ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

MẬT MÃ & AN NINH MẠNG

ĐỀ TÀI :

XÂY DỰNG CHƯƠNG TRÌNH MÃ HÓA VÀ GIẢI MÃ TẬP TIN

Giáo viên hướng dẫn: Nguyễn Hữu Hiếu
Lớp : L02
Họ và tên : Trần Tiến Vũ
MSSV : 1714023

Ho Chi Minh, 5/2020

Mục lục

1 Giới thiệu đề tài

- Đánh giá hiệu năng của các thư viện về Deep Learning như TensorFlow - Keras, Pytorch,... trên các loại GPU cards khác nhau.

2 Cơ Sở Lý thuyết

2.1 Mã hóa đối xứng

- Mã hóa đối xứng là một loại sơ đồ mã hóa trong đó một khóa giống nhau sẽ vừa được dùng để mã hóa, vừa được dùng để giải mã các tệp tin. Phương pháp mã hóa thông tin này đã được sử dụng khá phổ biến từ nhiều thập kỷ với mục đích tạo ra cách thức liên lạc bí mật giữa chính phủ với quân đội. Ngày nay, các thuật toán khóa đối xứng được ứng dụng rộng rãi trên nhiều hệ thống máy tính khác nhau nhằm tăng cường bảo mật cho dữ liệu.

2.1.1 Ưu điểm

Các thuật toán đối xứng vừa có khả năng cung cấp mức độ bảo mật khá cao, vừa có khả năng cho phép mã hóa và giải mã tin nhắn rất nhanh. Mức độ đơn giản về tương quan của các hệ thống đối xứng cũng là một ưu điểm về mặt logic bởi nó sử dụng ít năng lượng tính toán hơn so với các hệ thống bất đối xứng. Thêm vào đó, cấp độ bảo mật mà mã hóa đối xứng mang lại có thể được nhân rộng lên một cách đơn giản chỉ bằng việc tăng độ dài của các khóa. Với mỗi bit được thêm vào trong độ dài 1 khóa đối xứng, thì độ khó của việc phá vỡ mã hóa đó bằng tấn công brute force sẽ tăng lên theo cấp số mũ.

2.1.2 Nhược điểm

Mặc dù mã hóa đối xứng mang lại khá nhiều lợi ích rộng rãi, nhưng nó lại sở hữu một bất lợi khá lớn: vấn đề cố hữu trong việc truyền tải các khóa dùng để mã hóa và giải mã dữ liệu. Nếu các khóa này được chia sẻ lên các kết nối không an toàn thì nguy cơ bị can thiệp bởi một bên thứ 3 là rất lớn. Khi một người dùng không được ủy quyền chiếm được quyền truy cập một khóa đối xứng thì mọi dữ liệu được mã hóa bằng khóa đó sẽ bị xâm phạm. Để giải quyết vấn đề này, hiện này nhiều giao thức website đã sử dụng kết hợp cả mã hóa đối xứng và bất đối xứng nhằm thiết lập các kết nối an toàn. Giao thức mã hóa Bảo mật Tầng Vận tải (TLS) là một trong nhiều ví dụ điển hình được sử dụng để bảo mật cho phần lớn mạng internet ngày nay.

2.2 Gải thuật AES

- AES là một thuật toán "mã hóa khối" (block cipher) ban đầu được tạo ra bởi hai nhà mật mã học người Bỉ là Joan Daemen và Vincent Rijmen. Kể từ khi được công bố là một tiêu chuẩn, AES trở thành một trong những thuật toán mã hóa phổ biến nhất

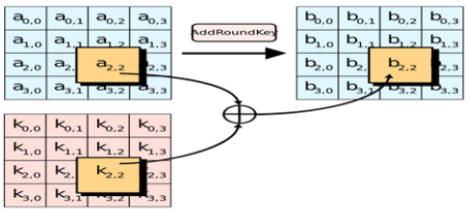
sử dụng khóa mã đối xứng để mã hóa và giải mã (một số được giữ bí mật dùng cho quy trình mở rộng khóa nhằm tạo ra một tập các khóa vòng). Ở Việt Nam, thuật toán AES đã được công bố thành tiêu chuẩn quốc gia TCVN 7816:2007 năm 2007 về Thuật toán mã hóa dữ liệu AES.

- AES là một thuật toán mã hóa khối đối xứng với độ dài khóa là 128 bít (một chữ số nhị phân có giá trị 0 hoặc 1), 192 bít và 256 bít tương ứng dọi là AES-128, AES-192 và AES-256. AES-128 sử dụng 10 vòng (round), AES-192 sử dụng 12 vòng và AES-256 sử dụng 14 vòng.

- Vòng lặp chính của AES thực hiện các hàm sau: SubBytes(), ShiftRows(), MixColumns() và AddRoundKey(). Ba hàm đầu của một vòng AES được thiết kế để ngăn chặn phân tích mã bằng phương thức "mập mờ" (confusion) và phương thức "khuếch tán" (diffusion), còn hàm thứ tư mới thực sự được thiết kế để mã hóa dữ liệu. Trong đó "khuếch tán" có nghĩa là các kiểu mẫu trong bản rõ (Dữ liệu đầu vào của phép mã hóa hoặc dữ liệu đầu ra của phép giải mã) được phân tán trong các bản mã (Dữ liệu đầu ra của phép mã hóa hoặc dữ liệu đầu vào của phép giải mã), "mập mờ" nghĩa là mối quan hệ giữa bản rõ và bản mã bị che khuất. Một cách đơn giản hơn để xem thứ tự hàm AES là: Trộn từng byte (SubBytes), trộn từng hàng (ShiftRows), trộn từng cột (MixColumns) và mã hóa (AddRoundKey).

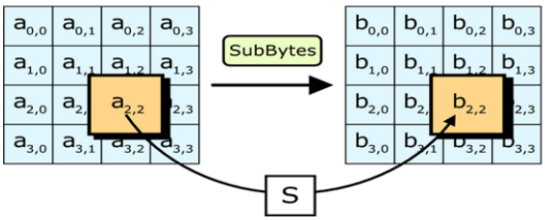
Quá trình mã hóa:

- Hàm AddRoundKey:
  - Được áp dụng từ vòng lặp thứ 1 tới vòng lặp Nr
  - Trong biến đổi Addroundkey(), một khóa vòng được cộng với state bằng một phép XOR theo từng bit đơn giản.
  - Mỗi khóa vòng gồm có 4 từ (128 bit) được lấy từ lịch trình khóa. 4 từ đó được cộng vào mỗi cột của state



Hình 1: Hàm AddRoundKey

- Hàm SubBytes:
  - Biến đổi SubBytes() thay thế mỗi byte riêng rẽ của state Sr,c bằng một giá trị mới S' r,c sử dụng bảng thay thế (S – box) được xây dựng ở trên.



Hình 2: Hàm SubBytes

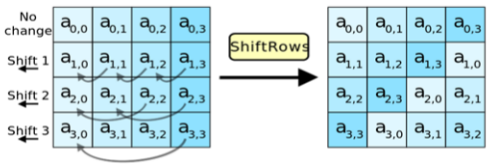
- Hàm ShiftRow:
  - Trong biến đổi ShiftRows(), các byte trong ba hàng cuối cùng của trạng thái được dịch vòng đi các số byte khác nhau (độ lệch). Cụ thể:

$$S'r, c = Sr, (c + shift(r, Nb))modNb(Nb = 4)$$

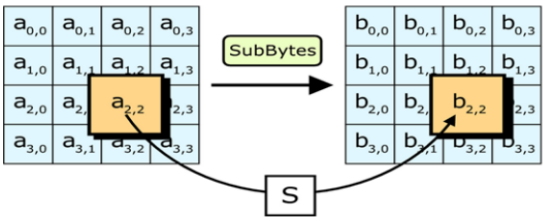
  - Trong đó giá trị dịch shift (r, Nb) phụ thuộc vào số hàng r như sau:

$$Shift(1, 4) = 1, shift(2, 4) = 2, shift(3, 4) = 3.$$

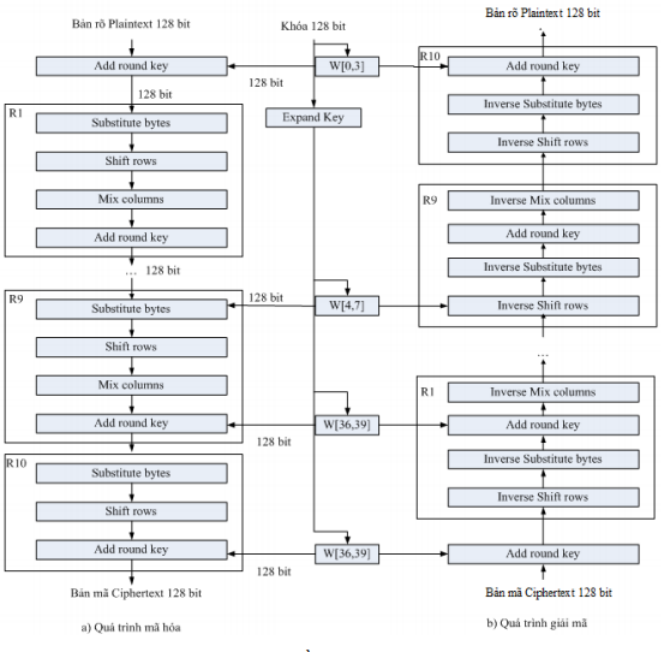
  - Hàng đầu tiên không bị dịch, ba hàng còn lại bị dịch tương ứng:
    - ☐ Hàng thứ 1 giữ nguyên.
    - ☐ Hàng thứ 2 dịch vòng trái 1 lần.
    - ☐ Hàng thứ 3 dịch vòng trái 2 lần.
    - ☐ Hàng thứ 4 dịch vòng trái 3 lần.



Hình 3: Hàm ShiftRow



Hình 4: Hàm SubBytes



Hình 5: Thuật toán AES 128bit

3 Phân tích yêu cầu hệ thống

3.1 Yêu cầu chức năng

- Mã hóa tất cả file dạng abc.xyz thành dạng file .bat, mã hóa folder, mã hóa nhiều file đồng thời.
- Có tùy chọn tự động sinh khóa hoặc dùng khóa do người dùng chọn.
- Giải mã file với khóa được cung cấp

3.2 Yêu cầu phi chức năng

- Thời gian thực thi ngắn
- Dung lượng file mã hóa/giải mã
- Giao diện đơn giản, dễ sử dụng

4 Hiện thực

4.1 Thư viện lập trình

- javax.crypto là thư viện java định nghĩa các lớp và Interfaces cho các tác vụ sinh khóa, mã hóa và giải mã. Figure 6 hiển thị các lớp phân cấp của package này.

- Các lớp được sử dụng trong chương trình:

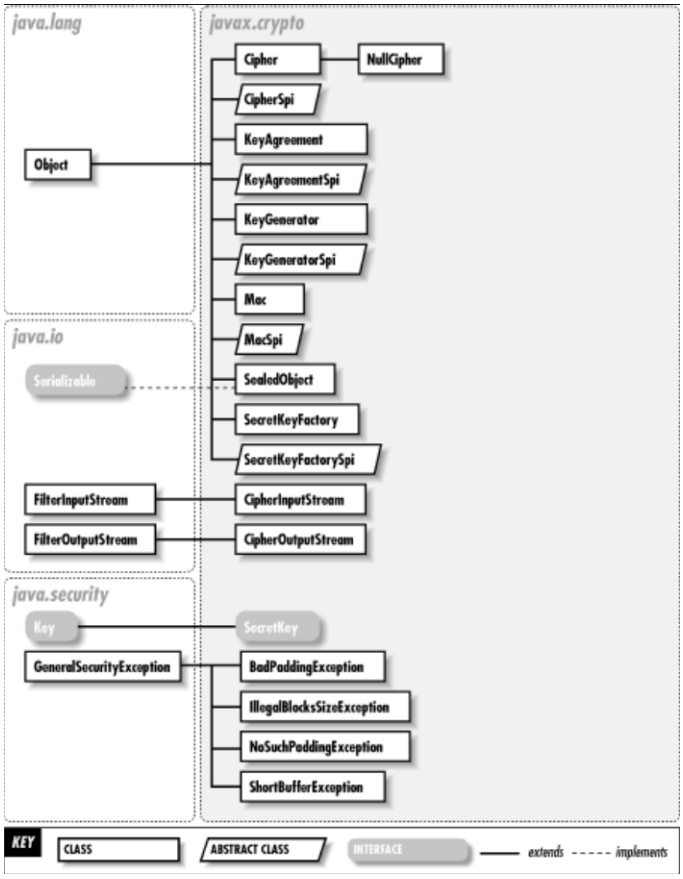
- Cipher thực hiện các tác vụ mã hóa và giải mã dùng khóa bí mật
- KeyGenerator cung cấp API cho việc tạo SecretKey (khóa đối xứng), là một tham số truyền vào của phương thức Cipher.init
- SecretKeySpec tạo khóa đối xứng từ file key nhập của người dùng

4.2 Cấu trúc dữ liệu

- Dùng cấu trúc Queue để lưu trữ các Path của file cần xử lí. Mỗi khi tác vụ encrypt hoặc decrypt được gọi, chương trình tiếp tục thực thi đến khi hàng đợi xử lí queue rỗng.
- Các file trong hàng đợi xử lí được đóng gói bởi lớp java.io.File
- Hiện thực class AES tạo đối tượng mã hóa/giải mã và cung cấp tác vụ mã hóa/giải mã

4.3 Một số hàm quan trọng

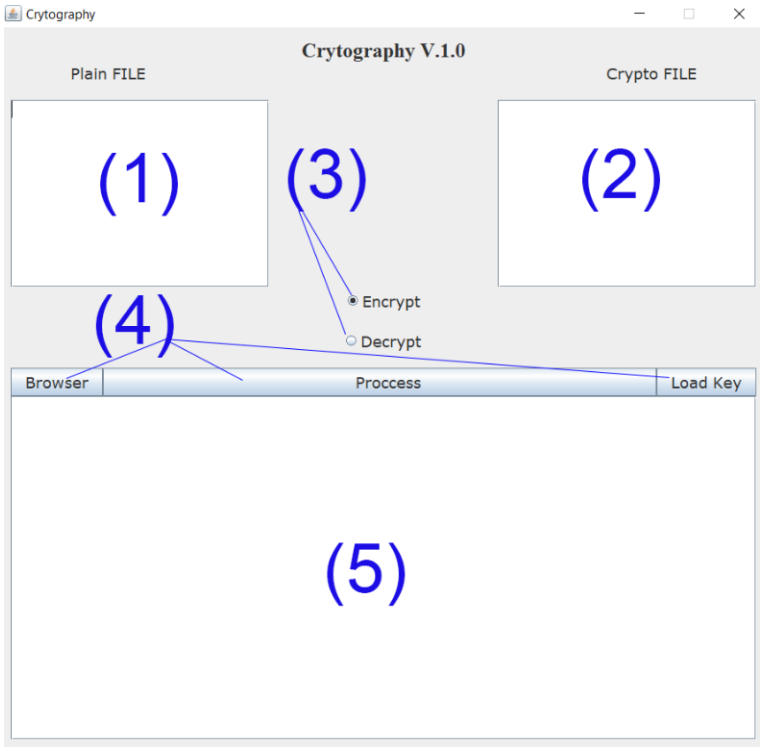
- Hàm Encrypt() / Decrypt() trong class AES : thực hiện quá trình khởi tạo đối tượng Cipher với key được cung cấp và gọi đến hàm proccessFile(...) để tiến hành mã hóa/giải mã file nằm ở đâu hàng đợi xử lí.
- Hàm proccessFile() thực hiện các quá trình mã hóa/giải mã tương ứng với đối tượng Cipher truyền vào và lưu file (secrKey) vào đường dẫn do người dùng tùy chọn



Hình 6: Kiến trúc thư viện javax.crypt

Tên hàm	Input	Output	Chức năng
Encrypt()		File mã hóa dạng .en, file khóa đối xứng, mã md5 hash code	Mã hóa file ban đầu thành file mã hóa có định dạng .en, lưu khóa đối xứng và mã xác thực toàn vẹn dữ liệu md5 cùng nơi với file mã hóa
Decrypt()		File được giải mã	Giải mã file .en được mã hóa bằng key do người dùng cung cấp
proccessFile()	Cipher ci,String inFile, String outFile		

4.4 Giao diện



Hình 7: Giao diện chương trình

1. Hiển thị tên file cần giải mã
2. Hiển thị tên file cần giải mã.
3. Các tùy chọn mã hóa hoặc giải mã
4. Các chức năng tùy chọn file, mã hóa/giải mã, load key
5. Hiển thị trạng thái các file trong hàng chờ xử lí.

5 Đánh giá kết quả, nhận xét

5.1 Đánh giá kết quả

5.1.1 Tính toàn vẹn dữ liệu

File ban đầu được xử lí với thuật toán MD5 (Message-Digest Algorithm), kết quả cho ra là một Checksum với kích thước cố định (32 ký tự hexa) tương ứng với 128 bit. Chỉ cần một sai khác bất kì trong dữ liệu sau khi decrypt người dùng sẽ được cảnh báo sau khi quá trình decrypt kết thúc. Do đó đảm bảo được yêu cầu về tính toàn toàn vẹn dữ liệu.

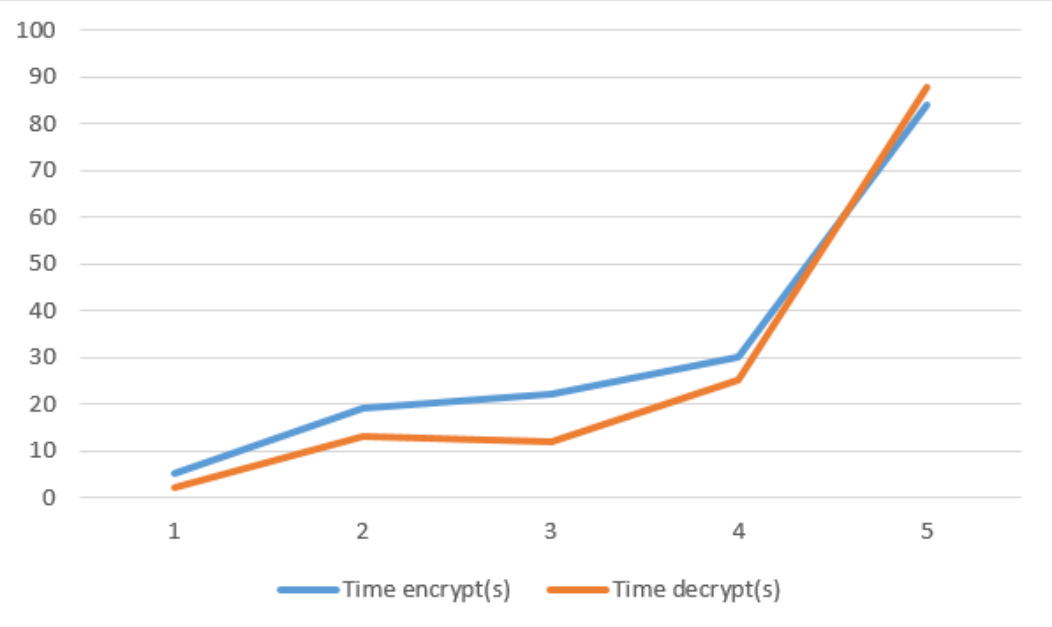
5.1.2 Độ dài file mã hóa/giải mã

Giải thuật AES không có bất kì ràng buộc nào về dung lượng file xử lí, chỉ phụ thuộc vào tài nguyên máy tính đang sử dụng. Dung lượng file input tối đa đã được test thành công là 7Gb.

5.1.3 Thời gian thực thi

Thời gian thực hiện các tác vụ mã hóa, giải mã của AES tương đối nhanh, phục thuộc nhiều vào dung lượng file và cấu hình của máy tính đang chạy chương trình. Tương quan thời gian thực thi giữa các file nhỏ với các file có dung lượng lớn tăng theo tuyến tính nhưng vẫn nằm trong ngưỡng chấp nhận được. (7Gb $\sim$ 1m28s)

STT	Size (Mb)	Encrypt time (s)	Decrypt time (s)
1	6	5	2
2	276	19	13
3	950	22	12
4	1845	30	25
5	7680	84	88



Hình 8: Đo thời gian thực thi của chương trình

5.2 Hướng phát triển

- Tích hợp đa luồng để tăng tốc độ xử lí cho chương trình.
- Cải thiện giao diện, hiển thị tiến độ thực thi chương trình.
- Phát triển số lượng giải thuật, hiện thực giải thuật 3DES hoặc mã hóa bất đối xứng RSA, thêm tùy chọn cho người dùng.
- Hiện tại chương trình chỉ chạy trên hệ điều hành Window nên khả năng tương thích, linh hoạt còn kém, cần cải tiến để có thể chạy trên các hệ điều hành phổ biến như Linux, MacOS.

6 Kết luận

Là phiên bản đầu tiên nên Cryptool còn nhiều hạn chế về trải nghiệm người dùng nhưng cơ bản đã đáp ứng được nhu cầu mã hóa, giải mã các file với hầu hết các định dạng file. Cho thấy khả năng về tốc độ tuyệt vời của giải thuật mã hóa bất đối xứng, tận dụng được điểm mạnh của AES về tốc độ và kích thước để giải quyết bài toán một cách linh hoạt.

Tài liệu tham khảo

[1] "**https://www.researchgate.net/**", last access: 27/05/2020.

[2] "**https://ieeexplore.ieee.org/document/7853276**", last access: 27/05/2020.

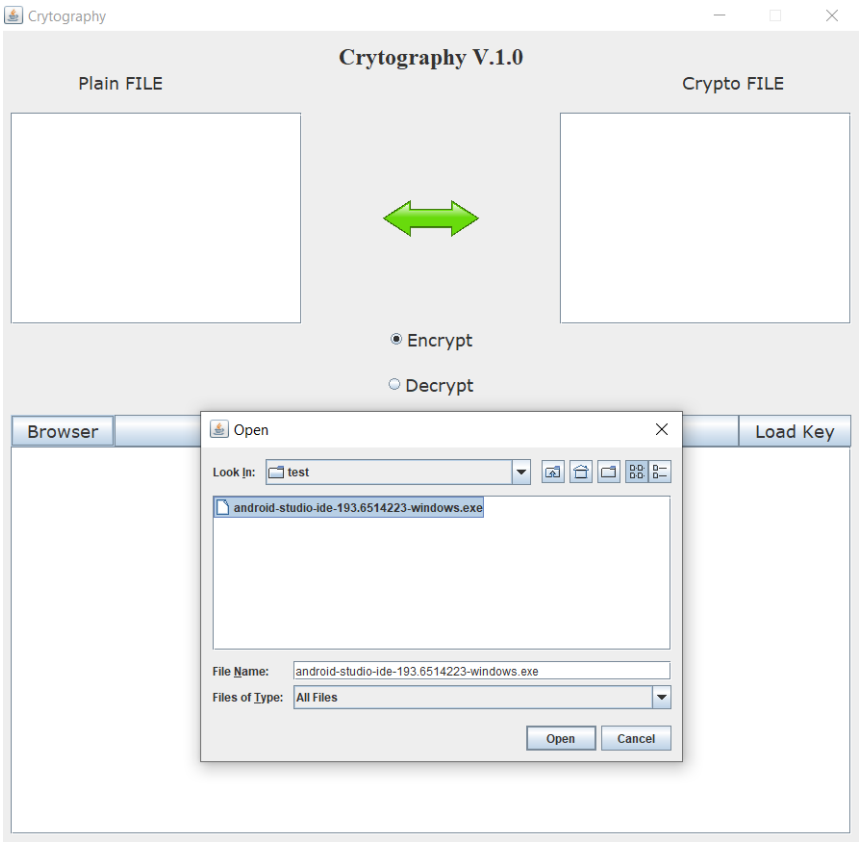
[3] "**https://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html**", last access: 27/05/2020.

[4] ,"**https://www.programcreek.com/java-api-examples/javax.crypto.Cipher**", last access: 27/05/2020.

A Phụ lục

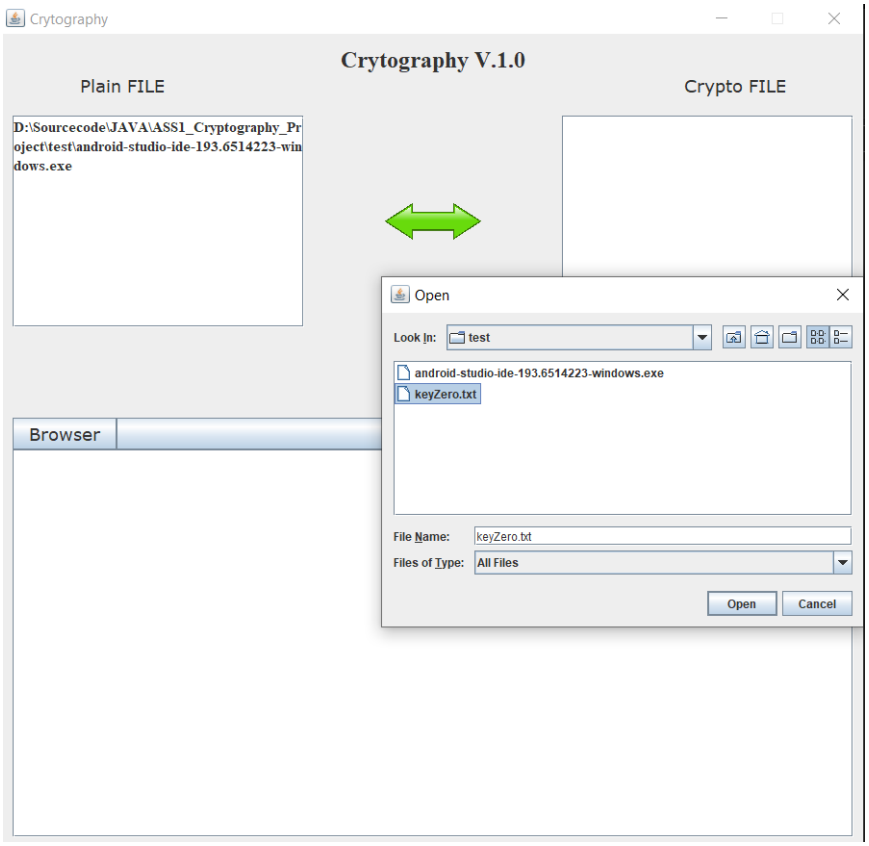
A.1 Hướng dẫn sử dụng

1. Mở file *Cryptool.exe*
2. Chọn mode Encrypt (mặc định) ⇒ giao diện hiển thị giống như Hình 9

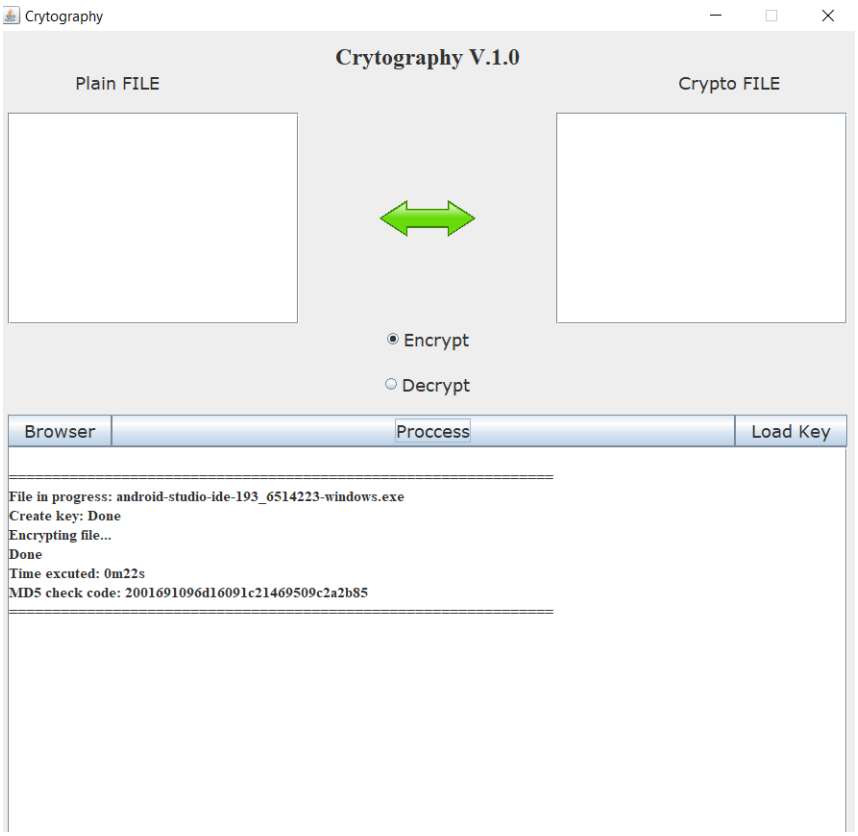


Hình 9: Loading file

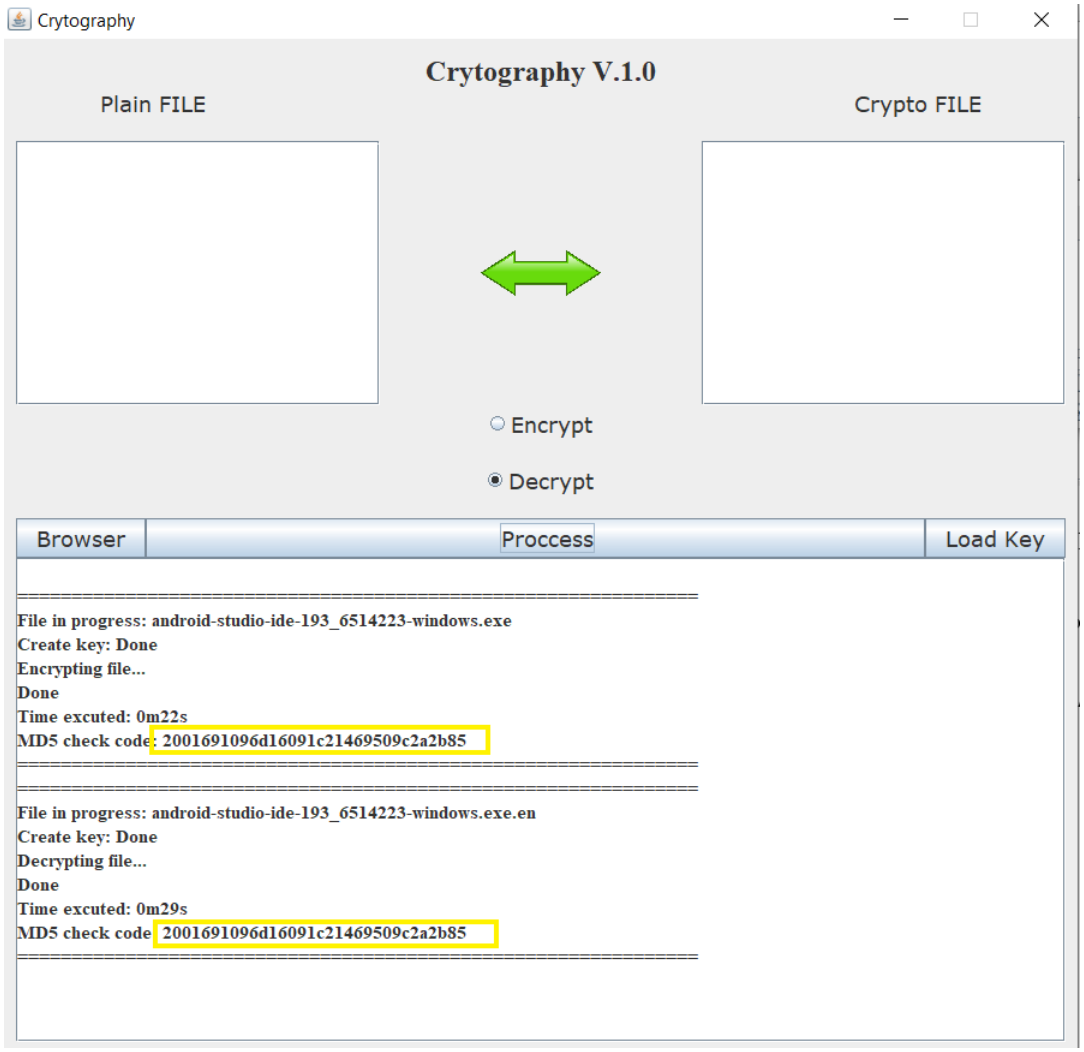
3. Tùy chọn load key hoặc nếu không chọn file key thì key sẽ được sinh tự động.
4. Chọn button proccess ⇒ chọn vị trí lưu file mã hóa (chú ý: có 3 file được tạo thành gồm: file mã hóa đuôi *encrypted_abc.xyz.en*, file key dạng *skey_abc*, file *MD5 Checksum* dạng *skey_abc* với *abc.xyz* là tên file ban đầu). Kết quả sau khi hoàn tất quá trình Encrypt đươc chỉ ra ở Hình 11.
5. Decrypt file: Chuyển mode sang *Decrypt*, Click *Browser* chọn file cần mã hóa, sau đó Click chọn *Load Key*, chọn file *skey* tương ứng với file cần decrypt, Click *Proccess*, kết quả được hiển thị ở Hình 12
6. Tương tự với folder, đầu tiên chương trình sẽ nén folder được chọn thành dạng file .zip rồi thực thi các quá trình *Encrypt*/*Decrypt* tương tự như đã làm ở các mục nêu đã nêu ở trên.



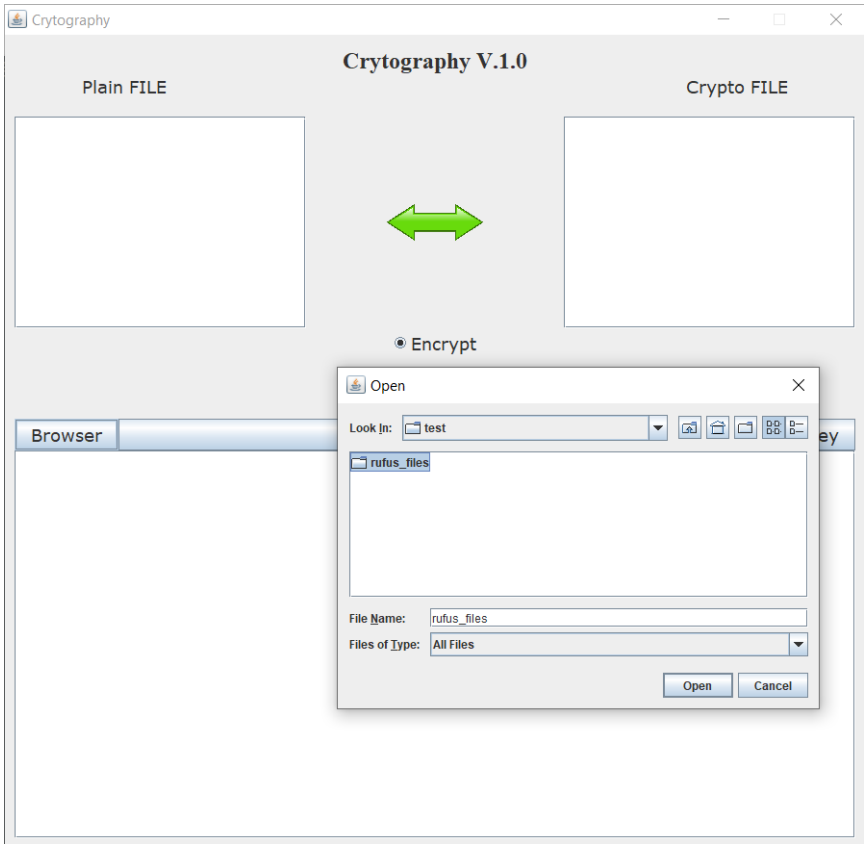
Hình 10: Loading key



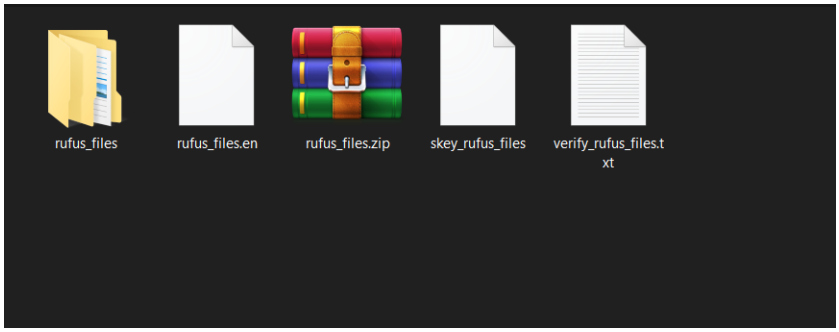
Hình 11: Encrypted result



Hình 12: Decrypted result



Hình 13: Encrypt folder



Hình 14: Kết quả sau khi encrypt folder