BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: AN TOÀN BẢO MẬT THÔNG TIN

Đề tài 39: Trình bày các phương pháp mã hóa cổ điển, xây dựng chương trình mô phỏng thuật toán Affine

Sinh viên thực hiện	Lớp	Khóa
Nguyễn Trí Dũng	DCCNTT 13.10.16	K13
Nguyễn Trung Chính	DCCNTT 13.10.16	K13
Trần Văn Nam	DCCNTT 13.10.16	K13
Vũ Văn Phong	DCCNTT 13.10.16	K13
Đổng Trung Đức	DCCNTT 13.10.16	K13

Bắc Ninh, năm 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: AN TOÀN BẢO MẬT THÔNG TIN

Đề tài 39: Trình bày các phương pháp mã hóa cổ điển, xây dựng chương trình mô phỏng thuật toán Affine

STT	Sinh viên thực hiện	Mã sinh viên	Điểm bằng số	Điểm bằng chữ
1	Nguyễn Trí Dũng	20223155		
2	Nguyễn Trung Chính	20222999		
3	Trần Văn Nam	20222996		
4	Vũ Văn Phong	20222998		
5	Đồng Trung Đức	20222877		

CÁN BỘ CHẤM 1
(Ký và ghi rõ họ tên)

CÁN BỘ CHẤM 2
(Ký và ghi rõ họ tên)

LỜI MỞ ĐẦU

Mật mã học là một ngành khoa học về mã hóa dữ liệu nhằm bảo mật thông tin. Mã hóa dữ liệu là một quá trình mà các dữ liệu dạng văn bản gốc được chuyển thành văn bản mật mã để làm nó không thể đọc được. Ngày nay, để đảm bảo sự an toàn và bí mật của các thông tin quan trọng, nhạy cảm, vấn đề mã hóa dữ liệu ngày càng trở nên cấp thiết và được nhiều người quan tâm.

Có nhiều phương pháp mã hóa dữ liệu được đưa ra. Vậy làm thế nào để đánh giá được một phương pháp mã hóa nào là tốt? Có nhiều phương pháp đánh giá nhưng phương pháp tốt nhất và trực quan nhất là phương pháp phân tích trực tiếp bản mã khi không có khóa mã trong tay mà người ta thường gọi là thám mã (Cryptanalysis).

Có thể chia các phương pháp mã hóa dữ liệu thành hai hệ mật mã cơ bản: Hệ mật mã cổ điển với các hệ mật mã như hệ mã Caesar, Affine, thay thế, Vigenere… và hệ mật mã hiện đại với hệ mã đối xứng (DES - Data Encryption Standard) và hệ mã bất đối xứng (RSA – Rivest, Shamir, Adleman).

Với mỗi hệ mật mã ta có những phương pháp thám mã tương ứng. Hệ mật mã cổ điển đến nay không còn được sử dụng nhiều nhưng chính hệ mật mã này là nền tảng cho sự phát triển của mật mã hiện đại. Việc nghiên cứu hệ mật mã cổ điển có ý nghĩa quan trọng hỗ trợ việc nghiên cứu các hệ thống mã hiện đại, vì vậy nhóm chúng em lựa chọn nghiên cứu hệ mã cổ điển.

MỤC LỤC

DANH MỤC HÌNH ẢNH

CHƯƠNG I: GIỚI THIỆU CÁC MÃ HÓA CỔ ĐIỂN

1.1 Affine Cipher

Affine Cipher là một phương pháp mã hóa thay thế dựa trên sự kết hợp giữa phép nhân và phép cộng theo mô-đun. Mã hóa Affine sử dụng hai tham số (thường gọi là khóa) để mã hóa mỗi ký tự trong văn bản gốc.

1.1.1 Nguồn gốc

a, Người phát minh

Affine Cipher là một hệ thống mã hóa toán học, và không rõ ai là người phát minh cụ thể. Tuy nhiên, nó dựa trên các nguyên lý toán học đã được biết đến từ lâu trong lý thuyết số.

b, Ứng dụng lịch sử

Mã hóa Affine từng được sử dụng trong các hệ thống mã hóa cổ điển để bảo mật các thông điệp. Tuy nhiên, với sự xuất hiện của các phương pháp giải mã hiện đại, Affine Cipher không còn được dùng phổ biến do tính bảo mật tương đối thấp.

1.1.2 Khái niệm cơ bản

a, Bảng chữ cái

Affine Cipher sử dụng một bảng chữ cái tiêu chuẩn gồm 26 chữ cái tiếng Anh (A-Z). Các ký tự trong bảng chữ cái được ánh xạ vào các số nguyên từ 0 đến 25.

b, Cấu trúc

Phương trình mã hóa trong Affine Cipher là:
$E(x)=(a*x+b) \bmod 26$
Trong đó:

E(x) là ký tự được mã hóa.

a và b là các khóa mã hóa (với điều kiện UCLN(a,26)=1).

x là vị trí của ký tự trong bảng chữ cái (ví dụ: A=0, B=1,... Z=25).

c, Quy trình mã hóa

Chọn khóa: Người mã hóa chọn hai số nguyên aaa và bbb, với điều kiện aaa và 26 phải nguyên tố cùng nhau (tức là UCLN(a,26)=1UCLN(a, 26) = 1UCLN(a,26)=1).

Chuyển đổi ký tự thành số: Mỗi ký tự trong văn bản gốc được ánh xạ vào một số từ 0 đến 25.

Áp dụng công thức mã hóa: Áp dụng công thức $E(x)=(a\times x+b) \mod 26$ để mã hóa từng ký tự.

Chuyển đổi số trở lại ký tự: Các số sau khi mã hóa được chuyển đổi lại thành ký tự trong bảng chữ cái.

Quy trình giải mã ngược lại với công thức: $D(x)=a-1\times(x-b) \mod 26$

1.1.3 Tính bảo mật

a, Ưu điểm

Đơn giản và dễ triển khai: Phương pháp mã hóa Affine có cấu trúc đơn giản, dễ hiểu và dễ thực hiện trong các ứng dụng mã hóa cổ điển.

Mã hóa đa biến: Sự kết hợp giữa phép nhân và phép cộng giúp Affine Cipher mạnh mẽ hơn so với các phương pháp mã hóa chỉ dựa trên phép dịch chuyển, như Caesar Cipher.

b, Nhược điểm

Dễ bị tấn công phân tích tần số: Do chỉ có 26 ký tự, Affine Cipher vẫn dễ bị phá mã thông qua các kỹ thuật phân tích tần số. Kẻ tấn công có thể phân tích sự xuất hiện thường xuyên của các ký tự để xác định khóa.

Số lượng khóa hạn chế: Chỉ có một số giá trị nhất định của aaa và bbb có thể sử dụng, vì aaa phải nguyên tố cùng 26, nên số lượng khóa có thể sử dụng bị giới hạn.
Affine Cipher cung cấp một mức bảo mật cơ bản, nhưng không được khuyến nghị sử dụng trong các ứng dụng mã hóa hiện đại.

1.2 Vigenère Cipher

Vigenère Cipher là một phương pháp mã hóa dựa trên việc sử dụng bảng chữ cái Vigenère cùng với một từ khóa để mã hóa văn bản. Đây là một trong những hệ mã mật thay thế nổi tiếng nhất trong lịch sử mật mã học. Ưu điểm lớn của Vigenère Cipher là khả năng kháng lại các cuộc tấn công bằng phân tích tần suất, vốn là điểm yếu của các hệ mã đơn bảng như Caesar Cipher. Nhờ vào việc sử dụng một từ khóa để thay đổi cách mã hóa

của từng ký tự, Vigenère Cipher cung cấp một mức độ bảo mật cao hơn so với các phương pháp mã hóa đơn giản.

1.2.1 Nguồn gốc

a, Người phát minh

Vigenère Cipher được đặt theo tên của Blaise de Vigenère (1523-1596), một nhà ngoại giao người Pháp. Ông đã phát triển và hoàn thiện phương pháp mã hóa này, giới thiệu nó vào năm 1586. Tuy nhiên, cần lưu ý rằng một phiên bản sớm hơn của hệ mã này đã được mô tả bởi nhà toán học người Ý Giovan Battista Bellaso từ năm 1553. Mặc dù Vigenère là người phổ biến rộng rãi phương pháp này, Bellaso mới là người đầu tiên đưa ra ý tưởng về việc sử dụng từ khóa để mã hóa văn bản.

b, Lịch sử phát triển

Năm 1553: Giovan Battista Bellaso mô tả phương pháp mã hóa sử dụng từ khóa trong cuốn sách của mình. Phương pháp này sau đó được hoàn thiện bởi Blaise de Vigenère và trở nên nổi tiếng với tên gọi "Vigenère Cipher."

Năm 1586: Blaise de Vigenère phổ biến phương pháp này, từ đó nó được đặt theo tên ông.

c, Ứng dụng lịch sử

Vigenère Cipher đã được sử dụng rộng rãi bởi các nhà ngoại giao và quân đội trong nhiều thế kỷ. Phương pháp này từng được coi là "mã không thể phá vỡ" (le chiffre indéchiffrable) cho đến khi bị Charles Babbage và Friedrich Kasiski phá giải vào thế kỷ 19.

1.2.2 Khái niệm cơ bản

a, Bảng chữ cái

Bảng Vigenère là một ma trận kích thước 26x26, trong đó mỗi hàng là một dịch chuyển của bảng chữ cái chuẩn (A-Z). Bảng này được sử dụng để mã hóa văn bản bằng cách thay thế mỗi ký tự của văn bản bằng một ký tự khác dựa trên từ khóa. Mỗi hàng trong bảng Vigenère đại diện cho một bảng chữ cái được dịch chuyển theo một số vị trí nhất định so với bảng chữ cái chuẩn.

b, Cấu trúc

Mỗi hàng trong bảng Vigenère là một phiên bản dịch chuyển của bảng chữ cái chuẩn. Ví dụ, hàng đầu tiên là bảng chữ cái không thay đổi (A-Z), hàng thứ hai là bảng chữ cái dịch chuyển một ký tự (B-ZA), hàng thứ ba dịch chuyển hai ký tự (C-ZAB).

c, Quy trình mã hóa

- Văn bản gốc và từ khóa được sắp xếp sao cho mỗi ký tự của từ khóa ứng với một ký tự của văn bản gốc. Nếu từ khóa ngắn hơn văn bản, nó sẽ được lặp lại liên tục.
- Mỗi ký tự trong văn bản gốc được thay thế bằng một ký tự từ bảng Vigenère, tương ứng với vị trí của ký tự đó trong từ khóa.
- Công thức mã hóa: (Ci = (Pi + Ki) mod 26)
- Ví dụ:
    + Văn bản gốc (Plaintext): ATTACKATDAWN
    + Từ khóa (Key): LEMON
    + Mã hóa (Ciphertext): LXFOPVEFRNHR
- Quá trình mã hóa:
    + A (vị trí 0 trong bảng chữ cái) + L (vị trí 11) = L
    + T (vị trí 19) + E (vị trí 4) = X
    + T (vị trí 19) + M (vị trí 12) = F
    + Và tiếp tục lặp lại như vậy.
- Quy trình giải mã: Giải mã Vigenère Cipher ngược lại với quy trình mã hóa, bằng cách sử dụng từ khóa để truy xuất ký tự gốc từ bảng chữ cái Vigenère.

1.2.3 Tính bảo mật

a, Ưu điểm

Khó bị phá mã hơn so với mã hóa Caesar: Vigenère Cipher sử dụng nhiều bảng chữ cái thay vì chỉ một bảng như trong mã hóa Caesar, làm cho việc phân tích tần suất trở nên khó khăn hơn. Điều này giúp tăng cường bảo mật so với mã hóa Caesar, vốn chỉ sử dụng một bảng chữ cái duy nhất.

Dễ triển khai: Thuật toán mã hóa và giải mã của Vigenère khá đơn giản. Việc lặp lại từ khóa và mã hóa hoặc giải mã từng ký tự chỉ cần một số vòng lặp, làm cho quá trình triển khai trở nên dễ dàng.

Tăng cường bảo mật: Khi sử dụng từ khóa dài và phức tạp, mã hóa Vigenère có thể cung cấp mức độ bảo mật cao hơn. Từ khóa càng dài và phức tạp, việc phá mã càng

trở nên khó khăn, nhờ vào việc mỗi ký tự của văn bản gốc được mã hóa bằng nhiều dịch chuyển khác nhau.

b, Nhược điểm

Dễ bị phá mã nếu từ khóa ngắn: Nếu từ khóa ngắn hoặc dễ đoán, mã hóa Vigenère có thể bị phá mã bằng phương pháp phân tích tần suất. Ví dụ, nếu từ khóa là "A", mã hóa Vigenère sẽ trở thành mã hóa Caesar, dễ bị phá vỡ.

Không an toàn trước các phương pháp hiện đại: Với sự phát triển của các phương pháp mã hóa hiện đại như AES và RSA, mã hóa Vigenère không còn được coi là an toàn. Các phương pháp hiện đại sử dụng các thuật toán phức tạp hơn và cung cấp mức độ bảo mật cao hơn.

Phụ thuộc vào từ khóa: Bảo mật của mã hóa Vigenère phụ thuộc rất nhiều vào độ dài và độ phức tạp của từ khóa. Nếu từ khóa bị lộ hoặc quá đơn giản, toàn bộ hệ thống mã hóa có thể bị phá vỡ.

1.3 Caesar Cipher

Caesar Cipher, còn được gọi là Caesar Shift, là một phương pháp mã hóa cổ điển, trong đó mỗi ký tự trong văn bản gốc được thay thế bằng một ký tự khác trong bảng chữ cái theo một khoảng cách cố định.

1.3.1 Nguồn gốc

a, Người phát minh

Caesar Cipher được cho là phát minh bởi Julius Caesar, vị hoàng đế La Mã. Ông sử dụng mã hóa này để truyền tải các thông điệp quân sự bí mật cho cấp dưới.

b, Lịch sử phát triển

Sau khi Julius Caesar sử dụng phương pháp này, nó được áp dụng rộng rãi trong quân đội và các cuộc trao đổi bí mật khác. Tuy nhiên, sau này với sự phát triển của các phương pháp mã hóa phức tạp hơn, Caesar Cipher dần mất đi giá trị chiến lược trong quân sự và ngoại giao.

c, Ứng dụng lịch sử

Caesar Cipher đã được sử dụng trong nhiều giai đoạn lịch sử khác nhau, đặc biệt là trong thời kỳ đế chế La Mã. Ngoài ra, nó còn được sử dụng trong các phương pháp mã hóa đơn giản để bảo vệ thông tin ở các giai đoạn sau.

1.3.2 Khái niệm cơ bản

a, Bảng chữ cái

Caesar Cipher hoạt động trên cơ sở bảng chữ cái của ngôn ngữ đang sử dụng (ví dụ: bảng chữ cái tiếng Anh gồm 26 ký tự). Mỗi ký tự trong bảng chữ cái được thay thế bằng một ký tự khác cách nó một số bước nhất định.

b, Cấu trúc

Phương pháp này sử dụng một khóa mã hóa duy nhất là số bước dịch chuyển (shift). Ví dụ, với dịch chuyển là 3, chữ "A" sẽ được thay thế bằng chữ "D", "B" sẽ thành "E", và cứ tiếp tục như vậy.

c, Quy trình mã hóa

Bước 1: Chọn một số bước dịch chuyển (key) từ 1 đến số ký tự trong bảng chữ cái (ví dụ: từ 1 đến 25 đối với tiếng Anh).

Bước 2: Với mỗi ký tự trong văn bản gốc, tìm vị trí của nó trong bảng chữ cái.

Bước 3: Dịch chuyển vị trí ký tự đó theo số bước đã chọn để tìm ra ký tự mã hóa tương ứng.

Bước 4: Thay thế ký tự gốc bằng ký tự mã hóa và hoàn thành văn bản mã hóa.

1.3.3 Tính bảo mật

a, Ưu điểm

Đơn giản và dễ sử dụng: Caesar Cipher rất dễ hiểu và dễ triển khai, không yêu cầu tính toán phức tạp.

Nhanh chóng: Quá trình mã hóa và giải mã diễn ra nhanh chóng vì chỉ cần dịch chuyển ký tự.

Có thể sử dụng trong môi trường hạn chế về tài nguyên: Trong các môi trường mà việc triển khai các thuật toán mã hóa phức tạp không khả thi, Caesar Cipher có thể là một giải pháp.

b, Nhược điểm

Dễ bị phá mã: Vì chỉ có một số hữu hạn các dịch chuyển (tối đa là 25 trong bảng chữ cái tiếng Anh), kẻ tấn công có thể dễ dàng thử tất cả các khóa và phá mã bằng cách tấn công vét cạn (brute-force).

Thiếu độ phức tạp: Không có sự thay đổi hoặc biến hóa trong cấu trúc của văn bản mã hóa, nên Caesar Cipher không thể bảo vệ tốt trước các phương pháp phân tích tần số.

Không thích hợp cho thông tin nhạy cảm: Với khả năng bảo mật yếu, phương pháp này không thể được sử dụng để bảo vệ các thông tin quan trọng trong thời đại hiện đại.

1.4 Playfair Cipher

Playfair Cipher là một phương pháp mã hóa dựa trên sự thay thế các cặp ký tự trong văn bản gốc bằng cách sử dụng một bảng chữ cái 5x5. Nó là một trong những phương pháp mã hóa đa ký tự sớm nhất, thay vì mã hóa từng ký tự riêng lẻ như Caesar Cipher.

1.4.1 Nguồn gốc

a, Người phát minh

Playfair Cipher được phát minh bởi Charles Wheatstone vào năm 1854, nhưng nó được đặt tên theo Baron Lyon Playfair, một người bạn của Wheatstone, người đã giúp phổ biến phương pháp này.

b, Lịch sử phát triển

Playfair Cipher được phát minh trong thời kỳ Victoria ở Anh và được quân đội Anh sử dụng trong Chiến tranh Boer và Thế chiến I. Phương pháp này đã trở thành một trong những hệ thống mã hóa chữ ghép đầu tiên được ứng dụng rộng rãi.

c, Ứng dụng lịch sử

Playfair Cipher đã được quân đội sử dụng để mã hóa thông tin quân sự trong các cuộc chiến tranh. Nó có thể truyền tải thông điệp nhanh chóng và khó bị phá hơn so với các phương pháp mã hóa đơn ký tự, nhưng đã bị thay thế bởi các phương pháp mã hóa hiện đại hơn sau khi các thuật toán giải mã phức tạp hơn ra đời.

1.4.2 Khái niệm cơ bản

a, Bảng chữ cái

Playfair Cipher sử dụng một bảng chữ cái 5x5 để mã hóa. Trong bảng này, các chữ cái từ A đến Z được sắp xếp, với I và J được coi là cùng một ký tự để phù hợp với bảng 5x5.

b, Cấu trúc

Bảng chữ cái 5x5 được tạo ra bằng cách chọn một từ khóa. Các chữ cái trong từ khóa sẽ được điền vào bảng trước, sau đó các chữ cái còn lại sẽ được điền vào cho đến khi bảng hoàn thành.

c, Quy trình mã hóa

Bước 1: Chia văn bản cần mã hóa thành các cặp chữ cái (digraphs). Nếu một cặp chữ có 2 chữ giống nhau (ví dụ: "LL"), chèn một ký tự đệm như "X" giữa chúng. Nếu số chữ lẻ, thêm một chữ "X" vào cuối.

Bước 2: Tìm hai chữ cái của từng cặp trong bảng chữ cái. Nếu hai chữ cái nằm trên cùng một hàng, thay thế chúng bằng chữ cái bên phải (quay vòng lại nếu đến cuối hàng).

Nếu hai chữ cái nằm trên cùng một cột, thay thế chúng bằng chữ cái ở phía dưới (quay vòng lại nếu đến cuối cột).

Nếu hai chữ cái nằm ở vị trí hình chữ nhật, thay thế chúng bằng hai chữ cái ở hai góc đối diện của hình chữ nhật.

Bước 3: Lắp ghép các chữ cái đã được mã hóa thành văn bản mã hóa cuối cùng.

1.4.3 Tính bảo mật

a, Ưu điểm

Mã hóa theo cặp chữ: Playfair Cipher mã hóa theo cặp chữ thay vì từng chữ riêng lẻ, giúp tăng tính phức tạp và bảo mật so với các phương pháp mã hóa đơn ký tự như Caesar Cipher.

Khó phá bằng tấn công phân tích tần số: Vì mã hóa dựa trên các cặp chữ, Playfair Cipher làm giảm tính hiệu quả của các phương pháp tấn công dựa trên tần số ký tự.

b, Nhược điểm

Dễ bị tấn công vét cạn (brute-force): Dù khó bị phá bằng phương pháp tấn công tần số, Playfair Cipher vẫn có thể bị phá mã với các kỹ thuật hiện đại như vét cạn hoặc phân tích mật mã với số lượng văn bản mã hóa lớn.

Không an toàn với lượng thông tin lớn: Khi sử dụng cho lượng văn bản lớn, Playfair Cipher có xu hướng dễ bị khai thác điểm yếu do sự lặp lại của các cặp chữ cái trong thông điệp mã hóa.

Phụ thuộc vào từ khóa: An ninh của phương pháp này phụ thuộc vào việc giữ bí mật từ khóa và quá trình tạo bảng chữ cái mã hóa.

CHƯƠNG II: NGUYÊN LÝ HOẠT ĐỘNG

2.1 Khóa

Hệ mã Affine là một loại mã hóa đối xứng thuộc nhóm mã hóa theo phép dịch tuyến tính. Trong hệ mã này, mỗi ký tự trong bản rõ (plaintext) sẽ được biến đổi thành ký tự mã hóa (ciphertext) bằng một công thức tuyến tính phụ thuộc vào hai tham số: khóa a và b.

2.1.1 Khái niệm khóa

Trong mã hóa Affine, một cặp khóa (key) bao gồm hai số nguyên a và b:

Khóa a: Đây là hệ số nhân trong công thức mã hóa. Nó có vai trò điều chỉnh cách thức mã hóa từng ký tự dựa trên vị trí của chúng trong bảng chữ cái. Điều kiện quan trọng đối với khóa a là nó phải có ước chung lớn nhất (UCLN) với 26 bằng 1, vì 26 là số lượng ký tự trong bảng chữ cái tiếng Anh. Điều này đảm bảo rằng tồn tại một nghịch đảo của a trong phép nhân modulo 26, để quá trình giải mã có thể thực hiện được. Nếu không có nghịch đảo, việc giải mã sẽ không thể hoàn thành.

Khóa b: Đây là hằng số dịch trong công thức mã hóa. Nó được cộng thêm vào kết quả của phép nhân giữa giá trị ký tự và khóa a, sau đó thực hiện phép chia dư (mod 26) để đảm bảo rằng kết quả nằm trong phạm vi 26 ký tự của bảng chữ cái tiếng Anh.

Công thức mã hóa cho ký tự đơn trong hệ mã Affine là: E(x)=(a.x+b) mod26

Trong đó:

x là giá trị số nguyên của ký tự (A = 0, B = 1, ..., Z = 25).

E(x) là ký tự mã hóa.

a là khóa nhân, phải thỏa mãn điều kiện UCLN(a, 26) = 1.

b là khóa dịch, là một số nguyên tùy ý từ 0 đến 25.

Quá trình giải mã được thực hiện với công thức: $D(y) = a^{-1}.(y - b) \bmod 26$

Trong đó:

y là ký tự đã mã hóa.

a^{-1} là nghịch đảo modular của a theo modulo 26, nghĩa là $a^{-1}.a \equiv 1$

2.1.2 Sinh khóa

Việc sinh khóa trong hệ mã Affine phụ thuộc vào hai bước:

Chọn giá trị khóa a:

Để đảm bảo rằng quá trình giải mã có thể thực hiện được, giá trị của a phải được chọn sao cho nó thỏa mãn điều kiện UCLN(a, 26) = 1. Các giá trị hợp lệ của a trong

phạm vi từ 1 đến 25 là: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, và 25. Những giá trị này đều có nghịch đảo modular theo modulo 26.

Chọn giá trị khóa b:

Khóa b là một số nguyên tùy ý từ 0 đến 25. Giá trị của b không cần thỏa mãn điều kiện nào đặc biệt, do đó có 26 giá trị khả thi cho b.

Việc kết hợp giữa các giá trị của a và b cho phép sinh ra nhiều cặp khóa khác nhau. Số lượng khóa khả thi là 12 (số giá trị có thể của a) nhân với 26 (số giá trị có thể của b), tức là có 312 cặp khóa khả thi trong hệ mã Affine. Mỗi cặp khóa sẽ xác định một phương pháp mã hóa và giải mã duy nhất.

2.2 Mã thay thế

Hệ mật này đã được sử dụng hàng trăm năm. Trò chơi đố chữ "cryptogram" trong các bài báo là những ví dụ về MTT.

Trên thực tế MTT cổ thể lấy cả P và C đều là bộ chữ cái tiếng anh, gồm 26 chữ cái. Ta dùng Z26 trong MDV vì các phép mã và giải mã đều là các phép toán đại số. Tuy nhiên, trong MTT, thích hợp hơn là xem phép mã và giải mã như các hoán vị của các kí tự.

Cho P = C = Z26. K chứa mọi hoán vị có thể của 26 kí hiệu 0,1,…..,25. Với mỗi phép hoán vị $\pi \in$ K, ta định nghĩa : $e\pi(x) = \pi(x)$ và $d\pi(y) = \pi\text{-}1(y)$

Trong đó π-1 là hoán vị ngược của π.

Sau đây là một ví dụ về phép hoán vị ngẫu nhiên π tạo nên một hàm mã hoá (cũng như trước, các kí hiệu của bản rõ được viết bằng chữ thường còn các kí hiệu của bản mã là chữ in hoa).

a	b	c	d	e	f	g	h	i	j	k	l	m
X	N	Y	A	H	P	O	G	Z	Q	W	B	T

n	o	p	q	r	s	t	u	v	w	x	y	z
S	F	L	R	C	V	M	U	E	K	J	D	I

Như vậy, $e\pi$ (a) = X, $e\pi$(b) = N,. . . . Hàm giải mã là phép hoán vị ngược. Điều này được thực hiện bằng cách viết hàng thứ hai lên trước rồi sắp xếp theo thứ tự chữ cái. Ta nhận được:

A	B	C	D	E	F	G	H	I	J	K	L	M
d	l	r	y	v	o	H	e	z	x	w	p	t

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	g	f	j	q	n	M	u	s	k	a	c	i

Bởi vậy dπ (A) = d, dπ(B) = l,.. .

Mỗi khoá của MTT là một phép hoán vị của 26 kí tự. Số các hoán vị này là 26!, lớn hơn 4 *1026 là một số rất lớn. Bởi vậy, phép tìm khoá vét cạn không thể thực hiện được, thậm chí bằng máy tính. Tuy nhiên, sau này sẽ thấy rằng MTT có thể dễ dàng bị thám mã bằng các phương pháp khác.

2.3 Mã affine

Một trường hợp đặc biệt của MTT là mã Affine được mô tả dưới đây. trong mã Affine, ta giới hạn chỉ xét các hàm mã có dạng: e(x) = ax + b mod 26 và a,b ϵ Z26

Các hàm này được gọi là các hàm Affine (chú ý rằng khi a = 1, ta có MDV).

Để việc giải mã có thể thực hiện được, yêu cầu cần thiết là hàm Affine phải là đơn ánh. Nói cách khác, với bất kỳ y ϵ Z26, ta muốn có đồng nhất thức sau: ax + b ≡ y (mod 26) phải có nghiệm x duy nhất. Đồng dư thức này tương đương với: ax ≡ y-b (mod 26)

Vì y thay đổi trên Z26 nên y-b cũng thay đổi trên Z26. Bởi vậy, ta chỉ cần nghiên cứu phương trình đồng dư: ax ≡ y (mod 26) (y ϵ Z26).

Ta biết rằng, phương trình này có một nghiệm duy nhất đối với mỗi y khi và chỉ khi UCLN(a,26) = 1 . Trước tiên ta giả sử rằng, UCLN(a,26) = d >1. Khi đó, đồng dư thức ax ≡ 0 (mod 26) sẽ có ít nhất hai nghiệm phân biệt trong Z26 là X = 0 và X = 26/d. Trong trường hợp này, e(x) = ax + b mod 26 không phải là một hàm đơn ánh và bởi vậy nó không thể là hàm mã hoá hợp lệ.

Ví dụ, do UCLN(4,26) = 2 nên 4x +7 không là hàm mã hoá hợp lệ: x và x+13 sẽ mã hoá thành cùng một giá trị đối với bất kì X ϵ Z26.

Ta giả thiết UCLN(a,26) = 1. Giả sử với x1 và x2 nào đó thỏa mãn:

ax1 ≡ ax2 (mod 26)

Khi đó: a(x1 - x2) ≡ 0(mod 26)

Bởi vậy: 26 | a(x1 - x2)

Bây giờ ta sẽ sử dụng một tính chất của phép chia sau: Nếu UCLN(a,b)=l và a | bc thì a | c. Vì 26 | a(xr x2) và UCLN(a,26) = 1 nên ta có:

26 | (x1 - x2)

Tức là : x1 ≡ x2 (mod 26)

Tới đây ta chứng tỏ rằng, nếu UCLN(a,26) = 1 thì một đồng dư thức dạng ax ≡ y (mod 26) chỉ có (nhiều nhất) một nghiệm trong Z26. Do đó , nếu ta cho X thay đổi trên

Z26 thì ax mod 26 sẽ nhận được 26 giá trị khác nhau theo modulo 26 và đồng dư thức ax = y (mod 26) chỉ có một nghiệm y duy nhất.

Không có gì đặc biệt đối với số 26 trong khẳng định này. Bởi vậy, bằng cách tương tự ta có thể chứng minh được.

2.3.1 Định nghĩa 1

Đồng dư thức ax =b mod m chỉ có một nghiệm duy nhất x ϵ Zm với mọi b ϵ Zm khi và chỉ khi UCLN(a,m) = 1.

Vì 26 = 2 * l3 nên các giá trị a ϵ Z26 thoả mãn UCLN(a,26) = 1 là a = 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23 và 25. Tham số b có thể là một phần tử bất kỳ trong Z26. Như vậy, mã Affine có 12x26 = 312 khoá có thể (dĩ nhiên con số này quá nhỏ để bảo đảm an toàn).

Bây giờ ta sẽ xét bài toán chung với modulo m. Ta cần một định nghĩa khác trong lý thuyết số.

Giả sử a ≥ 1 và m ≥ 2 là các số nguyên. UCLN(a,m) = 1 thì ta nói rằng a và m lànguyên tố cùng nhau. Số các số nguyên trong Zm nguyên tố cùng nhau với m thường được kỷ hiệu là ø(m) (hàm này được gọi là hàm Euler).

Một kết quả quan trọng trong lý thuyết số cho ta giá trị của ø(m) theo các thừa số trong phép phân tích theo luỹ thừa các số nguyên tố của m. (Một số nguyên p >1 là số nguyên tố nếu nó không có ước dương nào khác ngoài 1 và p. Mọi số nguyên m >1 có thể phân tích được thành tích của các luỹ thừa các số nguyên tố theo cách duy nhất. Ví dụ 60 = 2 3 * 3 * 5 và 98 = 2 * 7 2).

2.3.2 Định nghĩa 2

Giả sử: $m = \prod Pi$

Trong đó các số nguyên tố pi khác nhau và ei >0 ,1

Định lý này cho thấy rằng, số khóa trong mã Affine trên Zm bằng m*ø(m), trong đó ø(m) được cho theo công thức trên. (Số các phép chọn của b là m và số các phép chọn của a là ø(m) với hàm mã hoá là e(x) = ax + b). Ví dụ, khi m = 60, Ø(60) = 2 * 2 * 4 = 16 và số các khoá trong mã Affine là 960.

Bây giờ ta sẽ xét xem các phép toán giải mã trong mật mã Affine với modulo m = 26. Giả sử UCLN(a,26) = 1. Để giải mã cần giải phương trình đồng dư y =ax+b (mod 26) theo X. Từ thảo luận trên thấy rằng, phương trình này có một nghiệm duy nhất trong Z26. Tuy nhiên ta vẫn chưa biết một phương pháp hữu hiệu để tìm nghiệm. Điều cần thiết ở đây là có một thuật toán hữu hiệu để làm việc đó. Rất may là một số kết quả tiếp sau về số học modulo sẽ cung cấp một thuật toán giải mã hữu hiệu cần tìm.

Giả sử a ϵ Zm . Phần tử nghịch đảo (theo phép nhân) của a là phần tử a-1 ϵ Zm sao cho aa-1 ≡ a-1a ≡1 (mod m).

Bằng các lý luận tương tự như trên, có thể chứng tỏ rằng a có nghịch đảo theo modulo m khi và chỉ khi UCLN(a,m) =1, và nếu nghịch đảo này tồn tại thì nó phải là duy nhất. Ta cũng thấy rằng, nếu b = a-1 thì a = b-1. Nếu p là số nguyên tố thì mọi phần tử khác không của Zp đều có nghịch đảo. Một vành trong đó mọi phần tử đều có nghịch đảo được gọi là một trường.

Trong Z26, chỉ bằng phương pháp thử và sai cũng có thể tìm được các nghịch đảo của các phần tử nguyên tố cùng nhau với 26: 1-1 = 1, 3-1 = 9, 5-1 = 21, 7-1 = 15 , 11-1 = 19, 17-1 =23, 25-1 = 25. (Có thể dễ dàng kiểm chứng lại điều này, ví dụ: 7 * 15 = 105 ≡ 1 mod 26, bởi vậy 7-1 = 15).

Xét phương trình đồng dư y ≡ ax+b (mod 26). Phương trình này tương đương với

ax ≡ y-b (mod 26)

Vì UCLN(a,26) =1 nên a có nghịch đảo theo modulo 26. Nhân cả hai vế của đồng dư thức với a-1 ta có: a-1(ax) ≡ a-1(y-b) (mod 26)

Áp dụng tính kết hợp của phép nhân modulo: a-1(ax) ≡ (a-1a)x ≡ 1x ≡ x.

Kết quả là x ≡ a-1 (y-b) (mod 26). Đây là một công thức tường minh cho x. Như vậy hàm giải mã là: d(y) ≡ a-1(y-b) mod 26

Cho $\mathcal{P} = \mathcal{C} = Z_{26}$ và giả sử

$\mathcal{P} = \{ (a,b) \in Z_{26} \times Z_{26} : UCLN(a,26) = 1 \}$

Với $K = (a,b) \in \mathcal{K}$, ta định nghĩa:

$$e_K(x) = ax + b \bmod 26$$

và

$$d_K(y) = a^{-1}(y-b) \bmod 26,$$

$x,y \in Z_{26}$

Hình 2. 1 Hệ mã Affine

Ví dụ minh họa:

Giả sử K = (3,6). Ta có 3-1 = 9 (vì 3*9 = 27 mod 26 = 1). Hàm mã hóa là: eK(x) = 3x + 6. Và hàm giải mã tương ứng là: dK(y) = 9(y-6) = 9y – 2

Ở đây, tất cả các phép toán đều thực hiện trên Z26. Ta sẽ kiểm tra liệu dK(eK(x)) = x với mọi x ϵ Z26 không? Dùng các tính toán trên Z26, ta có:

dK(eK(x)) = dK(3x + 6)

= 9(3x + 6) – 2

= x + 54 – 2

= x

Ta mã hóa bản rõ “xung”. Trước tiên biến đổi các chữ x, u, n, g thành các thặng dư theo modulo 26, Hoặc vị trí của các chữ trong bảng chữ cái tiếng anh:

a b c d e f g h i j k l m n o p q r s t u v w x y z

Ta được các số tương ứng là: 23, 20, 13, 6. Bây giờ ta sẽ mã hóa :

3*23 + 6 mod 26 = 75 mod 26 = 23.

3*20 + 6 mod 26 = 66 mod 26 = 14.

3*13 + 6 mod 26 = 45 mod 26 = 19.

3*6 + 6 mod 26 = 24 mod 26 = 24.

Vậy 4 kí hiệu của bản mã là 23, 14, 19, 24 tương ứng xâu kí tự là : "**xoty**"

Bên cạnh đó chúng ta sẽ có cách giải mã những ký tự vừa mã hóa ở trên. Đó là:

Việc giải mã sẽ áp dụng thuật toán giải mã dK đã tính toán ở trên :

9*23 – 2 mod 26 = 205 mod 26 = 23.

9*14 – 2 mod 26 = 124 mod 26 = 20.

9*19 – 2 mod 26 = 169 mod 26 = 13.

9*24 – 2 mod 26 = 214 mod 26 = 6.

Bản rõ sau khi giải mã là "**xung**".

CHƯƠNG III: DEMO CHƯƠNG TRÌNH AFFINE

3.1 Ý tưởng thực hiện

Để thực hiện mã hóa Affine trong ứng dụng Windows Form bằng ngôn ngữ C#, chúng ta sẽ tạo một giao diện đơn giản cho phép người dùng nhập vào văn bản cần mã hóa (plaintext), các khóa a và b, sau đó hiển thị kết quả mã hóa (ciphertext). Các bước chính của quá trình này bao gồm việc xử lý giao diện người dùng và triển khai thuật toán mã hóa Affine.

Ứng dụng cho phép người dùng nhập văn bản và khóa mã hóa, và sau đó chọn giữa hai tùy chọn: mã hóa hoặc giải mã. Kết quả của quá trình này sẽ được hiển thị ngay trên giao diện người dùng. Chương trình sử dụng các thành phần giao diện cơ bản của Windows Forms, bao gồm:

Hộp điều kiển phân trang (TabControl): Để người dùng lựa chọn một trong hai tác vụ. Mã hóa hoặc giải mã của chương trình.

Hộp văn bản nhập liệu (TextBox): Để người dùng nhập văn bản cần mã hóa (‘txtPtoE’) hoặc giải mã (‘txtEtoD’). Nhập khóa a (`kda`) và khóa b (`kdb`). Và hiển thị kết quả (`txtAfE`).

Nút bấm (Button): Để thực hiện hành động mã hóa (`btnEncode`) hoặc giải mã (‘btnDecode’).

3.2 Quy trình hoạt động của chương trình

Khởi chạy chương trình

Chọn chức năng: Người dùng tích chọn trang để chọn giữa mã hóa hoặc giải mã.

Nhập dữ liệu: Người dùng nhập văn bản vào `txtPtoE’` và khóa a `kda` - khóa b ‘kdb’.

Thực hiện thao tác: Khi người dùng bấm nút `btnEncode` để mã hóa hoặc nếu bên giải mã sẽ bấm nút ‘btnDecode’’ , chương trình sẽ kiểm tra và chạy chương trình.

Hiển thị kết quả: Kết quả được hiển thị trong `txtAfE`.

3.2.1 Hàm mã hóa

```
// Ham ma hoa
if ( kea.Text == "" || keb.Text == "" ) {
    MessageBox.Show ( "Phải nhập khóa." );
} else {
    try {
        int a = int.Parse ( kea.Text );
        int b = int.Parse ( keb.Text );

        // Kiểm tra điều kiện của khóa
        if ( a <= 0 || a > 25 || b < 0 || b > 25 || UCLN ( a, 26 ) != 1 ) {
            MessageBox.Show ( "Khóa không hợp lệ." );
        } else {
            string plain = txtPtoE.Text;
            string enc = "";

            for ( int i = 0 ; i < plain.Length ; i++ ) {
                char currentChar = plain [i];

                if ( currentChar != ' ' ) {
                    // Xử lý ký tự chữ hoa
                    if ( char.IsUpper ( currentChar ) ) {
                        enc += ( char ) ( ( ( a * ( currentChar - 'A' ) + b ) % 26 ) + 'A' );
                    }
                    // Xử lý ký tự chữ thường
                    else if ( char.IsLower ( currentChar ) ) {
                        enc += ( char ) ( ( ( a * ( currentChar - 'a' ) + b ) % 26 ) + 'a' );
                    }
                } else {
                    enc += " ";
                }
            }

            txtAfE.Text = enc;
        }
    } catch ( Exception ) {
        MessageBox.Show ( "Khóa không hợp lệ." );
    }
}
```

Hình 3. 1 Hàm Mã hoá

a, Chức năng

Hàm này thực hiện mã hóa Affine cho một chuỗi ký tự (văn bản) do người dùng nhập, dựa trên hai khóa a và b mà người dùng cung cấp. Hàm sẽ kiểm tra tính hợp lệ của khóa, sau đó mã hóa từng ký tự của chuỗi văn bản theo công thức mã hóa Affine và trả kết quả mã hóa lên giao diện.

b, Chi tiết hoạt động

Bước 1: Kiểm tra khóa đầu vào

Hàm kiểm tra xem người dùng đã nhập khóa a và b chưa. Nếu một trong hai giá trị khóa trống, một thông báo yêu cầu nhập khóa sẽ xuất hiện.

Sau đó, hai giá trị khóa a và b được chuyển đổi từ kiểu chuỗi (string) sang kiểu số nguyên (int) bằng phương thức int.Parse().

Bước 2: Kiểm tra tính hợp lệ của khóa

Hàm kiểm tra xem giá trị của khóa a có nằm trong khoảng từ 1 đến 25 và khóa b có nằm trong khoảng từ 0 đến 25 hay không.

Đặc biệt, khóa a phải thỏa mãn điều kiện UCLN(a, 26) = 1 (tức là a và 26 phải là hai số nguyên tố cùng nhau), để đảm bảo tồn tại nghịch đảo modular của a trong phép mã hóa Affine. Nếu khóa không hợp lệ, hàm sẽ thông báo lỗi.

Bước 3: Mã hóa văn bản

Văn bản cần mã hóa (chuỗi plain) được lấy từ txtPtoE.Text.

Một biến chuỗi enc được khởi tạo để lưu kết quả của chuỗi đã mã hóa.

Vòng lặp qua từng ký tự của chuỗi:

Hàm duyệt qua từng ký tự của chuỗi plain:

Nếu ký tự không phải là dấu cách:

Hàm kiểm tra ký tự hiện tại là chữ hoa hay chữ thường bằng cách sử dụng các hàm char.IsUpper() và char.IsLower().

Với ký tự chữ hoa:

Công thức mã hóa được áp dụng là:

$E(x)=(a\cdot (x-'A')+b) \bmod 26+'A'$

Trong đó, x là giá trị ASCII của ký tự, và 'A' là giá trị ASCII của ký tự A.

Với ký tự chữ thường:

Công thức tương tự nhưng với ký tự bắt đầu là 'a':

$E(x)=(a\cdot (x-'a')+b) \bmod 26+'a'$

Nếu ký tự là dấu cách, nó sẽ được giữ nguyên và thêm vào chuỗi kết quả.

Sau khi mã hóa toàn bộ chuỗi, kết quả mã hóa được lưu vào biến enc.

Bước 4: Hiển thị kết quả mã hóa

Kết quả mã hóa (enc) được hiển thị lên giao diện thông qua txtAfE.Text.

Bước 5: Xử lý ngoại lệ

Trong quá trình chuyển đổi giá trị hoặc mã hóa, nếu xảy ra lỗi (ví dụ: nhập sai định dạng), một thông báo "Khóa không hợp lệ" sẽ được hiển thị để người dùng biết có lỗi xảy ra.

3.2.2 Hàm giải mã

```
// Ham giai ma
txtP.Clear ( );
if ( kda.Text == "" || kdb.Text == "" ) {
    MessageBox.Show ( "Phải nhập khóa." );
} else {
    try {
        int a = int.Parse ( kda.Text );
        int b = int.Parse ( kdb.Text );

        // Kiểm tra điều kiện của khóa
        if ( a <= 0 || a > 25 || b < 0 || b > 25 || UCLN ( a, 26 ) != 1 ) {
            MessageBox.Show ( "Khóa không hợp lệ." );
        } else {
            if ( a == 0 && b == 0 ) {
                // Trường hợp thử tất cả các khóa (a = 0, b = 0)
                for ( int i = 1 ; i < 26 ; i++ ) {
                    for ( int j = 0 ; j < 26 ; j++ ) {
                        if ( UCLN ( i, 26 ) == 1 ) {
                            string enc = txtEtoD.Text;
                            string Pl = "a = " + i + ", b = " + j + " : ";

                            for ( int k = 0 ; k < enc.Length ; k++ ) {
                                char currentChar = enc [k];

                                if ( currentChar != ' ' ) {
                                    if ( char.IsLower ( currentChar ) )
                                        Pl += ( char ) ( nghichdao ( i ) * ( currentChar - 'a' - j + 26 ) % 26 + 'a' );
                                    else if ( char.IsUpper ( currentChar ) )
                                        Pl += ( char ) ( nghichdao ( i ) * ( currentChar - 'A' - j + 26 ) % 26 + 'A' );
                                } else {
                                    Pl += " ";
                                }
                            }
                            Pl = Pl + "\n";
                            txtP.Text += Pl;
                        }
                    }
                }
            } else {
                if ( a != 0 ) {
                    string enc = txtEtoD.Text;
                    string Pl = "";

                    for ( int i = 0 ; i < enc.Length ; i++ ) {
                        char currentChar = enc [i];

                        if ( currentChar != ' ' ) {
                            if ( char.IsLower ( currentChar ) )
                                Pl += ( char ) ( nghichdao ( a ) * ( currentChar - 'a' - b + 26 ) % 26 + 'a' );
                            else if ( char.IsUpper ( currentChar ) )
                                Pl += ( char ) ( nghichdao ( a ) * ( currentChar - 'A' - b + 26 ) % 26 + 'A' );
                        } else {
                            Pl += " ";
                        }
                    }
                    txtP.Text = Pl;
                }
            }
        }
    } catch ( Exception ) {
        MessageBox.Show ( "Khóa không hợp lệ." );
    }
}
```

Hình 3. 2 Hàm Giải mã

a, Chức năng

Hàm giải mã Affine này được sử dụng để giải mã một chuỗi ký tự đã được mã hóa bằng phương pháp Affine. Người dùng cung cấp hai khóa a và b, và dựa vào đó hàm sẽ thực hiện giải mã từng ký tự của văn bản mã hóa theo công thức giải mã của hệ mã Affine.

b, Chi tiết hoạt động

Bước 1: Xóa kết quả cũ

Dòng lệnh txtP.Clear(); dùng để xóa dữ liệu văn bản đã được giải mã từ trước đó, chuẩn bị cho quá trình giải mã mới.

Bước 2: Kiểm tra khóa đầu vào

Hàm kiểm tra xem người dùng đã nhập khóa a và b hay chưa. Nếu chưa nhập, hàm hiển thị thông báo yêu cầu nhập khóa.

Bước 3: Kiểm tra tính hợp lệ của khóa

Sau khi nhận khóa a và b, hàm chuyển đổi chúng từ chuỗi ký tự sang số nguyên bằng cách sử dụng int.Parse().

Hàm kiểm tra khóa a và b phải nằm trong phạm vi hợp lệ:

a phải nằm trong khoảng từ 1 đến 25.

b phải nằm trong khoảng từ 0 đến 25.

a phải thỏa mãn điều kiện UCLN(a, 26) = 1 (tức là a và 26 phải là hai số nguyên tố cùng nhau) để có thể tìm được nghịch đảo modular của a.

Nếu khóa không hợp lệ, hàm sẽ thông báo cho người dùng.

Bước 4: Xử lý trường hợp đặc biệt (a = 0 và b = 0)

Khi khóa a = 0 và b = 0, hàm sẽ thực hiện việc thử tất cả các cặp khóa có thể (thử mọi giá trị từ a = 1 đến 25 và b = 0 đến 25). Đây là cách để thử tìm ra khóa mã hóa bằng cách thử tất cả các tổ hợp khóa có thể.

Với mỗi cặp giá trị của a và b, hàm kiểm tra xem a có thỏa mãn điều kiện UCLN(a, 26) = 1. Nếu có, nó thực hiện giải mã chuỗi văn bản mã hóa với cặp khóa hiện tại.

Bước 5: Giải mã văn bản

Văn bản mã hóa (biến enc) được lấy từ txtEtoD.Text.

Biến Pl được khởi tạo để chứa kết quả giải mã.

Vòng lặp qua từng ký tự của chuỗi:

Hàm duyệt qua từng ký tự của chuỗi mã hóa:

Nếu ký tự không phải là dấu cách:

Ký tự chữ thường: Sử dụng công thức giải mã: $D(y)=a^{-1} \cdot (y-b) \bmod 26$

Trong đó:

y là giá trị mã ASCII của ký tự hiện tại.

a^{-1} là nghịch đảo modular của a (tính bằng hàm nghichdao(a)).

b là khóa dịch chuyển.

Ký tự chữ hoa: Công thức tương tự nhưng ký tự bắt đầu từ 'A'.

Nếu ký tự là dấu cách, nó được giữ nguyên.

Sau khi giải mã toàn bộ chuỗi, kết quả được lưu vào biến Pl.

Bước 6: Hiển thị kết quả

Nếu khóa không phải là cặp a = 0 và b = 0, hàm sẽ hiển thị kết quả giải mã lên giao diện thông qua txtP.Text.

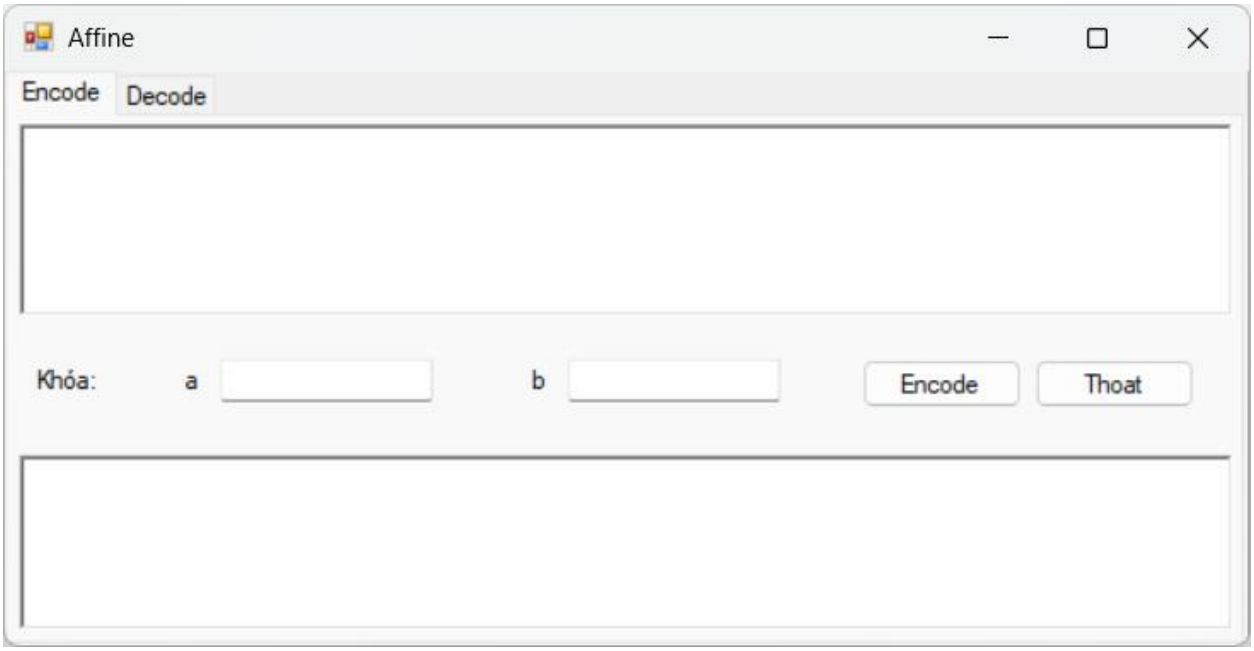
Trong trường hợp thử tất cả các cặp khóa, kết quả cho từng cặp khóa sẽ được hiển thị theo định dạng "a = ..., b = ... : kết quả giải mã".

Bước 7: Xử lý ngoại lệ

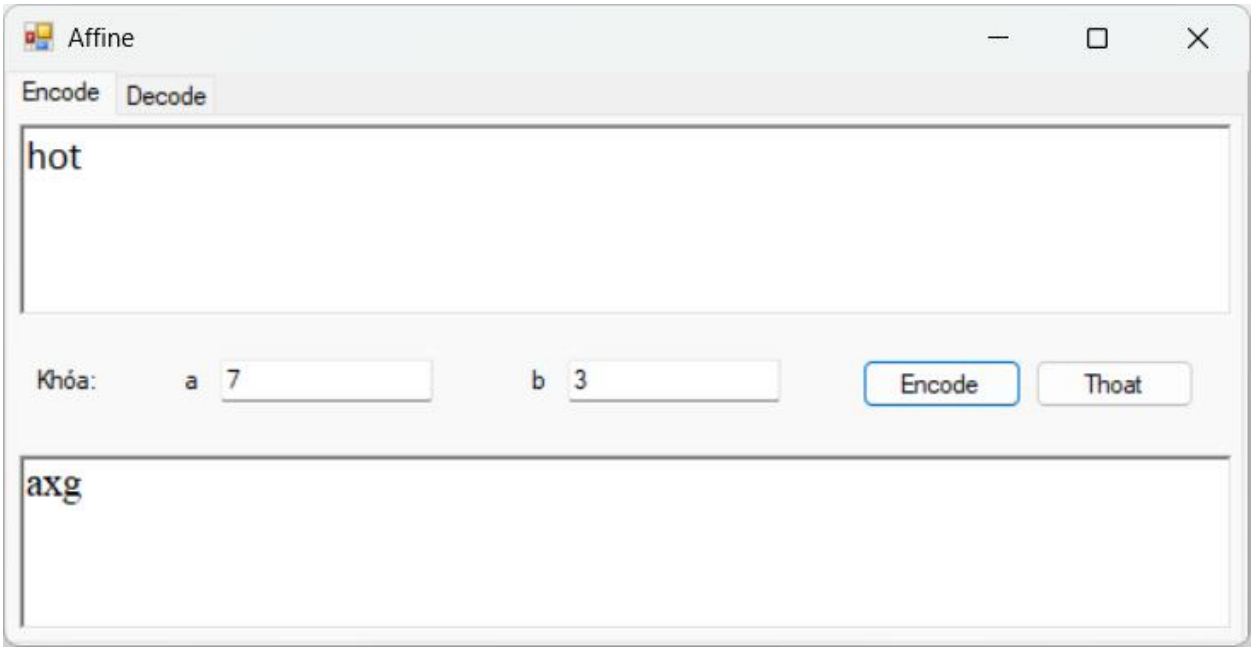
Nếu trong quá trình chuyển đổi hoặc giải mã xảy ra lỗi, một thông báo "Khóa không hợp lệ" sẽ xuất hiện để báo cho người dùng về vấn đề xảy ra.

3.3 Kết quả thu được

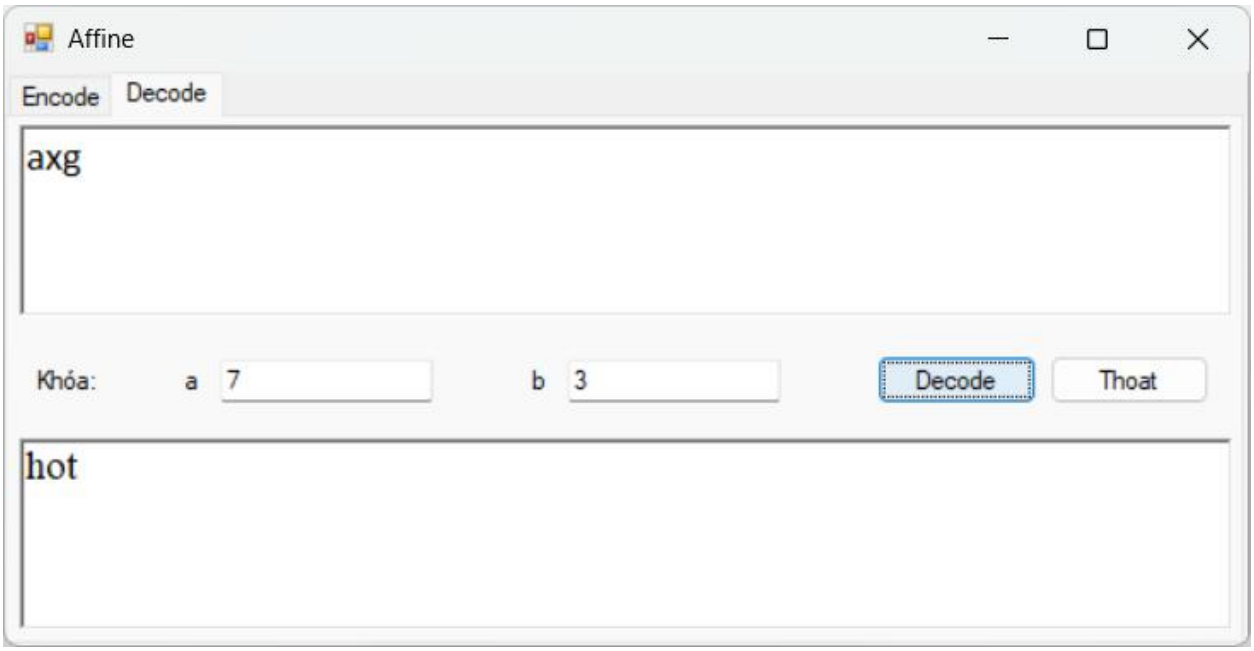
Chương trình hoạt động tốt sau khi hoàn thành và khởi chạy. Dưới đây là thành quả thu được của chúng tôi:



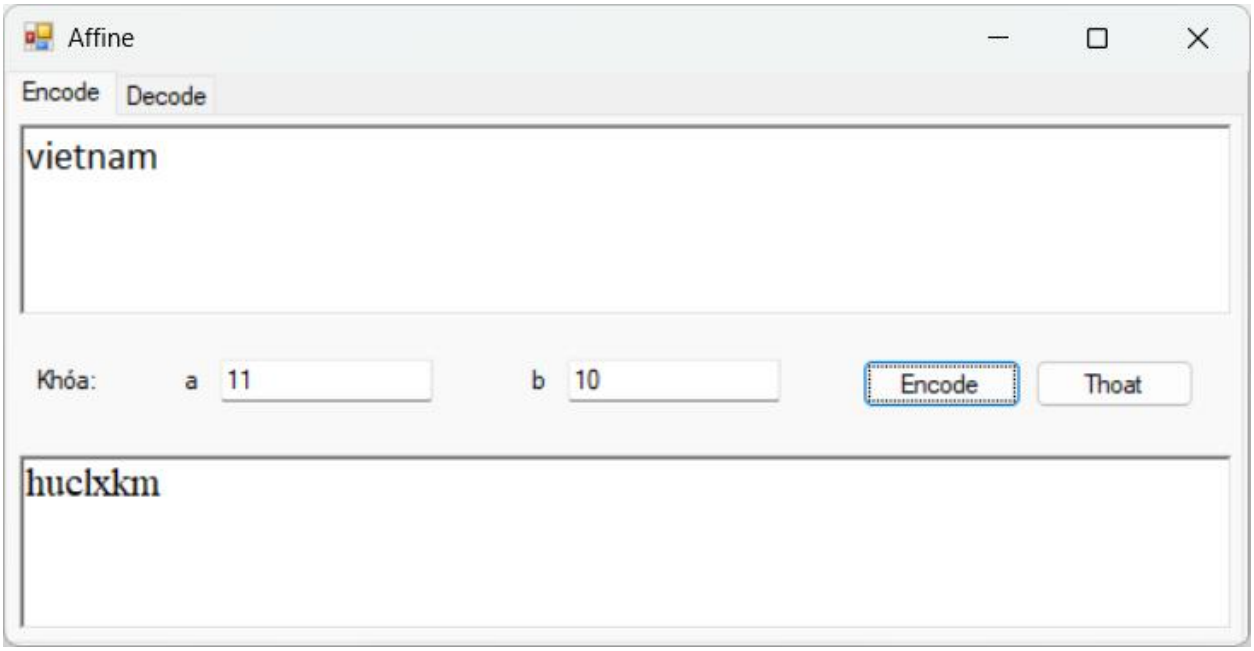
Hình 3. 3 Chương trình khởi chạy



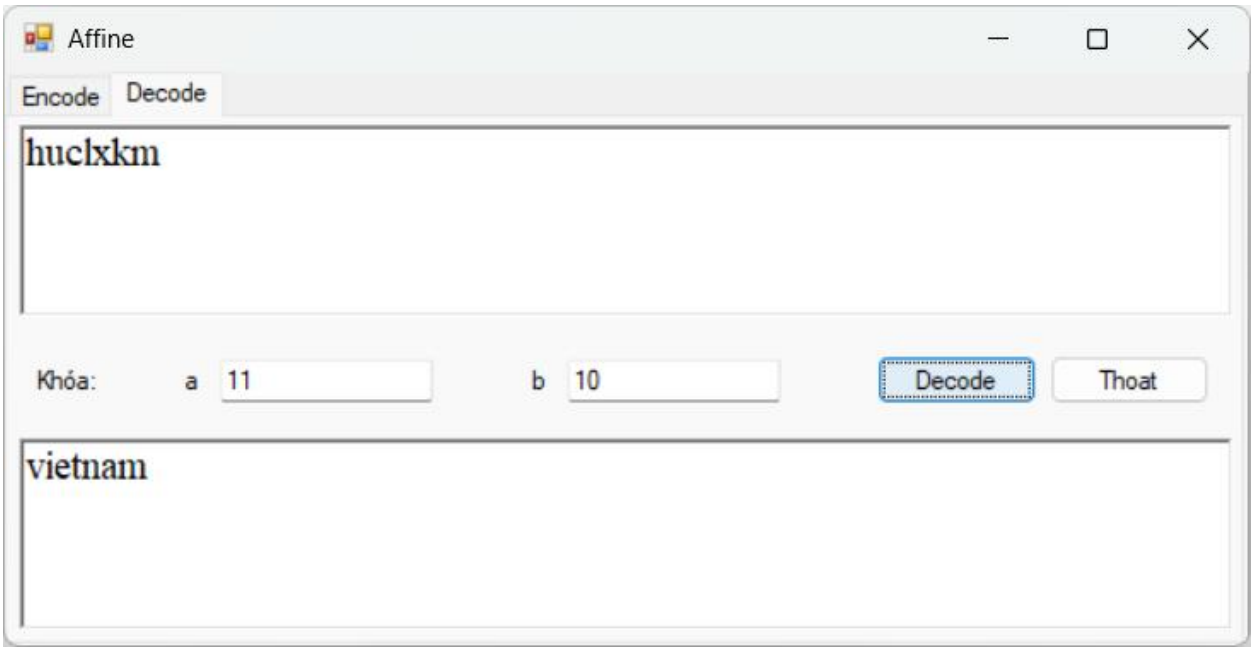
Hình 3. 4 Mẫu thử mã hoá (1)



Hình 3. 5 Mẫu thử Giải mã (1)



Hình 3. 6 Mẫu thử Mã hoá (2)



Hình 3. 7 Mẫu thử Giải mã (2)

CHƯƠNG IV: KẾT LUẬN

4.1 Kết quả đạt được

- Tổng quan về mã hóa cổ điển:

Đã nghiên cứu và trình bày chi tiết về các phương pháp mã hóa cổ điển, bao gồm mã hóa Caesar, mã hóa Vigenère, và đặc biệt là mã hóa Affine.

Phân tích ưu nhược điểm của từng phương pháp, giúp hiểu rõ hơn về ứng dụng và tính bảo mật của chúng.

- Xây dựng chương trình mô phỏng thuật toán Affine:

Thiết kế và triển khai chương trình mô phỏng thuật toán Affine bằng ngôn ngữ lập trình Python (hoặc ngôn ngữ khác phù hợp).

Chương trình cho phép người dùng nhập văn bản cần mã hóa, cùng với hai khóa (a và b) và thực hiện mã hóa theo công thức: $E(x)=(ax+b) \bmod 26$, trong đó

- Kiểm thử và đánh giá:

Chương trình đã được kiểm thử với nhiều bộ dữ liệu khác nhau, cho kết quả chính xác và nhanh chóng.

Đã thực hiện mã hóa và giải mã thành công, chứng minh được tính đúng đắn của thuật toán.

- Tài liệu và hướng dẫn sử dụng:

Soạn thảo tài liệu hướng dẫn chi tiết về cách sử dụng chương trình, bao gồm các bước cài đặt, nhập dữ liệu và kiểm tra kết quả.

- Những bài học rút ra:

Nâng cao hiểu biết về các khái niệm mã hóa và an ninh thông tin.

Cải thiện kỹ năng lập trình và khả năng giải quyết vấn đề thông qua việc xây dựng chương trình mô phỏng.

4.2 Hạn chế

Mặc dù mã hóa Affine là một phương pháp mã hóa cổ điển đơn giản và dễ hiểu, nó có một số hạn chế đáng kể:

Độ bảo mật thấp: Mã hóa Affine chỉ dựa trên hai tham số khóa đơn giản, điều này làm cho nó dễ bị tấn công bằng phương pháp phân tích tần suất. Vì mỗi ký tự trong bảng chữ cái được ánh xạ theo một quy tắc cố định, kẻ tấn công có thể phân tích tần suất xuất hiện của các ký tự trong văn bản mã hóa để tìm ra khóa.

Không đủ phức tạp cho dữ liệu lớn: Mã hóa Affine chỉ phù hợp với văn bản ngắn và không có khả năng đối phó với các loại dữ liệu hiện đại như tệp hình ảnh, âm thanh,

hoặc video. Do tính chất đơn giản của nó, thuật toán không thể đáp ứng các yêu cầu về mã hóa dữ liệu lớn hoặc dữ liệu nhạy cảm.

Không có tính kháng mã hóa hiện đại: Các thuật toán mã hóa hiện đại như AES (Advanced Encryption Standard) hay RSA có độ bảo mật rất cao và được sử dụng phổ biến trong bảo mật thông tin hiện đại. Mã hóa Affine không thể cạnh tranh về mặt bảo mật hoặc tính hiệu quả với các thuật toán này.

Dễ bị tấn công brute-force: Vì khóa a chỉ có thể là các số nguyên nguyên tố cùng nhau với 26 (chỉ có 12 giá trị khả dĩ), kết hợp với khóa b (có 26 giá trị), tổng số khóa có thể kiểm tra bằng brute-force là rất nhỏ (chỉ 312 khả năng), khiến mã hóa Affine không thể chống lại các tấn công kiểu brute-force.

4.3 Hướng phát triển

Mặc dù mã hóa Affine có những hạn chế, nó vẫn có thể là nền tảng cho các hướng phát triển và nghiên cứu trong tương lai, đặc biệt khi áp dụng trong môi trường giáo dục hoặc nghiên cứu về các hệ mã hóa cổ điển:

Kết hợp với các phương pháp mã hóa khác: Mã hóa Affine có thể được kết hợp với các thuật toán mã hóa phức tạp hơn để tạo ra một phương pháp mã hóa lai. Ví dụ, việc kết hợp Affine với mã hóa hoán vị hoặc các phương pháp mã hóa đa lớp có thể cải thiện độ an toàn.

Nghiên cứu các biến thể của mã hóa Affine: Một hướng phát triển có thể là nghiên cứu các biến thể của thuật toán Affine bằng cách thay đổi phép toán hoặc mở rộng tập hợp các ký tự. Việc này có thể tăng tính phức tạp của thuật toán và giúp chống lại các tấn công phân tích tần suất.

Sử dụng trong giáo dục: Mã hóa Affine là một phương pháp lý tưởng để giảng dạy về các nguyên tắc cơ bản của mật mã học. Các khóa học về mã hóa có thể sử dụng Affine để giúp sinh viên hiểu về các nguyên tắc của mã hóa và giải mã trước khi chuyển sang các hệ mã phức tạp hơn.

Ứng dụng trong hệ thống mã hóa đơn giản: Mặc dù không phù hợp cho các ứng dụng yêu cầu bảo mật cao, mã hóa Affine vẫn có thể sử dụng trong các hệ thống yêu cầu mã hóa đơn giản và không cần bảo mật phức tạp, chẳng hạn như trong các trò chơi hoặc các ứng dụng giáo dục.

Tăng cường các tính năng bảo mật: Một số tính năng bảo mật như việc thêm tính ngẫu nhiên vào quá trình mã hóa, sử dụng nhiều khóa hoặc các thuật toán hoán vị phức tạp hơn có thể giúp tăng cường khả năng bảo mật của mã hóa Affine.

Tóm lại, mã hóa Affine tuy có nhiều hạn chế về bảo mật, nhưng nó vẫn là một công cụ hữu ích trong việc nghiên cứu, giáo dục và có tiềm năng phát triển thêm khi kết hợp với các phương pháp mã hóa phức tạp khác.

DANH MỤC THAM KHẢO

[1] https://nam.name.vn/bai-11-ma-hoa-co-dien-ma-hoa-affine.html (Truy cập lần cuối vào 10:20 ngày 11/09/2024)

[2] https://cryptii.com/pipes/affine-cipher (Truy cập lần cuối vào 09:45 ngày 10/09/2024)

[3] https://vi.wikipedia.org/wiki/M%E1%BA%ADt_m%C3%A3_Affine (Truy cập lần cuối vào 11:25 ngày 10/09/2024)

[4] https://www.dcode.fr/affine-cipher?__r=1.dc3f42975767e81926aae0fb915fba6e (Truy cập lần cuối vào 15:35 ngày 09/09/2024)

[5] https://drx.home.blog/2018/07/21/series-mat-ma-03-ma-affine/ (Truy cập lần cuối vào 18:22 ngày 08/09/2024)

[6] https://post.nghiatu.com/tutorial/cryptography-with-python/cryptography-with-python-affine-cipher/mat-ma-voi-python-mat-ma-affine (Truy cập lần cuối vào 22:23 ngày 11/09/2024)