ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Mật mã và an ninh mạng

"ỨNG DỤNG MÃ HÓA VÀ GIẢI MÃ"

Giảng viên: Nguyễn Hữu Hiếu
Bài tập lớn: Số 1
SV thực hiện: Nguyễn Thanh Tuấn
MSSV: 1613907

TP. HỒ CHÍ MINH, THÁNG 5/2020

Mục lục

Danh sách hình vẽ

1 Tóm tắt

Những nội dung được trình bày trong báo cáo này bao gồm:

- Tìm hiểu các giải thuật mã hóa: Đối xứng(DES,AES), Bất đối xứng(RSA).
- Phân tích yêu cầu
- Thiết kế và hiện thực chương trình
- Đánh giá ưu, nhược điểm của chương trình
- Hướng phát triển tương lai cho chương trình
- Hướng dẫn sử dụng

2 Giới thiệu cơ sở lý thuyết

2.1 Giải thuật DES

- DES (Data Encryption Standard) là một mật mã khối đối xứng được tiêu chuẩn hóa trong FIPS 46-3. DES có kích thước khối dữ liệu cố định là 8 byte. Các khóa của nó dài 64 bit, mặc dù 8 bit đã được sử dụng cho tính toàn vẹn (hiện tại chúng bị bỏ qua) và không đóng góp cho bảo mật. Do đó, độ dài khóa hiệu quả chỉ là 56 bit. DES không bao giờ bị phá vỡ bằng mật mã, nhưng độ dài khóa của nó quá ngắn theo tiêu chuẩn ngày nay và có thể bị vét cạn với sự hỗ trợ của những máy tính hiện đại.
- DES: Data Encryption Standard
- Là giải thuật mã hóa đối xứng phổ biến.
- Mỗi thông điệp (message) được chia thành những khối (block) 64 bits
- Khóa có 56 bits
- Có thể bị tấn công bằng giải thuật vét cạn khóa (Brute-force or exhaustive key search)

2.2 Giải thuật AES

- AES: Advanced Encryption Standard
- Là giải thuật mã hóa đối xứng.
- AES còn gọi là Rijndael, tên đặt theo hai nhà mật mã học thiết kế ra giải thuật là Daemen và Rijmen
- Rijndael là giải thuật mã hóa theo khối. Tuy nhiên, khác với DES, Rijndael có thể làm việc với dữ liệu và khóa có độ dài block là 128, 192 hoặc 256 bit.

2.3 Giải thuật RSA

- Là giải thuật mã hóa bất đối xứng. - RSA: tên được đặt theo tên 3 nhà phát minh ra giải thuật Rivest, Shamir và Adleman.
- Thuật toán sử dụng 2 khóa có quan hệ toán học với nhau: khóa công khai và khóa bí mật.
- Khóa công khai được công bố rộng rãi cho mọi người và được dùng để mã hóa.
- Khóa bí mật dùng để giải mã.

3 Công việc đã làm

3.1 Phân tích yêu cầu

Trong bài tập lớn này, sinh viên được yêu cầu hiện thực một chương trình mã hóa và giải mã tập tin hoặc thư mục trên máy tính sử dụng các giải thuật mã hóa phổ biến như DES, AES, RSA. Em xây dựng chương trình cho phép mã hóa và giải mã tập tin hoặc toàn bộ tập tin trong thư mục sử dụng một trong ba giải thuật mã hóa DES, AES, RSA.
Để hiện thực được chương trình, em chọn ngôn ngữ lập trình Python và module PyQt5 xây dựng giao diện người dùng (GUI).
Lý do chọn Python để hiện thực chương trình:

- Là một ngôn ngữ có hình thức sáng sủa, cấu trúc roc ràng, cú pháp ngắn gọn.
- Có trên tất cả các nền tảng hệ điều hành từ UNIX, MS – DOS, Mac OS, Windows và Linix và các OS khác thuộc họ Unix.
- Tương thích mạnh mẽ với Unix, hardware, thirt-party software với số lượng thư viện khổng lồ (400 triệu người sử dụng).
- Python có tộc độ xử lý nhanh.

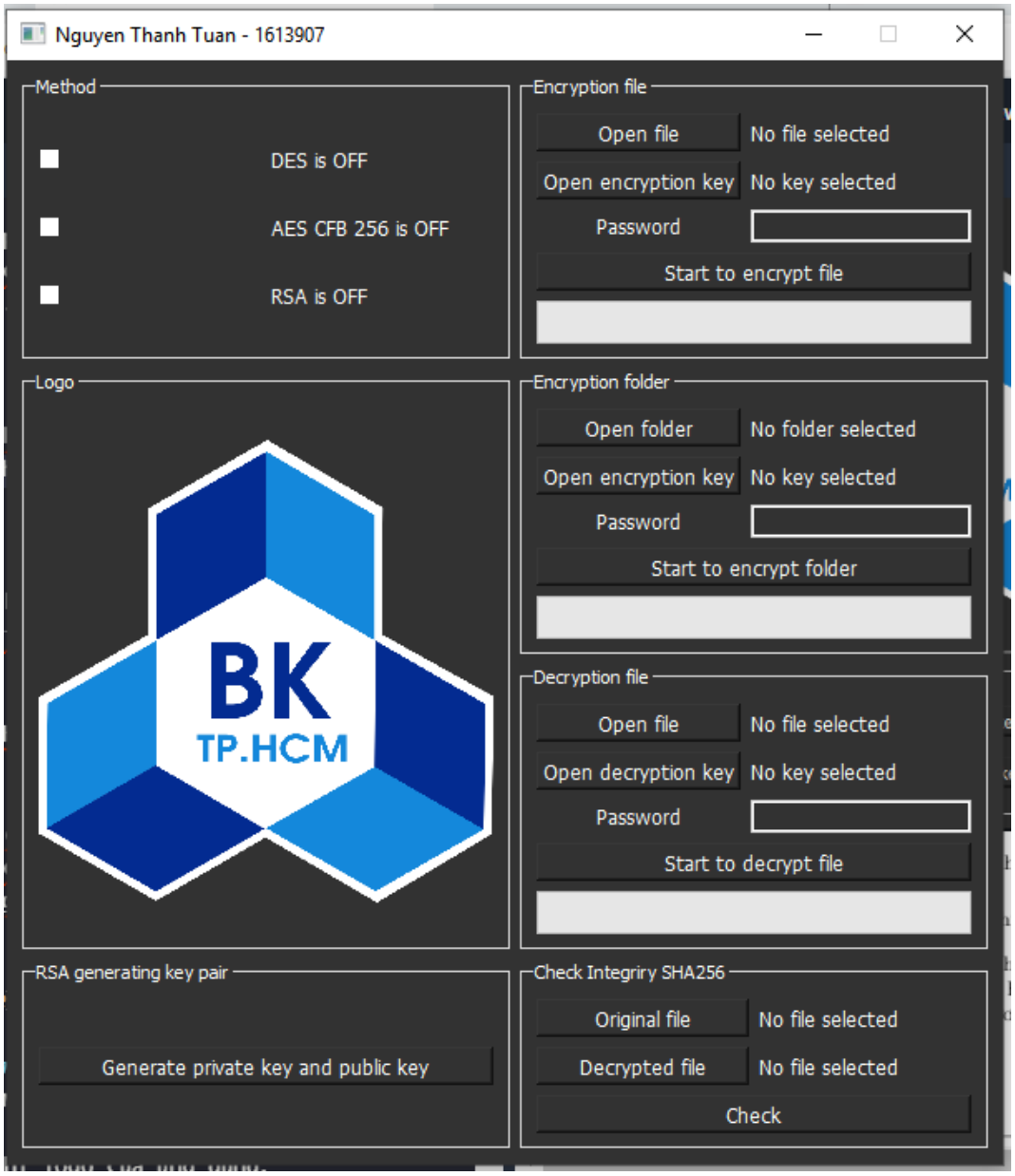
3.2 Thiết kế chương trình

3.2.1 Ngôn ngữ sử dụng

- Ngôn ngữ lập trình Python.
- Thư viện pycryptodome để thực hiện giải thuật mã hóa và giải mã.
- Module PyQt5 để thiết kế giao diện cho chương trình.

3.2.2 Thiết kế giao diện

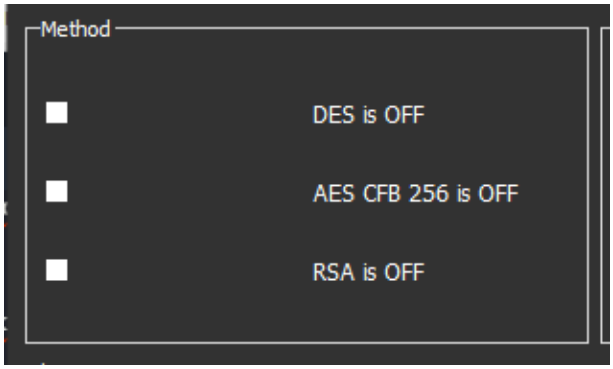
Giao diện chương trình được thiết kế bằng module PyQt5 của Python.



Hình 1: Giao diện chính của ứng dụng

Giao diện bao gồm 6 phần chính:

- Method bao có 3 phương thức mã hóa cho người dùng tùy chọn, bao gồm DES, AES CFB 256 và RSA. Mỗi lượt mã hóa hoặc giải mã, người dùng chỉ được sử dụng duy nhất một phương thức trong ba phương thức trên.

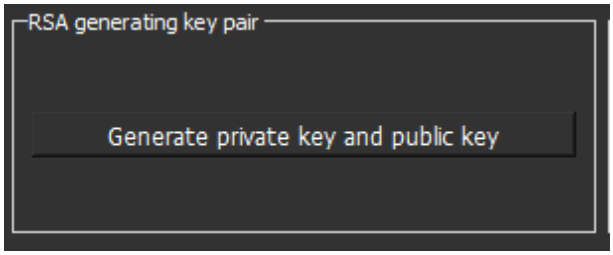


Hình 2: Hộp Method

- Logo hiển thị logo của ứng dụng.

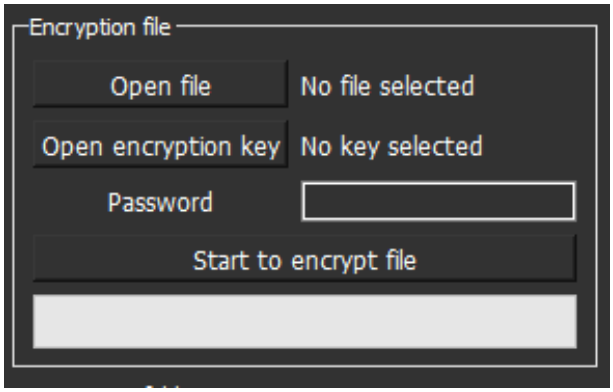
Hình 3: Hộp Logo

- RSA generating key pair để tạo ra cặp khóa riêng tư và công khai cho giải thuật RSA. Chức năng này chỉ sử dụng được khi phương thức mã hóa RSA được chọn.



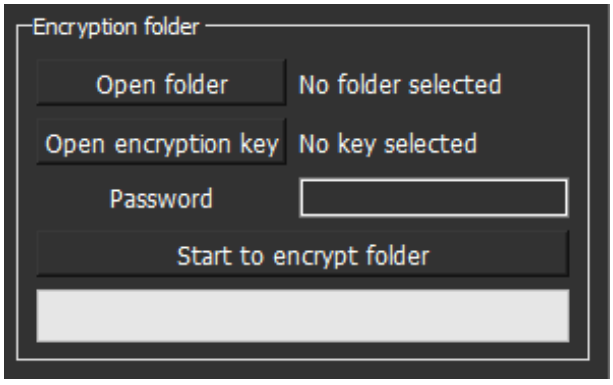
Hình 4: Hộp RSA generating key pair

- Encryption file để mã hóa một file bất kỳ được chọn bởi người dùng.



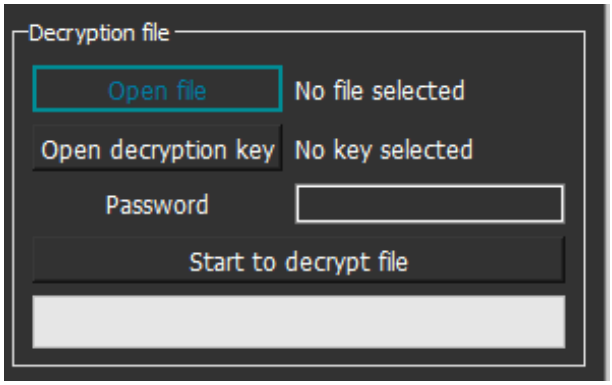
Hình 5: Hộp Encryption file

- Encryption folder để mã hóa tất cả những file trong một thư mục bất kỳ do người dùng chọn.



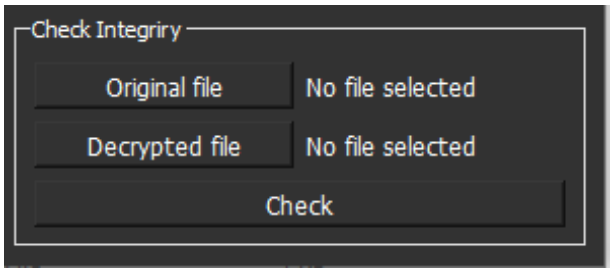
Hình 6: Hộp Encryption folder

- Decryption file để giải mã một file bất kỳ được chọn bởi người dùng.



Hình 7: Hộp Decryption file

- Check integrity để kiêm tra tính toàn vẹn của tập tin ban đầu và tập tin sau khi được giải mã.



Hình 8: Hộp integrity

3.3 Hiện thực chương trình

3.3.1 Hiện thực và đánh giá hàm mã hóa DES

- Hiện thực DES trong Python: Sau khi đọc khóa trong tập tin text (.txt), chương trình cho phép người dùng nhập 1 đoạn mật khẩu. Đoạn mật khẩu này kết hợp với khóa trong tập tin text qua việc sử dụng PBKDF 2(Hàm phát sinh khóa với độ dài cho trước để hạn chế khả năng bị tấn công vét cạn). Sau đó, chọn buffer_size = 64kb. Sau đó tạo ra cipher object và bắt đầu mã hóa dữ liệu.

- Đánh giá giải thuật DES: Mã hóa được nhiều loại tập tin (hình ảnh, âm thanh, video, tập tin .docs, tập tin .ppt, ...), mã hóa tập tin kích thước lớn với thời gian chạy ấn tương.

```
def DES_Encryptor(self, file_path, key_path, password, progress_bar):
    encrypt_salt_location = key_path
    file_in = open(encrypt_salt_location, "rb")
    salt_from_file = file_in.read()
    file_in.close()
    encrypt_key = PBKDF2(password, salt_from_file, dkLen=8)
    file_to_encrypt = file_path
    file_to_encrypt_size = os.path.getsize(file_to_encrypt)
```

```
    buffer_size = 65536 # bytes =  64kb
    num_of_iteration = file_to_encrypt_size // buffer_size
    temp = num_of_iteration // 100
    input_file = open(file_to_encrypt, 'rb')
    output_file = open(file_to_encrypt + '.encrypted', 'wb' )
    cipher_encrypt = DES.new(encrypt_key, DES.MODE_OFB)
    output_file.write(cipher_encrypt.iv)
    buffer = input_file.read(buffer_size)
    count = 0
    value = 1
    while len(buffer) > 0:
        count = count + 1
        if temp == 0 :
            progress_bar.setValue(100)
        else:
            if count % temp == 0:
                progress_bar.setValue(value)
                value = value + 1
        ciphered_bytes = cipher_encrypt.encrypt(buffer)
        output_file.write(ciphered_bytes)
        buffer = input_file.read(buffer_size)
    input_file.close()
    output_file.close()
    # os.remove(file_to_encrypt)
```

3.3.2 Hiện thực và đánh giá hàm mã hóa AES

- Hiện thực AES trong Python:
- Đánh giá giải thuật AES: Mã hóa được nhiều loại tập tin (hình ảnh, âm thanh, video, tập tin .docs, tập tin .ppt, ...), mã hóa tập tin kích thước lớn với thời gian chạy ấn tượng.

```
def AES_Encryptor(self, file_path, key_path, password, progress_bar):
    encrypt_salt_location = key_path
    file_in = open(encrypt_salt_location, "rb")
    salt_from_file = file_in.read()
    file_in.close()
    encrypt_key = PBKDF2(password, salt_from_file, dkLen=32)
    file_to_encrypt = file_path
    file_to_encrypt_size = os.path.getsize(file_to_encrypt)
    buffer_size = 65536 # Tạo buffer
    num_of_iteration = file_to_encrypt_size // buffer_size
    temp = num_of_iteration // 100
    input_file = open(file_to_encrypt, 'rb')
    output_file = open(file_to_encrypt + '.encrypted', 'wb' )
    cipher_encrypt = AES.new(encrypt_key, AES.MODE_CFB)
    output_file.write(cipher_encrypt.iv)
    buffer = input_file.read(buffer_size)
    count = 0
    value = 1
```

```
    while len(buffer) > 0:
        count = count + 1
        if temp == 0 :
            progress_bar.setValue(100)
        else:
            if count % temp == 0:
                progress_bar.setValue(value)
                value = value + 1
        ciphered_bytes = cipher_encrypt.encrypt(buffer)
        output_file.write(ciphered_bytes)
        buffer = input_file.read(buffer_size)
    input_file.close()
    output_file.close()
    # os.remove(file_to_encrypt)
```

3.3.3 Hiện thực và đánh giá hàm mã hóa RSA

- Hiện thực RSA trong Python:
- Đánh giá giải thuật RSA: Chỉ mã hóa được tập tin có kích thước nhỏ, thời gian xử lý chậm.

```
def rsa_encrypt_blob(self, file_path, public_key_path, progress_bar):
    fd = open(public_key_path, "rb")
    public_key = fd.read()
    fd.close()
    fd = open(file_path, "rb")
    blob = fd.read()
    fd.close()
    rsa_key = RSA.importKey(public_key)
    rsa_key = PKCS1_OAEP.new(rsa_key)
    blob = zlib.compress(blob)
    file_to_encrypt_size = os.path.getsize(file_path)
    num_of_iteration = file_to_encrypt_size // 470
    temp = num_of_iteration // 100
    chunk_size = 470
    offset = 0
    end_loop = False
    encrypted =  "".encode()
    count = 0
    value = 1
    while not end_loop:
        count = count + 1
        if temp == 0 :
            progress_bar.setValue(100)
        else:
            if count % temp == 0:
                progress_bar.setValue(value)
                value = value + 1
        chunk = blob[offset:offset + chunk_size]
        if len(chunk) % chunk_size != 0:
```

```
            end_loop = True
            chunk += " ".encode() * (chunk_size - len(chunk))
        encrypted += rsa_key.encrypt(chunk)
        offset += chunk_size
    if progress_bar.value() < 100:
        progress_bar.setValue(100)
    return base64.b64encode(encrypted)
```

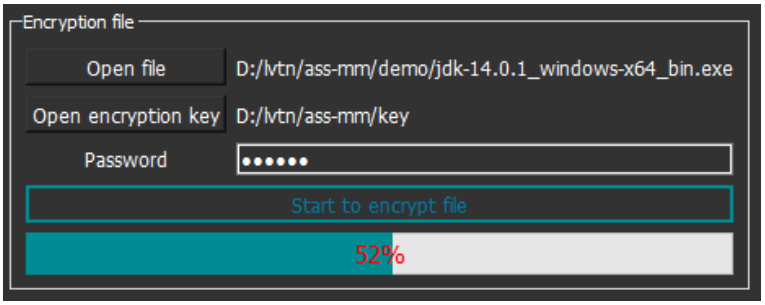
3.4 Kết quả đạt được

3.4.1 Tính năng cơ bản của chương trình

- Mã hóa file hình ảnh (.png, .jpeg, .jpg), file âm thanh (.mp3), file video (.mp4), file văn bản (.pdf, .word), file .ppt, file .exe ,...
- Giải mã những tập tin đã mã hóa.
- Cho phép người dùng nhập mật khẩu trong mã hóa DES và AES.
- Cho phép sinh cặp khóa công khai và riêng tư cho giải thuật RSA.
- Cho phép kiểm tra tính toàn vẹn dữ liệu giữa tập tin gốc và tập tin được giải mã.

3.4.2 Tính năng nâng cao của chương trình

- Mã hóa toàn bộ tập tin trong một thư mục được chọn
- Hiển thị thanh trạng thái trong quá trình mã hóa và giải mã



Hình 9: Hiển thị trạng thái quá trình mã hóa file

3.5 Hạn chế của chương trình

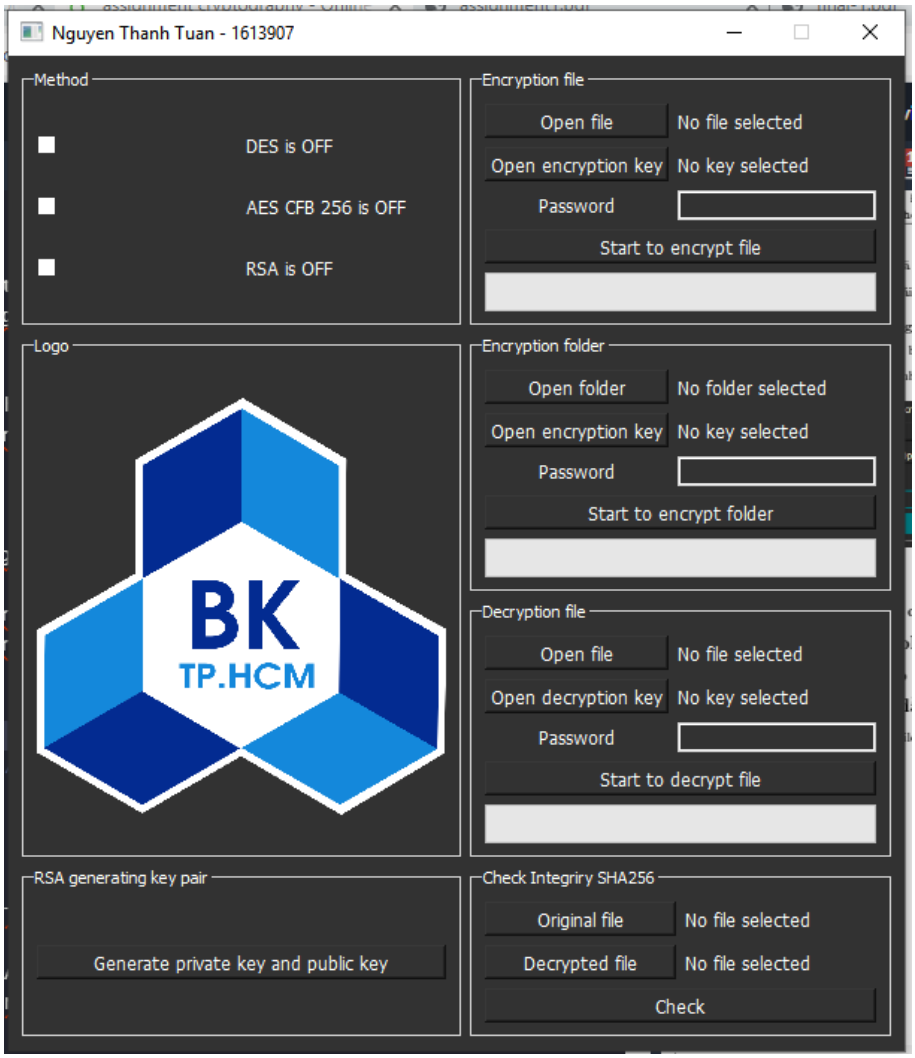
Giải thuật RSA chưa mã hóa được tập tin có kích thước lớn.
Thời gian mã hóa và giải mã của giải thuật RSA còn lâu.

4 Hướng phát triển trong tương lai

Cải thiện giải thuật RSA để chương trình có thể mã hóa được tập tin có kích thước lớn.

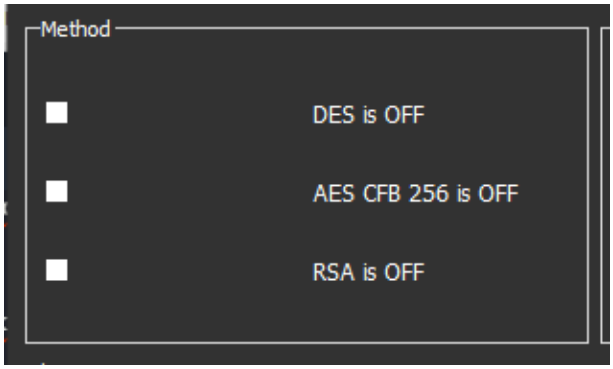
5 Hướng dẫn sử dụng

Người dùng mở file program.exe để khởi động chương trình.
Xem video hướng dẫn tại https://youtu.be/GZMVE-4Zj3I
Source tại https://github.com/tuanbieber/assignment_mm



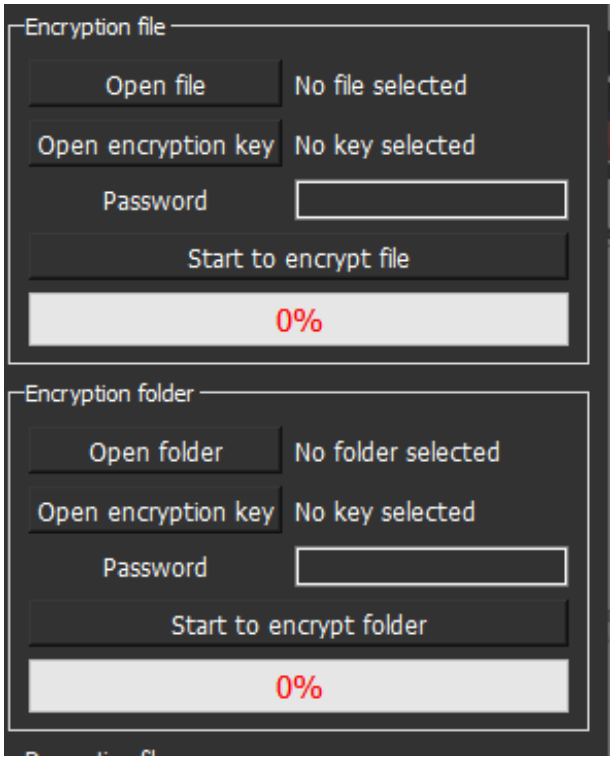
Hình 10: Giao diện chính của chương trình

Ở góc trên cùng bên trái là nơi người dùng chọn phương thức mã hoá (Chương trình hỗ trợ 3 phương thức là DES, AES và RSA).



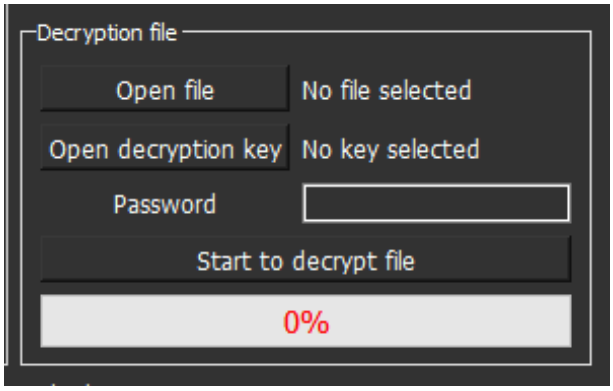
Hình 11: Hộp Method

Sau khi chọn phương thức mã hóa, người dùng chọn file hoặc folder cần mã hóa, chọn khóa mã hóa và nhập mật khẩu (Nếu là phương thức mã hóa DES hoặc AES).



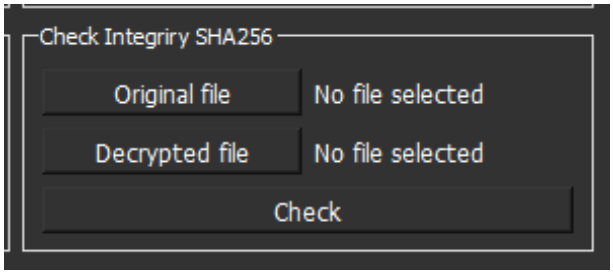
Hình 12: Chọn tệp tin hoặc thư mục cần mã hóa

Để giải mã, người dùng chọn tập tin cần giải mã và chọn khóa tương ứng. Sau đó bấm nút "Start to decrypt file".



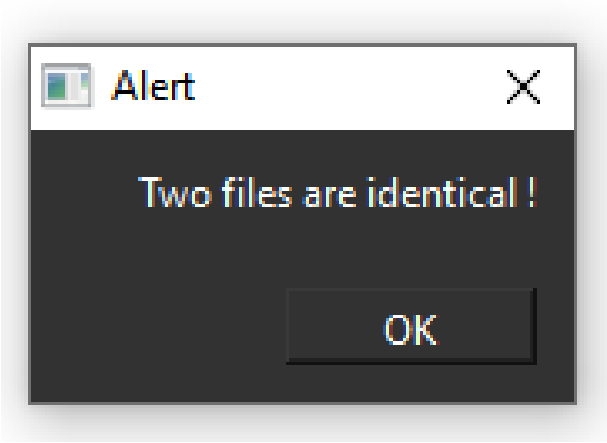
Hình 13: Chọn tệp tin cần giải mã

Sau khi giải mã xong, nếu người dùng cần kiểm tra tính toàn vẹn của dữ liệu thì chọn hộp thoại "Check Integrity SHA256" và chọn lần lượt tâp tin gốc, tập tin sau khi đã giải mã và bấm nút "Check" để kiểm tra.

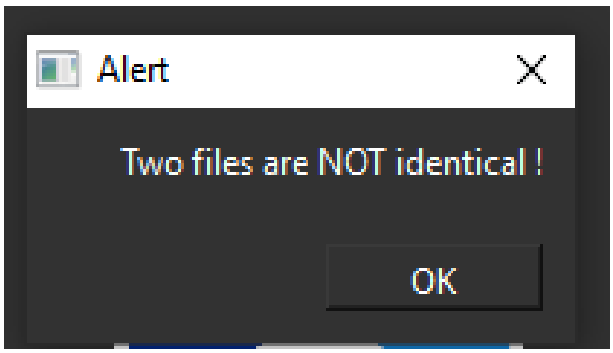


Hình 14: Kiểm tra tính toàn vẹn của tập tin gốc và tập tin được giải mã

Nếu giá trị hash của 2 tập tin là như nhau thì chương trình sẽ thông báo 1 tập tin giống nhau (Đảm bảo tính toàn vẹn), ngược lại thông báo 2 tập tin không giống nhau (Không đảm bảo tính toàn vẹn).

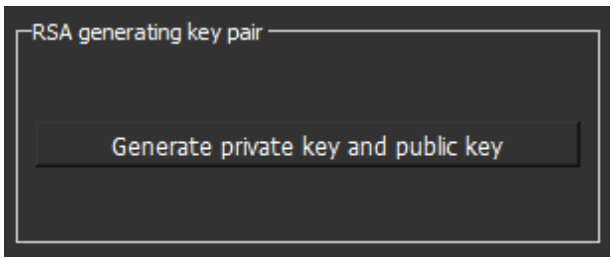


Hình 15: Chương trình thông báo đảm bảo tính toàn vẹn



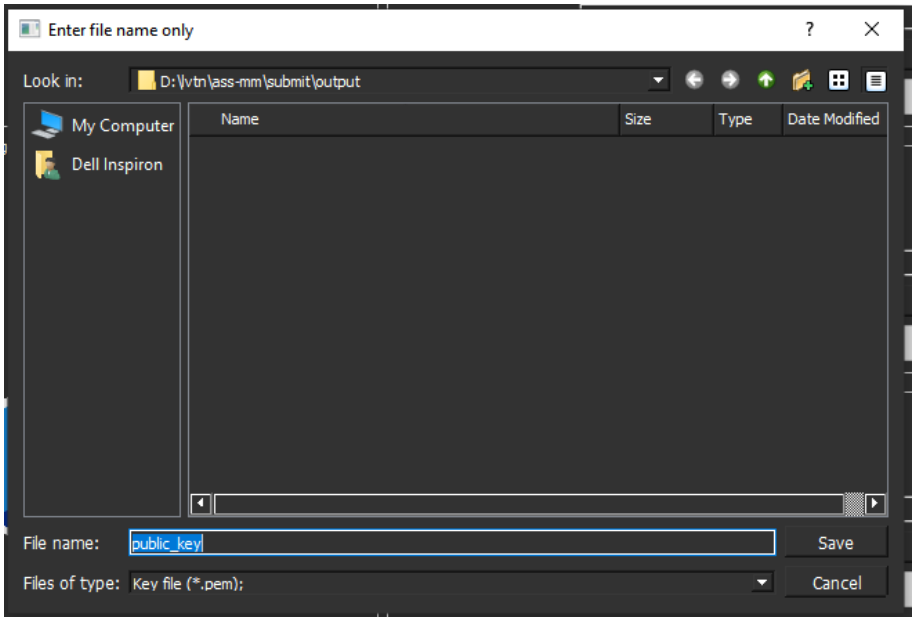
Hình 16: Chương trình thông báo không đảm bảo tính toàn vẹn

Để tạo một căp khóa công khai và khóa riêng tư cho thuật toán mã hóa RSA, người dùng bấm nút "Generate private key and public key" trong hộp thoại "RSA generating key pair".



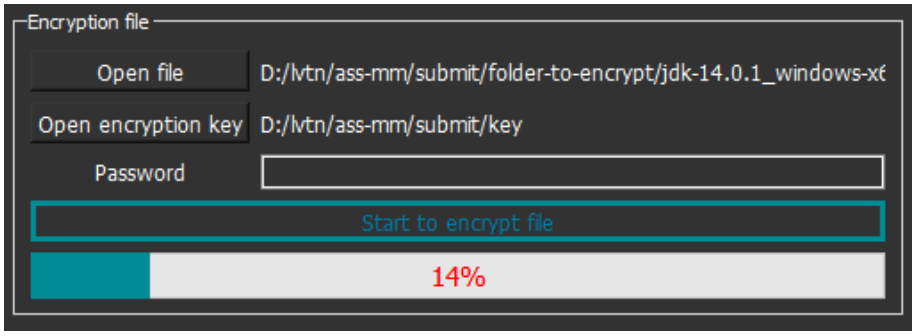
Hình 17: Chương trình thông báo không đảm bảo tính toàn vẹn

Người dụng chọn nơi lưu trữ cặp khóa, việc tạo khóa sẽ tốn một thời gian nhất định, khi hoàn thành chương trình sẽ hiện hộp thoại thông báo.



Hình 18: Chọn nơi lưu trữ khóa

Trong suốt quá trình mã hóa tập tin, mã hóa thư mục và giải mã tập tin sẽ có 1 thanh progress hiển thị quá trình mã hóa, giải mã. Khi mã hóa hoặc giải mã xong chương trình sẽ hiện thông báo.



Hình 19: Thanh hiển thị trạng thái mã hóa

Tài liệu

[1] http://itplus-academy.edu.vn/Uu-va-nhuoc-diem-cua-Python-trong-lap-trinh.html

[2] https://nitratine.net/blog/post/encryption-and-decryption-in-python/

[3] https://nitratine.net/blog/post/asymmetric-encryption-and-decryption-in-python/

[4] https://medium.com/@ismailakkila/black-hat-python-encrypt-and-decrypt-with-rsa-cryptography-

[5] https://pycryptodome.readthedocs.io/en/latest/src/cipher/des.html