

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Báo cáo

ĐỒ ÁN 1

Môn học: An ninh máy tính

CSC15003 _ 22MMT

Sinh viên:

Lê Hoàng Đạt
Nguyễn Hồ Đăng Duy
Phạm Quang Duy

Giảng viên hướng dẫn:

Lê Giang Thanh
Lê Hà Minh
Phan Quốc Kỳ

Mục lục

1 Thông tin sinh viên	2
2 Giới thiệu	2
3 Phân công chi tiết	3
3.1 Giai đoạn 1: Tìm hiểu và lên kế hoạch	3
3.2 Giai đoạn 2: Triển khai chức năng	3
3.3 Giai đoạn 3: Hoàn thành các chức năng và gộp mã	4
3.4 Giai đoạn 4: Kiểm thử, viết báo cáo và quay video demo	5
4 Kiến trúc hệ thống	6
4.1 Sơ đồ tổng thể	6
4.2 Thư mục và các modules chính	6
5 Chi tiết chức năng và kỹ thuật bảo mật	8
5.1 Đăng ký tài khoản người dùng	8
5.2 Đăng nhập và Xác thực đa yếu tố (MFA)	12
5.3 Quản lý khoá RSA cá nhân	16
5.4 QR Code Public Key	20
5.5 Cập nhật thông tin tài khoản	26
5.6 Mã hoá tập tin gửi người khác	31
5.7 Giải mã tập tin	34
5.8 Ký số tập tin	37
5.9 Xác minh chữ ký	41
5.10 Phân quyền tài khoản	45
5.11 Ghi log bảo mật	49
5.12 Chia nhỏ tập tin lớn	52
5.13 Kiểm tra trạng thái khoá	54
5.14 Tìm kiếm public key	57
5.15 Giới hạn đăng nhập	60
5.16 Tùy chọn định dạng lưu file	63
5.17 Khôi phục tài khoản	65
6 Kiểm thử ứng dụng	68
Tài liệu tham khảo	75

1 Thông tin sinh viên

Nhóm gồm có 3 thành viên:

- 22127060 - Lê Hoàng Đạt - 22127060@student.hcmus.edu.vn
- 22127085 - Nguyễn Hồ Đăng Duy - 22127085@student.hcmus.edu.vn
- 22127088 - Phạm Quang Duy - 22127088@student.hcmus.edu.vn

2 Giới thiệu

Secure Vault là một ứng dụng mô phỏng hệ thống bảo mật theo mô hình thực tế, được xây dựng nhằm mục tiêu bảo vệ dữ liệu cá nhân và hỗ trợ truyền tin an toàn giữa người dùng. Ứng dụng tích hợp nhiều kỹ thuật bảo mật hiện đại như mã hóa RSA/AES, xác thực đa yếu tố (MFA), ký số – xác minh chữ ký, QR Code, cùng với cơ chế phân quyền, kiểm tra trạng thái khóa, ghi log bảo mật,... Người dùng có thể:

- Đăng ký, đăng nhập bảo mật với OTP/TOTP
- Tạo và quản lý cặp khóa RSA cá nhân
- Mã hóa và giải mã tập tin với AES + RSA
- Ký số tập tin và xác minh tính toàn vẹn
- Tạo và quét QR chứa public key để chia sẻ
- Theo dõi trạng thái khóa, phân quyền người dùng (admin/user)
- Cập nhật thông tin, khôi phục tài khoản khi quên mật khẩu

Hệ thống được xây dựng với ngôn ngữ `Python`, sử dụng `Flask` cho backend, kết hợp `MySQL/JSON` để lưu trữ dữ liệu, cùng các thư viện bảo mật như `pycryptodome`, `pyotp`, `qrcode` ,...

GitHub Repository: https://github.com/YuD1405/Secure_Vault

3 Phân công chi tiết

3.1 Giai đoạn 1: Tìm hiểu và lên kế hoạch

Giai đoạn 1: Tìm hiểu và lên kế hoạch		
Thành viên	Nhiệm vụ	Thời hạn
Phạm Quang Duy	Tóm tắt thông tin đồ án, chia thành các modules	16/06
Phạm Quang Duy	Tạo repository github và chốt công nghệ	16/06
Phạm Quang Duy	Chia task	16/06
Cả nhóm	Chốt công nghệ	16/06

3.2 Giai đoạn 2: Triển khai chức năng

QUẢN LÝ NGƯỜI DÙNG			
Thành viên	Yêu cầu	Nhiệm vụ	Thời hạn
Nguyễn Hồ Đăng Duy	1	Đăng ký người dùng	17/06 - 19/06
Nguyễn Hồ Đăng Duy	2	Đăng nhập và xác thực MFA	18/06 - 20/06
Phạm Quang Duy	5	Cập nhật thông tin tài khoản	20/06 - 21/06
Nguyễn Hồ Đăng Duy	10	Phân quyền Admin	20/06 - 23/06
Nguyễn Hồ Đăng Duy	15	Giới hạn login	19/06 - 20/06
Phạm Quang Duy	17	Khôi phục tài khoản	17/06 - 23/06
Nguyễn Hồ Đăng Duy	UI	Frontend: Form accounts, navigation	29/01 - 09/03

MÃ HÓA VÀ GIẢI MÃ			
Thành viên	Yêu cầu	Nhiệm vụ	Thời hạn
Lê Hoàng Đạt	3	Quản lí khóa	17/06 - 19/06
Lê Hoàng Đạt	4	Tạo và quét QR cho pub key	24/06 - 25/06
Lê Hoàng Đạt	6	Mã hóa tập tin gửi	20/06 - 21/06
Lê Hoàng Đạt	7	Giải mã tập tin	21/06 - 22/06
Lê Hoàng Đạt	12	Chia nhỏ tập tin khi mã hóa	23/06 - 24/06
Lê Hoàng Đạt	16	Tùy chọn định dạng lưu	21/06 - 22/06
Phạm Quang Duy	UI	Giao diện chức năng dashboard	17/06 - 25/06

SIGNING & LOGGING & UTILS			
Thành viên	Yêu cầu	Nhiệm vụ	Thời hạn
Phạm Quang Duy	8	Ký số	17/06 - 18/06
Phạm Quang Duy	9	Xác minh ký số	19/06 - 20/06
Phạm Quang Duy	11	Log bảo mật	21/06 - 22/06
Lê Hoàng Đạt	13	Kiểm tra trạng thái khóa	18/06 - 19/06
Phạm Quang Duy	14	Tìm pub Key	23/06 - 23/06
Nguyễn Hồ Đăng Duy	UI	Logging interface	23/06 - 27/06

3.3 Giai đoạn 3: Hoàn thành các chức năng và gộp mã

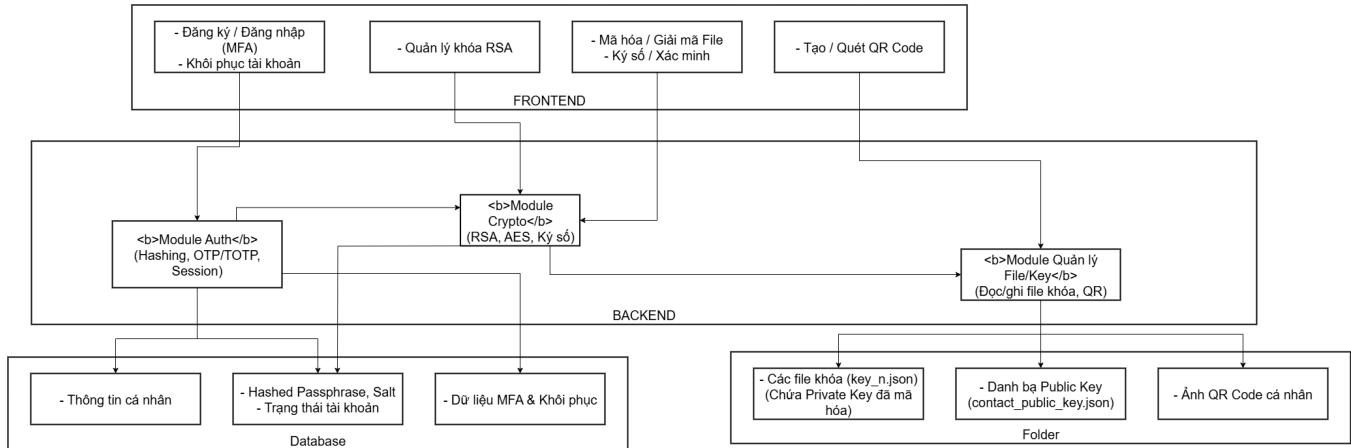
Giai đoạn 3: Hoàn thành các chức năng và merge code		
Thành viên	Nhiệm vụ	Thời hạn
Nguyễn Hồ Đăng Duy	Kết nối: Group 1: User management → main UI	01/07 - 03/07
Lê Hoàng Đạt	Kết nối: Group 2: Encrypt và Decrypt → main UI	03/07 - 06/07
Phạm Quang Duy	Kết nối: Group 3: Logging và Utils → main UI	06/07 - 08/07
Cả nhóm	Kết nối toàn bộ chức năng (flow giữa các group chức năng)	08/07

3.4 Giai đoạn 4: Kiểm thử, viết báo cáo và quay video demo

Giai đoạn 4: Test chức năng, quay video và hoàn thiện report		
Thành viên	Nhiệm vụ	Thời hạn
Nguyễn Hồ Đăng Duy	Test toàn bộ flow chương trình	08/07 - 09/07
Cả nhóm	Chỉnh sửa sau test	09/07
Cả nhóm	Report	08/07 - 10/07
Cả nhóm	Record video	11/07 - 12/07

4 Kiến trúc hệ thống

4.1 Sơ đồ tổng thể



Hình 1: Sơ đồ kiến trúc hệ thống

4.2 Thư mục và các modules chính

Dồ án được tổ chức rõ ràng theo mô hình **Flask MVC** và phân chia modules theo chức năng bảo mật:

- `main.py` - Điểm khởi chạy của ứng dụng Flask
- `.env` - Tập tin cấu hình (thông tin DB, email, secret key,...)
- `data/` - Chứa dữ liệu hệ thống
 - `key_manage/`
 - `keys/`
 - `qr/`
 - `temp_key/`
- `flaskapi/` - Backend API
 - `app.py` - Cấu hình Flask app và đăng ký route
 - `routes/` - Chứa các tệp định nghĩa route theo nhóm chức năng
- `frontend/` - Các file HTML render giao diện với Jinja2
- `log/` - Ghi toàn bộ hành vi quan trọng: đăng nhập, mã hóa, ký số,... kèm thời gian, email, trạng thái

- `modules/` - Các modules chức năng chính
 - `auth/` - Hash passphrase, sinh OTP, xác thực MFA
 - `crypto/` - Mã hóa RSA/AES, ký số và xác minh chữ ký
 - `utils/` - Hàm tiện ích chung: log sự kiện, mail, QR,...
- `mySQL/` - Các chương trình để tạo database, table cho cơ sở dữ liệu MySQL
- `report/` - Báo cáo và video demo
- `README.md` - Hướng dẫn cài đặt và chạy thử đồ án. Mô tả các chức năng và công nghệ sử dụng.

5 Chi tiết chức năng và kỹ thuật bảo mật

5.1 Đăng ký tài khoản người dùng

Mục tiêu

Cho phép người dùng tạo tài khoản bằng cách điền đầy đủ thông tin cá nhân và **passphrase** (đảm bảo đủ an toàn). Thông tin sau đó được kiểm tra tính hợp lệ, hash **passphrase** và lưu vào cơ sở dữ liệu. Đồng thời, hệ thống sẽ tạo một **cặp khóa cá nhân RSA** và sinh một mã **recovery code** để cung cấp cho người dùng. **Private key** sẽ được mã hóa thành hai bản riêng biệt:

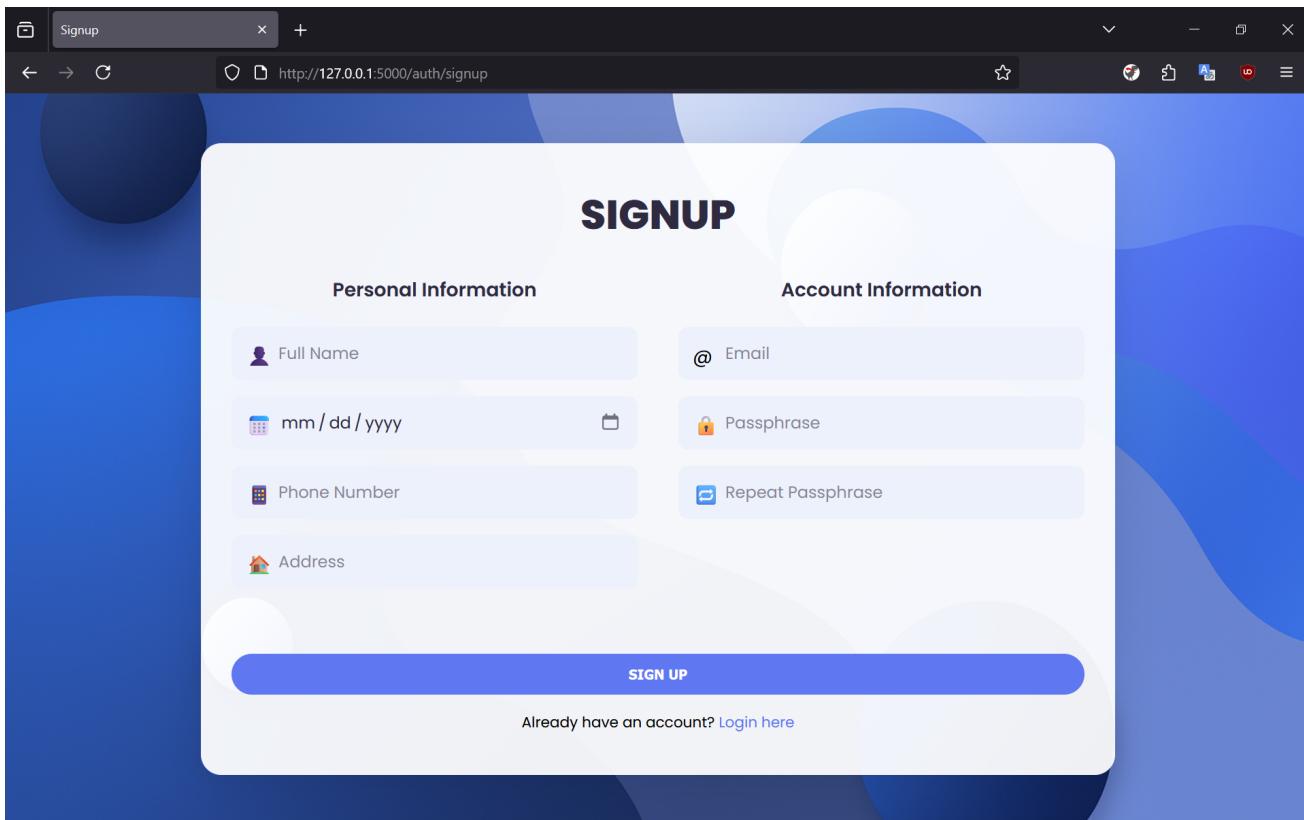
- Một bản được mã hóa bằng **AES key** dẫn xuất từ **passphrase**
- Một bản khác được mã hóa bằng **AES key** dẫn xuất từ **recovery code**

Cả hai bản đều được lưu trữ an toàn trên hệ thống. Điều này cho phép người dùng có thể truy xuất lại khóa cá nhân trong trường hợp quên **passphrase**, miễn là còn giữ **recovery code**.

Giao diện

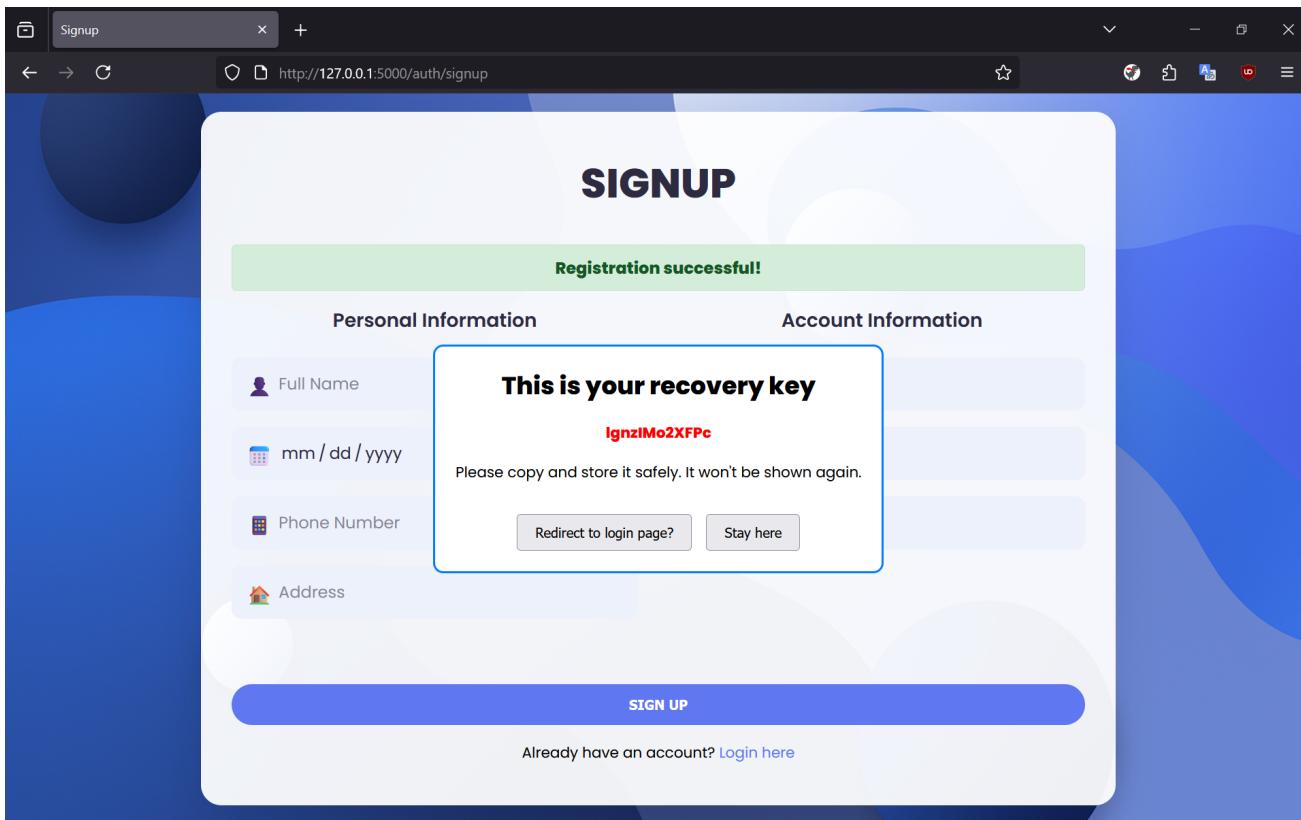
Giao diện **/auth/signup** là form HTML bao gồm các trường:

- Email
- Họ tên
- Ngày sinh
- Số điện thoại
- Địa chỉ
- Passphrase và xác nhận passphrase



Hình 2: Giao diện Signup

Khi đăng ký thành công, hệ thống sẽ hiển thị mã khôi phục tài khoản, yêu cầu người dùng lưu lại mã này để phục hồi nếu mất mật khẩu.



Hình 3: Giao diện popup recovery key

Quy trình thực hiện

1. Người dùng nhập thông tin và submit form → POST `/signup`
2. Flask gọi hàm `register_user(request.form)` trong `logic.py`
3. Thông tin được kiểm tra và chuẩn hóa:
 - Email hợp lệ (`is_valid_email`)
 - Ngày sinh đúng định dạng (`is_valid_date`)
 - Số điện thoại gồm đúng 10 chữ số (`is_valid_phone`)
 - Passphrase đủ mạnh (`is_strong_passphrase`)
4. Nếu hợp lệ:
 - Sinh `salt` ngẫu nhiên
 - Băm `passphrase` kết hợp với salt bằng `SHA-256`
 - Sinh `recovery_code` để dùng cho khôi phục
 - Sinh `mfa_secret` phục vụ TOTP trong MFA
5. Thực hiện `INSERT` bản ghi vào bảng `users` trong CSDL

6. Ghi log hành động: `log_user_action(email, "Register", "Success")`

7. Trả về kết quả thành công và hiển thị mã khôi phục

Chi tiết kỹ thuật và thư viện bảo mật

1. Hashing passphrase với Salt

Thư viện: `hashlib, os`

Kỹ thuật:

- Salt được sinh bằng `os.urandom(16).hex()` → đảm bảo tính ngẫu nhiên mạnh.
- `passphrase` được hash bằng `SHA-256(passphrase + salt)` trước khi lưu xuống DB.

2. Validator kiểm tra đầu vào

Thư viện: `re, datetime`

Kỹ thuật:

- Regex kiểm tra định dạng email, số điện thoại.
- Kiểm tra `passphrase` phải ít nhất 8 ký tự, chứa chữ hoa, số, ký hiệu.
- Loại bỏ ký tự nguy hiểm khỏi input (`sanitize_input()`) → giảm rủi ro XSS / SQLi / Code Injection.

3. Tạo mã khôi phục

Thư viện: `secrets`

Kỹ thuật:

```
1 def generate_recovery_code(length_bytes=16):  
2     return secrets.token_urlsafe(length_bytes)  
3
```

- Sinh chuỗi ngẫu nhiên gồm chữ + số, chỉ hiển thị 1 lần.
- Người dùng được yêu cầu lưu lại → dùng khi mất `passphrase`.
- Được mã hóa bằng `AES key` tạo từ `passphrase` và lưu trong DB (`users.encrypted_recovery_code`)

5.2 Đăng nhập và Xác thực đa yếu tố (MFA)

Mục tiêu

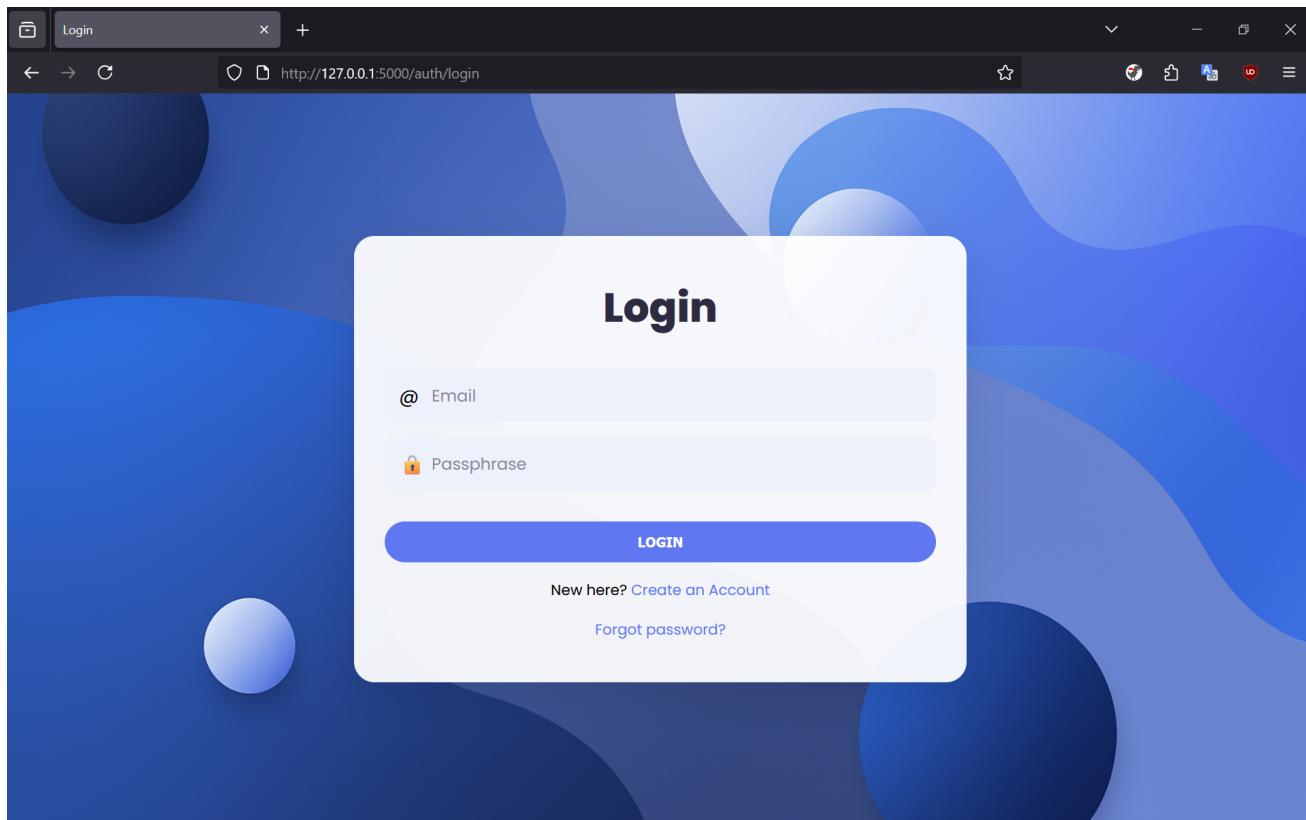
Cho phép người dùng đăng nhập bằng cách nhập `email` và `passphrase`. Nếu thông tin hợp lệ, hệ thống sẽ yêu cầu xác minh OTP qua email hoặc mã TOTP (Google Authenticator). Hệ thống hỗ trợ các biện pháp bảo mật nâng cao:

- Bảo vệ bằng xác thực 2 yếu tố
- Tự động khóa tài khoản 5 phút sau 5 lần đăng nhập sai
- Cho phép admin khóa tài khoản thủ công
- Ghi log toàn bộ hành vi đăng nhập

Giao diện

Giao diện `/auth/login` là form HTML bao gồm 2 trường:

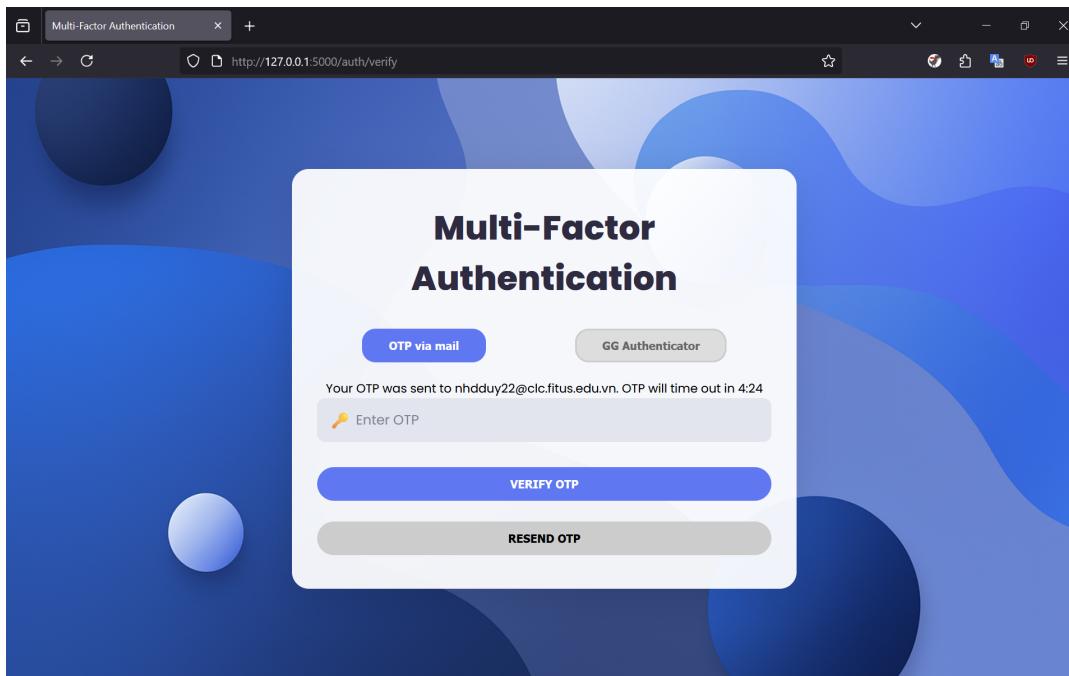
- Email
- Passphrase



Hình 4: Giao diện Login

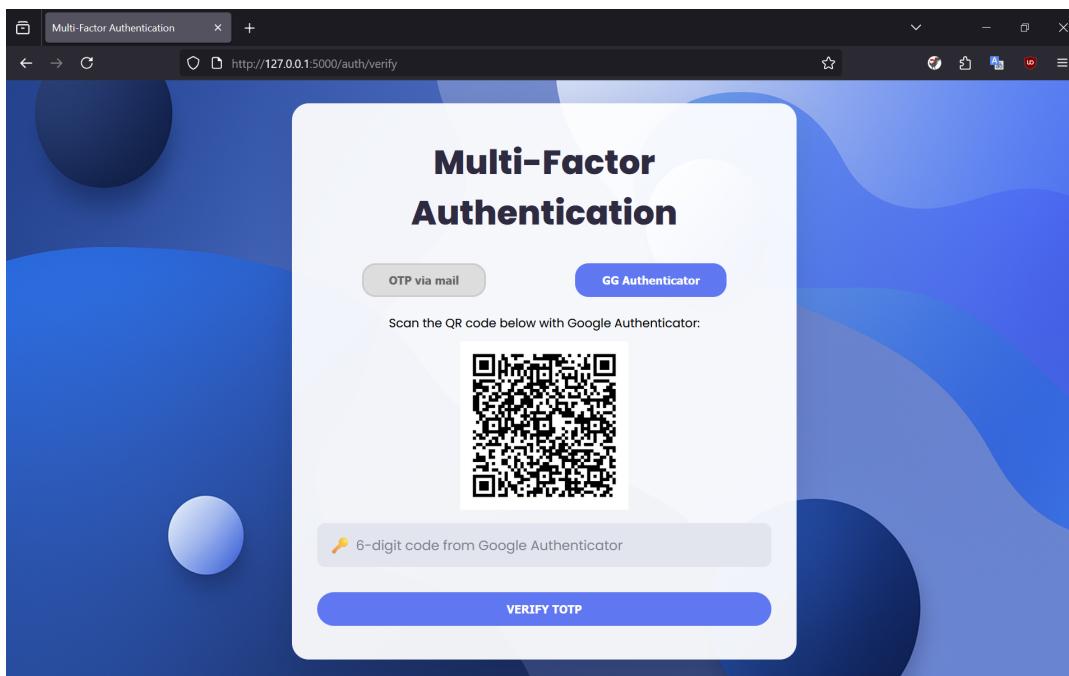
Sau khi đăng nhập thành công, người dùng được chuyển đến `/auth/verify` để xác thực MFA bằng 1 trong 2 phương thức:

1. OTP gửi qua email (mặc định)



Hình 5: Xác thực bằng OTP gửi qua mail

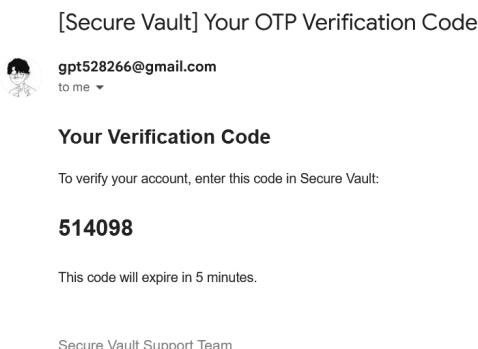
2. Mã TOTP (QR code cho Google Authenticator)



Hình 6: Xác thực bằng TOTP qua QR code

Quy trình thực hiện

1. Người dùng submit form `/auth/login` với `email` và `passphrase`.
2. Flask gọi hàm `process_login(email, passphrase)` trong `logic.py`
3. Chương trình thực hiện:
 - Kiểm tra người dùng tồn tại hay không.
 - So sánh `hash(passphrase + salt)`.
 - Nếu sai, tăng `failed_attempts` và cập nhật `last_failed_login`.
 - Nếu sai hơn 5 lần trong 2 phút → Gán `is_locked = 1`, trả về trạng thái **locked**.
 - Nếu tài khoản bị khóa bởi admin (`last_failed_login = null` và `is_locked = 1`) → trả về **locked by admin**.
 - Nếu các thông tin đăng nhập đúng → Reset `failed_attempts`, ghi log, chuyển đến bước xác thực MFA.
4. Người dùng được chuyển đến `/auth/verify`, chọn xác thực OTP qua email hoặc quét QR TOTP (mặc định là gửi OTP qua email đã đăng ký).
5. Nếu chọn OTP:
 - Gọi `generate_and_send_otp(email)` → sinh mã 6 chữ số, gửi qua email, lưu DB kèm thời gian hết hạn.



Hình 7: Cấu trúc mail

- Kiểm tra bằng `verify_otp_code(email, input_code)`.
6. Nếu chọn TOTP:
 - Sinh mã QR từ `generate_qr_code(email)` → dùng cho ứng dụng Google Authenticator.
 - Kiểm tra bằng `verify_totp_code(email, input_code)`.
7. Nếu đúng → Lưu `session['user_id']` và chuyển đến `auth/dashboard`

Chi tiết kỹ thuật và thư viện bảo mật

1. Hashing passphrase với Salt

Thư viện: `hashlib`

Kỹ thuật:

- Salt đã được sinh bằng `os.urandom(16).hex()` →.
- Kiểm tra `passphrase` được người dùng nhập vào sau khi hash có giống với chuỗi được lưu trong DB hay không.

2. Gửi mã OTP qua email

Thư viện: `random`, `datetime`, `smtplib`, `email`

Kỹ thuật:

- Sinh mã 6 chữ số ngẫu nhiên.
- Lưu `expires_at = now + 5 minutes`
- Dùng `email.mime.text` và `email.mime.multipart` để format tin nhắn gửi email.
- Tạo tài khoản gmail đã xác thực 2 yếu tố và tiến hành tạo `app password` và lưu vào `.env` với cấu trúc như sau

```
1 SMTP_USER=<sender mail>
2 SMTP_PASS=<app password>
```

- OTP được tạo sẽ lưu vào DB và gửi bằng SMTP → người dùng cần kiểm tra email để nhận OTP.
- Người dùng nhập mã OTP → Server sẽ lấy bản `otp_code` mới nhất và kiểm tra xem mã có đúng và còn hạn hay không → Nếu hợp lệ, người dùng xác thực thành công

3. TOTP

Thư viện: `pyotp`, `qrcode`, `base64`

Kỹ thuật:

- Mỗi tài khoản có một `mfa_secret` (chuỗi `base32`) sinh ngẫu nhiên và được lưu trong bảng `users`.
- Dùng `pyotp.totp.TOTP(mfa_secret).provisioning_uri()` để tạo ra URI định dạng chuẩn TOTP.
- Sinh ảnh QR code từ URI trên.
- Người dùng sử dụng Google Authenticator để quét mã trên.
- Người dùng nhập 6 số do app tạo ra mỗi 30 giây.
- Server sẽ dùng lại `mfa_secret` từ DB để sinh lại mã đúng tại thời điểm đó. Nếu mã số giống nhau thì người dùng xác thực thành công.

5.3 Quản lý khóa RSA cá nhân

Mục tiêu

Cho phép người dùng tạo, quản lý và sử dụng cặp khóa **RSA 2048** bit phục vụ các thao tác bảo mật như mã hóa tập tin, ký số,...

Private key được mã hóa bằng **AES-256** và public key được lưu dưới dạng PEM và chia sẻ qua QR

Giao diện

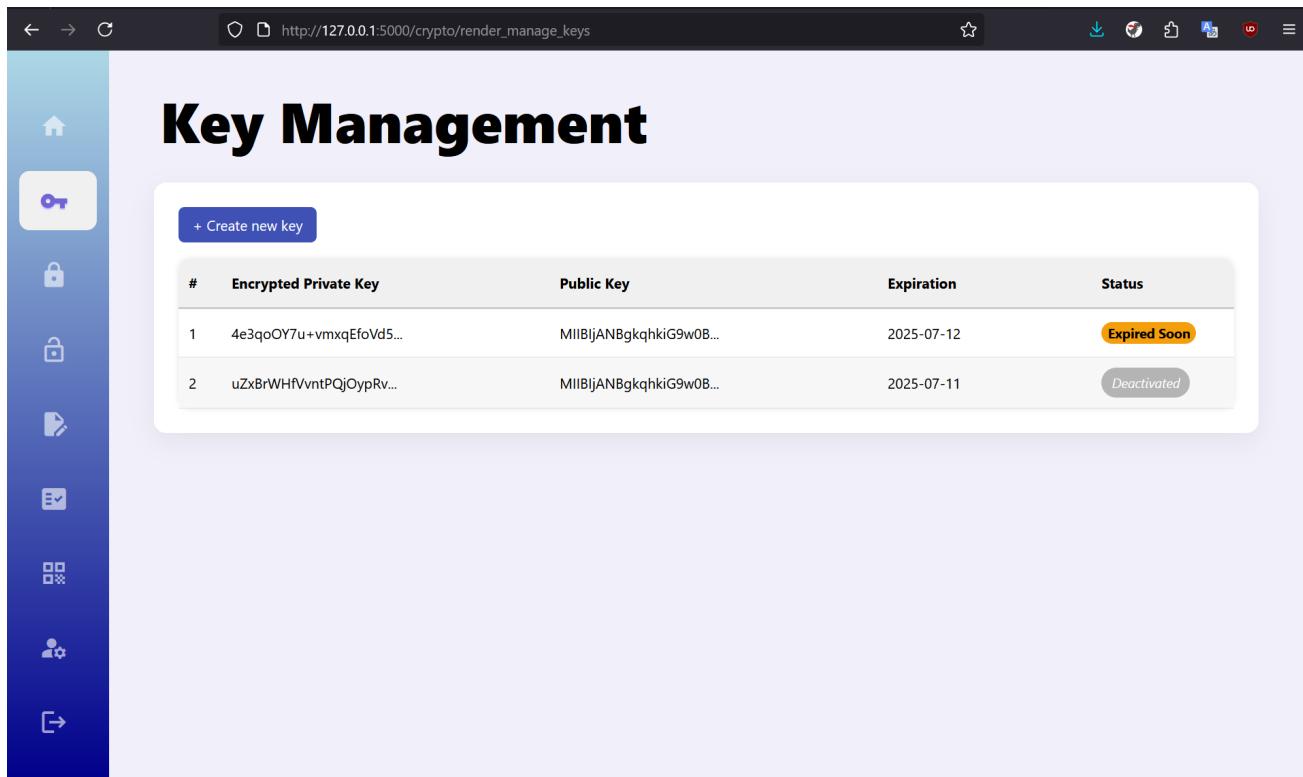
Giao diện `/crypto/render_manage_keys` hiển thị:

- Danh sách các khóa người dùng đã tạo, kèm ngày hết hạn và trạng thái.
- Nút `Create new key` để sinh cặp khóa mới

The screenshot shows a web browser window with the URL `http://127.0.0.1:5000/crypto/render_manage_keys`. The page title is "Key Management". On the left, there is a vertical sidebar with icons for Home, Key, Lock, Padlock, Copy, Paste, and others. The main content area displays a table of keys. The table has columns: #, Encrypted Private Key, Public Key, Expiration, and Status. There are two rows:

#	Encrypted Private Key	Public Key	Expiration	Status
1	4e3qoOY7u+vmxqEfoVd5...	MIIIBIjANBgkqhkiG9w0B...	2025-07-12	Active
2	uZxBrWHFVvntPQjOypRv...	MIIIBIjANBgkqhkiG9w0B...	2025-07-11	Deactivated

Hình 8: 2 key đã tạo, key 1 còn hạn và key 2 đã hết hạn



Hình 9: Giao diện key sắp hết hạn

Quy trình thực hiện

Khi truy cập vào `/manage_keys`

- Kiểm tra `session['email']` và `session['passphrase']`.
- Gọi `derive_aes_key(passphrase, salt)` để tạo AES key từ `passphrase`.
- Gọi `check_and_manage_own_keys(email, aes_key)`:
 - Nếu chưa có khóa → tạo khóa mới.
 - Nếu khóa đã hết hạn (quá 90 ngày) → tự động tạo khóa mới.
- Trả về danh sách các khóa trong thư mục `data/key_manage/<sha256(email)>/`.

Khi người dùng nhấn nút "**Create new key**", hệ thống sẽ thực hiện các bước sau:

- Gọi API `/regenerate_key`
- Backend gọi hàm `regenerate_key_handling(email, passphrase, aes_key, salt)`, bao gồm:
 - Gọi `create_new_key(email, aes_key)` để:
 - Hủy kích hoạt khóa cũ bằng cách cập nhật `status = deactivated`

- Tạo cặp khóa mới sử dụng thuật toán **RSA-2048**
- Mã hóa private key bằng thuật toán **AES-GCM** với khóa được dẫn xuất từ `passphrase`
- Chuỗi kết quả được lưu dưới dạng `base64(nonce + ciphertext)`
- Public key được lưu dưới định dạng PEM, kèm theo metadata như `expiry_date`, `owner_email`
- Tất cả được ghi vào file `key_<n>.json` (ví dụ: `key_1.json`, `key_2.json`) với hạn sử dụng mặc định là **90 ngày** kể từ ngày tạo
- Backend tiếp tục thực hiện phần mã hóa private key bằng `recovery code`:
 - Giải mã chuỗi `recovery code` đã lưu trong cơ sở dữ liệu bằng `passphrase`
 - Sử dụng `recovery code` để mã hóa lại private key (tạo một bản khác)
 - Ghi bản mã hóa này vào file riêng (phục vụ khôi phục tài khoản khi quên `passphrase`)

Quá trình này giúp đảm bảo mỗi người dùng luôn có một cặp khóa mới nhất ở trạng thái `active`, đồng thời hỗ trợ phục hồi an toàn thông qua recovery code nếu cần.

Chi tiết kỹ thuật và thư viện bảo mật

1. Sinh cặp khóa RSA

Thuật toán: RSA-2048 bit

Thư viện: `cryptography.hazmat.primitives.asymmetric.rsa`

Public key được lưu dưới định dạng PEM:

```
1 public_key = public_bytes(
2     encoding=serialization.Encoding.PEM,
3     format=serialization.PublicFormat.SubjectPublicKeyInfo
4 )
```

2. Mã hóa private key bằng AES-GCM

Thuật toán: AES-256 với chế độ GCM.

Khóa AES sinh từ `passphrase` thông qua `PBKDF2-HMAC-SHA256`

Thư viện:

- `cryptography.hazmat.primitives.ciphers.aead.AESGCM`
- `cryptography.hazmat.primitives.kdf.pbkdf2.PBKDF2HMAC`

Chi tiết mã hóa:

```
1 aes_key = PBKDF2HMAC(...).derive(passphrase.encode())
2 nonce = os.urandom(12) # 96-bit nonce
3 encrypted = AESGCM(aes_key).encrypt(nonce, private_pem, None)
```

Output: `base64(nonce + ciphertext)`

3. Giải mã khi sử dụng

Khi ký số, giải mã file,... hệ thống sẽ gọi `get_active_private_key()`:

- Từ `passphrase` và `salt` → derive lại AES key

- Dùng AES-GCM để giải mã private key
- Load lại object RSA

4. Hạn sử dụng khóa

Thư viện: `datetime`, `timedelta`

Chi tiết:

- Mỗi cặp khóa được gắn `creation_date` và `expiry_date` (90 ngày)
- Kiểm tra hạn sử dụng khi `/manage_keys` hoặc khi mã hóa
- Nếu hết hạn → hệ thống tự động gọi `regenerate_key_handling` để thay thế

5. Lưu trữ file theo người dùng

Thư viện: `pathlib.Path`, `hashlib.sha256`

Chi tiết:

- File khóa lưu trong: `data/key_manage/<sha256(email)>/`
- Mỗi file là một cặp khóa (`public_info` + `private_info`)
- Truy cập file an toàn, riêng biệt theo hash

5.4 QR Code Public Key

Mục tiêu

Cung cấp phương thức chia sẻ public key an toàn và thuận tiện giữa người dùng thông qua mã QR.

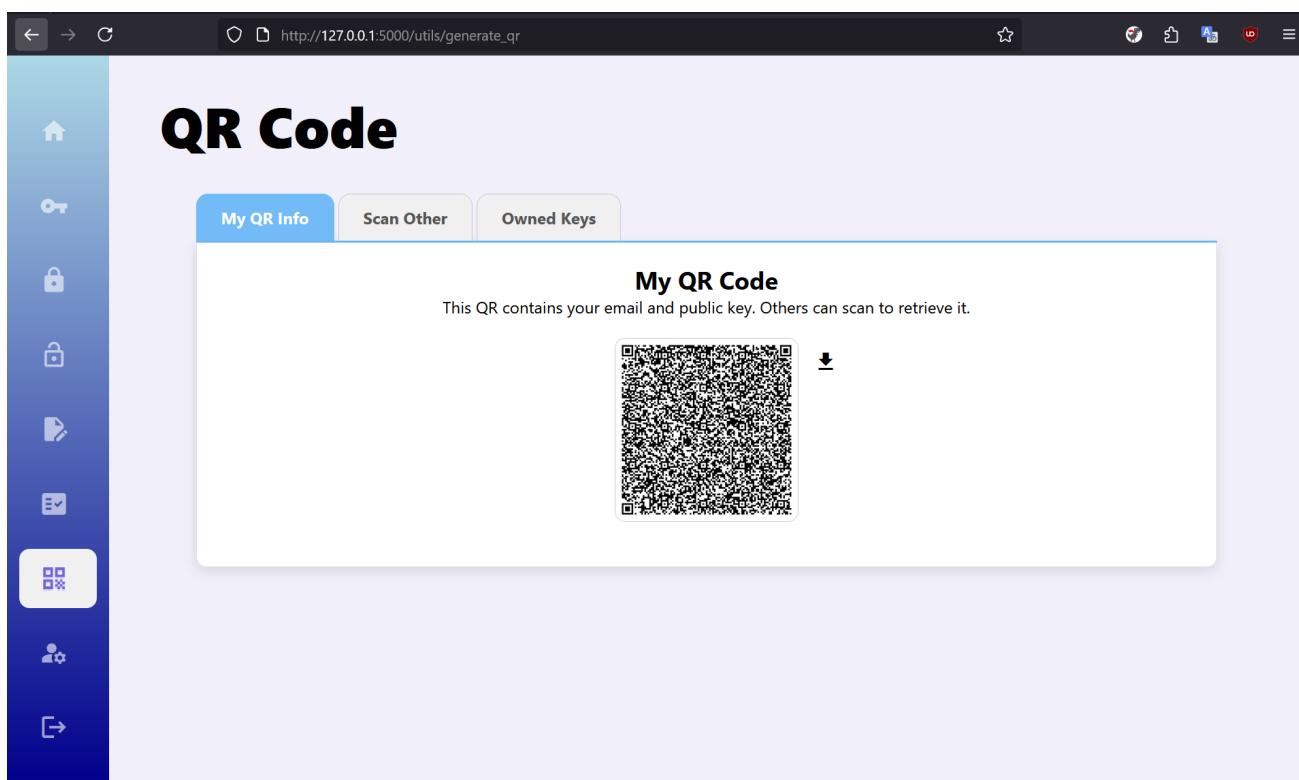
Người dùng có thể:

- Tạo mã QR chứa email + public key + các thông tin khác của mình
- Quét QR của người khác để lưu khóa công khai vào danh bạ
- Quản lý danh sách các khóa đã quét

Giao diện

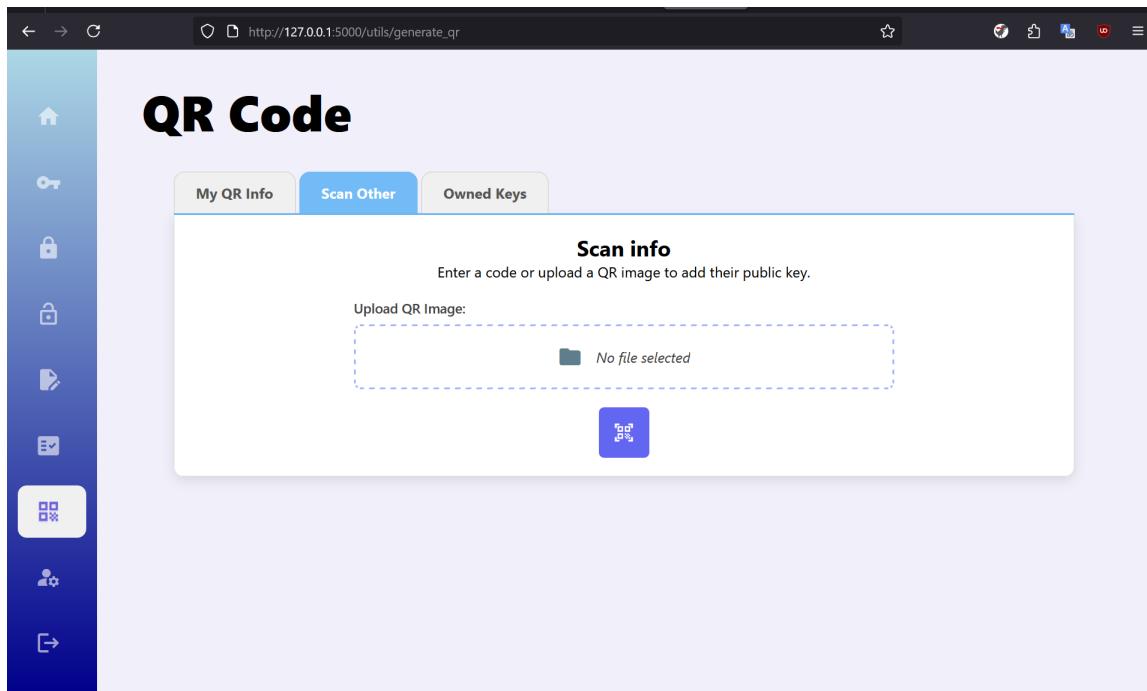
Có 3 tab chính trong trang `/utils/generate_qr`

1. **My QR Info:** Hiển thị QR chứa thông tin khóa công khai của tài khoản đang đăng nhập



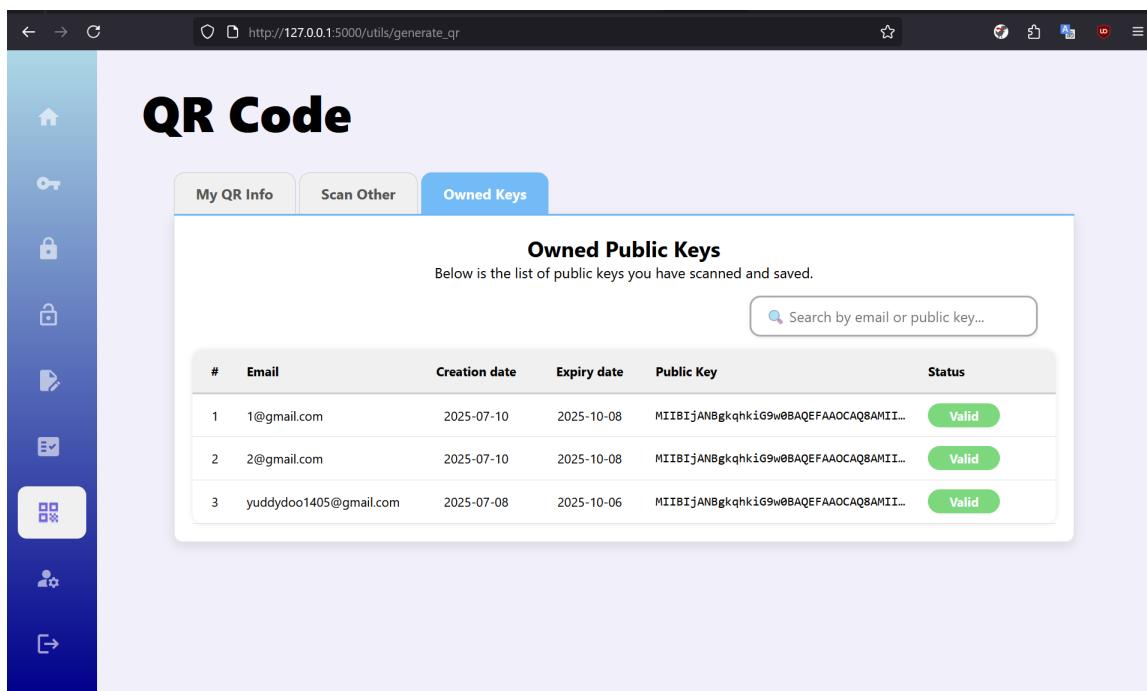
Hình 10: Tab My QR Info

2. Scan Other: Cho phép quét ảnh QR của người dùng khác

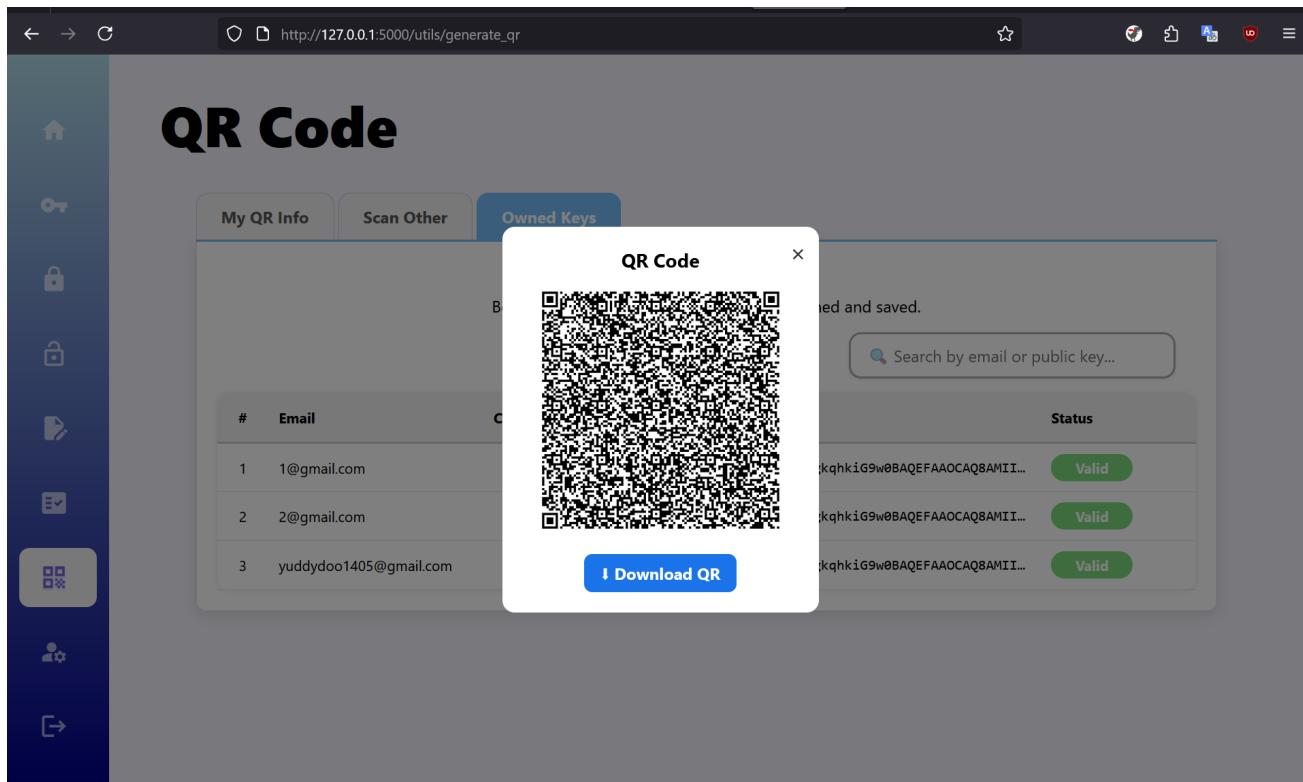


Hình 11: Tab Scan Other

3. Owned Keys: Hiển thị danh sách các public key đã quét từ QR. Khi nhấp vào 1 dòng, chương trình sẽ hiển thị ra QR của người dùng tương ứng



Hình 12: Tab Owned Keys



Hình 13: Xem QR của người dùng trong Owned Keys

Quy trình thực hiện

a) Tạo QR Code từ Public Key

Chương trình sẽ gọi `generate_public_info_qr(email, qr_path)` khi người dùng vào `/generate_qr`. Cụ thể:

1. Lấy public key từ `get_active_public_info(email)`:

- Đọc từ file `key_<n>.json` mới nhất trong thư mục `data/key_managesha256(email)>/`
- Chỉ lấy khóa `status == "active"`

2. Encode public key PEM:

```
public_key_b64 = base64.b64encode(public_key_pem.encode()).decode()
```

3. Tạo dữ liệu JSON cần encode QR:

```

1  {
2      "email": "user@example.com",
3      "creation_date": "2025-07-01T12:00:00",
4      "expired_date": "2025-09-30T12:00:00",
5      "public_key_b64": "<base64 string>"
6  }
```

4. Sinh QR bằng thư viện `qrcode`

5. Hiển thị ảnh QR lên giao diện:

- Tải từ route: /qr_image/<email_hash> do JS gọi /my_qr_url
- Được gắn vào trên giao diện

b) Quét ảnh QR của người khác để lưu public key

Chương trình sẽ gọi /decode_qr khi người dùng nhấn Scan QR. Cụ thể:

1. Upload file ảnh QR (PNG, JPG):

- Lấy request.files['qr_code_file']
- Đọc dữ liệu ảnh và chuyển thành numpy array để decode

2. Giải mã QR bằng pyzbar

3. Kiểm tra dữ liệu:

- Có đủ: email, creation_date, public_key_b64, expiry_date
- Không trùng current_user_email

4. Decode public key bằng base64.b64decode

5. Lưu vào danh bạ khóa người dùng hiện tại:

- File đích data/key_manage/<hash_email>/contact_public_key.json
- Dùng hàm: add_contact_public_key(user_dir, contact_email, public_info)

c) Quản lý các public key đã quét

1. Kiểm tra session['email']

2. Tìm thư mục người dùng bằng get_user_dir(email)

3. Mở file contact_public_key.json

4. Đọc toàn bộ các public key đã lưu, parse thành list.

5. Trả về dưới dạng JSON:

```
1  {
2      "success": true,
3      "data": [
4          {
5              "owner_email": "alice@example.com",
6              "public_key_pem": "-----BEGIN PUBLIC KEY-----...",
7              "creation_date": "2025-07-01",
8              "expiry_date": "2025-09-30",
9              "qr_image": "data:image/png;base64,..."
10         },
11         ...
12     ]
13 }
```

6. Frontend dùng `data` để render bảng

Chi tiết kỹ thuật và thư viện bảo mật

1. Tạo mã QR chứa public key cá nhân

Thư viện:

- `qrcode` – tạo QR từ chuỗi JSON
- `base64` – mã hóa public key PEM
- `json` – serialize dữ liệu QR
- `hashlib.sha256` – băm email để lưu QR riêng từng người

Chi tiết tạo:

```

1 json_data = {
2     "email": email,
3     "creation_date": ...,
4     "expired_date": ...,
5     "public_key_b64": base64.b64encode(public_key.encode()).decode()
6 }
7 qr_img = qrcode.make(json.dumps(json_data))
8 qr_img.save("data/qr/<hash_email>/my_qr.png")

```

2. Đọc và xử lý ảnh QR của người khác

Thư viện:

- `cv2` – đọc ảnh QR từ file ảnh (PNG, JPG)
- `pyzbar` – giải mã QR code
- `base64`, `json` – phân tích nội dung QR

Chi tiết đọc và xác thực:

```

1 decoded = pyzbar.decode(cv2_img)
2 qr_json = json.loads(decoded[0].data.decode("utf-8"))
3 public_key_pem = base64.b64decode(qr_json["public_key_b64"]).decode()

```

Dữ liệu được kiểm tra kỹ: email, ngày tạo, và không được trùng với chính người dùng hiện tại.

3. Lưu danh bạ public key

Thư viện:

- `pathlib.Path` – tạo thư mục riêng theo user
- `hashlib.sha256` – hash email làm thư mục
- `json` – lưu danh bạ ở dạng JSON

Chi tiết:

- Danh bạ lưu trong: `data/key_manage/<sha256(email)>/contact_public_key.json`

- Mỗi liên hệ là 1 entry theo định dạng:

```
1 {
2     "bob@example.com": {
3         "owner_email": "...",
4         "public_key_pem": "...",
5         "creation_date": "...",
6         "expiry_date": "..."
7     }
8 }
```

4. Hiển thị danh sách Owned Keys

Thư viện:

- `Flask` – route `/owned_keys` trả JSON danh bạ
- `JavaScript` – render dữ liệu vào bảng HTML

Chi tiết:

```
1 fetch("/owned_keys")
2     .then(res => res.json())
3     .then(data => renderPublicKeyTable(data.data));
```

Người dùng có thể tìm kiếm theo email hoặc nội dung khóa bằng hàm JS `filterPublicKeys()`

5.5 Cập nhật thông tin tài khoản

Mục tiêu

Chức năng cập nhật tài khoản cho phép người dùng chỉnh sửa thông tin cá nhân (họ tên, ngày sinh, địa chỉ, số điện thoại), cũng như thay đổi mật khẩu (passphrase). Việc này đảm bảo:

- Người dùng có thể chủ động thay đổi thông tin cá nhân khi cần.
- Cho phép cập nhật passphrase và tự động mã hóa lại khóa RSA, recovery code bằng passphrase mới.
- Dảm bảo tính bảo mật và tính nhất quán dữ liệu khóa trong hệ thống.

Giao diện

Giao diện tại `/render_update_account` bao gồm:

- Form nhập các trường: Họ tên, ngày sinh, địa chỉ, số điện thoại.
- Tùy chọn nhập mật khẩu hiện tại và mật khẩu mới nếu muốn đổi passphrase.
- Nút `Save changes` sẽ gửi POST đến API `/update_account` và hiển thị thông báo kết quả.

Hình 14: Giao diện cập nhật thông tin

Quy trình thực hiện

Bước 1 - Người dùng nhập thông tin Người dùng điền thông tin cần thay đổi, có thể chọn thay đổi passphrase hoặc không.

Bước 2 - Gửi yêu cầu cập nhật Form gửi POST đến route `/update_account`, kèm theo session để xác định người dùng.

Bước 3 - Kiểm tra và xử lý Server thực hiện:

- Kiểm tra dữ liệu đầu vào và định dạng.
- Kiểm tra passphrase cũ và mới nếu có yêu cầu đổi mật khẩu.
- Hash passphrase mới và cập nhật.
- Nếu đổi mật khẩu, khóa riêng RSA sẽ được giải mã bằng passphrase cũ và mã hóa lại bằng passphrase mới. Và cũng tương tự với recovery code sẽ được lấy từ database, giải mã và mã hóa lại với passphrase mới

Bước 4 - Ghi log và phản hồi Ghi log cập nhật tài khoản và trả kết quả thành công hoặc thất bại cho giao diện frontend.

The screenshot shows a user interface for updating account information. On the left is a sidebar with icons for home, key, lock, address, phone, email, and user. The main area has a title 'Update Account Information'. It contains fields for 'Full Name' (Phạm Quang Duy), 'Date of Birth' (11/11/1111), 'Address' (nhà của 1), 'Phone Number' (1111111111) and 'Email (read-only)' (1@gmail.com). Below these is a 'Change Passphrase' section with 'Old Passphrase' and 'New Passphrase' fields. A green button at the bottom right says 'Save Changes'. A green notification bar at the top right says 'Account information has been updated!'. There are three small circular icons on the right side of the main area.

Hình 15: Giao diện cập nhật thông tin thành công

Chi tiết kỹ thuật và thư viện bảo mật**1. Kiểm tra thông tin đầu vào**

- Sử dụng các hàm xác thực định dạng: `is_valid_email()`, `is_valid_date()`, `is_valid_phone()`.
- Bắt buộc điền đầy đủ thông tin: email, họ tên, ngày sinh, địa chỉ, số điện thoại.

2. Kiểm tra và cập nhật passphrase

- Nếu có ý định đổi passphrase, yêu cầu nhập cả pass cũ và pass mới.
- Pass mới phải khác pass cũ và đạt yêu cầu bảo mật qua hàm `is_strong_passphrase()`.
- So sánh hash passphrase cũ với DB bằng `hash_with_salt()`.
- Nếu hợp lệ, hash passphrase mới và cập nhật vào DB.

3. Re-encrypt RSA key với passphrase mới

- Hàm `re_encrypt_private_key_with_new_passphrase()` dùng để giải mã khóa RSA bằng passphrase cũ và mã hóa lại bằng passphrase mới.
- Nếu lỗi trong quá trình này, cập nhật DB sẽ được rollback và trả lỗi.
- Passphrase mới được cập nhật vào session để sử dụng cho các thao tác sau.

```
--- [BẮT ĐẦU QUÁ TRÌNH TÁI MÃ HOÁ PRIVATE KEY CHO 1@gmail.com] ---
Giải mã thành công private key bằng passphrase cũ.
Mã hoá lại thành công private key bằng passphrase mới.
Đã cập nhật thành công file key_1.json.
127.0.0.1 - - [11/Jul/2025 10:21:59] "POST /auth/update_account HTTP/1.1" 200 -
```

Hình 16: Mã hóa passphrase mới thành công

4. Cập nhật recovery key (khóa khôi phục)

Để đảm bảo người dùng có thể khôi phục khóa riêng sau khi quên passphrase, hệ thống sử dụng một **recovery key** (AES key phụ) được mã hóa bằng passphrase hiện tại. Khi người dùng thay đổi passphrase, recovery key cần được giải mã và mã hóa lại như sau:

- **Bước 1: Giải mã recovery key bằng passphrase cũ**
 - Truy vấn trường `encrypted_recovery_key` từ bảng `users`.
 - Derive khóa AES từ passphrase cũ bằng hàm `derive_aes_key(pass1, salt)`.
 - Giải mã recovery key hiện tại bằng AES.
- **Bước 2: Mã hóa lại recovery key bằng passphrase mới**
 - Derive AES key mới từ passphrase mới: `derive_aes_key(pass2, salt)`.
 - Mã hóa recovery key vừa giải mã bằng AES key mới.
 - Lưu bản mã hóa mới vào trường `encrypted_recovery_key`.
- **Lưu ý bảo mật:**
 - Nếu giải mã hoặc mã hóa recovery key thất bại, toàn bộ quá trình cập nhật sẽ bị rollback và trả lỗi.
 - Mọi thao tác đều được log bằng `log_user_action(..., "Fail", ...)` hoặc `log_internal_event()` với mức `error`.

5. Ghi log và bảo mật

- Mọi thao tác đều được ghi bằng `log_user_action()` với trạng thái và chi tiết lỗi.
- Lỗi truy cập không có session sẽ chuyển hướng về trang đăng nhập.
- Dữ liệu passphrase luôn được hash bằng salt, không lưu dưới dạng thô.

6. Xử lý lỗi

- Nếu thiếu session → lỗi 401 hoặc redirect về trang đăng nhập.
- Nếu định dạng sai hoặc passphrase cũ không đúng → trả lỗi rõ ràng.
- Nếu không có thay đổi gì → trả thông báo: "No changes were made."
- Nếu cập nhật thành công nhưng RSA re-encryption thất bại → rollback và log lỗi mức độ error.

The screenshot shows the 'Update Account Information' form. On the left is a vertical sidebar with icons for home, key, lock, file, user, and export. The main area has fields for Full Name (Phạm Quang Duy), Date of Birth (11/11/1111), Address (nhà của 1), Phone Number (1111111111) and Email (read-only) (1@gmail.com). Below these is a 'Change Passphrase' section with 'Old Passphrase' and 'New Passphrase' fields, both containing dots. A red box at the top right says: 'New passphrase is too weak. It must contain at least 8 characters, an uppercase letter, a number, and a special symbol.' A green 'Save Changes' button is at the bottom right.

Hình 17: Thông báo lỗi khi kiểm tra đầu vào

This screenshot is identical to Figure 17, showing the same 'Update Account Information' form. The red error message at the top right now says: 'Current passphrase is incorrect.' The rest of the interface, including the sidebar and form fields, remains the same.

Hình 18: Thông báo lỗi khi kiểm tra đầu vào

The screenshot shows a user interface for updating account information. On the left is a vertical sidebar with icons for home, key, lock, file, email, and user. The main area has a title 'Update Account Information'. It contains fields for 'Full Name' (Phạm Quang Duy), 'Date of Birth' (11/11/1111), 'Address' (nhà của 1), 'Phone Number' (1111111111), and 'Email (read-only)' (1@gmail.com). Below these is a 'Change Passphrase' section with 'Old Passphrase' and 'New Passphrase' fields. A green 'Save Changes' button is at the bottom right. In the top right corner, there's a message 'No changes were made.'

Hình 19: Thông báo khi không có thay đổi

5.6 Mã hóa tập tin gửi người khác

Mục tiêu

Cho phép người dùng mã hóa tập tin và gửi cho người nhận cũnghệ bằng cách sử dụng **public key đã lưu**. Tập tin sẽ được mã hóa bằng **AES-256-GCM** với session key ngẫu nhiên, sau đó session key được mã hóa bằng **RSA-OAEP** sử dụng public key của người nhận.

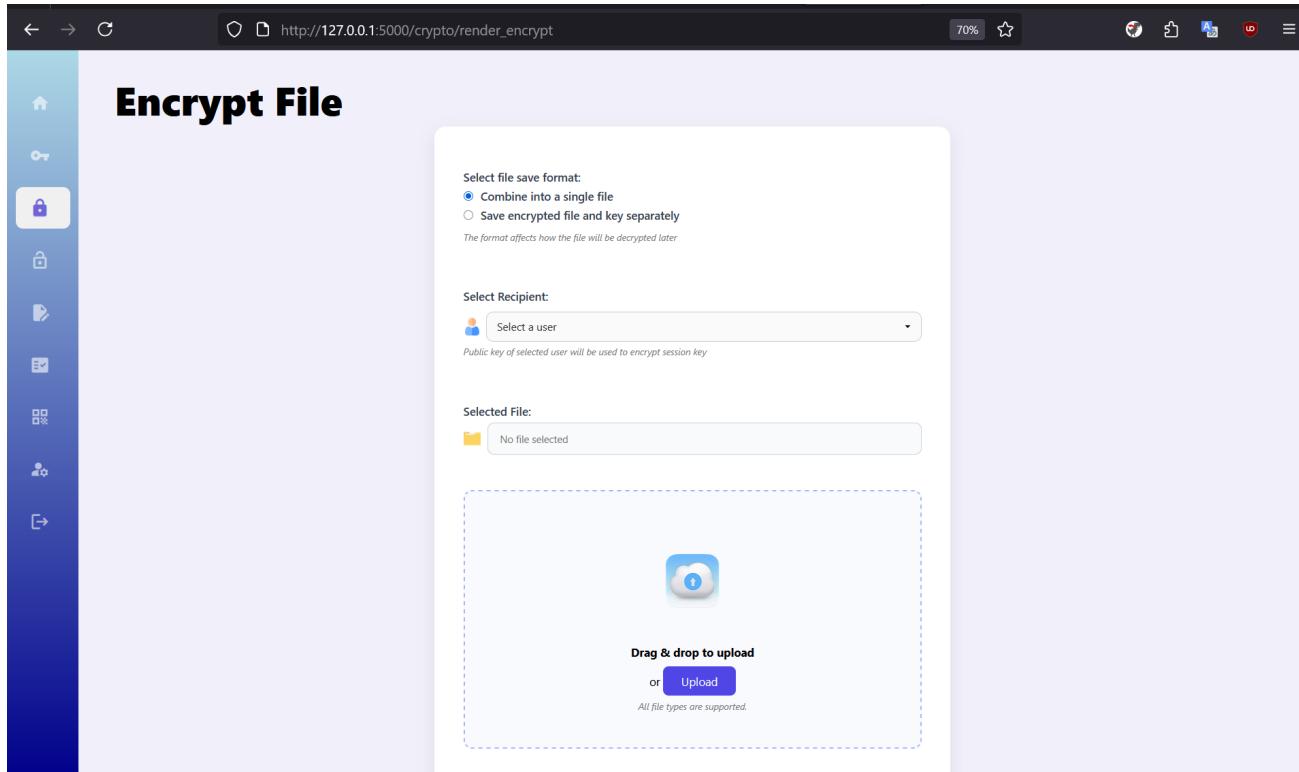
Người gửi có thể chọn một trong hai cách sau:

1. Gộp thành một file mã hóa duy nhất **.enc**
2. Tách riêng file dữ liệu **.enc** và key **.key** và lưu dưới định dạng **.zip**

Giao diện

Giao diện tại route **/render_encrypt** gồm các thành phần:

- Tùy chọn lưu: combine (1 file **.enc**) hoặc split (**.enc** + **.key**)
- Dropdown chọn người nhận (danh sách contact đã quét)
- Drag & Drop upload
- Nút "Encrypt file"



Hình 20: Giao diện mã hóa file

Quy trình thực hiện

a) Người dùng chọn:

- Tập tin muôn mã hóa (`file_to_encrypt`)
- Người nhận (dropdown `recipient_email`)
- Định dạng lưu (`save_format = combined / split`)

b) Server xử lý qua API `encrypt_file`:

1. Kiểm tra đầu vào:

- Xác thực phiên làm việc (`session`)
- Lấy public key của người nhận từ `contact_public_key.json`
- Kiểm tra hạn sử dụng của public key

2. Sinh session key và metadata:

- Session key: `os.urandom(32)` → AES-256
- Mã hóa session key bằng public key của người nhận (RSA-OAEP)
- Lưu metadata gồm:

```
1 {
2     "sender_email": "...",
3     "original_filename": "...",
4     "timestamp_utc": "...",
5     "algorithm": "AES-256-GCM",
6     "encrypted_session_key_b64": "...",
7     "mode": "chunked/full",
8     ...
9 }
```

3. Phân biệt theo kích thước:

- $\leq 5\text{MB}$ → mã hóa toàn bộ
- $> 5\text{MB}$ → chia khối 1MB, mỗi khối mã hóa độc lập với nonce riêng

4. Lưu định dạng tùy chọn:

- Gộp (`combined`): ghi `metadata + encrypted_content` vào `.enc`
- Tách riêng (`split`): `.enc` chứa dữ liệu, `.key` chứa metadata JSON

Chi tiết kỹ thuật và thư viện bảo mật

1. Sinh session key và metadata

Thư viện:

- `os.urandom(32)` – sinh khóa AES ngẫu nhiên (256-bit)
- `base64` – mã hóa khóa
- `json` – lưu thông tin metadata

2. Mã hóa dữ liệu bằng AES-256-GCM

Thuật toán: AES-GCM Nonce: 12 byte (`os.urandom(12)`)

Thư viện: `cryptography.hazmat.primitives.ciphers.aead.AESGCM`

Mỗi khối:

```
1 nonce = os.urandom(12)
2 ciphertext = aesgcm.encrypt(nonce, chunk_data, None)
```

3. Mã hóa khóa phiên bằng RSA-OAEP

Thuật toán: RSA-OAEP + SHA-256

Thư viện:

- `cryptography.hazmat.primitives.asymmetric.padding.OAEP`
- `serialization.load_pem_public_key()`

```
1 encrypted_session_key = recipient_public_key.encrypt(
2     session_key,
3     asym_padding.OAEP(
4         mgf=asym_padding.MGF1(hashes.SHA256()),
5         algorithm=hashes.SHA256(),
6         label=None
7     )
8 )
```

4. Chia khối file lớn

Nguồn: 5MB

Kích thước khối: 1MB

Thư viện: `io`, `os`, `stream.read()`

Cơ chế:

- Mỗi khối: [4 byte độ dài] + nonce + ciphertext
- Viết tuần tự vào file `.enc`

5.7 Giải mã tập tin

Mục tiêu

Cho phép người dùng giải mã tập tin được gửi từ người khác bằng public key của người gửi và private key của chính mình.

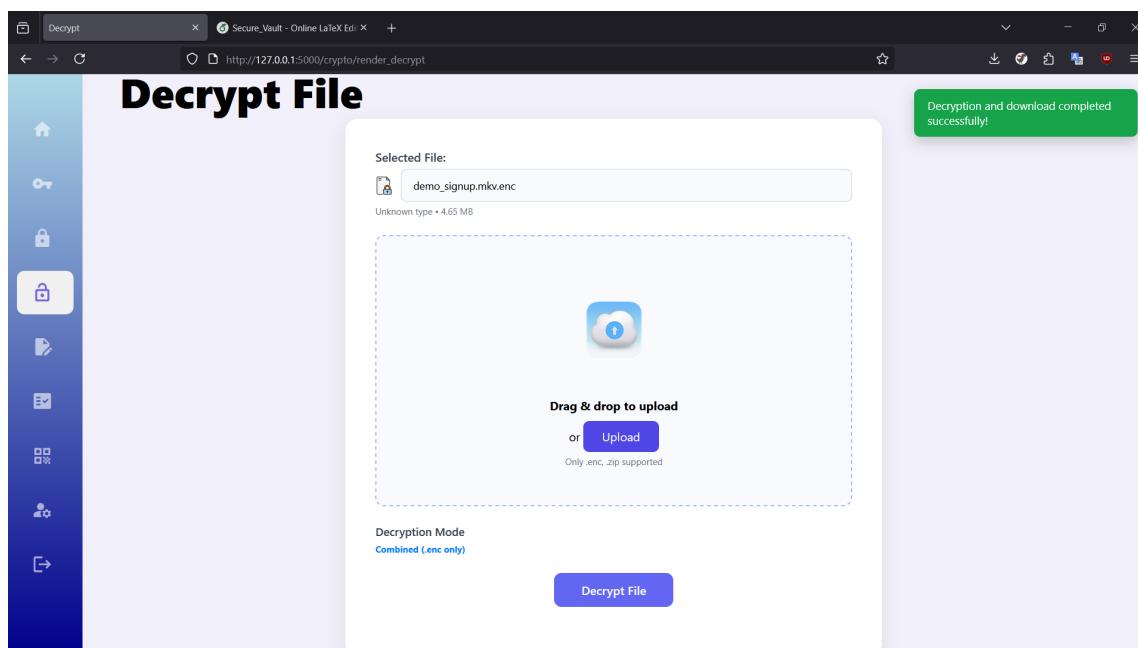
Hệ thống hỗ trợ 2 chế độ giải mã:

- File mã hoá gộp (`.enc`)
- File `zip` chứa hai file mã hóa được tách riêng (`.enc` + `.key`)

Giao diện

Giao diện tại route `/render_decrypt` gồm các thành phần:

- Drag & Drop upload
- Decryption Mode: Tự động nhận diện kiểu encrypt
- Nút "Decrypt file"



Hình 21: Giao diện giải mã file

Quy trình thực hiện

1. Người dùng upload file mã hoá (`.enc` hoặc `.zip`)
2. Backend (`/crypto/decrypt`) nhận và kiểm tra:
 - Phiên hoạt động (`session`)

- Tìm private key đang hoạt động của người dùng
- Xác thực file có đúng định dạng mã hoá hay không

3. Xử lý metadata:

- Nếu chế độ `combined`: đọc metadata từ phần đầu file
- Nếu chế độ `split`: load file `.key` chứa metadata JSON

4. Giải mã session key: Lấy `encrypted_session_key_b64` → giải mã bằng private key RSA (OAEP)

5. Dùng AES-GCM để giải mã nội dung file:

- Nếu file chia khối: giải mã từng khối (có nonce riêng)
- Nếu file toàn khối: giải mã toàn bộ với 1 nonce

6. Trả lại file gốc (tên và định dạng từ `metadata.original_filename`)

Chi tiết kỹ thuật và thư viện bảo mật

1. Giải mã session key bằng RSA-OAEP

Thư viện:

- `cryptography.hazmat.primitives.asymmetric.rsa`
- `OAEP padding` và `SHA256`

Chi tiết:

```

1 private_key.decrypt(
2     encrypted_session_key,
3     padding.OAEP(
4         mgf=padding.MGF1(algorithm=hashes.SHA256()),
5         algorithm=hashes.SHA256(),
6         label=None
7     )
8 )
```

Session key là AES key gốc dùng để giải mã dữ liệu.

2. Giải mã dữ liệu bằng AES-GCM

Thư viện:

- `AESGCM`
- `base64` – decode nonce và ciphertext

Giải mã từng khối:

```
1 AESGCM(session_key).decrypt(nonce, ciphertext, None)
```

Trong chế độ chia khối:

- Đọc [4 byte] độ dài → nonce (12 byte) → ciphertext

- Lặp lại cho đến hết file

3. Tải về tập tin gốc

Sau khi giải mã thành công: - Gửi response dạng `send_file` với tên gốc (`original_filename`)

- Nếu giải mã lỗi → bắt và hiển thị thông báo phù hợp

4. Bảo vệ file giải mã

- Private key người dùng được giải mã bằng passphrase trước khi dùng - Kiểm tra trạng thái khóa (`status == "active"`) - Giới hạn chỉ người nhận có private key tương ứng mới giải mã được

5. Hỗ trợ giải nén file .zip

Hệ thống sẽ tự động phát hiện định dạng `.zip`, và thực hiện giải nén tạm thời vào thư mục `data/temp/` để xử lý.

Thư viện sử dụng:

- `zipfile` – mở và giải nén file zip
- `tempfile`, `os` – xử lý thư mục tạm
- `pathlib` – truy xuất file vừa giải nén

Chi tiết xử lý:

```
1 with zipfile.ZipFile(zip_file_stream, 'r') as zip_ref:  
2     zip_ref.extractall(temp_dir)  
3  
4 enc_path = next(temp_dir.glob("*.enc"))  
5 key_path = next(temp_dir.glob("*.key"))
```

Bảo mật:

- Chỉ chấp nhận file zip chứa đúng 2 file: `.enc` và `.key`
- Kiểm tra MIME type và đuôi file → từ chối file lạ
- Xóa toàn bộ thư mục tạm sau khi xử lý để tránh lưu file nhạy cảm

5.8 Ký số tập tin

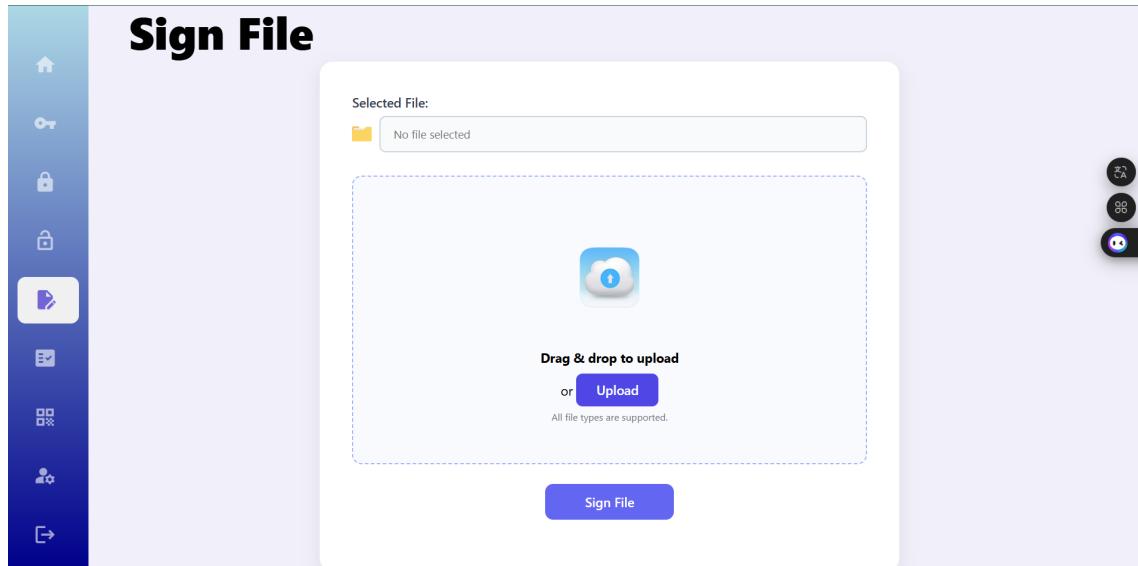
Mục tiêu

Chức năng ký số cho phép người dùng xác minh tính **toàn vẹn** và **nguồn gốc xác thực** của tập tin. Người gửi sử dụng private key RSA để ký tập tin, từ đó sinh ra một file chữ ký **.sig**. Người nhận có thể xác minh bằng public key tương ứng.

Giao diện

Giao diện `/utils/sign_file` cho phép:

- Chọn file bất kỳ để ký
- Submit và tải về file chữ ký **.sig**
- Hiển thị thông báo (toast) khi ký thành công hoặc lỗi

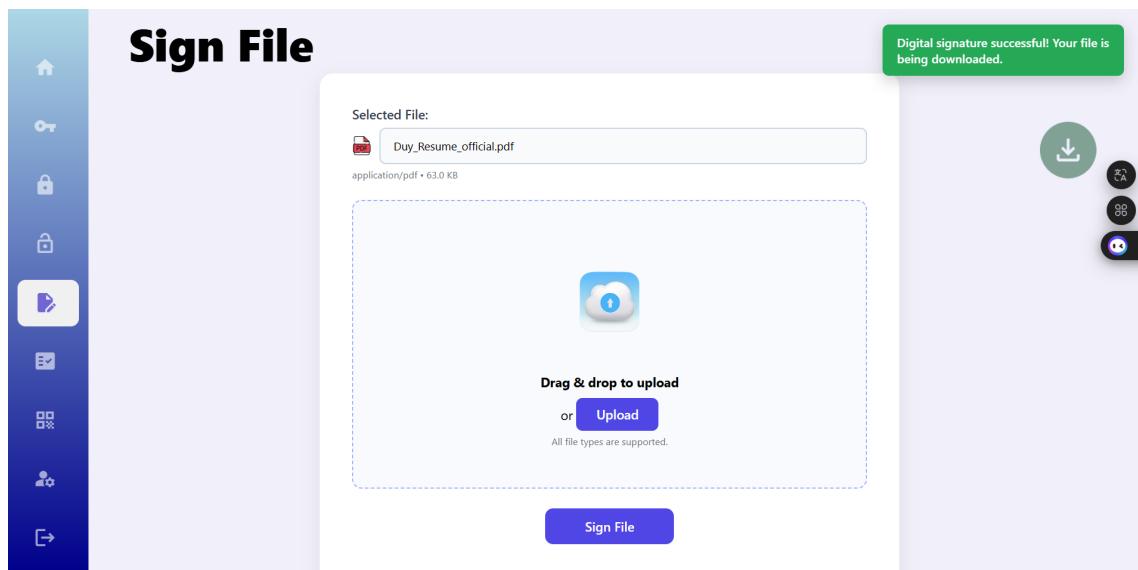


Hình 22: Giao diện ký số và nút tải file chữ ký

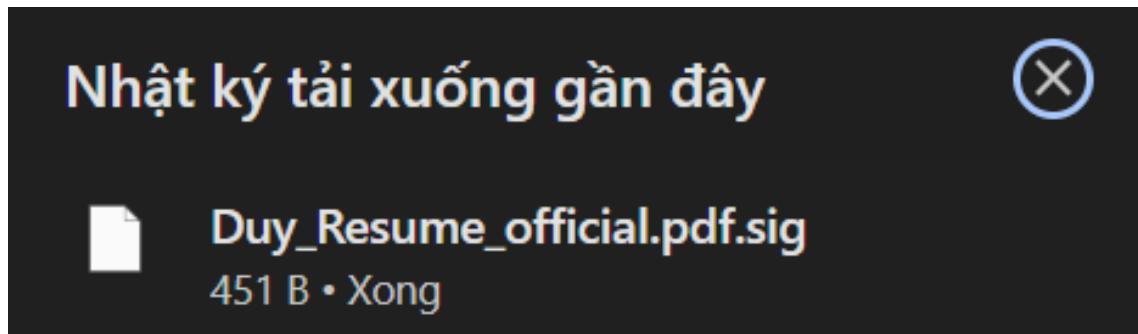
Quy trình thực hiện

Bước 1 - Frontend: Người dùng chọn file và submit form. File được gửi lên server bằng `fetch()` qua `FormData`:

- Gửi POST đến `/utils/sign_file`
- Header `Content-Type` là `multipart/form-data`
- Khi server trả về file **.sig**, frontend tạo link download động và tự động tải về



Hình 23: Ký số thành công



Hình 24: Tự động tải file chữ ký

Bước 2 - Flask Route: Route `/sign_file` kiểm tra:

1. Đã đăng nhập và có session
2. Có passphrase trong session để giải mã khóa riêng
3. Tìm salt người dùng → derive AES key → giải mã private key
4. Gọi hàm `digital_sign_file(file, private_key)`
5. Trả về file .sig cho trình duyệt dưới dạng `application/octet-stream`

Bước 3 - Xử lý ký số (Modules): Hàm `digital_sign_file()` thực hiện:

1. Đọc toàn bộ nội dung file → tính hash SHA-256
2. Ký bằng `private_key.sign(...)` sử dụng PKCS1v15 + SHA-256
3. Base64 hóa chữ ký → tạo JSON `{"filename": ..., "signature": ..., "timestamp": ...}`

4. Ghi file chữ ký .sig vào thư mục data/signature/

```
{
  "timestamp": "2025-07-11T10:51:36.036912",
  "filename": "Duy_Resume_official.pdf",
  "signature": "10gcX9kkDu3nSwp4p2TBjJ7edvggYOa5l+GLS3b1HNC1iVMz8bZdOPePJ618koSxn6W7bbCRTNsE9rAs/"
}
```

Hình 25: Ghi file chữ ký

Chi tiết kỹ thuật và thư viện bảo mật

1. Ký số SHA-256 + RSA: Quá trình ký số bắt đầu bằng việc băm nội dung tập tin bằng SHA-256, sau đó dùng khóa riêng RSA để tạo chữ ký số.

- Dùng `hashlib.sha256(file_bytes).digest()` để tính hash
- Dùng `cryptography.hazmat.primitives.asymmetric.rsa` để ký
- Padding: `PKCS1v15()`, Hash: `SHA256()`

2. Mã hóa và giải mã private key: Private key được bảo vệ bằng AES sử dụng passphrase người dùng. Khi cần ký, passphrase được lấy từ session để giải mã khóa.

- Private key được AES hóa bằng passphrase người dùng khi tạo khóa
- Để ký, passphrase được lấy từ session và dùng để giải mã khóa

3. Ghi log: Hệ thống ghi lại đầy đủ mọi thao tác ký dưới dạng log bảo mật – gồm cả hành động của người dùng và sự kiện nội bộ hệ thống.

- **Log người dùng:** sử dụng `log_user_action()` ghi lại email, hành động, trạng thái và lỗi nếu có
- **Log nội bộ:** `log_internal_event()` lưu các sự kiện kỹ thuật như “Signed filename successfully...”

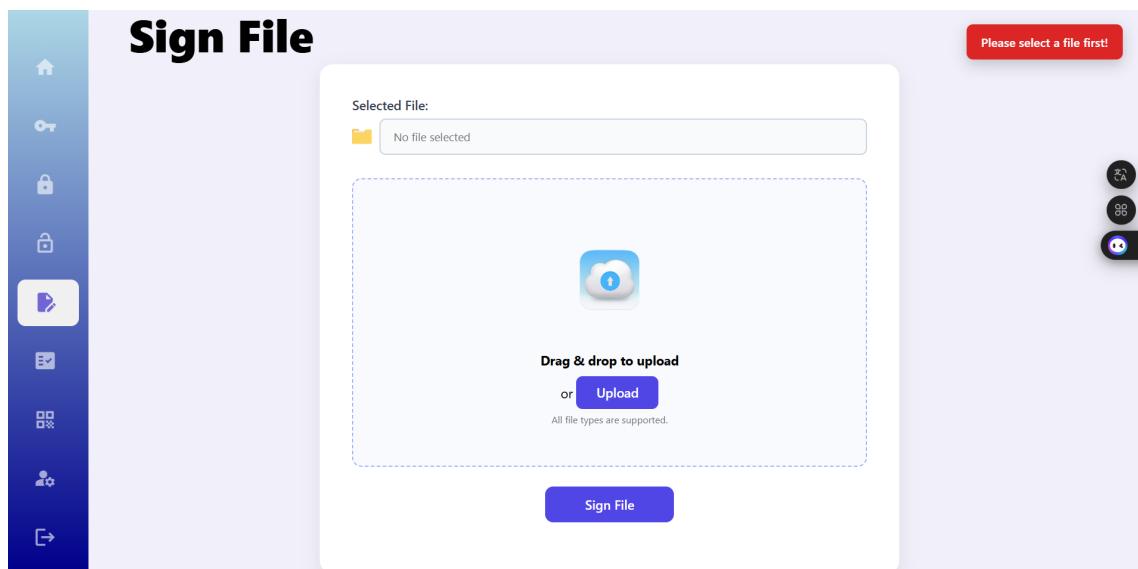
4. Bảo mật tổng thể: Quy trình ký số đảm bảo private key không bao giờ rời khỏi server, ngăn rò rỉ khóa, đồng thời có thể truy vết mọi hoạt động.

- Tránh truyền private key về phía client – xử lý toàn bộ phía server
- Tất cả chữ ký lưu kèm timestamp và filename để tăng khả năng truy vết
- Chỉ người dùng đã đăng nhập mới được phép truy cập chức năng ký

5. Xử lý lỗi và báo lỗi chi tiết: Hệ thống đảm bảo mọi tình huống lỗi trong quá trình ký số đều được kiểm tra, phản hồi rõ ràng cho frontend, đồng thời ghi log đầy đủ để phục vụ giám sát bảo mật.

- Kiểm tra session: Nếu chưa đăng nhập → trả lỗi 401 với thông báo "You must be logged in to access this page.".

- Kiểm tra file đầu vào: Nếu không có file hoặc tên file rỗng → trả lỗi 400: "No file provided.".
- Kiểm tra passphrase: Nếu thiếu → trả lỗi 401: "Passphrase not found.".
- Kiểm tra lỗi phát sinh khi đọc salt, giải mã khóa riêng hoặc ký file:
 - Nếu bất kỳ bước nào bị lỗi → trả lỗi 500 kèm thông báo: "Error during digital signing: <error>".
 - Toàn bộ lỗi được ghi log với level="error" để phục vụ phân tích.
- Ghi log thất bại bằng log_user_action(..., "Fail", ...) với lý do chi tiết cho từng trường hợp.



Hình 26: Báo lỗi nếu chưa chọn file

5.9 Xác minh chữ ký

Mục tiêu

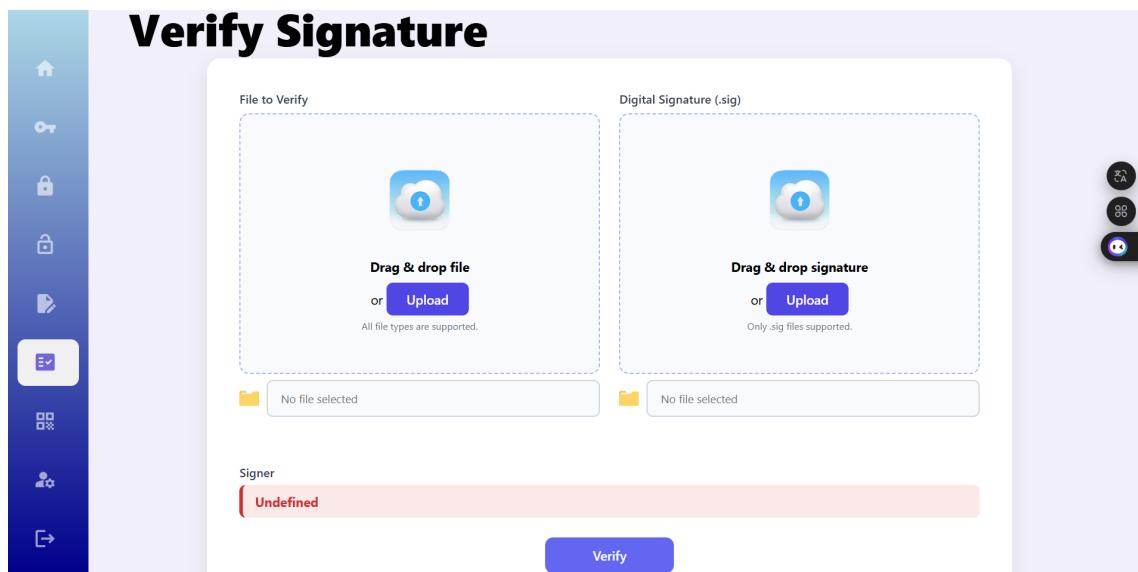
Chức năng xác minh chữ ký cho phép người dùng kiểm tra tính toàn vẹn và nguồn gốc của một tập tin bất kỳ thông qua file chữ ký ‘.sig’ do người gửi tạo ra. Việc xác minh này giúp đảm bảo:

- Tập tin không bị chỉnh sửa sau khi ký.
- Người nhận có thể xác định chính xác ai là người ký.
- Thời điểm ký được xác minh và hiển thị lại rõ ràng.

Giao diện

Giao diện xác minh tại trang `/crypto/verify` bao gồm:

- Trường upload 2 file: tập tin gốc cần xác minh và file chữ ký ‘.sig’.
- Nút “Xác minh chữ ký” gửi yêu cầu lên server bằng `fetch()`.
- Kết quả xác minh hiển thị thông tin người ký và thời điểm ký nếu hợp lệ.
- Toast thông báo xác minh thành công hoặc thất bại.



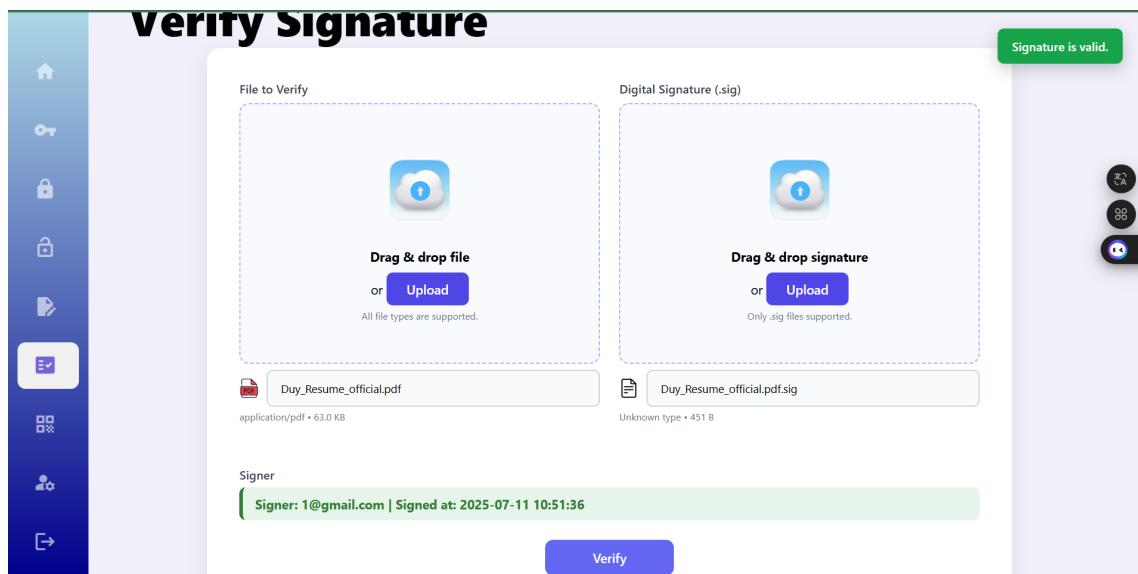
Hình 27: Giao diện xác minh ký số

Quy trình thực hiện

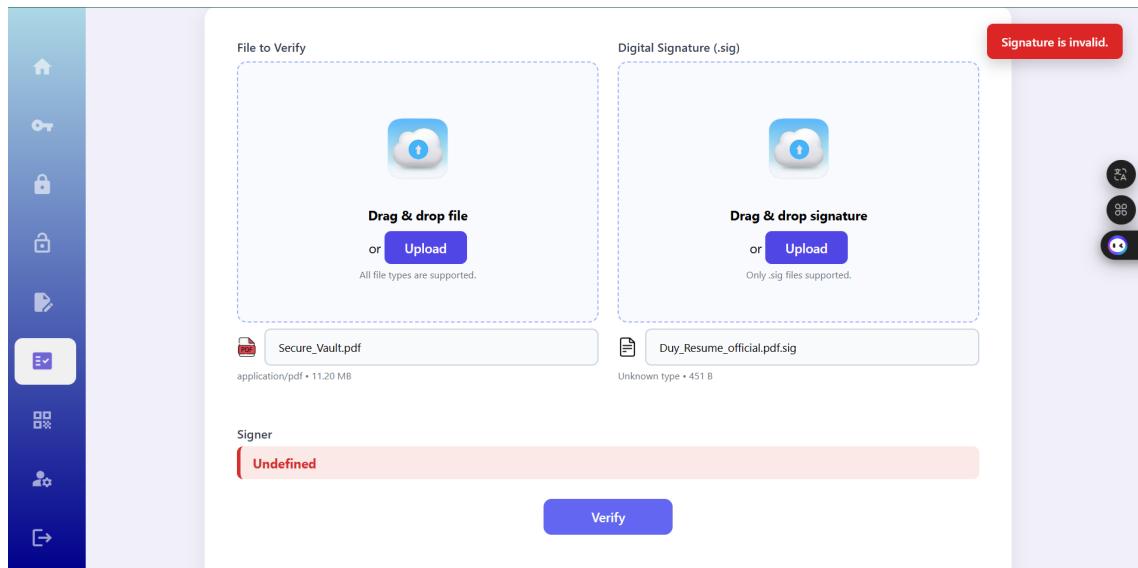
Bước 1 - Frontend: Người dùng chọn tập tin và file chữ ký ‘.sig’, sau đó gửi lên server bằng `FormData`.

- Gửi POST đến `/utils/verify_signature`.

- Gửi kèm 2 trường: `file_to_verify` và `signature`.
- Nếu kết quả hợp lệ → hiển thị tên người ký và thời điểm ký.



Hình 28: Xác minh ký số thành công



Hình 29: Chữ ký không được xác minh

Bước 2 - Flask Route: Route xử lý tại `/verify_signature`.

1. Kiểm tra session và sự tồn tại của cả 2 file.
2. Đọc nội dung file chữ ký và parse JSON.
3. Đọc danh sách public key từ file `contact_public_key.json`.

4. Gọi hàm `verify_signature(...)` để xác minh chữ ký.
5. Ghi log kết quả (thành công hoặc thất bại) và trả về thông tin người ký.

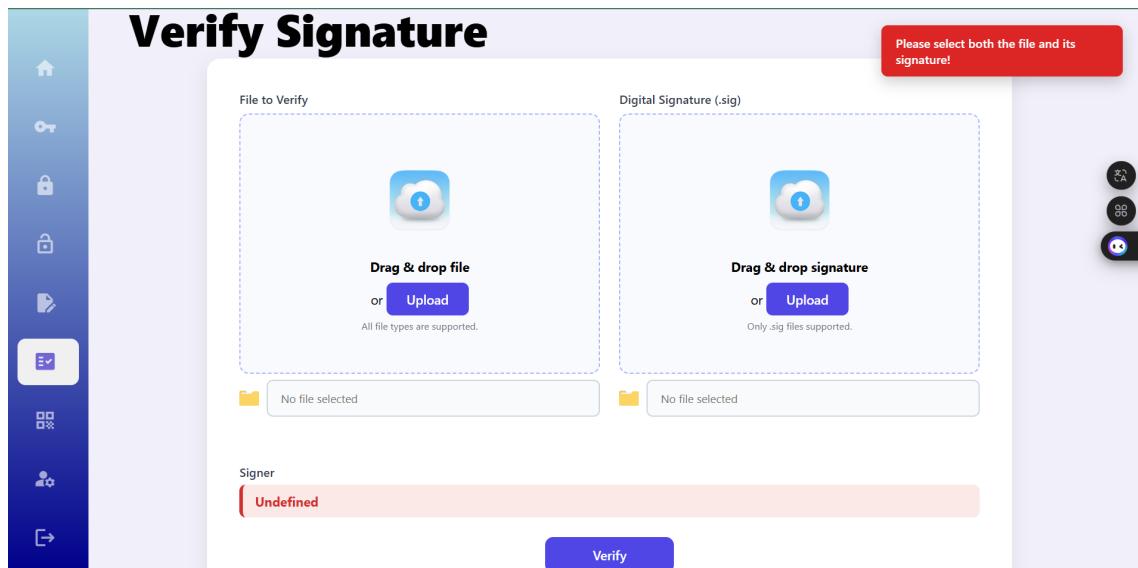
Bước 3 - Xử lý xác minh (Modules): Hàm `verify_signature()` thực hiện:

1. Tính hash SHA-256 của tập tin gốc.
2. Dò từng public key trong danh sách để thử xác minh.
3. Nếu khớp → trả về email người ký và thời gian.
4. Nếu không khớp → trả lỗi “Signature is invalid”.

Chi tiết kỹ thuật và thư viện bảo mật

- 1. Phân tích file chữ ký:** File chữ ký ‘.sig’ là JSON chứa chữ ký (dạng base64) và timestamp. Hệ thống decode và parse dữ liệu để chuẩn bị xác minh.
 - Đọc bằng `.read().decode('utf-8')` và xử lý JSON với `json.loads(...)`.
 - Giải mã chữ ký với `base64.b64decode(...)`.
- 2. Duyệt danh sách public key:** Server đọc file `contact_public_key.json` chứa public key của các người gửi.
 - Public key lưu dưới dạng PEM, được parse bằng `serialization.load_pem_public_key()`.
 - Với mỗi key, thử gọi `public_key.verify(...)` để kiểm tra.
- 3. Tính hash và xác minh:** Server tính hash SHA-256 của file gốc và dùng RSA-PKCS1v15 để xác minh chữ ký.
 - Hash bằng: `hashlib.sha256(file_data).digest()`.
 - Padding: `PKCS1v15()`, Hash: `SHA256()`.
 - Nếu đúng → xác minh thành công và log email + timestamp.
- 4. Xử lý lỗi và bảo mật:** Hệ thống có các bước kiểm tra và log lỗi đầy đủ:
 - Báo lỗi nếu thiếu file, sai định dạng hoặc không tìm thấy key.
 - Ghi log bằng `log_user_action()` và `log_internal_event()` theo mức độ `info`, `warning`, hoặc `error`.
 - Không lưu lại nội dung file, chỉ log metadata để đảm bảo riêng tư.
- 5. Xử lý lỗi và báo lỗi chi tiết:** Hệ thống được thiết kế để phát hiện và phản hồi rõ ràng với các lỗi có thể xảy ra trong quá trình xác minh chữ ký, đồng thời ghi log đầy đủ theo từng tình huống cụ thể.
 - **Kiểm tra đăng nhập:** Nếu session không tồn tại, trả về mã lỗi 401 cùng thông báo: “You must be logged in to access this page.”.
 - **Thiếu file:** Nếu người dùng không gửi cả file cần xác minh hoặc file chữ ký → trả lỗi 400 kèm thông báo: “No file or signature provided”.

- **Tên file trống:** Nếu `file.filename == ""` → trả lỗi 400 với thông báo: "Tên file trống".
- **Lỗi định dạng chữ ký:**
 - Nếu không đọc hoặc decode được nội dung file chữ ký JSON → trả lỗi 400, thông báo: "Invalid signature file format".
 - Log lỗi này ở mức `error` vì có thể liên quan đến file bị sửa hoặc không hợp lệ.
- **Thiếu danh sách public key:**
 - Nếu không tồn tại file `contact_public_key.json` → trả lỗi 400 với thông báo: "Public key does not exist.".
 - Lỗi được log lại kèm tên người dùng để hỗ trợ debug.
- **Chữ ký không hợp lệ:**
 - Nếu không có public key nào xác minh được chữ ký → trả lỗi 400 với thông báo: "Signature is invalid.".
 - Thông tin file được log kèm để phục vụ điều tra.
- **Log chi tiết:** Mỗi lỗi đều được ghi bằng `log_user_action(...)` với trạng thái "Fail" và ghi rõ lý do, đồng thời có thể bổ sung thêm `log_internal_event()` để phân tích kỹ thuật.



Hình 30: Kiểm tra có đủ file đầu vào

5.10 Phân quyền tài khoản

Mục tiêu

Quản trị viên có quyền truy cập đặc biệt nhằm đảm bảo hệ thống vận hành an toàn:

- Quản lý người dùng cuối (user)
- Khóa hoặc mở khóa tài khoản có hành vi bất thường
- Theo dõi nhật ký hành động trên toàn hệ thống

Giao diện

1. Trang Admin Dashboard (/admin_dashboard)

- Hiển thị toàn bộ log bảo mật (security.log)
- Hỗ trợ lọc, tìm kiếm theo email, action, trạng thái

The screenshot shows a web browser window with the URL `http://127.0.0.1:5000/auth/admin_dashboard`. The main title is "Admin Dashboard". On the left, there is a sidebar with icons for shield, user, and export. The main content area is titled "Security Logs" and contains a search bar with placeholder "Tìm kiếm log...". Below is a table with the following data:

#	Level	Time	Email	Action	Status	Message
87	ERROR	2025-09-10 08:30:31	1@gmail.com	Login	Fail	Wrong email or password
88	ERROR	2025-09-10 08:30:45	1@gmail.com	Login	Fail	Wrong email or password
89	INFO	2025-09-10 08:30:56	1@gmail.com	Login	Success	Login successfully
90	INFO	2025-09-10 08:30:59	1@gmail.com	Send OTP	Success	OTP sent via email
91	WARNING	2025-09-10 08:31:06	1@gmail.com	TOTP Verify	Fail	Wrong 6-digit TOTP

Hình 31: Giao diện Admin Dashboard

2. Trang List Users (admin_manage_account)

- Bảng danh sách người dùng: email, họ tên, số điện thoại, trạng thái
- Popup chi tiết khi click 1 hàng: ngày tạo, địa chỉ, số lần sai mật khẩu, lần đăng nhập gần nhất

- Nút Lock / Unlock tương ứng với từng user

#	Fullname	Email	Phone	DOB	Status
1	ABC	abcABC@clc.fitus.edu.vn	0123456798	2004-02-03	LOCKED
2	scriptalert1/script	abc@gmail.com	0999999999	2002-12-01	ACTIVE
3	Pham Quang Duy	pqduy22@gmail.com	0123456789	2002-12-01	ACTIVE
4	Nguyen Ho Dang Duy	nhduy22@clc.fitus.edu.vn	0372174241	2004-06-07	ACTIVE

Hình 32: Giao diện Admin Manage Account

User Details

Email: pqduy22@gmail.com
Fullname: Pham Quang Duy
DOB: Sun, 01 Dec 2002 00:00:00 GMT
Phone: 0123456789
Address: 227 Nguyen Van Cu
Created At: Tue, 08 Jul 2025 21:33:30 GMT
Failed Attempts: 0
Last Failed Login: Never

Lock **Close**

#	Fullname	Email	Phone	DOB	Status
1	ABC	abcABC@clc.fitus.edu.vn	0123456798	2004-02-03	LOCKED
2	scriptalert1/script	abc@gmail.com	0999999999	2002-12-01	ACTIVE
3	Pham Quang Duy	pqduy22@gmail.com	0123456789	2002-12-01	ACTIVE
4	Nguyen Ho Dang Duy	nhduy22@clc.fitus.edu.vn	0372174241	2004-06-07	ACTIVE

Hình 33: Popup chi tiết thông tin account

Quy trình thực hiện

1. **Phân quyền khi đăng nhập:** Trong `/login`, nếu `role` là `admin`, chuyển hướng về `/admin_dashboard`:
2. **Xem log toàn hệ thống:**
 - Tải dữ liệu từ file `logs/security.log`
 - Parse ra các dòng: thời gian, email, action, trạng thái, message
 - Dùng JavaScript để render bảng và lọc kết quả
3. **Xem danh sách tài khoản:**
 - Gọi `fetch_all_users()` → lấy toàn bộ người dùng không phải `Administrator`
 - Trả về kết quả hiển thị qua `admin_manage_account.html`
4. **Khóa / Mở tài khoản:** Khi admin bấm nút Lock/Unlock:
 - Gửi form POST gồm `email` và `action`
 - Backend gọi `toggle_user_lock(email, lock=True/False)`
 - Cập nhật cột `is_locked` trong bảng `users`

Chi tiết kỹ thuật và thư viện bảo mật

1. Phân quyền truy cập

Thư viện:

- `session["role"]` – lưu vai trò người dùng
- `Flask redirect, url_for` – điều hướng tùy vai trò

Chi tiết:

```
1 if session.get("role") == "admin":  
2     return redirect("/admin_dashboard")  
3 else:  
4     return redirect("/dashboard")
```

2. Quản lý tài khoản người dùng

Thư viện:

- `modules.utils.manage_account.fetch_all_users()`
- `toggle_user_lock(email, lock:bool)` – cập nhật SQL

Truy vấn SQL:

```
1 SELECT email, fullname, dob, phone, ... FROM users WHERE fullname != '  
    Administrator';  
2 UPDATE users SET is_locked = 1 WHERE email = 'alice@example.com';
```

3. Ghi log khi thao tác

Thư viện:

- `log_user_action(email, action, status, message, level)`

Ví dụ log:

```
1 [2025-07-01 11:05:12] | admin@example.com | Lock Account | Success |  
alice@example.com
```

4. Xem toàn bộ Security Log

Thư viện:

- `read_security_logs()` – đọc file `logs/security.log`
- `JavaScript` – lọc kết quả bằng từ khoá

Định dạng log:

```
1 [TIMESTAMP] | email | ACTION | STATUS | MESSAGE
```

5.11 Ghi log bảo mật

Mục tiêu

Chức năng ghi log bảo mật nhằm đảm bảo toàn bộ hành động của người dùng và các sự kiện nội bộ trong hệ thống đều được theo dõi, ghi nhận và lưu trữ dưới dạng thống nhất. Việc này phục vụ:

- Truy vết các hành động quan trọng: đăng nhập, ký số, xác minh,...
- Giám sát lỗi bảo mật và cảnh báo bất thường.
- Làm bằng chứng trong điều tra sự cố hoặc kiểm toán.

Giao diện

Giao diện tại trang /admin_dashboard cung cấp:

- Bảng hiển thị đầy đủ log: thời gian, mức độ (INFO, WARNING, ERROR), email người dùng, hành động, trạng thái và chi tiết.
- Tính năng tìm kiếm realtime và nút Reload để làm mới dữ liệu.
- Dữ liệu được render động thông qua fetch X-Requested-With = XMLHttpRequest.

#	Level	Time	Email	Action	Status	Message
1	INFO	2025-07-09 10:54:14	yuddyd oo1405@gmail.com	Logout	Success	Logout successfully
2	INFO	2025-07-09 14:13:28	yuddyd oo1405@gmail.com	Login	Success	Login successfully
3	INFO	2025-07-09 14:13:28	yuddyd oo1405@gmail.com	Send OTP	Success	OTP sent via email

Hình 34: Bảng log của admin

Quy trình thực hiện

1. **Người dùng thực hiện hành động** Mỗi khi người dùng thao tác (ví dụ: đăng nhập, ký file, xác minh), hàm `log_user_action()` được gọi để ghi log với thông tin đầy đủ: email, hành động, trạng thái, chi tiết và mức độ severity.

```
[2025-07-09 10:54:14] [INFO] [yuddydoo1405@gmail.com] Action=Logout | Status=Success | Details=Logout successfully
[2025-07-09 14:13:28] [INFO] [yuddydoo1405@gmail.com] Action=Login | Status=Success | Details=Login successfully
[2025-07-09 14:13:32] [INFO] [yuddydoo1405@gmail.com] Action=Send OTP | Status=Success | Details=OTP sent via email
[2025-07-09 14:13:50] [INFO] [admin@fitus.edu.vn] Action=Login | Status=Success | Details=Login successfully
[2025-07-09 14:14:02] [INFO] [admin@fitus.edu.vn] Action=Send OTP | Status=Success | Details=OTP sent via email
[2025-07-09 14:14:18] [INFO] [admin@fitus.edu.vn] Action=TOTP Verify | Status=Success
[2025-07-09 14:13:50] [INFO] [admin@fitus.edu.vn] Action=Login | Status=Success | Details=Login successfully
[2025-07-09 14:14:02] [INFO] [admin@fitus.edu.vn] Action=Send OTP | Status=Success | Details=OTP sent via email
[2025-07-09 14:14:18] [INFO] [admin@fitus.edu.vn] Action=TOTP Verify | Status=Success\[2025-07-09 15:05:33] [INFO] [admin@fitus.edu.vn] Action=Login | Status=Success | Details=Login successfully
[2025-07-09 17:31:42] [INFO] [yuddydoo1405@gmail.com] Action=Login | Status=Success | Details=Login successfully
[2025-07-09 17:31:46] [INFO] [yuddydoo1405@gmail.com] Action=Send OTP | Status=Success | Details=OTP sent via email
[2025-07-09 17:31:58] [WARNING] [yuddydoo1405@gmail.com] Action=OTP Verify | Status=Fail | Details=Invalid or expired OTP
[2025-07-09 17:32:07] [INFO] [yuddydoo1405@gmail.com] Action=OTP Verify | Status=Success
[2025-07-09 17:32:10] [INFO] [yuddydoo1405@gmail.com] Action=Get User Info | Status=Success | Details=User profile returned
[2025-07-09 17:33:31] [INFO] [yuddydoo1405@gmail.com] Action=Get User Info | Status=Success | Details=User profile returned
[2025-07-09 17:34:35] [INFO] [yuddydoo1405@gmail.com] Action=Get User Info | Status=Success | Details=User profile returned
[2025-07-09 17:37:58] [INFO] [yuddydoo1405@gmail.com] Action=Get User Info | Status=Success | Details=User profile returned
[2025-07-09 17:38:08] [INFO] [yuddydoo1405@gmail.com] Action=Get User Info | Status=Success | Details=User profile returned
[2025-07-09 17:39:04] [INFO] [yuddydoo1405@gmail.com] Action=Get User Info | Status=Success | Details=User profile returned
[2025-07-09 17:39:05] [WARNING] [yuddydoo1405@gmail.com] Action=Update Account Info | Status=Fail | Details>No changes were made.
[2025-07-09 17:39:07] [WARNING] [yuddydoo1405@gmail.com] Action=Update Account Info | Status=Fail | Details>No changes were made.
[2025-07-09 17:39:08] [WARNING] [yuddydoo1405@gmail.com] Action=Update Account Info | Status=Fail | Details>No changes were made.
[2025-07-09 17:39:08] [WARNING] [yuddydoo1405@gmail.com] Action=Update Account Info | Status=Fail | Details>No changes were made.
[2025-07-10 07:53:24] [INFO] [1@gmail.com] Action=Register | Status=Success | Details=Registration successful!
[2025-07-10 07:53:57] [INFO] [1@gmail.com] Action=Login | Status=Success | Details=Login successfully
[2025-07-10 07:54:00] [INFO] [1@gmail.com] Action=Send OTP | Status=Success | Details=OTP sent via email
```

Hình 35: Log cho người dùng - Routes

2. Hệ thống nội bộ phát sinh sự kiện Các module nội bộ (như mã hóa, xác minh chữ ký) có thể gọi log_internal_event() để ghi lại log kỹ thuật chi tiết.

```
[2025-07-10 07:54:37] [INFO] [digital_signature]: Signed Duy_Resume_official.pdf successfully and saved to data/signature\Duy_Resume_official.pdf.sig.
[2025-07-10 07:56:39] [INFO] [digital_signature]: Verified signature for Duy_Resume_official.pdf successfully.
[2025-07-10 07:59:23] [INFO] [digital_signature]: Verified signature for Duy_Resume_official.pdf successfully.
[2025-07-11 10:50:51] [INFO] [digital_signature]: Signed Duy_Resume_official.pdf successfully and saved to data/signature\Duy_Resume_official.pdf.sig.
[2025-07-11 10:50:59] [INFO] [digital_signature]: Signed Duy_Resume_official.pdf successfully and saved to data/signature\Duy_Resume_official.pdf.sig.
[2025-07-11 10:51:15] [INFO] [digital_signature]: Signed Duy_Resume_official.pdf successfully and saved to data/signature\Duy_Resume_official.pdf.sig.
[2025-07-11 10:51:36] [INFO] [digital_signature]: Signed Duy_Resume_official.pdf successfully and saved to data/signature\Duy_Resume_official.pdf.sig.
[2025-07-11 11:02:56] [INFO] [digital_signature]: Verified signature for Duy_Resume_official.pdf successfully.
[2025-07-11 11:03:02] [INFO] [digital_signature]: Verified signature for Duy_Resume_official.pdf successfully.
[2025-07-11 11:03:40] [WARNING] [digital_signature]: Failed to verify signature for AIoT-Group5-Slide.pptx.pdf.
[2025-07-11 11:03:44] [WARNING] [digital_signature]: Failed to verify signature for AIoT-Group5-Slide.pptx.pdf.
[2025-07-11 11:03:46] [WARNING] [digital_signature]: Failed to verify signature for AIoT-Group5-Slide.pptx.pdf.
[2025-07-11 11:04:23] [WARNING] [digital_signature]: Failed to verify signature for Secure_Vault.pdf.
[2025-07-11 11:06:49] [WARNING] [digital_signature]: Failed to verify signature for Secure_Vault.pdf.
[2025-07-11 11:07:06] [WARNING] [digital_signature]: Failed to verify signature for Secure_Vault.pdf.
[2025-07-11 11:07:36] [WARNING] [digital_signature]: Failed to verify signature for Secure_Vault.pdf.
```

Hình 36: Log cho debug - Modules

3. Lưu vào file log Log người dùng và log nội bộ để debug sẽ được lưu vào các file riêng

- Log người dùng được ghi vào log/security.log
- Log module nội bộ được ghi vào log/debug_log.log

4. Giao diện đọc log Khi người dùng truy cập trang /admin_dashboard, Flask route đọc và phân tích nội dung từ file log, chuyển thành danh sách JSON và hiển thị trên giao diện.

Chi tiết kỹ thuật và thư viện bảo mật

1. Hàm log người dùng log_user_action(email, action, status, details, level) ghi log theo format thống nhất:

- Ví dụ log: [2025-07-09 10:15:23] [INFO] [user@example.com] Action=Sign File | Status=Success | Details=file=report.pdf

- Sử dụng thư viện chuẩn `logging`, ghi vào `log/security.log`

2. Log nội bộ hệ thống `log_internal_event(module, message, level)` dùng cho debug các module như crypto, xác minh,...

- Ví dụ: `[crypto]: Signature verified successfully.`
- Ghi vào `log/debug_log.log` với format chi tiết hơn để phục vụ debug.

3. Route hiển thị log Route `/log_security` thực hiện:

- Đọc file `log/security.log`, tách thành các trường: `timestamp`, `level`, `user`, `action`, `status`, `details`
- Trả về JSON nếu là AJAX, hoặc render giao diện nếu là truy cập thường

4. Mức độ log hỗ trợ Gồm 4 level chính:

- **INFO** – hành động thành công hoặc hợp lệ
- **WARNING** – thao tác sai, lỗi thường gặp
- **ERROR** – lỗi hệ thống hoặc dữ liệu bất thường
- **DEBUG** – dành cho log kỹ thuật nội bộ (chỉ module log mới dùng)

5. Định dạng và chuẩn hóa log Mọi log đều tuân thủ format:

`[timestamp] [LEVEL] [email] Action=... | Status=... | Details=...`

Điều này giúp dễ dàng phân tích bằng tool, lọc log, và hỗ trợ audit.

5.12 Chia nhỏ tập tin lớn

Mục tiêu

Đảm bảo hệ thống có thể xử lý các tập tin lớn (trên 5MB) bằng cách **chia nhỏ thành các khối** trước khi mã hoá. Mỗi khối sẽ được mã hoá riêng biệt bằng AES-GCM để tăng tính an toàn và dễ xử lý bộ nhớ.

Quy trình thực hiện

1. Khi người dùng tải lên file và nhấn “Encrypt File”
2. Hệ thống kiểm tra dung lượng file
3. Nếu cần chia nhỏ file:
 - Đọc file theo từng khối `chunk_size = 1MB`
 - Với mỗi khối:
 - Sinh `nonce = os.urandom(12)`
 - Mã hoá bằng AES-GCM với cùng session key
 - Ghi độ dài khối (4 byte) + nonce + ciphertext vào file đích
4. Toàn bộ metadata và thông tin session key được lưu kèm theo

Chi tiết kỹ thuật và thư viện bảo mật

Điều kiện: Áp dụng khi tập tin có kích thước > 5MB

Thuật toán mã hóa: AES-256-GCM, mỗi khối dùng `nonce` riêng biệt
Thư viện:

- `cryptography.hazmat.primitives.ciphers.aead.AESGCM`
- `os.urandom` – sinh nonce ngẫu nhiên 12 byte cho mỗi khối
- `struct.pack` – encode độ dài khối (4 byte đầu)
- `stream.read()` – đọc file theo từng khối 1MB

Chi tiết xử lý từng khối:

```

1 chunk = file_stream.read(CHUNK_SIZE)
2 nonce = os.urandom(12)
3 ciphertext = AESGCM(aes_key).encrypt(nonce, chunk, None)
4 out_stream.write(struct.pack(">I", len(ciphertext)))
5 out_stream.write(nonce + ciphertext)

```

Cấu trúc file xuất ra:

```
1 [block_len][nonce][ciphertext] // L p l i c h o t n g k h i
```

Ưu điểm bảo mật:

- Mỗi khối có nonce riêng → đảm bảo không lặp nonce trong AES-GCM

- Không cần load toàn bộ file vào RAM → tránh rò rỉ
- Có thể kiểm tra từng phần độc lập khi giải mã

5.13 Kiểm tra trạng thái khóa

Mục tiêu

Hệ thống tự động kiểm tra tình trạng khóa RSA của người dùng mỗi lần truy cập dashboard hoặc khi thực hiện thao tác yêu cầu mã hóa/ký số. Nếu khóa đã hết hạn (sau 90 ngày), hệ thống sẽ tự động:

- Hủy kích hoạt khóa cũ
- Tạo khóa mới và lưu trữ theo đúng định dạng

Đảm bảo người dùng luôn có 1 khóa **active**, còn hạn sử dụng để thực hiện các chức năng mã hóa/ký.

Giao diện

Trong tab **RSA Keys**, hệ thống hiển thị:

- Danh sách các khóa đã tạo
- Trạng thái: Active, Expired Soon và Deactivated
- Ngày hết hạn

The screenshot shows a web-based application titled "Key Management". On the left, there is a vertical sidebar with icons for home, key management, locks, and other system functions. The main area displays a table of RSA keys. The columns are labeled: #, Encrypted Private Key, Public Key, Expiration, and Status. There are three rows of data:

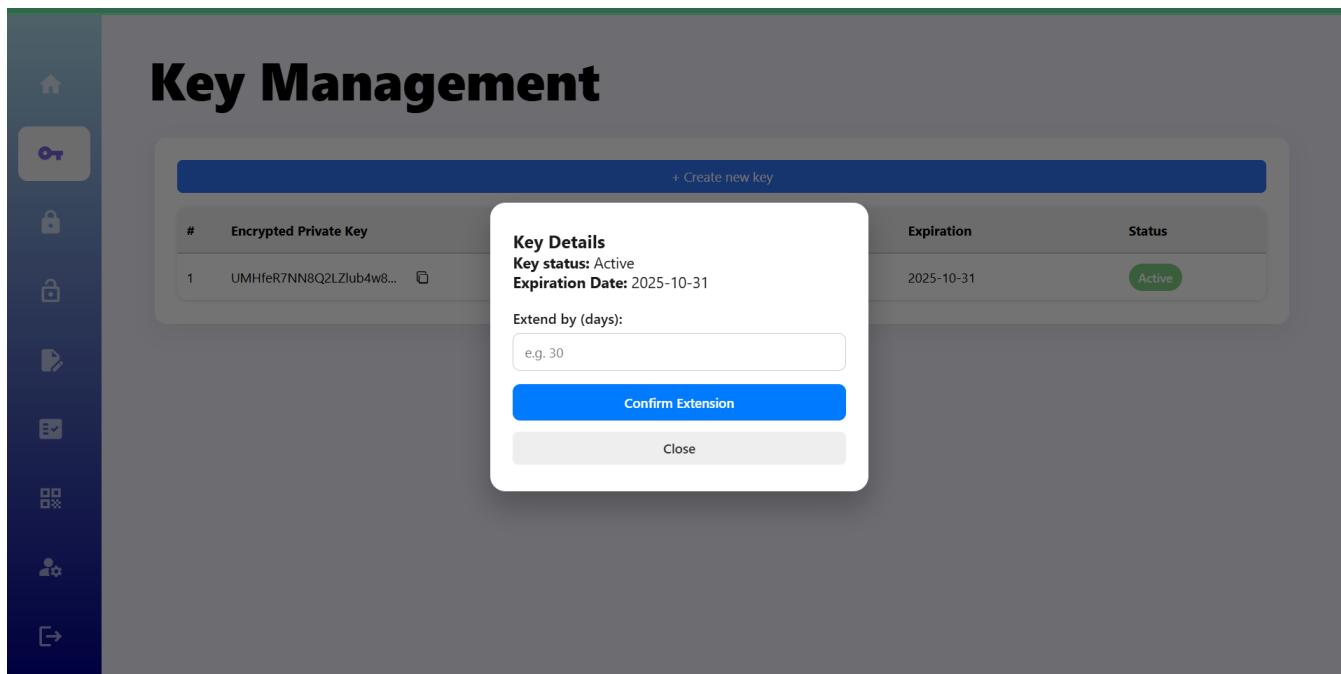
#	Encrypted Private Key	Public Key	Expiration	Status
1	5t0ofyzmxpls2eezFda...	MIIIBIjANBgkqhkiG9w0B...	2025-07-14	Expired Soon
2	4e3qoOY7u+vmxqEf0Vd5...	MIIIBIjANBgkqhkiG9w0B...	2025-07-12	Deactivated
3	uZxBrWHfVntPQjOypRv...	MIIIBIjANBgkqhkiG9w0B...	2025-07-11	Deactivated

Hình 37: Trạng thái khóa

Người dùng không cần thao tác gì thêm – hệ thống kiểm tra và xử lý tự động phía server

Quy trình thực hiện

1. **Tại Flask Backend:** Gọi hàm `check_and_manage_own_keys(email, aes_key)` mỗi khi truy cập `/manage_keys` hoặc mã hóa/ký số tập tin
2. Hàm trên sẽ tìm đường dẫn file key mới nhất:
 - Nếu người dùng chưa có khóa thì sẽ tạo mới bằng `create_new_key(email, aes_key)`
 - Nếu khóa đã hết hạn → Tạo khóa mới và vô hiệu hóa khóa cũ
3. Mỗi khóa có thời hạn 90 ngày.
4. Hệ thống đảm bảo:
 - Chỉ 1 khóa ở trạng thái **active**
 - Khi còn một ngày nữa là hết hạn, giao diện sẽ hiển thị **Expiring Soon**
 - Các khóa cũ đã hết hạn sẽ chuyển thành **deactivated**
5. Chương trình cho phép gia hạn khóa khi click vào hàng khóa còn đang kích hoạt ('Active' / 'Expiring soon'), được xử lý bởi hàm `extend_key_expiry(email, days_to_add)`



Hình 38: Gia hạn khóa

Chi tiết kỹ thuật và thư viện bảo mật

1. Cấu trúc lưu khóa

Mỗi cặp khóa RSA lưu trong 1 file JSON riêng tại:

`data/key_manage/<sha256(email)>/key_<n>.json`

Nội dung gồm:

- `public_info` (email, public key PEM, creation_date, expiry_date)
- `private_info` (private key đã mã hóa AES)
- `status`: active hoặc deactivated

2. Kiểm tra hạn khóa

Thư viện: `datetime`

Cách so sánh:

```
1 expiry = datetime.fromisoformat(expiry_date_str)
2 if datetime.now() > expiry:
3     create_new_key(...)
```

3. Vô hiệu hóa cũ

Mỗi khi tạo khóa mới, nếu đã có khóa cũ → cập nhật:

```
1 key_data['status'] = 'deactivated'
2 write_json_file(old_key_path, key_data)
```

4. Sinh khóa mới tự động

Thư viện:

- `cryptography.hazmat.primitives.asymmetric.rsa` – Tạo khóa
- `AESGCM` – Mã hóa private key
- `PBKDF2HMAC` – Sinh AES key từ passphrase

5.14 TÌM KIẾM PUBLIC KEY

Mục tiêu

Chức năng này cho phép người dùng tìm kiếm trong danh sách các public key đã lưu (của các liên hệ khác). Việc quản lý và tìm nhanh public key giúp người dùng dễ dàng sử dụng trong quá trình mã hóa, xác minh chữ ký, và chia sẻ an toàn.

Giao diện

Giao diện tại tab `Owned Public Keys` trong dashboard bao gồm:

- Bảng hiển thị danh sách các public key đã lưu: tên người gửi, email, timestamp và public key.
- Ô tìm kiếm theo email hoặc public key (tìm mờ, không phân biệt hoa thường).
- Dữ liệu được lấy động từ API `/owned_keys`.

The screenshot shows a dashboard titled 'QR Code' with a sidebar containing various icons. The main area is titled 'Owned Public Keys' and displays a table of saved public keys. The table has columns for #, Email, Creation date, Expiry date, Public Key, and Status. One entry is shown: #1, Email 1@gmail.com, Creation date 2025-07-10, Expiry date 2025-10-08, Public Key MII... (partial), and Status Valid. There is also a search bar labeled 'Search by email or public key...'.

#	Email	Creation date	Expiry date	Public Key	Status
1	1@gmail.com	2025-07-10	2025-10-08	MII... (partial)	Valid

Hình 39: Danh sách contact - public key đã lưu

Quy trình thực hiện

- Truy cập tab Owned Keys** Người dùng đăng nhập và truy cập trang dashboard → phần "Owned Public Keys" sẽ gửi request GET đến API `/owned_keys`.
- API lấy dữ liệu** Flask route `/owned_keys` thực hiện:
 - Kiểm tra phiên đăng nhập.
 - Đọc file `contact_public_key.json` theo thư mục cá nhân người dùng.
 - Chuyển danh sách public key về dạng JSON và trả về frontend.

- 3. Hiển thị và tìm kiếm**
- JavaScript tạo bảng dữ liệu từ kết quả JSON trả về.
 - Khi người dùng gõ từ khóa vào ô tìm kiếm, hàm `filterPublicKeys()` lọc các hàng phù hợp theo email hoặc chuỗi public key.

Chi tiết kỹ thuật và thư viện bảo mật

- 1. Cấu trúc lưu public key**
- Mỗi người dùng có một file riêng: `<user_email>/contact_public_key.json`
 - Dạng JSON: `{"abc@example.com": {"name": ..., "public_key_pem": ...}}`

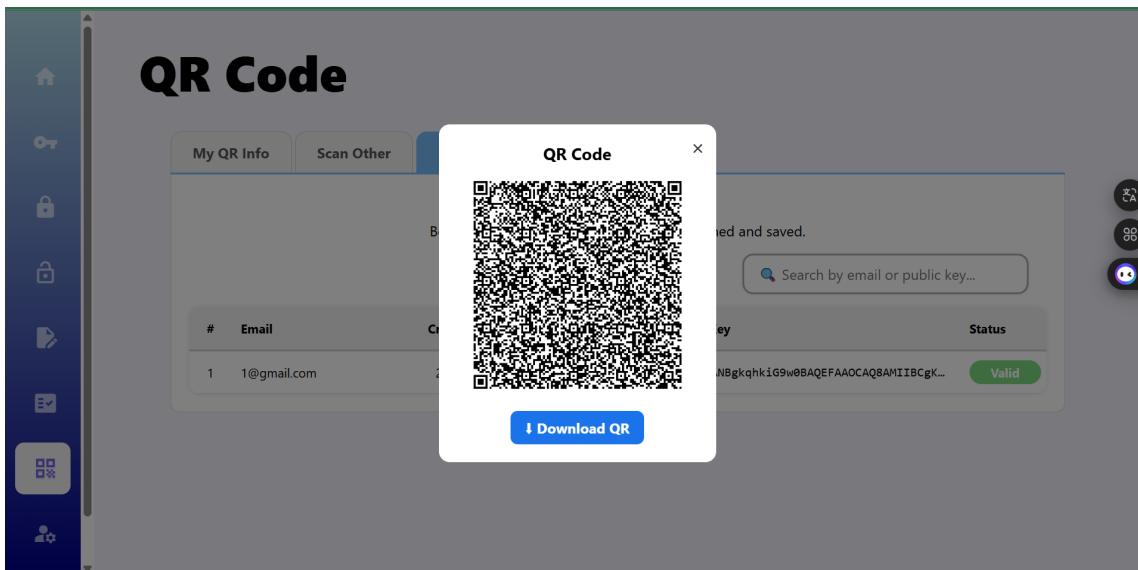
- 2. API trả dữ liệu public key** `/owned_keys` là route GET:

- Nếu chưa đăng nhập → trả lỗi `401 Unauthorized`.
- Nếu file danh bạ không tồn tại → trả danh sách rỗng.
- Ghi log trạng thái truy vấn bằng `log_user_action(...)` với số lượng public key tìm được.

- 3. Tìm kiếm phía client**
- Hàm `filterPublicKeys()` lọc dữ liệu theo từ khóa không phân biệt hoa thường.
 - Kiểm tra xem từ khóa xuất hiện trong email hoặc chuỗi public key PEM.
 - Lọc trực tiếp trên DOM → không gọi lại API khi tìm kiếm.

- 4. Hiển thị QR code public key** Hệ thống hỗ trợ hiển thị mã QR của public key khi người dùng click vào một contact cụ thể trong bảng danh sách. Điều này giúp việc chia sẻ public key giữa người dùng dễ dàng và nhanh chóng hơn (qua quét mã bằng điện thoại, Google Authenticator,...).

- Giao diện sử dụng sự kiện `onClick` hoặc nút `View QR` gắn với mỗi hàng trong bảng.
- Khi người dùng bấm chọn, mã public key PEM tương ứng sẽ được gửi đến route `/utils/generate_qr` (hoặc sinh trực tiếp frontend).
- QR code được sinh từ chuỗi PEM hoặc JSON chứa các thông tin liên hệ và khóa công khai.
- Kết quả được hiển thị trong một modal hoặc khung bên cạnh dòng đã chọn.
- Mã QR cho phép người khác dễ dàng quét để lưu lại public key → hỗ trợ xác thực và mã hóa an toàn.



Hình 40: Pop-up QR của contact đã lưu

5. **Bảo mật truy cập** Chỉ người dùng đã đăng nhập mới được phép truy vấn danh sách public key đã lưu.

- File JSON được lưu riêng trong thư mục người dùng → cô lập dữ liệu từng tài khoản.

5.15 Giới hạn đăng nhập

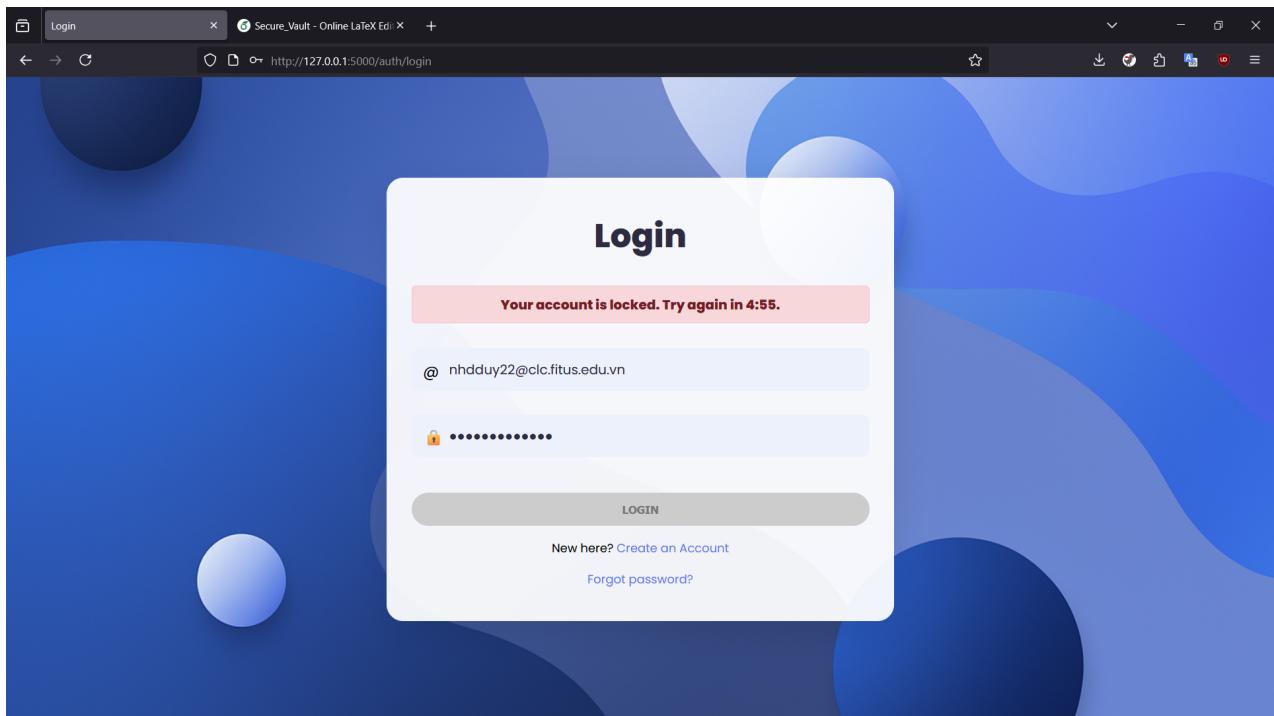
Mục tiêu

Đảm bảo an toàn tài khoản người dùng khỏi tấn công brute-force và hành vi truy cập trái phép bằng cách:

- **Khóa tạm thời 5 phút** nếu người dùng nhập sai mật khẩu **5 lần liên tiếp trong 2 phút**
- **Khóa vĩnh viễn bởi admin:** chỉ được mở lại thủ công từ dashboard quản trị

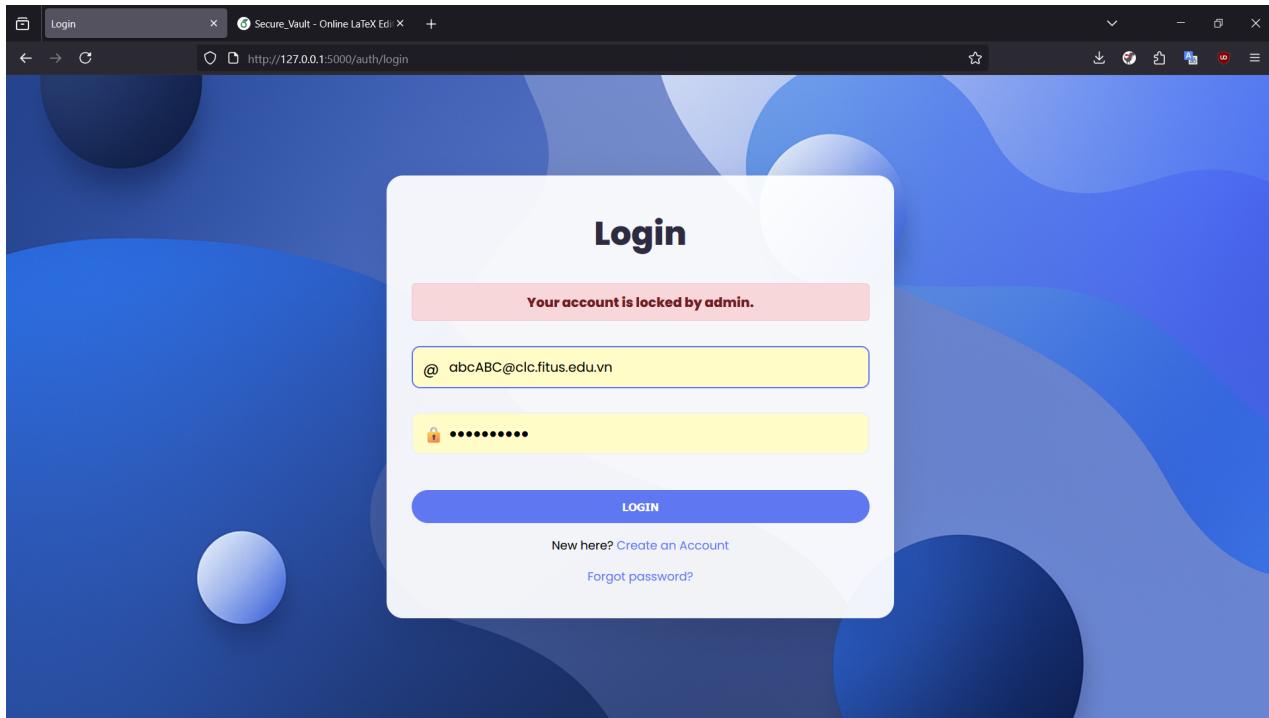
Giao diện

Khi người dùng nhập sai mật khẩu **5 lần liên tiếp trong 2 phút**, hệ thống sẽ tạm khóa khiến cho người dùng không thể đăng nhập tiếp được nữa. Chương trình cũng có đồng hồ đếm ngược và sẽ mở khóa khi hết thời gian



Hình 41: Giao diện bị lock trong 5 phút

Người dùng có hành vi bất thường có thể bị khóa vĩnh viễn bởi admin. Tài khoản chỉ có thể mở khóa tại giao diện admin



Hình 42: Giao diện bị lock vĩnh viễn bởi admin

Quy trình thực hiện

- Kiểm tra trạng thái khóa trước khi xử lý đăng nhập:** Khi người dùng bị khóa đăng nhập lại, chương trình sẽ kiểm tra:
 - Nếu bị khóa bởi admin thì sẽ từ chối đăng nhập và hiển thị thông báo tương ứng
 - Nếu lần đăng nhập cuối đã quá 5 phút, chương trình sẽ kiểm tra email và passphrase như bình thường
- Ghi nhận đăng nhập sai:** Nếu người dùng đăng nhập sai, chương trình sẽ ghi nhận `user['failed_attempts'] + 1`. Nếu `fail_attempts=5` thì sẽ cập nhật `is_locked=1` và lock người dùng 5 phút
- Reset khi nhập đúng mật khẩu:** Reset các biến đếm lỗi khi người dùng đã nhập đúng mật khẩu
- Reset timeout:** Nếu thời gian từ lần sai cuối lớn hơn 2 phút thì sẽ reset lại biến đếm số lần sai

Chi tiết kỹ thuật và thư viện bảo mật

1. Trường dữ liệu liên quan

Bảng `users` gồm:

- `failed_attempts` – số lần nhập sai liên tiếp
- `last_failed_login` – thời điểm nhập sai gần nhất

- `is_locked` – đánh dấu tài khoản bị khóa

2. Xử lý sai nhiều lần trong thời gian ngắn

Thư viện: `datetime, timedelta`

```
1 if user['last_failed_login']:  
2     time_diff = now - user['last_failed_login']  
3     if time_diff > timedelta(minutes=2):  
4         reset failed_attempts
```

3. Khóa tài khoản tạm thời 5 phút

Sau 5 lần sai:

```
1 if failed >= 5:  
2     is_locked = True  
3     UPDATE users SET is_locked = TRUE
```

Lần đăng nhập tiếp theo:

```
1 if now - last_failed_login < 5:  
2     reject login
```

4. Khóa bởi admin

•

5.16 Tùy chọn định dạng lưu file

Mục tiêu

Cho phép người dùng lựa chọn định dạng lưu file sau khi mã hoá:

- **Combined:** Tập tin duy nhất `.enc`, chứa cả metadata + dữ liệu
- **Split:** Hai tập tin: `.enc` chứa dữ liệu, `.key` chứa metadata + session key

Quy trình thực hiện

- Trong form `/encrypt_file`, người dùng chọn:

```
1 <input type="radio" name="save_format" value="combined" checked />
2 <input type="radio" name="save_format" value="split" />
```

- Backend kiểm tra:

```
1 if save_format == 'split':
2     save_metadata_as_key_file(...)
3 else:
4     combine_metadata_and_data(...)
```

- Tùy theo lựa chọn:

- Combined: File `.enc` gồm: [4 byte metadata_len] [metadata] [encrypted_data]
- Split:
 - File `.enc`: encrypted content
 - File `.key`: JSON chứa `encrypted_session_key`, `nonce`, `filename`, ...

Chi tiết kỹ thuật và thư viện bảo mật

Thư viện:

- `json` – serialize metadata chứa thông tin mã hóa, khóa
- `base64` – mã hóa khóa phiên và nonce
- `struct.pack(">I", len(metadata))` – định vị metadata trong file
- `os.path.join, with open(...)` – ghi file an toàn

Chi tiết Combined Mode:

```
1 metadata_bytes = json.dumps(metadata).encode()
2 file_enc.write(struct.pack(">I", len(metadata_bytes)))
3 file_enc.write(metadata_bytes)
4 file_enc.write(encrypted_content)
```

Chi tiết Split Mode:

```
1 with open(...enc, 'wb') as f_data:  
2     f_data.write(encrypted_content)  
3 with open(...key, 'w') as f_key:  
4     json.dump(metadata, f_key, indent=4)
```

Bảo mật:

- Session key luôn được mã hóa bằng RSA-OAEP trước khi lưu
- Nonce và AES key không lưu thô, luôn ở dạng base64
- Metadata chứa thông tin mã hóa (algorithm, timestamp, người gửi,...) để phục vụ giải mã và xác minh

5.17 Khôi phục tài khoản

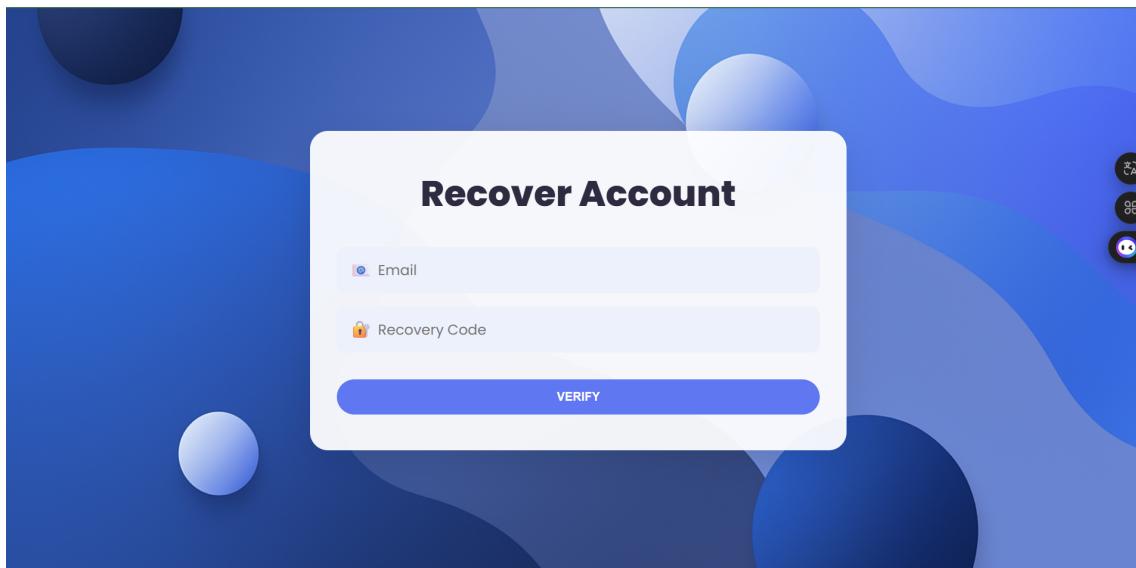
Mục tiêu

Chức năng khôi phục tài khoản cho phép người dùng đặt lại mật khẩu (passphrase) khi không còn nhớ mật khẩu cũ. Khác với đặt lại mật khẩu thông thường, hệ thống vẫn đảm bảo khôi phục được private key đã mã hóa trước đó – nhờ sử dụng một **recovery code** (AES key phụ) đã được lưu trữ an toàn từ khi đăng ký.

Giao diện

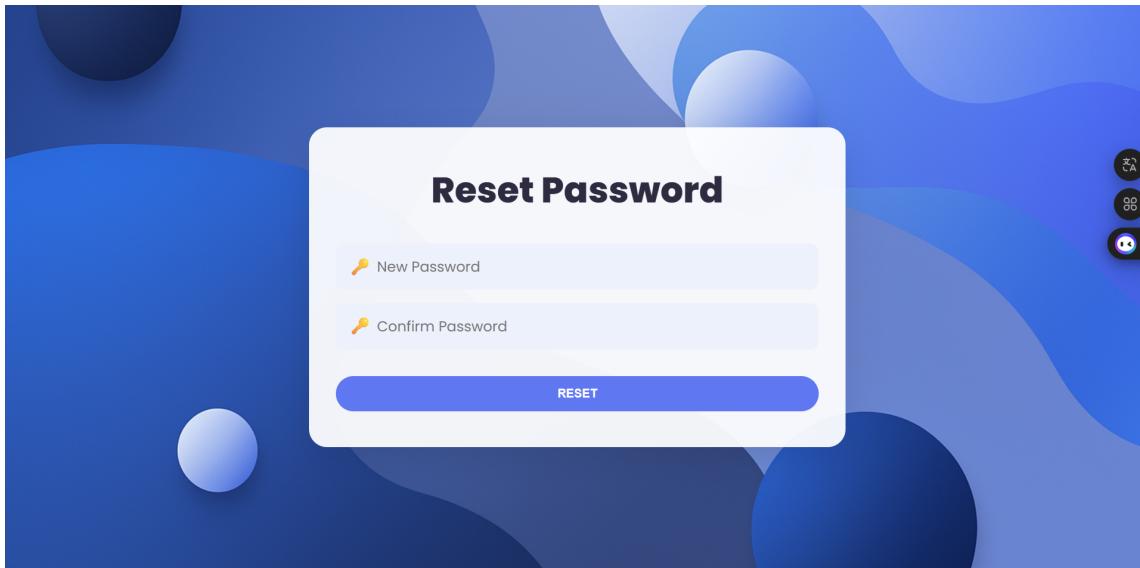
Giao diện khôi phục gồm 2 bước:

- **Bước 1:** Người dùng nhập email và recovery code để xác minh quyền sở hữu tài khoản.



Hình 43: Nhập mã khôi phục

- **Bước 2:** Nếu thành công, hệ thống cho phép nhập mật khẩu mới (passphrase), sau đó sẽ mã hóa lại khóa riêng và recovery key.



Hình 44: Nhập mật khẩu mới

Quy trình thực hiện

Bước 1 Xác minh recovery code

- Người dùng nhập recovery code → hệ thống derive AES key từ recovery code đó.
- Dùng AES key này để giải mã file `key_recovery` chứa private key hiện tại.
- Nếu giải mã thành công → recovery code hợp lệ.

Bước 2 Nhập mật khẩu mới

- Người dùng nhập passphrase mới.
- Hệ thống mã hóa lại private key hiện tại bằng AES key derive từ passphrase mới.
- Đồng thời, mã hóa lại recovery code bằng AES key derive từ passphrase mới → lưu vào cột `encrypted_recovery_code` trong database.

Bước 3 Cập nhật cơ sở dữ liệu

- Hash passphrase mới và cập nhật vào cột `hashed_passphrase`.
- Ghi log hành động khôi phục với trạng thái thành công hoặc thất bại.

Chi tiết kỹ thuật và thư viện bảo mật

1. Recovery code

Recovery code là một chuỗi ngẫu nhiên sinh ra khi đăng ký, dùng để derive AES key tạm thời cho mục đích khôi phục khóa.

- Được dùng để giải mã private key hiện tại khi muốn reset password.
- Được mã hóa bằng AES key derive từ passphrase và lưu dưới database.

2 . Kiểm tra tính hợp lệ của recovery code

- Dùng recovery code derive AES key: `derive_aes_key(recovery_code)`.
- Giải mã file `key_recovery` (chứa private key mã hóa) bằng AES key này.
- Nếu giải mã được → xác minh thành công.

3 . Đặt lại mật khẩu và mã hóa lại dữ liệu

- Derive AES key từ passphrase mới.
- Mã hóa lại private key bằng AES key mới.
- Mã hóa recovery code bằng AES key mới và lưu vào `encrypted_recovery_code`.
- Hash passphrase mới với salt → cập nhật vào bảng `users`.

4. Xử lý lỗi và báo lỗi chi tiết

Hệ thống đảm bảo mọi lỗi trong quá trình khôi phục tài khoản đều được phát hiện, phản hồi rõ ràng cho người dùng, đồng thời ghi log bảo mật để hỗ trợ điều tra.

- **Thiếu email hoặc recovery code:**

- Nếu không gửi đủ thông tin đầu vào → trả lỗi 400 với thông báo: "Missing recovery information".

- **Recovery code không hợp lệ:**

- Nếu recovery code không trùng khớp hoặc giải mã private key thất bại → trả lỗi 400, thông báo: "Invalid recovery code".
 - Lỗi này thường do nhập sai hoặc đã đổi passphrase trước đó mà chưa cập nhật lại recovery key.

- **Mật khẩu mới không hợp lệ:**

- Nếu passphrase mới không đủ mạnh (dưới 8 ký tự, không có ký tự đặc biệt, chữ hoa, số) → từ chối và trả thông báo cụ thể.

- **Lỗi mã hóa lại private key hoặc recovery key:**

- Nếu trong quá trình mã hóa lại private key bằng passphrase mới xảy ra lỗi → rollback thao tác DB và trả về lỗi "Re-encrypt RSA failed".
 - Nếu mã hóa lại recovery key lỗi → báo lỗi "Failed to encrypt recovery key" và không cập nhật thông tin.

- **Lỗi cập nhật CSDL:**

- Nếu xảy ra lỗi khi cập nhật passphrase hoặc encrypted recovery key trong database → trả lỗi "Database error" và log ở mức `error`.

- **Log đầy đủ:**

- Tất cả lỗi đều được ghi lại bằng `log_user_action(...)` với trạng thái "Fail" và nội dung chi tiết.
 - Các lỗi hệ thống (mã hóa thất bại, không đọc được recovery file, lỗi DB) được log với `level = "error"`.

6 Kiểm thử ứng dụng

Tất cả hình ảnh kết quả kiểm thử, video demo được cập nhật đầy đủ trong drive sau

6.1 Đăng ký tài khoản người dùng (/signup)

Mô tả kiểm thử	Input	Kết quả mong đợi
Đăng ký thành công	Email hợp lệ, Passphrase mạnh, các trường đầy đủ	Thông báo "Registration successful", pop-up <code>recovery_code</code>
Email sai định dạng	<code>abc@.com</code>	Báo lỗi "Invalid email format"
Passphrase yếu	<code>abc12345</code>	Báo lỗi "Passphrase too weak"
Trùng email đã đăng ký	Email đã có trong DB	Báo lỗi "Account already exists"
XSS/SQL injection	<code><script>, " OR "1"="1</code>	Bị sanitize, không lỗi hệ thống

6.2 Đăng nhập và xác thực đa yếu tố (/login → /verify)

Mô tả kiểm thử	Input	Kết quả mong đợi
Đăng nhập đúng pass → chờ xác thực OTP	Email và passphrase đúng	Chuyển hướng tới trang <code>/verify</code> để xác thực OTP hoặc TOTP
Sai pass < 5 lần	Email đúng, passphrase sai	Báo lỗi "Wrong email or password" và tăng <code>failed_attempts</code>
Sai pass 5 lần	Nhập sai liên tiếp 5 lần	Khóa tài khoản 5 phút, thông báo thời gian chờ
OTP hợp lệ trong thời gian	Mã OTP 6 chữ số từ email (trong vòng 5 phút)	Xác thực thành công, chuyển đến <code>/dashboard</code> hoặc <code>/admin_dashboard</code>
OTP sai hoặc hết hạn	Mã OTP sai hoặc quá hạn 5 phút	Báo lỗi "Invalid or expired OTP"
TOTP đúng từ Google Authenticator	Mã 6 chữ số từ ứng dụng	Xác thực thành công, chuyển trang chính
TOTP sai	Nhập sai mã TOTP	Báo lỗi "Invalid TOTP code"

6.3 Quản lý khóa RSA cá nhân (/manage_keys)

Mô tả kiểm thử	Input	Kết quả mong đợi
Tạo cặp khóa thành công	Bấm nút "Create new key" sau khi đăng nhập	Sinh file lưu private và public key mới, với ngày tạo và hạn 90 ngày
Kiểm tra trạng thái khóa	Tài khoản đã có khóa	Hiển thị trạng thái: Còn hạn 'Active' / Gần hết hạn 'Expiring Soon' / Hết hạn 'Deactivated'
Giải mã private key thành công	Passphrase đúng	Mở được private key để ký / giải mã
Giải mã private key thất bại	Passphrase sai	Báo lỗi "Unable to decrypt private key"

6.4 QR Code Public Key (/utils/qr)

Mô tả kiểm thử	Input	Kết quả mong đợi
Tạo QR thành công	Email có public key	Tạo mã QR base64, lưu file PNG
Quét QR thành công	File ảnh QR đúng định dạng	Hiển thị: "Successfully added {contact_email} to your contact list!" và thêm contact vào bảng 'Owned Keys'
Quét file ảnh sai định dạng	PNG không chứa mã QR hoặc bị lỗi json, thông tin	"Unable to read the image file. Please try again with a different format (PNG, JPG)." "No QR code found in the image." "QR code data is incomplete or invalid." "QR code content is not a valid JSON format."

6.5 Cập nhật thông tin tài khoản (/update_account)

Mô tả kiểm thử	Input	Kết quả mong đợi
Cập nhật thông tin thành công	Họ tên, địa chỉ, SĐT đúng định dạng	Lưu thay đổi thành công, reload dữ liệu
Không có thông tin mới được cập nhật	Giữ nguyên tất cả input	"No changes were made."
Đổi passphrase thành công	Pass cũ đúng, pass mới đủ mạnh	Passphrase thay đổi, AES key tự cập nhật, bản mã Recovery code được mã hóa lại theo passphrase mới - "Information has been successfully updated."
Các trường input sai định dạng	Số điện thoại sai định dạng	"Phone number must be exactly 10 digits."
Nhập thiếu trường passphrase cũ / mới	Nhập duy nhất 1 trường passphrase	Báo lỗi "Please enter both your current and new passphrase to change your password."
Pass cũ sai	Nhập sai passphrase hiện tại	Báo lỗi "Current passphrase is incorrect."
Pass cũ trùng pass mới	Nhập pass mới giống pass hiện tại	Báo lỗi "The new passphrase must be different from the current one."
Pass mới yếu	Mới <8 ký tự hoặc không đủ yêu cầu	Báo lỗi "New passphrase is too weak. It must contain at least 8 characters, an uppercase letter, a number, and a special symbol."

6.6 Mã hoá tập tin gửi người khác (/crypto/encrypt)

Mô tả kiểm thử	Input	Kết quả mong đợi
Mã hoá thành công (gộp file)	Chọn file + email người nhận có public key	Tạo file <code>.enc</code> chứa dữ liệu mã hoá và metadata
Mã hoá thành công (tách file)	Chọn lưu dạng tách	Sinh file <code>.enc</code> và <code>.key</code> riêng biệt
Người nhận không có public key	Nhập email chưa có key lưu trữ	Báo lỗi "No users found in your contact lists."
Metadata đúng định dạng	Sau khi mã hoá	Metadata gồm người gửi, thời gian, thuật toán, định dạng

6.7 Giải mã tập tin (/crypto/decrypt)

Mô tả kiểm thử	Input	Kết quả mong đợi
Giải mã thành công (file gộp)	File <code>.enc</code> gộp	Giải mã thành công, tự động download file mã hóa
Giải mã thành công (file tách)	<code>.zip</code> bao gồm: <code>.enc</code> + <code>.key</code>	Tự động download file gốc đúng nội dung
Sai khóa	Passphrase không đúng hoặc thiếu <code>.key</code>	Báo lỗi "Decryption failed" hoặc "Unable to decrypt key"
Tự động nhận dạng định dạng file	Dù là gộp hay tách	Tự động phân tích đúng định dạng và giải mã phù hợp

6.8 Ký số tập tin (/crypto/sign)

Mô tả kiểm thử	Input	Kết quả mong đợi
Ký số thành công	Chọn file bất kỳ + passphrase đúng	Tạo file chữ ký <code>.sig</code> theo định dạng SHA-256 + RSA
Thiếu private key hoặc passphrase sai	Không giải mã được private key	Báo lỗi "Error during digital signing"
Log ký số đúng	Sau ký thành công	Ghi log: email người ký, thời gian ký, file đã ký

6.9 Xác minh chữ ký (/crypto/verify)

Mô tả kiểm thử	Input	Kết quả mong đợi
Xác minh đúng chữ ký	File gốc + file <code>.sig</code> đúng	Hiển thị email người ký, ngày ký, thông báo hợp lệ
Chữ ký sai hoặc không khớp	File bị thay đổi hoặc <code>.sig</code> giả mạo	Báo lỗi "Invalid signature"
Không có public key người ký	Người ký chưa có key trong danh sách	Báo lỗi "Public key does not exist."
Sai format chữ ký	File khác định dạng ở mục (<code>.sig</code>)	Báo lỗi "Invalid signature file format"
Thiếu input	Không nhập file / Nhập thiếu file	Báo lỗi "Please select both the files and its signature."

6.10 Phân quyền tài khoản

Mô tả kiểm thử	Input	Kết quả mong đợi
Admin truy cập trang quản lý	Người dùng có role = admin	Hiển thị danh sách tài khoản, log, nút khóa/mở tài khoản
Người thường truy cập trang admin	role = user	Báo lỗi "Access denied" và chuyển hướng về login
Khoá / mở tài khoản người dùng	Bấm nút khóa/mở trong giao diện admin	Cập nhật trạng thái khóa, tài khoản, lưu log thao tác

6.11 Ghi log bảo mật (security.log hoặc bảng log_activity)

Mô tả kiểm thử	Input	Kết quả mong đợi
Ghi log đăng nhập thành công	Email + passphrase đúng	Log sự kiện "Login success" với trạng thái "Success"
Ghi log đăng nhập sai	Nhập sai passphrase	Ghi vào log với thông tin thất bại, timestamp, email
Ghi log ký số / cập nhật / mã hoá	Thao tác thành công các chức năng	Ghi đúng hành vi người dùng vào log theo chuẩn đã định

6.12 Chia nhỏ tập tin lớn khi mã hóa (>5MB)

Mô tả kiểm thử	Input	Kết quả mong đợi
File >5MB được chia nhỏ	Upload file >5MB	Hệ thống chia thành block 1MB, mã hóa từng block bằng AES-GCM
File nhỏ <5MB không chia	Upload file 2MB	Hệ thống mã hóa nguyên khối, không chia block
Kiểm tra toàn vẹn sau khi gộp lại	Giải mã file đã chia	Nội dung khôi phục đúng, không mất dữ liệu

6.13 Kiểm tra trạng thái khóa (/manage_keys)

Mô tả kiểm thử	Input	Kết quả mong đợi
Khóa còn hạn	Ngày tạo < 60 ngày	Hiển thị trạng thái "Active"
Khóa gần hết hạn	Ngày tạo 80–89 ngày	Hiển thị "Expiring Soon"
Khóa hết hạn	Ngày tạo > 90 ngày	Hiển thị "Deactivated"
Gia hạn khóa	Số ngày muốn gia hạn thêm khi chọn 1 khóa đang 'Active' hoặc 'Expiring Soon'	Hiển thị "Key successfully extended. New expiry date: 2025-10-31."

6.14 Tìm kiếm public key (/keys/search)

Mô tả kiểm thử	Input	Kết quả mong đợi
Tìm thấy thông tin	Nhập email / public key có trong danh sách contacts	Hiển thị: email, QR code, ngày tạo, hạn dùng
Không tìm thấy thông tin	Email / Public key không tồn tại trong DB	Không hiển thị thông tin nào

6.15 Giới hạn số lần đăng nhập (/login)

Mô tả kiểm thử	Input	Kết quả mong đợi
Sai 5 lần liên tiếp	Nhập sai passphrase 5 lần trong 2 phút	Khoá tài khoản trong 5 phút, hiển thị thông báo thời gian chờ
Sau 5 phút thử lại	Dợi hết thời gian khoá	Tài khoản được mở lại và đăng nhập thành công nếu đúng pass
Log mỗi lần sai	Nhập sai liên tiếp	Ghi log với timestamp, lý do và email liên quan

6.16 Tùy chọn định dạng lưu file (/crypto/encrypt)

Mô tả kiểm thử	Input	Kết quả mong đợi
Lưu dạng gộp file	Chọn tùy chọn "Combine into a single file" khi mã hoá	Sinh 1 file duy nhất chứa toàn bộ dữ liệu và metadata
Lưu dạng tách file	Chọn "Save encrypted file and key separately" khi mã hoá	Sinh 2 file riêng biệt: dữ liệu mã hoá và AES key vào 1 file zip
Tự động nhận dạng khi giải mã	Upload file (gộp hoặc tách) khi giải mã	Tự động phân tích định dạng và xử lý phù hợp

6.17 Khôi phục tài khoản (/recover_account, /verify_recovery)

Mô tả kiểm thử	Input	Kết quả mong đợi
Khôi phục thành công	Nhập đúng recovery code hiển thị sau đăng ký	Cho phép đổi passphrase mới
Mã khôi phục sai hoặc hết hiệu lực	Mã recovery không đúng hoặc đã dùng	Báo lỗi "Invalid recovery code"
Đổi pass thành công sau xác minh	Pass mới đủ mạnh	Lưu pass mới

Tài liệu

- [1] Mastering Signup Systems: A Developer's Guide to Building Secure and User-Friendly Login Experiences. Nblocks.dev, 2024. <https://www.nblocks.dev/blog/authentication/mastering-signup-systems-a-developers-guide-to-building-secure-and-user-friendly-login-experiences>
- [2] NeuralNine. (2022, December 3). Two-Factor Authentication (2FA) in Python. YouTube. <https://www.youtube.com/watch?v=o0XZZkI69E8>