

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

Trương Đức Hào – Đỗ Mạnh Hùng – Lê Hồng Hiền

BÁO CÁO ĐỒ ÁN CUỐI KỲ
TRIỂN KHAI CHỮ KÝ SỐ CRYSTALS-DILITHIUM
TRONG CẤP PHÁT VĂN BẰNG SỐ

CRYSTALS-DILITHIUM THE IMPLEMENTATION OF
DIGITAL SIGNATURE IN ISSUING DIGITAL DEGREES

CỦ NHÂN NGÀNH AN TOÀN THÔNG TIN

TP. HỒ CHÍ MINH, 2024

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

TRƯƠNG ĐỨC Hào – 22520407

ĐỖ MẠNH HÙNG – 22520500

LÊ HỒNG HIỀN – 22520416

BÁO CÁO ĐỒ ÁN CUỐI KỲ
TRIỂN KHAI CHỮ KÝ SỐ CRYSTALS-DILITHIUM
TRONG CẤP PHÁT VĂN BẰNG SỐ

**CRYSTALS-DILITHIUM THE IMPLEMENTATION OF
DIGITAL SIGNATURE IN ISSUING DIGITAL DEGREES**

CỦ NHÂN NGÀNH AN TOÀN THÔNG TIN

GIẢNG VIÊN HƯỚNG DẪN
NGUYỄN NGỌC TỰ

TP. HỒ CHÍ MINH, 2024

Mục Lục

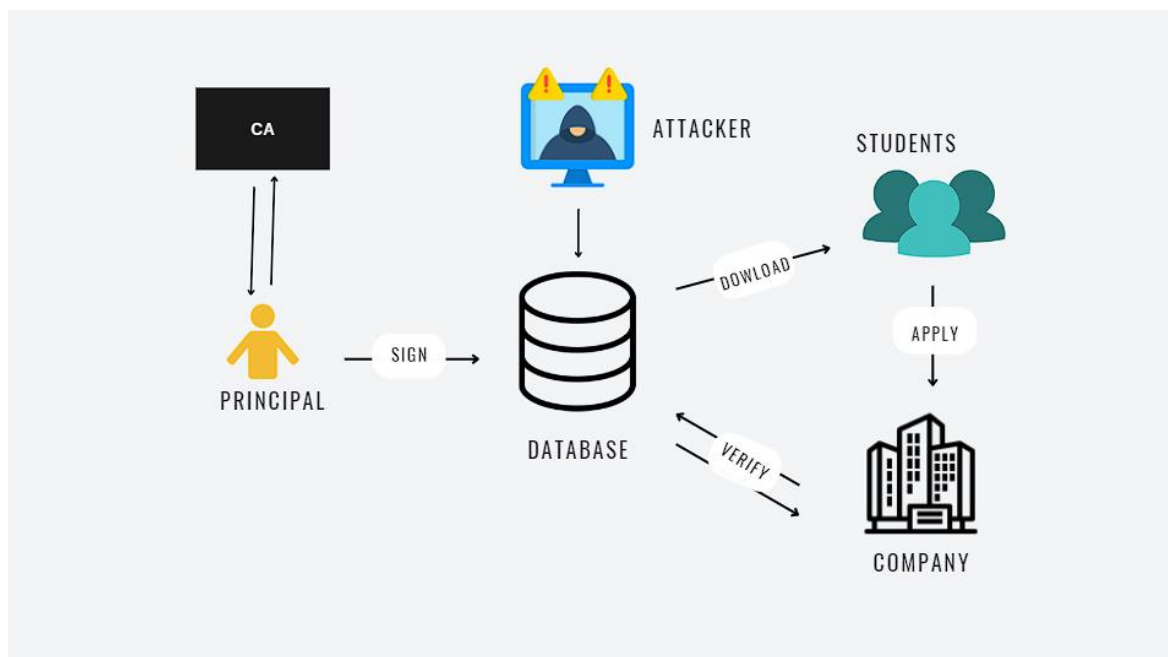
Chương 1. NGŨ CẢNH HỆ THỐNG VÀ CÁC THÀNH PHẦN	2
1.1. Ngũ cảnh hệ thống.....	2
1.1.1. Các thành phần trong hệ thống	2
1.1.2. Vai trò của các thành phần trong hệ thống	2
1.1.3. Tương tác giữa các thành phần trong hệ thống.....	3
Chương 2. TÀI SẢN, CÁC RỦI RO, YÊU CẦU BẢO MẬT	4
2.1. Tài sản.....	4
2.2. Rủi ro.....	4
2.3. Các yêu cầu bảo mật.....	4
Chương 3. CHI TIẾT GIẢI PHÁP VÀ DEMO	5
3.1. Kiến trúc hệ thống	5
3.2. Chi tiết giải pháp	5
3.2.1. Xác thực khi đăng nhập.....	5
3.2.2. Quản lý khóa riêng tư	5
3.2.3. Sử dụng thuật toán hậu lượng tử Dilithium:	5
3.3. CRYSTALS-DILITHIUM	6
3.4. Chi tiết Demo.....	6
3.4.1. Tạo cặp khóa sử dụng thuật toán Crystals-Dilithium, tạo CA key và CA-self signed certificate	7
3.4.2. Ứng dụng Web: Upload and Sign, Verify Signature	9

TÓM TẮT BÁO CÁO

Đề tài của nhóm tập trung vào việc ứng dụng chữ ký số hậu lượng tử trong việc cấp phát văn bằng điện tử nhằm đảm bảo an toàn và bảo mật trước các nguy cơ tiềm tàng từ máy tính lượng tử. Với sự phát triển của công nghệ lượng tử, các thuật toán mã hóa truyền thống như RSA hay ECC có thể bị phá vỡ, do đó việc nghiên cứu và ứng dụng các thuật toán mã hóa hậu lượng tử là cấp thiết. Đặt vấn đề làm sao xây dựng một hệ thống ký số sử dụng thuật toán Crystals-Dilithium cho cấp phát văn bằng điện tử ứng dụng trong các trường đại học, và các công ty khi tuyển dụng sinh viên có thể xác thực được văn bằng. Trong bản báo cáo, nhóm sẽ trình bày về ngữ cảnh hệ thống bao gồm các thành phần, vai trò của các thành phần, tương tác giữa các thành phần cho hệ thống thực tế, tài sản, rủi ro, các yêu cầu về bảo mật, chi tiết về giải pháp và tiến hành demo.

Chương 1. NGŨ CẢNH HỆ THỐNG VÀ CÁC THÀNH PHẦN

1.1. Ngũ cảnh hệ thống



Hình 2.1: Ngũ cảnh hệ thống

1.1.1. Các thành phần trong hệ thống

- Principal: Là hiệu trưởng của trường đại học hoặc người có thẩm quyền cấp văn bằng tốt nghiệp cho học sinh.
- Students: Là sinh viên được cấp văn bằng số, người sử dụng văn bằng số để xin việc,...
- Company: Tổ chức tuyển dụng sinh viên, cần sử dụng hệ thống để xác thực văn bằng.
- Database: Nơi lưu trữ dữ liệu là các văn bằng đã ký, public key của người ký (dưới dạng certificate).
- CA: Tổ chức có thẩm quyền, được tin tưởng để cấp digital certificate.
- Attacker: Cá nhân hoặc tổ chức xâm phạm Database với mục đích xấu.

1.1.2. Vai trò của các thành phần trong hệ thống

- Principal: Người ký văn bằng số, sử dụng secret key của mình để ký, tải văn bằng đã ký lên database của hệ thống.
- Students: Sinh viên tiến hành tải văn bằng đã ký được lưu ở database để sử dụng trong hồ sơ xin việc, tiến hành nộp văn bằng vào các công ty.

- Company: Truy cập hệ thống, dùng module verify để xác thực văn bản, xác thực certificate của người ký.
- Database: Lưu trữ văn bản sau khi ký, certificate chứa public key của người ký.
- CA: Cấp phát digital certificate để phân phối Public Key của Principal.

1.1.3. Tương tác giữa các thành phần trong hệ thống

- Principal sẽ tạo cặp key sử dụng thuật toán hậu lượng tử Crystals-Dilithium, sử dụng secret key để ký văn bản, gửi yêu cầu tạo certificate đến CA để có thể phân phối public key, sau đó tiến hành chuyển văn bản đã ký và certificate vào Database để lưu trữ.
- Students sẽ lên hệ thống và tải về văn bản của chính mình, sau đó nộp văn bản vào các công ty đang tuyển dụng.
- Company sẽ sử dụng module verify văn bản của hệ thống để xác thực văn bản với public key của người ký được lưu ở Database và chữ ký được gắn vào file văn bản.

Chương 2. TÀI SẢN, CÁC RỦI RO, YÊU CẦU BẢO MẬT

2.1. Tài sản

- Tài sản được lưu trữ ở Database: Bao gồm văn bản điện tử, chữ ký số, digital certificate.
- Khóa riêng tư của người ký: Khóa riêng tư được sử dụng để ký văn bản điện tử.

2.2. Rủi ro

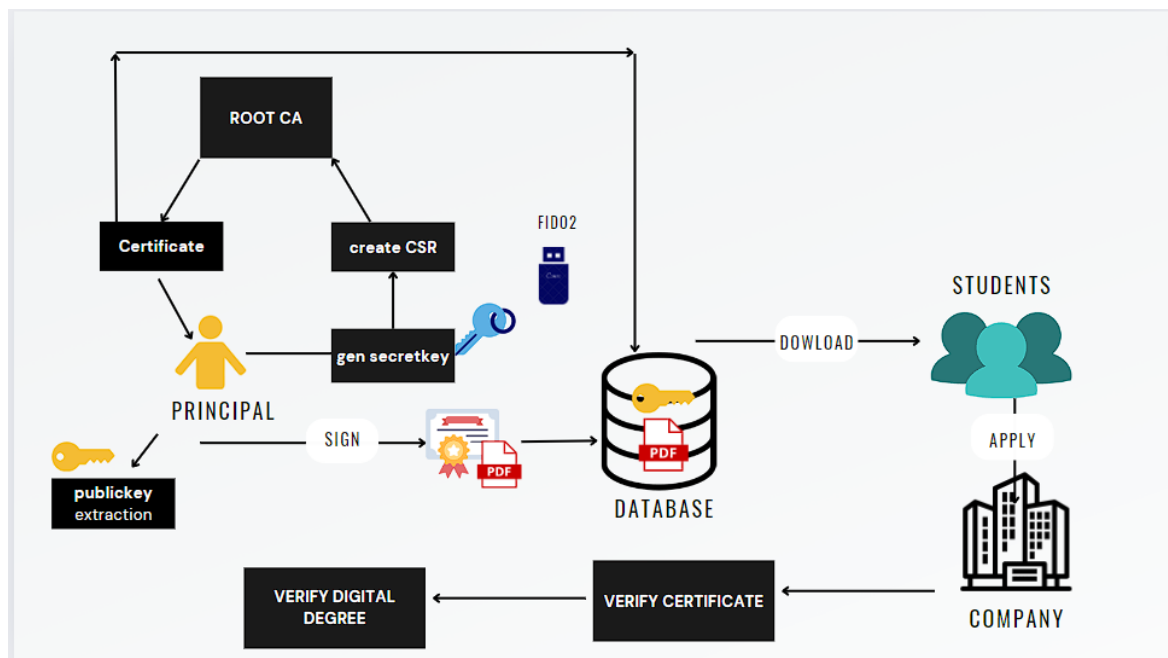
- Kẻ giả mạo, thay đổi nội dung của văn bản, nội dung chữ ký và khóa công khai được lưu trữ bên trong Database. Kẻ giả mạo có thể sửa đổi, sao chép, hoặc hủy hoại tài sản bên trong Database nếu không thể khôi phục sẽ mất mát tài nguyên gây ra tổn thất, ngoài ra hành vi sửa đổi có thể dẫn đến việc xác thực bị sai lệch gây ảnh hưởng lớn cho sinh viên.
- Khóa riêng tư của người ký có thể bị lộ hoặc đánh cắp gây nên việc có thể phát hành văn bản giả mạo.

2.3. Các yêu cầu bảo mật

- Tính toàn vẹn dữ liệu và xác thực: Phải đảm bảo dữ liệu bên trong Database không bị sửa đổi, hủy hoại, xác thực được văn bản.
- Khả năng chống lại máy tính lượng tử.
- Đảm bảo người ký không thể chối bỏ hành vi đã ký.
- Bảo vệ được khóa riêng tư của người ký.

Chương 3. CHI TIẾT GIẢI PHÁP VÀ DEMO

3.1. Kiến trúc hệ thống



Hình ảnh 3.1: Kiến trúc hệ thống

- Hiệu trưởng sẽ đăng nhập vào hệ thống, sử dụng khóa riêng tư để ký văn bằng điện tử theo định dạng PDF, sau đó chữ ký số và digital certificate của người ký sẽ được gắn vào Metadata của file PDF. Sinh viên khi truy cập hệ thống sẽ tìm kiếm văn bằng của mình và tải xuống, công ty khi nhận được văn bằng của học sinh sẽ tiến hành xác thực.

3.2. Chi tiết giải pháp

3.2.1. Xác thực khi đăng nhập

- Khi đăng nhập hiệu trưởng sẽ dùng tài khoản email và mật khẩu.

3.2.2. Quản lý khóa riêng tư

- Khóa riêng tư của người ký sẽ được bảo vệ bằng phương thức FIDO2, sử dụng USB token chuyên dụng lưu trữ.

3.2.3. Sử dụng thuật toán hậu lượng tử Dilithium:

- Sử dụng Dilithium trong việc tạo certificate và ký số để đảm bảo khả năng chống lại máy tính lượng tử.

3.3. CRYSTALS-DILITHIUM

- Dilithium là một trong những thuật toán chữ ký số hậu lượng tử (DSA), được NIST lựa chọn theo tiêu chuẩn. Dilithium dựa trên mô-đun lưới, được thiết kế để bảo vệ dữ liệu và xác thực danh tính của người ký, ngay cả trong môi trường có sự hiện diện của máy tính lượng tử.

Algorithm 1 Dilithium algorithm

```

1: procedure GENERATION
2:    $\mathbf{A} \leftarrow R_q^{k \times l}$ 
3:    $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$ 
4:    $\mathbf{t} := \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ 
5:   return  $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$ 

1: procedure SIGNATURE(sk, M)
2:    $\mathbf{z} := \perp$ 
3:   while  $\mathbf{z} = \perp$  do
4:      $\mathbf{y} \leftarrow S_{\gamma_1}^l$ 
5:      $\mathbf{w} := HighBits(\mathbf{A} \cdot \mathbf{y}, 2\gamma_2)$ 
6:      $c \in B_r := H(M \parallel \mathbf{w}_1)$ 
7:      $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ 
8:     if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|LowBits(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$  then
9:        $\mathbf{z} := \perp$ 
   return  $\sigma(\mathbf{z}, c)$ 

1: procedure VERIFY(pk, M,  $\sigma$ )
2:    $\mathbf{w}'_1 := HighBits(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$ 
3:   if then
4:     return  $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$  and  $c = H(M \parallel \mathbf{w}_1)$ 

```

Hình 3.3.1: Thuật toán Dilithium

3.4. Chi tiết Demo

- Platform: Windows
- Ngôn ngữ lập trình: Python
- Database: MongoDB
- Thư viện: Liboqspy, oqsprovider, openssl, pyPDF2

3.4.1. Tạo cặp khóa sử dụng thuật toán Crystals-Dilithium, tạo CA key và CA-self signed certificate

```
if __name__ == "__main__":
    try:
        sig_name = "Dilithium5"
        with oqs.oqs.Signature(sig_name) as signer:
            print("\nSignature details:")
            print(signer.details)

            signer_public_key = signer.generate_keypair()
            print("\nSigner public key:")
            print(signer_public_key.hex())

            signer_secret_key = signer.export_secret_key()
            print("\nSigner secret key:")
            print(signer_secret_key.hex())

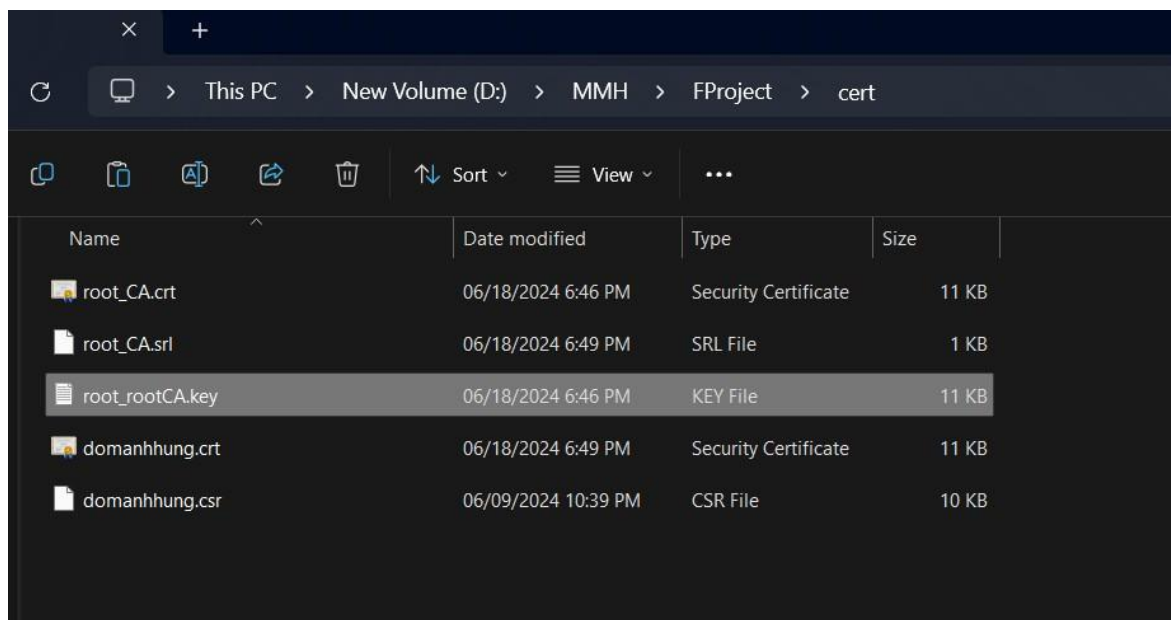
            # Tạo các tệp PEM
            directory = "D:\\MMH\\FinalProject\\keys"
            write_pem_file(directory, "domanhung_public_key.pem", "-----BEGIN PUBLIC KEY-----", "-----END PUBLIC KEY-----", signer_public_key)
            write_pem_file(directory, "domanhung_secret_key.pem", "-----BEGIN SECRET KEY-----", "-----END SECRET KEY-----", signer_secret_key)

            print("\nSuccessfully saved the key information to two files domanhung_public_key.pem and domanhung_secret_key.pem.")

    except Exception as e:
        print("Error:", e)
```

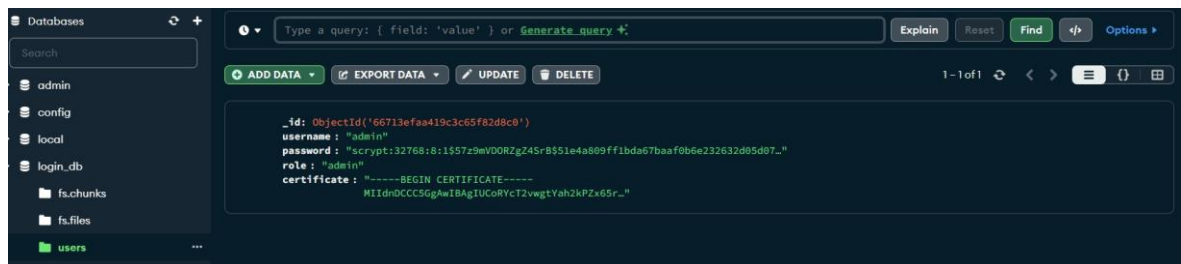
Hình 3.4.1.1: Tạo khóa

Sử dụng thư viện Liboqpython để tạo cặp khóa công khai và riêng tư cho người ký.



Hình 3.4.2.1: Tạo Certificate

Dùng thư viện Oqsprovider để tiến hành tạo certificate theo thuật toán Dilithium (2,3,5). Bao gồm rootCA certificate và certificate của người ký.



Hình 3.4.2.2: Certificate Database

Certificate chứa Public Key của Principal sẽ được lưu trữ ở Database.

```
def sig():
    Tk().withdraw()
    private_key_path = askopenfilename(title="Select the Private Key File")
    pdf_path = askopenfilename(title="Select the PDF File", filetypes=[("PDF files", "*.pdf")])

    private_key_base64 = read_pem_file(private_key_path)
    private_key = base64.b64decode(private_key_base64)
    pdf_content = read_pdf_text(pdf_path)

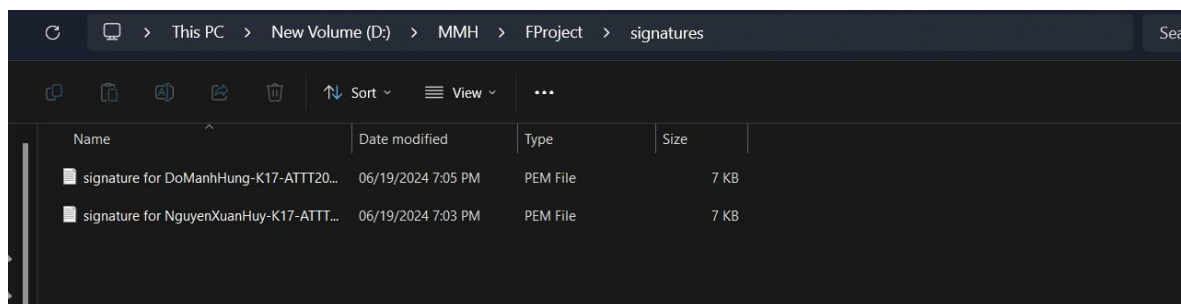
    sig = oqs.oqs.Signature("Dilithium5")
    sig.use_secret_key(private_key)
    signature = sig.sign(pdf_content)

    pdf_filename = os.path.basename(pdf_path)
    signature_filename = f"signature for {pdf_filename.replace('.pdf', '.pem')}"

    write_pem_file("D:\\MMH\\FProject\\signatures", signature_filename, "-----BEGIN SIGNATURE-----", "-----END SIGNATURE-----",
    print(f"Signature has been written to {signature_filename} successfully.")


if __name__ == "__main__":
    sig()
```

Hình 3.4.2.3: Signing



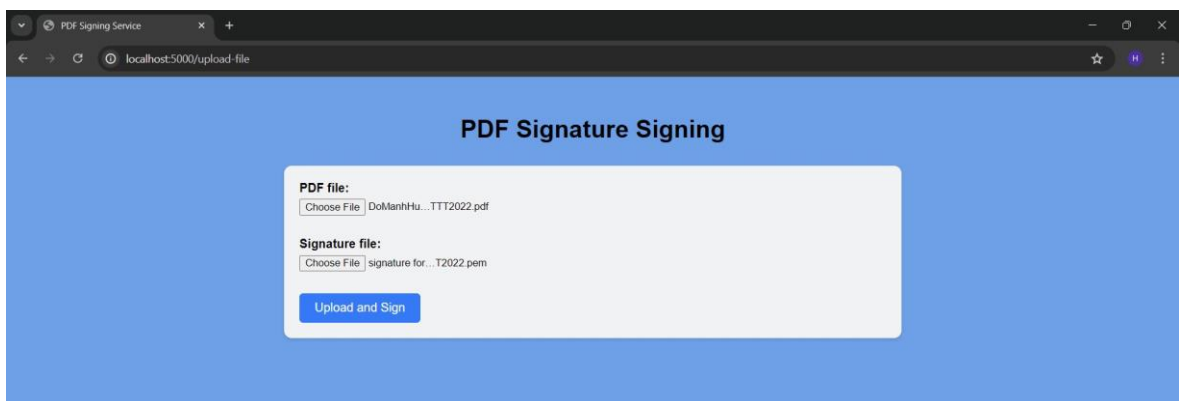
Hình 3.4.2.4: File chữ ký

3.4.2. Ứng dụng Web: Upload and Sign, Verify Signature



The image shows a login form titled "Login" on a light gray background. The form is a white card with a title "Login" in bold. Below the title, there is a text input field containing the username "admin". Below the username field is a password input field with masked characters ".....". At the bottom of the form is a green button labeled "Login".

Hình 3.4.2.1: Form đăng nhập cho Principal



The image shows a web browser window with the title "PDF Signing Service". The address bar shows "localhost:5000/upload-file". The main content area has a blue background and is titled "PDF Signature Signing". In the center, there is a white card with two sections: "PDF file:" and "Signature file:". Each section has a "Choose File" button and a file name. The PDF file is "DoManhHu...TTT2022.pdf" and the signature file is "signature for...T2022.pem". At the bottom of the card is a blue button labeled "Upload and Sign".

Hình 3.4.2.2: Upload and Sign

Sau khi đăng nhập, Principal tiến hành đăng văn bằng cùng file chữ ký số đã ký ở bước khởi tạo.

```

@app.route('/upload-file')
def uploadFile():
    return render_template('upload.html')

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'pdf' not in request.files or 'signature' not in request.files:
        return "No file part", 400

    pdf_file = request.files['pdf']
    signature_file = request.files['signature']
    certificate_file = request.files['certificate']

    if pdf_file.filename == '' or signature_file.filename == '' or certificate_file.filename == '':
        return "No selected file", 400

    pdf_path = os.path.join(app.config['UPLOAD_FOLDER'], pdf_file.filename)
    signature_path = os.path.join(app.config['UPLOAD_FOLDER'], signature_file.filename)
    certificate_path = os.path.join(app.config['UPLOAD_FOLDER'], certificate_file.filename)

    pdf_file.save(pdf_path)
    signature_file.save(signature_path)
    certificate_file.save(certificate_path)

    signed_pdf_path = os.path.join(app.config['UPLOAD_FOLDER'], 'signed_' + pdf_file.filename)

    add_signature_to_pdf(pdf_path, signature_path, certificate_path, signed_pdf_path)

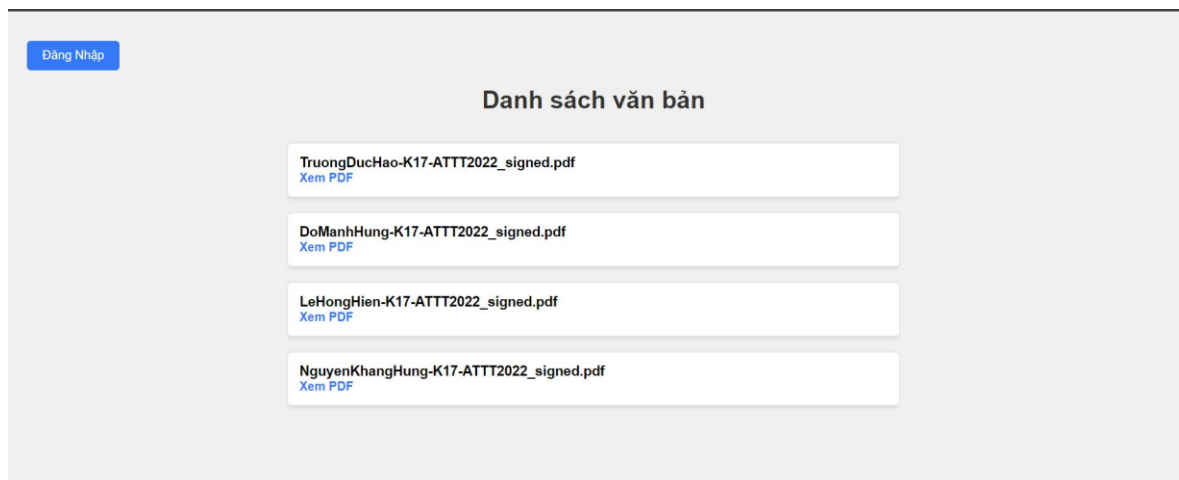
    unique_id = str(uuid.uuid4())

    with open(signed_pdf_path, 'rb') as signed_pdf:
        signed_pdf_id = fs.put(signed_pdf, filename=pdf_file.filename.split('.')[0] + '_signed.pdf', _id=unique_id)

    return redirect(url_for('list_files'))

```

Hình 3.4.2.3: Code Upload and Sign



Hình 3.4.2.4: Degrees List

Sau khi đăng tải và ký thì danh sách văn bằng đã ký sẽ được lưu trữ ở database và hiển thị trên web, từ đó sinh viên có thể tiến hành tìm kiếm văn bằng của mình để tiến hành tải xuống.

```
def add_signature_to_pdf(pdf_path, signature_path, certificate_path, output_path):
    signature = read_pem_file(signature_path)
    certificate = read_pem_file(certificate_path)

    pdf_reader = PdfReader(pdf_path)
    pdf_writer = PdfWriter()

    for page in pdf_reader.pages:
        pdf_writer.add_page(page)

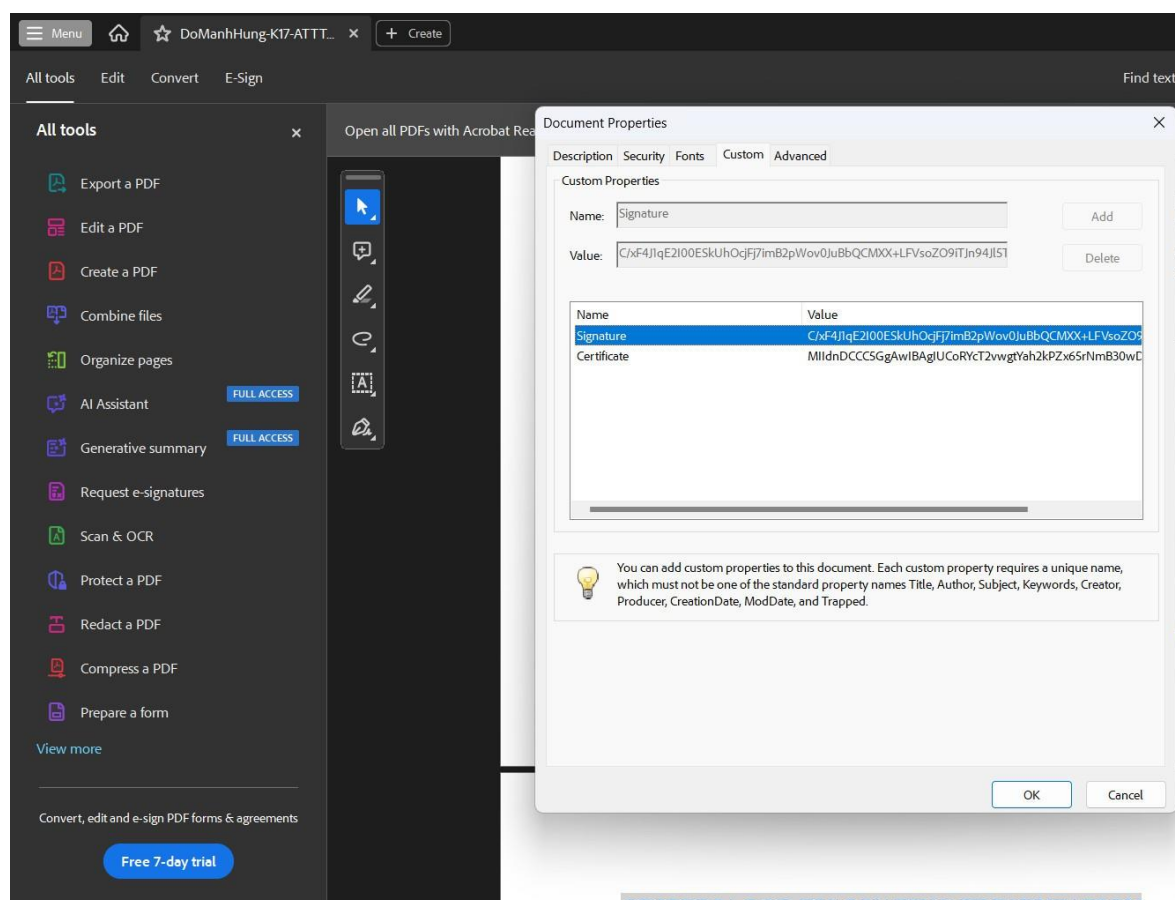
    metadata = pdf_reader.metadata
    updated_metadata = metadata.copy()
    updated_metadata['/Signature'] = signature
    updated_metadata['/Certificate'] = certificate

    pdf_writer.add_metadata(updated_metadata)

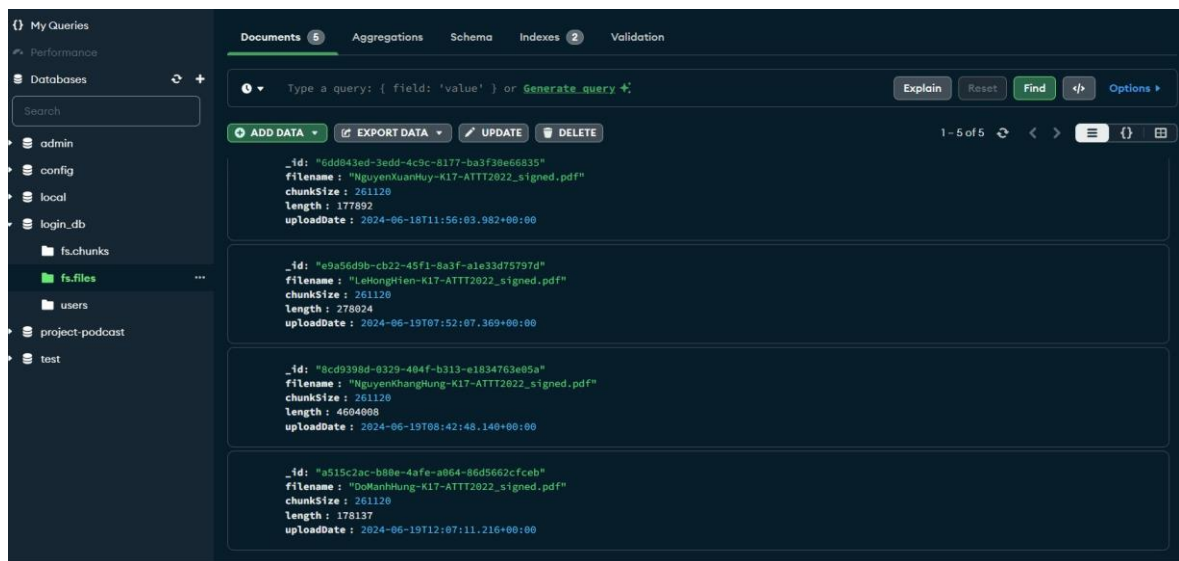
    with open(output_path, 'wb') as output_pdf:
        pdf_writer.write(output_pdf)
```

Hình 3.4.2.5: Văn bản gắn Metadata chứa trường Signature và Certificate

Văn bản sau khi ký sẽ được thêm trường Signature và Certificate vào Metadata PDF, là dấu hiệu để biết có phải là văn bản đã được ký hay không.

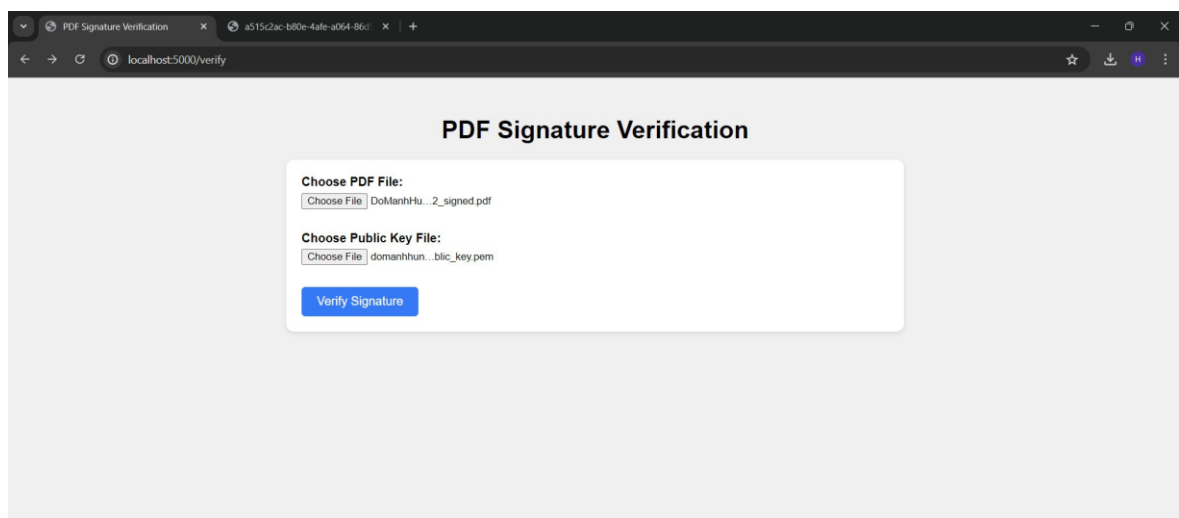


Hình 3.4.2.6: Signed Degree



Hình 3.4.2.7: Signed Degree MongoDB

Văn bằng đã ký sẽ được lưu trữ ở Database của hệ thống.



Hình 3.4.2.8: Module Verify

Các công ty cần xác thực văn bằng sẽ sử dụng module xác thực của hệ thống, với input là file văn bằng đã ký và Public Key của Principal được chứa trong Certificate lưu trữ trên database.

```
def verify_signature(pdf_path, public_key_path):
    public_key_base64 = read_pem_file(public_key_path)
    public_key = base64.b64decode(public_key_base64)

    pdf_content = read_pdf_text(pdf_path)

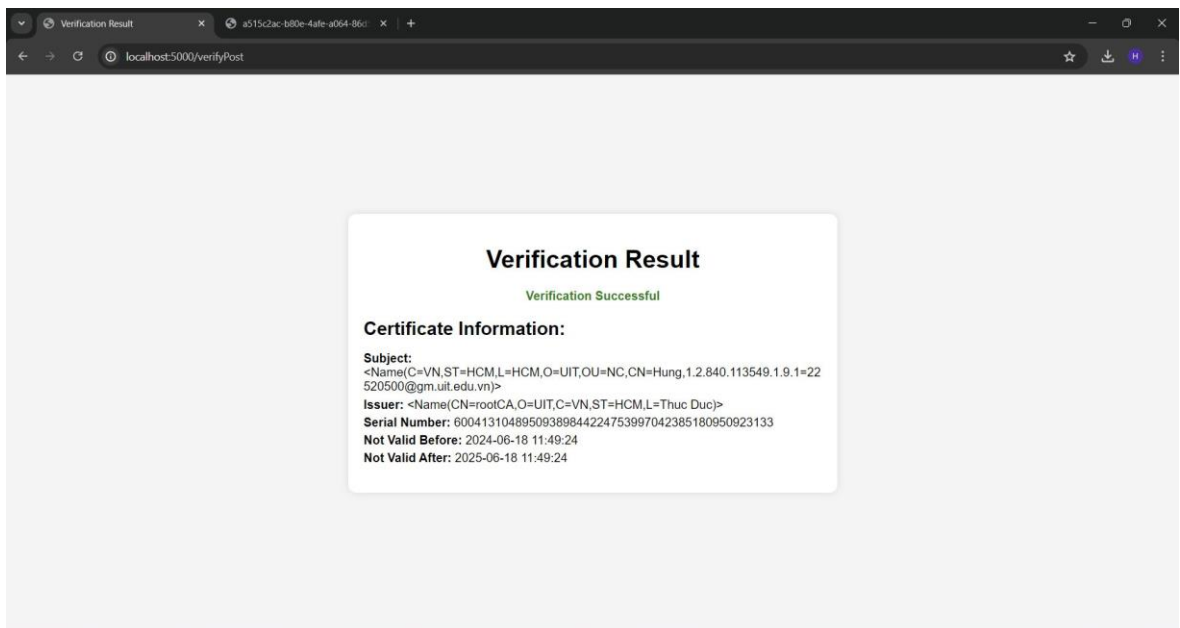
    signature_metadata = extract_signature_from_metadata(pdf_path)
    write_file("temp\\signature_metadata.pem", signature_metadata)
    signature_base64 = read_pem_file("temp\\signature_metadata.pem")
    signature = base64.b64decode(signature_base64)

    certificate_metadata = extract_certificate_from_metadata(pdf_path)
    write_file_with_header_footer("temp\\certificate_metadata.pem", certificate_metadata, "-----BEGIN CERTIFICATE-----", "-----END CERTIFICATE-----")
    cert_info = read_certificate_info("certificate_metadata.pem")

    ver = oqs.oqs.Signature("Dilithium5")
    is_valid = ver.verify(pdf_content, signature, public_key)
    print(f"Valid : {is_valid}")
    return is_valid, cert_info
```

Hình 3.4.2.9: Verify Code

Và đây là kết quả khi verify thành công, sẽ hiện ra người cấp văn bằng, rootCA cấp certificate. Ngược lại sẽ trả về giá trị Failed.



Hình 3.4.2.10: Verification result

TÀI LIỆU THAM KHẢO

- [1] Xavier Morales Rivero. (2022). Seamless transition to post-quantum resistant: Implementing digital signatures for PDF documents using PQ algorithms. Journal of Cryptographic Research. IEEE Access, 8, 18546-18557