaman login.
 - g. **register.html**: Halaman pendaftaran.
-

App.py:

```
import os
import hashlib
from flask import Flask, render_template, request, redirect, url_for,
flash, session
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64encode, b64decode

app = Flask(__name__)
app.secret_key = 'your_secret_key'

KEYS_DIRECTORY = 'data/keys'
CREDS_FILE = 'data/creds.txt'

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

def save_user_keys(username, private_key, public_key):
    username = username.lower()
    user_dir = os.path.join(KEYS_DIRECTORY, username)

    if not os.path.exists(user_dir):
        os.makedirs(user_dir)

    # Save private key
    private_key_file = os.path.join(user_dir, 'private_key.pem')
    with open(private_key_file, 'wb') as f:
```

```

        f.write(private_key)

    # Save public key
    public_key_file = os.path.join(user_dir, 'public_key.pem')
    with open(public_key_file, 'wb') as f:
        f.write(public_key)

def load_user_keys(username):
    username = username.lower()
    user_dir = os.path.join(KEYS_DIRECTORY, username)
    private_key_file = os.path.join(user_dir, 'private_key.pem')
    public_key_file = os.path.join(user_dir, 'public_key.pem')

    if os.path.exists(private_key_file) and \
os.path.exists(public_key_file):
        with open(private_key_file, 'rb') as f:
            private_key = f.read()
        with open(public_key_file, 'rb') as f:
            public_key = f.read()
        return private_key, public_key
    else:
        return None, None

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def save_user_credentials(username, hashed_password):
    with open(CREDS_FILE, 'a') as f:
        f.write(f"{username},{hashed_password}\n")

def check_credentials(username, password):
    hashed_input_password = hash_password(password)
    if os.path.exists(CREDS_FILE):
        with open(CREDS_FILE, 'r') as f:
            for line in f.readlines():
                stored_username, stored_hashed_password = \
line.strip().split(',')
                if stored_username == username and stored_hashed_password \
== hashed_input_password:
                    return True
    return False

@app.route('/')
def home():
    return render_template('home.html')

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

```

```

if request.method == 'POST':
    username = request.form['username'].lower()
    password = request.form['password']

    if check_credentials(username, password):
        session['username'] = username
        flash(f"Welcome, {username}!", "success")
        return redirect(url_for('dashboard'))
    else:
        flash("Invalid username or password. Please try again.",
"error")
        return redirect(url_for('login'))
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('username', None)
    flash("You have been logged out.", "success")
    return redirect(url_for('login'))

@app.route('/dashboard')
def dashboard():
    if 'username' not in session:
        flash("You need to login first.", "error")
        return redirect(url_for('login'))

    return render_template('dashboard.html', username=session['username'])

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username'].lower()
        password = request.form['password']

        private_key, public_key = load_user_keys(username)

        if private_key is None or public_key is None:
            hashed_password = hash_password(password)

            private_key, public_key = generate_rsa_keys()
            save_user_keys(username, private_key, public_key)

            save_user_credentials(username, hashed_password)

            flash(f"User {username} registered successfully!", "success")
            return redirect(url_for('login'))
        else:
            flash(f"User {username} already exists!", "error")

```

```

        return redirect(url_for('register'))
    else:
        return render_template('register.html')

@app.route('/kirim_pesan', methods=['GET', 'POST'])
def kirim_pesan():
    if 'username' not in session:
        flash("You need to login first.", "error")
        return redirect(url_for('login'))

    if request.method == 'POST':
        sender = session['username']
        recipient = request.form['recipient'].lower()
        message = request.form['message']

        _, recipient_public_key = load_user_keys(recipient)
        if recipient_public_key:
            encrypted_message = encrypt_message(recipient_public_key,
message)
            save_message(sender, recipient, encrypted_message)
            flash("Message sent successfully!", "success")
        else:
            flash("Recipient not found!", "error")

    users = [d for d in os.listdir(KEYS_DIRECTORY) if
os.path.isdir(os.path.join(KEYS_DIRECTORY, d))]
    return render_template('kirim_pesan.html', users=users)

@app.route('/inbox')
def inbox():
    if 'username' not in session:
        flash("You need to login first.", "error")
        return redirect(url_for('login'))

    username = session['username']
    private_key, _ = load_user_keys(username)
    if private_key is None:
        flash("User not found!", "error")
        return redirect(url_for('login'))

    messages = load_messages()
    user_messages = []

    if username in messages:
        for msg in messages[username]:
            decrypted_message = decrypt_message(private_key,
msg['message'])

```

```

        user_messages.append({
            'from': msg['from'],
            'message': decrypted_message
        })

    return render_template('inbox.html', username=username,
messages=user_messages)

# Encrypt message
def encrypt_message(public_key_str, message):
    public_key = RSA.import_key(public_key_str)
    cipher_rsa = PKCS1_OAEP.new(public_key)
    encrypted_message = cipher_rsa.encrypt(message.encode('utf-8'))
    return b64encode(encrypted_message).decode('utf-8')

# Decrypt message
def decrypt_message(private_key_str, encrypted_message):
    private_key = RSA.import_key(private_key_str)
    cipher_rsa = PKCS1_OAEP.new(private_key)
    decrypted_message = cipher_rsa.decrypt(b64decode(encrypted_message))
    return decrypted_message.decode('utf-8')

# Save message to file
def save_message(sender, recipient, encrypted_message):
    sender = sender.lower() # Normalize username to lowercase
    recipient = recipient.lower() # Normalize username to lowercase
    message_file = os.path.join('data', 'messages.txt')
    with open(message_file, 'a') as f:
        f.write(f"{sender},{recipient},{encrypted_message}\n")

# Load messages from file
def load_messages():
    messages = {}
    message_file = os.path.join('data', 'messages.txt')

    if os.path.exists(message_file):
        with open(message_file, 'r') as f:
            for line in f.readlines():
                sender, recipient, encrypted_message =
line.strip().split(',')
                if recipient not in messages:
                    messages[recipient] = []
                messages[recipient].append({
                    'from': sender,
                    'message': encrypted_message
                })
    return messages

```

```
if __name__ == '__main__':
    os.makedirs('data/keys', exist_ok=True)
    app.run(debug=True, port=5005)
```

2. Penjelasan Fungsi Utama

1. generate_rsa_keys()

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

- Menghasilkan sepasang kunci **RSA** (private key dan public key) dengan panjang kunci 2048 bit.
- **Private key** digunakan untuk dekripsi pesan, sedangkan **public key** digunakan untuk enkripsi.

2. save_user_keys(username, private_key, public_key)

```
def save_user_keys(username, private_key, public_key):
    user_dir = os.path.join(KEYS_DIRECTORY, username.lower())
    if not os.path.exists(user_dir):
        os.makedirs(user_dir)

    with open(os.path.join(user_dir, 'private_key.pem'), 'wb') as f:
        f.write(private_key)
    with open(os.path.join(user_dir, 'public_key.pem'), 'wb') as f:
        f.write(public_key)
```

- Menyimpan private key dan public key di folder pengguna yang dibuat secara otomatis.
- **Private key** disimpan di `private_key.pem`, dan **public key** disimpan di `public_key.pem`.

3. load_user_keys(username)

```
def load_user_keys(username):
```

```

user_dir = os.path.join(KEYS_DIRECTORY, username.lower())
private_key_file = os.path.join(user_dir, 'private_key.pem')
public_key_file = os.path.join(user_dir, 'public_key.pem')

if os.path.exists(private_key_file) and
os.path.exists(public_key_file):
    with open(private_key_file, 'rb') as f:
        private_key = f.read()
    with open(public_key_file, 'rb') as f:
        public_key = f.read()
    return private_key, public_key
return None, None

```

- Fungsi ini mengambil **private key** dan **public key** dari folder pengguna berdasarkan nama pengguna.

4. `hash_password(password)`

```

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

```

- Meng-hash password menggunakan algoritma **SHA-256**. Hasil hash disimpan di file `creds.txt`.

5. `save_user_credentials(username, hashed_password)`

```

def save_user_credentials(username, hashed_password):
    with open(CREDS_FILE, 'a') as f:
        f.write(f"{username},{hashed_password}\n")

```

- Menyimpan kredensial pengguna berupa username dan password yang sudah di-hash di file `creds.txt`.

6. `check_credentials(username, password)`

```

def check_credentials(username, password):
    hashed_input_password = hash_password(password)
    if os.path.exists(CREDS_FILE):
        with open(CREDS_FILE, 'r') as f:
            for line in f.readlines():

```



```

        stored_username, stored_hashed_password =
line.strip().split(',')
        if stored_username == username and stored_hashed_password
== hashed_input_password:
            return True
    return False

```

- Memeriksa apakah username dan password yang dimasukkan pengguna cocok dengan yang ada di `creds.txt`. Password pengguna di-hash terlebih dahulu sebelum dibandingkan.

3. Routing dan Fungsionalitas

1. Halaman Utama (/)

```

@app.route('/')
def home():
    return render_template('home.html')

```

- Halaman utama yang merender template `home.html`. Biasanya berisi tautan ke halaman login dan pendaftaran.

2. Halaman Login (/login)

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username'].lower()
        password = request.form['password']

        if check_credentials(username, password):
            session['username'] = username
            flash(f"Welcome, {username}!", "success")
            return redirect(url_for('dashboard'))
        else:
            flash("Invalid username or password. Please try again.",
"error")
            return redirect(url_for('login'))

```

```
return render_template('login.html')
```

- Memeriksa kredensial pengguna saat login. Jika valid, username disimpan di session dan pengguna diarahkan ke halaman dashboard. Jika tidak valid, pesan kesalahan ditampilkan.

3. Halaman Pendaftaran (/register)

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username'].lower()
        password = request.form['password']

        private_key, public_key = load_user_keys(username)

        if private_key is None or public_key is None:
            hashed_password = hash_password(password)
            private_key, public_key = generate_rsa_keys()
            save_user_keys(username, private_key, public_key)
            save_user_credentials(username, hashed_password)

            flash(f"User {username} registered successfully!", "success")
            return redirect(url_for('login'))
        else:
            flash(f"User {username} already exists!", "error")
            return redirect(url_for('register'))

    return render_template('register.html')
```

- Pengguna baru bisa mendaftar dengan memasukkan username dan password. Setelah itu, kunci RSA dihasilkan dan disimpan. Password di-hash dan disimpan di `creds.txt`.

4. Halaman Kirim Pesan (/ kirim_pesan)

```
@app.route('/kirim_pesan', methods=['GET', 'POST'])
def kirim_pesan():
    if 'username' not in session:
        flash("You need to login first.", "error")
        return redirect(url_for('login'))

    if request.method == 'POST':
        sender = session['username']
```

```

recipient = request.form['recipient'].lower()
message = request.form['message']

_, recipient_public_key = load_user_keys(recipient)
if recipient_public_key:
    encrypted_message = encrypt_message(recipient_public_key,
message)
    save_message(sender, recipient, encrypted_message)
    flash("Message sent successfully!", "success")
else:
    flash("Recipient not found!", "error")

users = [d for d in os.listdir(KEYS_DIRECTORY) if
os.path.isdir(os.path.join(KEYS_DIRECTORY, d))]
return render_template('kirim_pesan.html', users=users)

```

- Pengguna dapat mengirim pesan ke pengguna lain. Pesan akan dienkripsi menggunakan kunci publik penerima dan disimpan di `messages.txt`.

5. Halaman Inbox (`/inbox`)

```

@app.route('/inbox')
def inbox():
    if 'username' not in session:
        flash("You need to login first.", "error")
        return redirect(url_for('login'))

    username = session['username']
    private_key, _ = load_user_keys(username)
    if private_key is None:
        flash("User not found!", "error")
        return redirect(url_for('login'))

    messages = load_messages()
    user_messages = []

    if username in messages:

    for msg in messages[username]:
        decrypted_message = decrypt_message(private_key,
msg['message'])
        user_messages.append({
            'from': msg['from'],
            'message': decrypted_message
        })

```

```
    return render_template('inbox.html', username=username,
messages=user_messages)
```

- Pengguna dapat melihat pesan yang mereka terima. Pesan akan didekripsi menggunakan kunci privat pengguna.

4. Fungsi Enkripsi dan Dekripsi

1. `encrypt_message(public_key_str, message)`

```
def encrypt_message(public_key_str, message):
    public_key = RSA.import_key(public_key_str)
    cipher_rsa = PKCS1_OAEP.new(public_key)
    encrypted_message = cipher_rsa.encrypt(message.encode('utf-8'))
    return b64encode(encrypted_message).decode('utf-8')
```

- Fungsi ini mengenkripsi pesan menggunakan kunci publik penerima dan mengubahnya menjadi format base64 untuk penyimpanan.

2. `decrypt_message(private_key_str, encrypted_message)`

```
def decrypt_message(private_key_str, encrypted_message):
    private_key = RSA.import_key(private_key_str)
    cipher_rsa = PKCS1_OAEP.new(private_key)
    decrypted_message = cipher_rsa.decrypt(b64decode(encrypted_message))
    return decrypted_message.decode('utf-8')
```

- Fungsi ini mendekripsi pesan yang dienkripsi menggunakan kunci privat penerima.

5. Penyimpanan dan Pengambilan Pesan

1. `save_message(sender, recipient, encrypted_message)`

```
def save_message(sender, recipient, encrypted_message):
    message_file = os.path.join('data', 'messages.txt')
    with open(message_file, 'a') as f:
        f.write(f"{sender},{recipient},{encrypted_message}\n")
```

- Pesan yang terenkripsi disimpan di file `messages.txt` dengan format `pengirim, penerima, pesan`.

2. `load_messages()`

```
def load_messages():
    messages = {}
    message_file = os.path.join('data', 'messages.txt')

    if os.path.exists(message_file):
        with open(message_file, 'r') as f:
            for line in f.readlines():
                sender, recipient, encrypted_message =
line.strip().split(',')
                if recipient not in messages:
                    messages[recipient] = []
                messages[recipient].append({
                    'from': sender,
                    'message': encrypted_message
                })
    return messages
```

- Fungsi ini mengambil semua pesan yang disimpan di `messages.txt` dan mengorganisasikannya berdasarkan penerima.

Kesimpulan

Aplikasi ini menyediakan sistem pengiriman pesan terenkripsi menggunakan algoritma **RSA**. Setiap pengguna memiliki kunci privat dan publik yang disimpan secara terpisah. Pesan yang dikirim antar pengguna dienkripsi menggunakan kunci publik penerima, dan hanya dapat didekripsi menggunakan kunci privat penerima. Sistem ini memastikan keamanan komunikasi antar pengguna dengan menggunakan teknik enkripsi modern.

[Dashboard](#)[Kirim Pesan](#)[Inbox](#)[Logout](#)[Welcome, zidan](#)

Kirim Pesan

Welcome, zidan!

Send

Logged in as: zidan

[Home](#)[Login](#)[Register](#)

Login

You have been logged out.

Login

[Register here](#)

zidan's Inbox

From	Message
krisna	Testiung kirim email lumayan panjang, gimana trampilannya? moga2 bagus bgt