TRƯỜNG ĐẠI HỌC HÀNG HẢI VIỆT NAM

VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO

****

BÀI TẬP LỚN

AN TOÀN VÀ BẢO MẬT THÔNG TIN

Đề tài:

HỆ MÃ RC5

Ứng dụng trong chương trình mã hóa và giải mã

Sinh viên: Vũ Tiến Chung – 82418

GHVD: thầy Nguyễn Hữu Tuân

TRƯỜNG ĐẠI HỌC HÀNG HẢI VIỆT NAM

VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO

****

BÀI TẬP LỚN

AN TOÀN VÀ BẢO MẬT THÔNG TIN

1. Tên đề tài

HỆ MÃ RC5

2. Mục đích

Mã hóa và giải mã một chuỗi kĩ tự hoặc một văn bản.

3. Công việc cần thực hiện

- Khảo sát, xác định các vấn đề mà hệ thống yêu cầu.
- Phân tích, tìm hiểu các thuật toán.
- Thiết kế và lập trình cho phù hợp với yêu cầu.
- Demo hoạt động của trương trình
- Kết luận về ưu, nhược điểm.

4. Kết quả

- Kết quả bài tập lớn: có sản phẩm thực tế và báo cáo bài tập lớn hoàn thiện.

Hải phòng, ngày 12 tháng 12 năm 2022

CHƯƠNG I. GIỚI THIỆU BÀI TOÁN

1.1. Mã hóa thông tin là gì?

Trước khi tìm hiểu mã hoá thông tin là quá trình như thế nào thì chúng ta hãy xem mã hoá là gì nhé? Trong ngành mật mã học thì mã hóa chính là quá trình dùng để biến thông tin từ một dạng này sang dạng khác để ngăn chặn những người không có phận sự tiếp cận vào nguồn thông tin đó.

Bản thân việc mã hóa không thể ngăn chặn việc thông tin bị đánh cắp, có điều thông tin đó khi được lấy về cũng không thể dùng được, không đọc được hay hiểu được vì đã được làm biến dạng khó hiểu đi rồi.

Thông qua giải thích phía trên, ta có thể hiểu đơn giản mã hóa là một phương pháp nhằm bảo vệ thông tin cá nhân bằng cách chuyển đổi thông tin từ dạng có thể đọc và hiểu được 1 cách thông thường sang dạng thông tin không thể hiểu được theo cách thông thường. Và dĩ nhiên chỉ có người có quyền truy cập vào khóa giải mã hoặc có mật khẩu thì mới có thể đọc được nó.

Việc làm này giúp ta có thể bảo vệ nguồn thông tin được tốt hơn, đảm bảo an toàn trong việc truyền dữ liệu trên mạng Internet. Dữ liệu khi được mã hóa thành công thường gọi là ciphertext còn dữ liệu thông thường không được mã hóa thì gọi là plaintext.

1.2. Vì sao việc mã hóa thông tin lại quan trọng?

Việc mã hóa các thông tin, dữ liệu là để đảm bảo tính an toàn cho thông tin, đặc biệt là trong thời đại công nghệ số càng ngày càng phát triển như hiện nay. Với những giao dịch điện tử thì mã hoá có vai trò cực kỳ quan trọng, nó đảm bảo bí mật và toàn vẹn thông tin của người dùng khi thông tin được truyền trên mạng Internet. Mã hóa cũng chính là nền tảng cơ bản của kĩ thuật chữ ký điện tử và hệ thống PKI.

1.3. Chức năng của mã hóa thông tin là gì?

Như đã nói thì mục đích chính của việc mã hóa dữ liệu là để bảo vệ dữ liệu số khi nó được lưu trữ trên các hệ thống của máy tính và lan truyền qua Internet hay bất cứ các mạng máy tính khác.

Các thuật toán mã hóa thường sẽ cung cấp những yếu tố bảo mật then chốt như là xác thực, tính toàn vẹn và không thể thu hồi. Bước xác thực sẽ cho phép xác minh được nguồn gốc của dữ liệu, tính toàn vẹn và chứng minh rằng nội dung của dữ liệu sẽ không thể bị thay đổi kể từ khi nó vừa được gửi đi. Không thu hồi được nhằm đảm bảo rằng người đó không thể hủy việc gửi dữ liệu.

Quá trình mã hóa sẽ biến các nội dung này sang một dạng mới, vì thế sẽ làm tăng thêm một lớp bảo mật nữa cho dữ liệu. Như vậy cho dù dữ liệu của bạn có bị đánh cắp thì việc giải mã

dữ liệu này cũng là điều vô cùng khó khăn và gần như là không thể, không chỉ tốn nhiều nguồn lực để tính toán mà còn cần rất nhiều thời gian.

Với những công ty, tổ chức có quy mô lớn thì việc sử dụng mã hóa dữ liệu là điều vô cùng cần thiết. Điều này có thể sẽ giúp tránh được những thiệt hại lớn khi những thông tin bảo mật nếu vô tình bị lộ ra ngoài thì cũng khó lòng mà giải mã ngay lập tức được.

Hiện nay đang có rất nhiều ứng dụng tin nhắn đều sử dụng quy trình mã hóa nhằm bảo mật tin nhắn tới cho người dùng. Chúng ta có thể kể đến như là Facebook, WhatApps với loại mã hóa sử dụng có tên gọi là End-to-End.

1.4. Có những loại mã hóa thông tin nào?

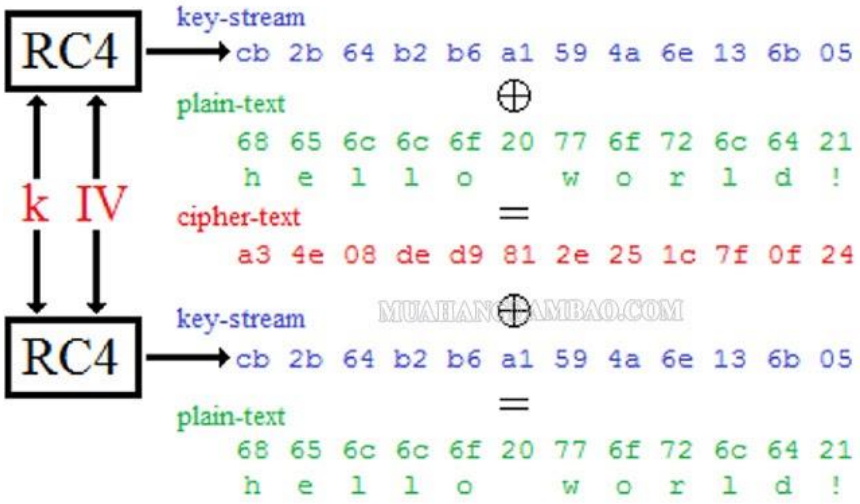
Hiện nay có 4 biện pháp mã hoá dữ liệu thông dụng nhất, cụ thể là:

mã hóa cổ điển:

Mã hóa cổ điển là cách mã hoá đơn giản nhất, tồn tại lâu nhất trên thế giới và không cần tới khóa bảo mật để mở. Chỉ cần người gửi và người nhận cùng hiểu và biết về thuật toán này là có thể giải được.

Ví dụ: Nếu như chúng ta dùng thuật toán đổi ký tự trong câu văn thành các ký tự liền kề trong bảng chữ cái thì chữ "tinh tế" sẽ được biến thành cụm "ujoi uf". Người nhận khi nhận được dòng chữ "ujoi uf" này thì chỉ việc dịch ngược lại là có thể giải được.

Tuy nhiên, giải pháp này lại được xem là không quá an toàn, vì nếu có một người thứ ba biết được thuật toán này thì xem như thông tin đã không còn bảo mật nữa. Việc giữ bí mật thuật toán trở nên vô cùng quan trọng và không phải ai cũng có thể đủ trọng trách để giữ bí mật đó một cách trọn vẹn nhất. Có khả năng cao người đó sẽ rò rỉ ra hoặc có ai đó rảnh rỗi ngồi giải ra thuật toán và nếu may mắn họ giải ra được thì xem như chúng ta đã thua cuộc.



Cách mã hoá theo dạng cổ điển

Mã hóa dạng một chiều (hash)

Phương pháp này được dùng để mã hóa những thứ không cần phải dịch lại ra nguyên bản gốc. Ví dụ, khi đang bạn đăng nhập vào muahangdambao.com thì mật khẩu mà bạn nhập vào sẽ được chuyển hoá thành một chuỗi dài các kí tự bằng một thứ được gọi là hash function, tạm dịch là hàm băm.

Chuỗi này sẽ được lưu ở trong cơ sở dữ liệu chứ không lưu mật khẩu thô của bạn nhằm tăng cao tính bảo mật. Lỡ như các hacker có trộm được dữ liệu thì cũng sẽ chỉ thấy những đoạn ký tự lộn xộn không theo 1 trật tự nào như là FIifsygXhYgBh5j47bhvyuuUIbZ chứ sẽ không thể phát hiện ra được password thật của bạn là gì.

Mỗi khi bạn đăng nhập thì hash function sẽ "băm" password thật của bạn thành 1 chuỗi ký tự rồi so sánh nó với những thứ đã có trong cơ sở dữ liệu, nếu khớp thì mới có thể tiến hành đăng nhập tiếp còn không thì sẽ báo lỗi. Chúng ta không cần phải dịch ngược chuỗi ký tự vô nghĩa nói trên ra lại thành password thật để làm gì cả vì đã có hash.

Nói thêm về phần hash function thì nhiệm vụ chính của nó sẽ là chuyển một chuỗi có độ dài bất kỳ thành các chuỗi ký tự có độ dài cố định. Ví dụ như nếu bạn quy định chuỗi ký tự sau khi được "băm" sẽ dài 10 ký tự thì dù đầu vào của bạn có là bao nhiêu chữ đi nữa thì kết quả mã hoá khi nhận được sẽ chỉ luôn là 10 và chỉ 10 ký tự mà thôi.

Đặc điểm nổi bật của hash function là ở trong cùng 1 điều kiện, dữ liệu đầu vào như nhau thì kết quả sau khi được băm cũng sẽ là y hệt như nhau. Nếu chỉ thay đổi một chút xíu thôi, có khi chỉ là 1 kí tự nhỏ thì chuỗi kết quả trả về cũng sẽ khác nhau hoàn toàn.

Cũng chính vì thế mà người ta thường dùng hash function để kiểm tra tính toàn vẹn của các dữ liệu. Ví dụ, trước khi bạn gửi một tập tin Word cho người khác thì có thể dùng mã hóa một chiều và tạo ra được các chuỗi sau băm là DFFGRYUBUfyehaudfuefu. Khi người đó tải tập tin này về máy, nếu nó băm và cũng nhận được chuỗi là DFFGRYUBUfyehaudfuefu thì có nghĩa là tập tin của bạn đã không hề bị can thiệp bởi các hacker còn nếu kết quả ra khác thì có nghĩa là trong quá trình truyền tải có thể đã xuất hiện lỗi và làm mất một phần dữ liệu hoặc tệ hơn nữa là có ai đó đã xén bớt hay thêm vào thứ gì đó vào nội dung rồi.

Hiện nay, hai thuật toán hash function được dùng nhiều nhất đó chính là MD5 và SHA. Nếu bạn tải 1 tập tin ở trên mạng về máy thì đôi khi sẽ thấy có dòng chữ MD5 do chính tác giả cung cấp, mục đích là để bạn có thể so sánh file đã tải về với file gốc xem có xuất hiện lỗi gì không.

Mã hóa dạng đối xứng (symmetric key encryption)

Chúng ta hãy cùng bắt đầu đi tìm hiểu về việc bảo mật có sử dụng khóa. Khóa ở đây sẽ được gọi là "key", nó chính là mấu chốt vô cùng quan trọng để thuật toán có thể nhìn vào và biết đường mã hóa để giải mã các dữ liệu.

Sơ đồ mã hoá dạng đối xứng

Nó cũng giống như cánh cửa nhà của bạn vậy, nếu bạn có chìa khóa thì bạn mới có thể nhanh chóng đi vào trong còn nếu không có khóa thì bạn vẫn có thể phá cửa hay kêu thợ sửa khóa tới giúp nhưng chắc chắn là sẽ tốn rất nhiều thời gian và công sức hơn rồi. Mỗi chìa khóa cho mỗi chiếc ổ khóa trên thế giới này là duy nhất với các đường rãnh không chìa nào giống với chìa nào và key được mã hóa cũng tương tự như vậy.

Ở phương pháp mã hóa đối xứng này thì chìa khóa để mã hóa và giải mã là giống như nhau nên người ta mới gọi nó là đối xứng và trong tiếng Anh có tên là symmetric. Theo một số tài liệu nghiên cứu khác thì mã hóa đối xứng là giải pháp đang được sử dụng phổ biến nhất hiện nay.

Giả sử bạn đang cần mã hóa một tập tin để gửi cho người khác thì quy trình sẽ được diễn ra như sau:

Bạn cần sử dụng một thuật toán mã hóa và khóa của mình để mã hóa file (cách tạo khóa tạm thời chúng ta không cần bàn đến, chủ yếu là dùng các giải thuật có tính ngẫu nhiên).

Bằng 1 cách nào đó, chúng ta sẽ giao cho người cần nhận file một khóa giống với mình, có thể là giao trước hoặc ngay sau khi mã xong hóa tập tin đều được.

Khi người đó nhận được tập tin, bạn hãy dùng khóa này để giải mã ra tập tin gốc và có thể đọc được nó 1 cách bình thường.

Vấn đề chính ở đây đó là bạn phải làm sao để chuyển khóa cho người nhận một cách an toàn nhất. Nếu khóa này bị lộ ra thì bất kỳ ai nắm nó trong tay cũng có thể xài thuật toán nói trên để giải ra mã của tập tin 1 cách dễ dàng, như vậy thì tính bảo mật sẽ không còn phát huy tác dụng nữa.

Ngày nay người ta thường xài password như là 1 dạng chìa khóa và bằng cách này bạn có thể nhanh chóng nhắn cho người nhận cùng 1 đoạn password đó để dùng làm khóa giải mã.

Các thuật toán mã hóa thường thấy hiện nay là DES và AES. Trong đó, AES là phổ biến nhất trên thế giới và nó được dùng để thay thế cho DES vốn đã xuất hiện từ những năm 1977. Hiện nay có rất nhiều cơ quan chính phủ trên thế giới quy định các tài liệu khi được gửi qua mạng phải sử dụng thuật toán AES để đảm bảo tính an toàn.

Thuật toán AES có thể dùng trong nhiều kích thước ô nhớ khác nhau để mã hóa thông tin và dữ liệu, thường thấy nhất sẽ là 128-bit và 256-bit, có một số có thể lên tới 512-bit và 1024-bit. Kích thước của ô nhớ càng lớn thì sẽ càng khó phá mã hơn bù lại việc giải mã và mã hóa cũng cần sử dụng nhiều năng lực xử lý hơn.

Hiện chế độ mã hóa mặc định của hệ điều hành Android 5.0 đang xài là AES 128-bit. Điều này có nghĩa là mỗi khi bạn chuẩn bị ghi các dữ liệu xuống bộ nhớ máy thì hệ điều hành sẽ mã hóa nó hoàn thiện rồi mới tiến hành ghi lại.

Tương tự như vậy, mỗi khi OS chuẩn bị đọc các dữ liệu thì Android sẽ phải giải mã trước rồi mới chuyển nó ra ngoài, khi đó thì hình ảnh mới có thể hiện ra được, các tập tin nhạc mới mở được và tài liệu mới có thể đọc được. Bằng cách này, nếu bạn có lỡ làm mất máy thì kẻ cắp cũng không thể xem trộm các dữ liệu của bạn (giả sử khi đó bạn đã khoá màn hình).

Nếu người lấy cắp có gỡ chip nhớ của bạn ra để đọc thì dữ liệu cũng đã bị mã hóa hết. Tất nhiên, hệ điều hành Android cũng xài key dạng symmetric (được tạo ra dựa vào password của bạn) và key đó còn được băm thêm một lần nữa bằng SHA 256-bit để làm tăng tính an toàn.

Cơ chế mã hóa của cả hai hệ điều hành Windows 10 và OS X có phần tương tự nhau, tức là xài AES và xài key tạo ra bằng password để dùng kết hợp thêm với SHA.

Mã hóa dạng bất đối xứng (public key encryption)

Nếu như ở trên thì khóa mã hóa và khóa giải mã đều giống nhau thì với phương pháp bất đối xứng này, hai khóa lại hoàn toàn khác nhau. Để có thể phân biệt được giữa hai khóa thì người ta sẽ gọi khóa mã hóa là public key còn khóa giải mã sẽ là private key.

Quy trình của mã hoá bất đối xứng

Public key đúng như cái tên của nó – mang tính chất "công cộng" và có thể được sử dụng để mã hóa các dữ liệu bởi bất kỳ ai. Tuy nhiên, chỉ những người nào nắm trong tay private key thì mới có khả năng giải mã dữ liệu này để xem mà thôi. Quy trình diễn ra mã hóa bất đối xứng như sau:

Bên nhận tin sẽ tạo ra một cặp public và private key. Người này sẽ giữ lại private key cho riêng mình và cất thật cẩn thận để không ai có thể phát hiện. Trong khi đó, public key thì sẽ được chuyển cho bên gửi (dưới hình thức email, copy vào USB, thẻ nhớ… v.v…) hoặc đăng tải đâu đó lên 1 mạng lưu trữ.

Bên gửi sẽ sử dụng public key để mã hóa các thông tin dữ liệu, sau đó gửi file đã được mã hóa lại cho bên nhận.

Bên nhận lúc này sẽ dùng tới private key đã lưu khi nãy để có thể giải mã dữ liệu và bắt đầu sử dụng. Rất là đơn giản đúng không nào?

Tuy nhiên, có 1 nhược điểm của mã hóa bất đối xứng cần được làm rõ đó là tốc độ giải mã sẽ chậm hơn khá nhiều so với phương thức đối xứng. Tức là chúng ta sẽ phải tốn nhiều năng lực xử lý của CPU hơn cũng như phải chờ lâu hơn, dẫn đến "chi phí" bỏ ra cao hơn. Khoảng thời gian lâu hơn là bao nhiêu thì lại còn tùy vào thuật toán, cách thức mã hóa và key mà bên gửi sử dụng.

Chính vì thế mà hiện tại có rất ít người mã hóa cả một file bằng phương pháp bất đối xứng. Thay vào đó họ sẽ dùng phương pháp bất đối xứng để mã hóa chính key được dùng trong mã hóa đối xứng (hoặc tạo ra key đó bằng cách tổng hợp lại public và private key của bên gửi và bên nhận).

Như đã nói ở trên thì mã hóa đối xứng có nhược điểm là key rất dễ bị lộ và khi bị là coi như xong, vậy thì giờ chúng ta cần mã hóa luôn cái key đó để đảm bảo an toàn và có thể gửi key được thoải mái hơn. Một khi đã giải được mã bất đối xứng để ra key gốc rồi thì bạn có thể tiến hành giải mã thêm lần nữa bằng phương pháp đối xứng để ra được file ban đầu. Một thuật toán mã hóa bất đối xứng thường được dùng hiện nay là RSA.

1.5. Thuật toán RC5

Ở đây mình tìm hiểu về hệ mã RC5. Thuật toán mã hóa RC5 do giáo sư Ronald Rivest của đạo học MIT công bố vào tháng 12 năm 1984. Đây là thuật toán mã hóa theo khóa bí mật. Ngay từ khi được giới thiệu RC5 được quan tâm rất nhiều do tính an toàn của nó. Ngày nay truyền dữ liệu thông qua một kênh yêu cầu bảo mật hơn. An ninh đạt được tầm quan trọng hơn chỉ đơn giản là truyền. Đảm bảo truyền yêu cầu giải thuật mã hóa. Các yêu cầu thực hiện phần cứng của các thuật toán tiêu thụ điện năng ít hơn, phân bổ nguồn lực, tái cấu hình, và kiến trúc hiệu quả và hiệu quả chi phí. Mã hóa RC5 có yêu cầu công suất thấp và độ phức tạp thấp và độ trễ thấp, độ xử lý nhanh được ứng dụng nhiều trong giao dịch mạng và thương mại điện tử

CHƯƠNG II. THUẬT TOÁN

2.1. Định nghĩa các giá trị

RC5 được xác định là RC5-w/b/r trong đó:

+w : kích thước khối cần được mã hóa (giá trị chuẩn là 32 bit, ngoài ra ta có thể chọn 16 hay 64 bit).

+r : số vòng lặp (giá trị từ 0,1, ,255)

+b : chiều dài khóa theo byte (0 đến 255)

Các giá trị thường dùng là : w = 32, r = 20, còn chiều dài khóa có thể 16, 24, hay 32 byte.

Đối với tất cả các biến, các thao tác RC5-w-r-b trên khối w-bit sử dụng các toán tử cơ bản sau:

a + b: phép cộng module 2w

a – b: phép trừ module 2w

a xor b: phép toán xor

a <<< b: phép toán quay trái a sang trái ít nhất log 2w bit của b

Trong thuật toán RC5 quá trình mã hóa và giải mã đều cần qua một quá trình quan trọng là quá trình mở rộng khóa.

2.2. Mở rộng khóa

Để tăng độ an toàn cũng như việc bảo vệ khóa bí mật cho người dùng. Việc mở rộng khóa là một chiều nên không thể suy ngược lại giá trị của khóa K khi biết được các giá trị của khóa mở rộng. Đây cũng chính là một đặc điểm nổi bật của thuật toán RC5.

Thuật toán mở rộng cho khóa K của người sử dụng thành một tập gồm 2(r+1) các khóa trung gian. Các khóa trung gian này được điền vào một bảng khóa mở rộng S. Do vậy, S là một bảng của t = 2(r+1) các giá trị nhị phân ngẫu nhiên được quyết định bởi khóa K. Nó sử dụng hai hằng số lý tưởng được định nghĩa:

Pw = Odd ((e - 2)2w)
Qw = Odd ((Ø- 1)2w

Trong đó:
e = 2.178281828459 (dựa trên số logarithms tự nhiên)

Ø = 1.618033988749 (tỉ lệ vàng)
Odd (x) là số nguyên lẻ gần x nhất
Một số giá trị khác :
t = 2(r + 1) : số phần tử của bảng khóa mở rộng S.
u = w/8 : u là số lượng các byte của khối w
c = b/u

Quá trình mở rộng khóa bao gồm các bước sau:

+Bước 1:

Chép khóa bí mật K[0, ,b-1] vào mảng L[0, ,c-1]. Thao tác này sử dụng u byte liên tục nhau của khóa K để điền vào cho L theo thứ tự từ byte thấp đến byte cao. Các byte còn lại trong L được điền vào giá trị 0. Trong trường hợp b = c = 0, chúng ta sẽ đặt c về 1 và L[0] về 0.

+ Bước 2:

Khởi tạo mảng S với một mẫu bit ngẫu nhiên đặc biệt, bằng cách dùng một phép tính số học module 2w được quyết định bởi hằng số lý tưởng PW và Qw.

S[0] = Pw
For i = 1 to t - 1 do
S[i] = S[i-1] + Qw

+ Bước 3 :
Trộn khóa bí mật của người sử dụng vào mảng L và S.
A = B = 0
i = j = 0
v = 3 * max{c,t}
For s=1 to v do {
A = S[i] = (S[i] + A + B) <<<3
B = L[j] = (L[j] + A + B) <<< (A + B)
i = (i + 1) mod (t)
j = (j + 1) mod (c)
}
Lưu ý rằng: hàm mở rộng khóa là một chiều, do vậy không dễ dàng tìm ra khóa K từ S.

Thuật toán mở rộng :
Input : khóa b được nạp và mảng c phần tử L[0, ,c-1]

Số vòng lặp r
Output : mảng khóa S[0, ,2r + 1]
S[0] = Pw
For i = 1 to t - 1 do
S[i] = S[i - 1] + Qw
A = B = 0
i = j = 0
V = 3 * max {c, t}
For s = 1 to v do {
A = S[i] = (S[i] + A + B) <<< 3
B = L[j] = (L[j] + A + B) <<< (A + B)

i = (i + 1) mod (t)
j = (j + 1) mod (c)
}

2.3. Quá trình mã hóa

Thuật toán sẽ mỗi lần mã hóa trên hai khối w bit, giả sử là A và B. Và sau quá trình mã hóa sẽ cho ra hai khối đã được mã hóa A' và B'.

Ban đầu A sẽ được cộng với giá trị khóa mở rộng S[0] và B sẽ được cộng với S[1]. Sau đó quá trình mã hóa sẽ thực hiện biến đổi A dựa vào giá trị của B bằng các phép toán Xor và quay tròn trái. Tiếp tục giá trị này sẽ được cộng tiếp với giá trị khóa mở rộng S[2]. Kết quả này được dùng để tiếp tục biến đổi giá trị của B giống như trên.

Toàn bộ quá trình này sẽ được thực hiện r lần. Kết quả cuối cùng ở bước r sẽ là giá trị đã được mã hóa A', B'.

Quá trình mã hóa có thể được minh họa như sau :

Thuật toán mã hóa:

Input : giá trị gốc được lưu trữ trong hai khối w-bit A, B

Số vòng lặp r

w-bit khóa vòng lặp S[0, ,2*r + 1]

Output : giá trị mã được lưu trong hai khối w-bit A', B'

A = A + S[0]

B = B + S[1]

For i = 1 to r do {

A = ((A XOR B) <<< B) + S[2i]

B = ((B XOR A) <<< A) + S[2i + 1]

}

A' = A

B' = B

Thuật toán giải mã :

Quá trình giải mã chính là quá trình đi ngược lại quá trình mã hóa để có được cái giá trị gốc.

Thuật toán giải mã như sau :

Input : giá trị mã được lưu trữ trong hai khối w-bit A', B'

Số vòng lặp r

w-bit khóa vòng lặp S[0, ,2r + 1]

Output : giá trị giải mã được lưu trong hai khối w-bit A, B

For i = r downto 1 do {

B' = ((B' - S[2i + 1]) >>> A') XOR A'

A' = ((A' - S[2i]) >>> B' XOR B'

}

B = B' - S[1]

A = A' - S[0]

2.4. Đánh giá

Thám mã RC5 :

+ Theo kết quả đánh giá độ an toàn của các thuật toán thì RC5 với 12 vòng lặp và mã hóa khối 64-bit thì cung cấp độ an toàn tương đương với thuật toán DES khi thử với phương pháp giả mã, 2 44 cho RC5 và 2 43 DES.

Bảng mô tả số thao tác cần thực hiện để thám mã RC5 mã hóa 64 bit. Số vòng lặp 4 6 8 10 12 14 16 18.

Thám mã Differential (với thông tin nguồn được chọn)

2
7
2
16
2
28
2
36
2
44
2
52
2
61
>

+ Khi số vòng lặp lên đến 18 thì việc thám mã trên lý thuyết là không thể thực hiện được (do đòi hỏi khoảng 2 128 thao tác cho khối 64 bit). Do việc tăng thêm số vòng lặp là tăng thêm độ an toàn cho RC5. Người ta nhận xét rằng RC5 với 16 vòng lặp và mã hóa khối 64 bit có thể cung cấp độ an toàn rất tốt để chống lại các thuật toán thám mã.

Ưu điểm :

+ RC5 là một thuật toán mã hóa khối với tốc độ nhanh được thiết kế cho việc sử dụng dễ dàng cho cả phần cứng lẫn phần mềm.

+ RC5 là một thuật toán được tham số hóa với : một biến mô tả kích thước khối, một biến cho số vòng quay, và một cho chiều dài khóa.

+ RC5 thì rất đơn giản: cơ chế mã hóa dựa trên ba toán tử chính : cộng, exclusive- or và quay. Vì thế, RC5 dễ cài đặt và phân tích hơn các thuật toán mã hóa khối khác.

+ Một đặc điểm nỗi bật khác của RC5 là các thao tác quay sử dụng chặt chẽ các dữ liệu phụ thuộc với nhau nhằm tránh được các phép thám mã tuyến tính và vi phân.

+ Cơ chế mở rộng khóa của RC5 là một chiều. Do vậy các hacker khó có thể phục hồi lại khóa chính ngay cả khi đã xác định được bộ khóa mở rộng.

+ Mỗi quá trình mã hóa và giải mã của RC5 được thực hiện trên hai khối w bit do vậy có thể tăng tốc độ mã hóa.

Khuyết điểm :

Trên thực tế cho đến năm 1998 thì chưa có cách thám mã nào có thể giải mã được RC5. Tuy nhiên một vài nghiên cứu lý thuyết đã cung cấp một vài cách thám mã có thể thực thi. Họ dựa vào đặc điểm là số lượng vòng lặp trong RC5 thì không phụ thuộc vào tất cả các bit trong một khối. Bên cạnh đó RC5 được thiết kế rất đơn giản do cơ chế mã hóa chỉ dựa vào các phép toán cộng, exclusive-or và quay.

CHƯƠNG III. TỔNG QUAN VỀ CHƯƠNG TRÌNH

Phần code của chương trình

```
from flask import Flask, render_template, request, redirect, jsonify, flash, url_for, send_from_directory

from werkzeug.utils import secure_filename

import os

import webbrowser

from RC5C import RC5C


UPLOAD_FOLDER = './uploads'

ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}

HOST = ""

app = Flask(__name__)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


def allowed_file(filename):

    return '.' in filename and \

        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


rc5 = 0

currentFile = ''


@app.route('/')

def index():

    global HOST

    HOST = request.host_url

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

```
@app.route('/key', methods=['POST'])
def key():
    global rc5
    key = request.form['key'].encode()
    w = int(request.form['w'])
    r = int(request.form['r'])
    rc5 = RC5C(key, w, r)
    return jsonify({"code": 0, "key": str(rc5.key)})


@app.route('/encrypt', methods=['POST'])
def encrypt():
    global rc5
    try:
        text = request.form.get('text')

        encryptText = rc5.encrypt(text.encode('utf-8'))

        string = ""

        t = [text[i:i+rc5.blockSize*2] for i in range(0, len(text), rc5.blockSize*2)]
        for i, j in zip(t, encryptText):
            if len(i) < (rc5.blockSize*2):
                i += " "*(rc5.blockSize*2 - len(i))
            string += ">> {}\t--> {}\n".format(i, j.hex())
        string += ">>\n>> encrypt: {}\n".format(b''.join(encryptText).hex())
        return jsonify({"code": 0, "data": string, "encryptCode": b''.join(encryptText).hex() })
```

```
except:
        return jsonify({"code": 1, "data": ">> Please assign key before encrypting\n"})
@app.route('/decrypt', methods=['POST'])
def decrypt():
    global rc5
    try:
        text = request.form.get('text')
        decryptText = rc5.decrypt(bytes.fromhex(text))
        utf8String = str(b''.join(decryptText), 'utf-8')
        string = ""
        b = [text[i:i+rc5.blockSize*4] for i in range(0, len(text), rc5.blockSize*2)]
        t = [utf8String[i:i+rc5.blockSize*2] for i in range(0,len(utf8String),rc5.blockSize*2)]
        for i, j in zip(b, t):
            if len(i) < (rc5.blockSize*2):
                i += " "*(rc5.blockSize*2 - len(i))
            string += ">> {}\t--> {}\n".format(i, j)
        string += ">>\n>> decrypt: {}\n".format(utf8String)
        return jsonify({"code": 0, "data": string})
    except:
        return jsonify({"code": 1, "data": ">> Please enter hex string\n", "decryptCode":
utf8String})
@app.route('/readFile', methods=['POST'])
def readFile():
    global currentFile
    if request.method == 'POST':
        if 'file' not in request.files:
            flash('No file part')
```

```
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            currentFile = "./{}".format(url_for('uploaded_file', filename=filename))
            with open(currentFile,'rb') as input:
                return jsonify({"code": 0, "data": ">> File content with hex: {}\n>> Go to {}{} to view\n".format(input.read(-1).hex(), HOST,format(url_for('uploaded_file', filename=filename)))})
    return


@app.route('/encryptFile', methods=['POST'])
def encryptFile():
    global currentFile
    path = currentFile.rsplit('.', 1)[1].lower()
    outputPath = 'uploads/out.{}'.format(path)
    try:
        with open(currentFile,'rb') as input, open("./{}".format(outputPath),'wb') as output:
            text, encryptOut = encryptFile(input, output)
            currentFile = "./{}".format(outputPath)
            return jsonify({"code": 0, "data": ">> File encrypt: {}\n>> Go to {}{} to view\n".format(b''.join(encryptOut).hex(),HOST,outputPath), "url": "{}{}".format(HOST,outputPath)})
    except:
```

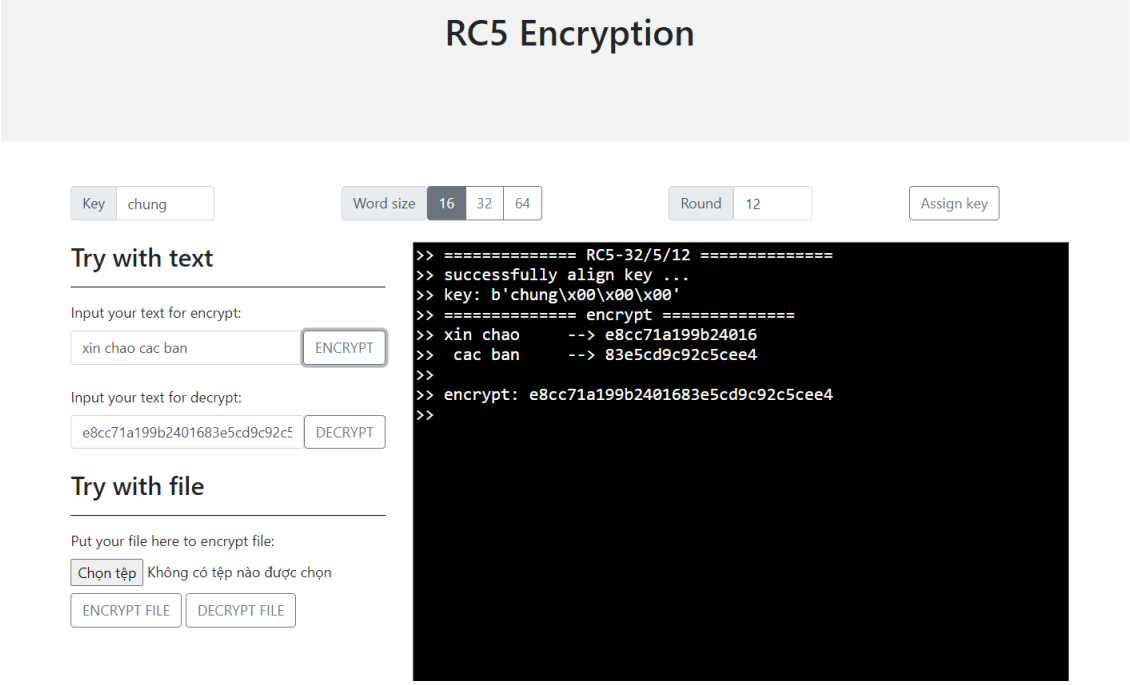
```
        return jsonify({"code": 1, "data": ">> Please assign key before encrypting\n"})


@app.route('/decryptFile', methods=['POST'])
def decryptFile():
    global currentFile
    path = currentFile.rsplit('.', 1)[1].lower()
    outputPath = 'uploads/decrypt.{}'.format(path)
    try:
        with open(currentFile,'rb') as input, open('./{}'.format(outputPath),'wb') as output:
            text, decryptOut = decryptFile(input, output)
            return jsonify({"code": 0, "data": ">> File decrypt: {}\n>> Go to {}{} to
view\n".format(b".join(decryptOut).hex(),HOST, outputPath), "url":
"{}{}".format(HOST,outputPath)})
    except:
        return jsonify({"code": 1, "data": ">> Please assign key before decrypting\n"})


@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'],
                               filename)


def encryptFile(input, output):
    global rc5
    print(rc5)
    text = input.read(-1)
    encryptOut = rc5.encrypt(text)
    output.write(b".join(encryptOut))
```

```
    return text, encryptOut


def decryptFile(input, output):
    global rc5
    text = input.read(-1)
    decryptOut = rc5.decrypt(text)
    output.write(b''.join(decryptOut))
    return text, decryptOut


if __name__ == "__main__":
    app.run(debug=True)
```

Kết quả

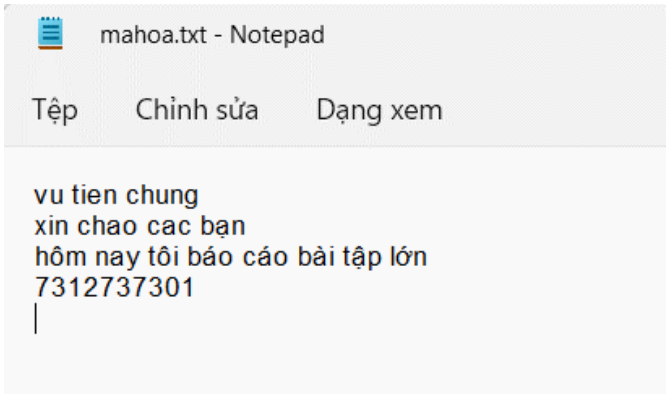
- Mã hóa một chuỗi text:

Đây là kết quả sau khi mã hóa:

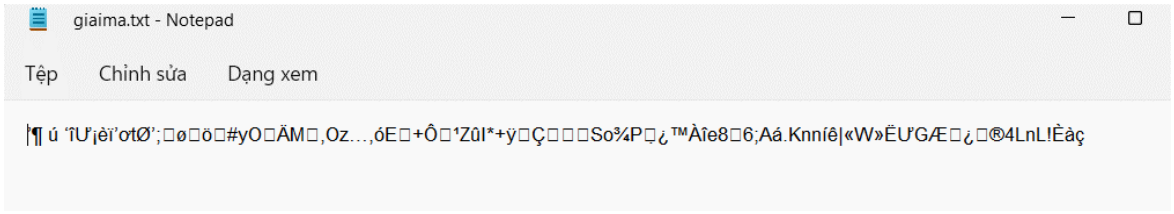


Giải mã thì ta bấm nút Decrypt trên giao diện chương trình.

- Mã hóa file:
  - File cần mã hóa:



  - Sau khi mã hóa bằng chương trình:



  - Giải mã file vừa rồi thì tao và chương trình và thao tác.

KẾT LUẬN

Đã chạy được chương trình.

Chương trình đã đáp ứng được nhu cầu cơ bản của đề tài.

CÁC TƯ LIỆU THAM KHẢO

https://www.youtube.com/watch?v=snmYELRQvME

https://vi.wikipedia.org/

https://github.com/

https://stackoverflow.com/