

LỜI NÓI ĐẦU

Ngày nay với sự xuất hiện của máy tính, các tài liệu văn bản giấy tờ và các thông tin quan trọng đều được số hóa và xử lý trên máy tính, được truyền đi trong một môi trường mà mặc định là không an toàn. Do đó yêu cầu về việc có một cơ chế, giải pháp để bảo vệ sự an toàn và bí mật của các thông tin nhạy cảm, quan trọng ngày càng trở nên cấp thiết. Mật mã học chính là ngành khoa học đảm bảo cho mục đích này. Khó có thể thấy một ứng dụng Tin học có ích nào lại không sử dụng các thuật toán mã hóa thông tin.

Chính vì nhu cầu cần thiết của mã hóa thông tin, với sự giúp đỡ của thầy giáo TS. Ngô Quốc Dũng – khoa Công nghệ và An ninh thông tin, em đã tiến hành tìm hiểu về “*Hệ mã hoá truyền thống và bất đối xứng – các điểm mạnh và yếu của chúng*” để thực hiện báo cáo kết quả môn học “*Lý thuyết mật mã*”.

Chân thành cảm ơn sự giúp đỡ của thầy đã giúp em hoàn thành bài báo cáo này!

MỤC LỤC

CHƯƠNG I: TỔNG QUAN VỀ MÃ HÓA.....	1
1.1 CÁC KHÁI NIỆM.....	1
1.1.1 Mã hóa, giải mã.....	1
1.1.2 Đánh giá tính bảo mật của các hệ mật mã.....	3
1.2 LỊCH SỬ	4
CHƯƠNG II: MÃ HÓA ĐỐI XỨNG.....	6
2.1 GIỚI THIỆU.....	6
2.1.1 Hệ mã hóa đối xứng	6
2.1.2 Mô hình mã hóa đối xứng	8
2.2 MỘT SỐ HỆ MÃ ĐỐI XỨNG.....	8
2.2.1 Caesar cipher.....	8
2.2.2 Vigenère	11
2.2.3 Data Encryption Standard (DES).....	11
2.2.4 Triple DES & AES.....	15
CHƯƠNG III: MÃ HÓA BẤT ĐỐI XỨNG.....	17
3.1 GIỚI THIỆU.....	17
3.1.1 Mã hóa bất đối xứng.....	17
3.1.2 Lịch sử.....	18
3.1.3 Độ an toàn.....	19
3.1.4 Ứng dụng.....	20
3.2 MỘT SỐ HỆ MÃ HÓA BẤT ĐỐI XỨNG.....	20
TÀI LIỆU THAM KHẢO.....	31



CHƯƠNG I: TỔNG QUAN VỀ MÃ HÓA

1.1 Các khái niệm

1.1.1 Mã hóa, giải mã

Mật mã là một lĩnh vực khoa học chuyên nghiên cứu về các phương pháp và kỹ thuật đảm bảo an toàn và bảo mật trong truyền tin liên lạc với giả thiết sự tồn tại của các thế lực thù địch, những kẻ muốn ăn cắp thông tin để lợi dụng và phá hoại. Tên gọi trong tiếng Anh, Cryptology được dẫn giải nguồn gốc từ tiếng Hy Lạp, trong đó kryptos nghĩa là “che giấu”, logos nghĩa là “từ ngữ”.

Cụ thể hơn, các nhà nghiên cứu lĩnh vực này quan tâm xây dựng hoặc phân tích (để chỉ ra điểm yếu) các giao thức mật mã (cryptographic protocols), tức là các phương thức giao dịch có đảm bảo mục tiêu an toàn cho các bên tham gia (với giả thiết môi trường có kẻ đối địch, phá hoại).

Ngành Mật mã (*cryptology*) thường được quan niệm như sự kết hợp của 2 lĩnh vực con:

1. Sinh, chế mã mật (*cryptography*): nghiên cứu các kỹ thuật toán học nhằm cung cấp các công cụ hay dịch vụ đảm bảo an toàn thông tin
2. Phá giải mã (*cryptanalysis*): nghiên cứu các kỹ thuật toán học phục vụ phân tích phá mật mã và/hoặc tạo ra các đoạn mã giả nhằm đánh lừa bên nhận tin.

Hai lĩnh vực con này tồn tại như hai mặt đối lập, “đấu tranh để cùng phát triển” của một thể thống nhất là ngành khoa học mật mã (*cryptology*). Tuy nhiên, do lĩnh vực thứ hai (*cryptanalysis*) ít được phổ biến quảng đại nên dần dần, cách hiểu chung hiện nay là đánh đồng hai thuật ngữ *cryptography* và *cryptology*. Theo thói quen chung này, hai thuật ngữ này có thể dùng thay thế nhau. Thậm chí *cryptography* là thuật ngữ ưa dùng, phổ biến trong mọi sách vở phổ biến khoa học, còn *cryptology* thì xuất hiện trong một phạm vi hẹp của các nhà nghiên cứu học thuật thuần túy.



Mặc dù trước đây hầu như mật mã và ứng dụng của nó chỉ phổ biến trong phạm vi hẹp, nhưng với sự phát triển vũ bão của công nghệ thông tin và đặc biệt là sự phổ biến của mạng Internet, các giao dịch có sử dụng mật mã đã trở nên rất phổ biến. Chẳng hạn, ví dụ điển hình là các giao dịch ngân hàng trực tuyến hầu hết đều được thực hiện qua mật mã. Ngày nay, kiến thức ngành mật mã là cần thiết cho các cơ quan chính phủ, các khối doanh nghiệp và cả cho cá nhân. Một cách khái quát, ta có thể thấy mật mã có các ứng dụng như sau:

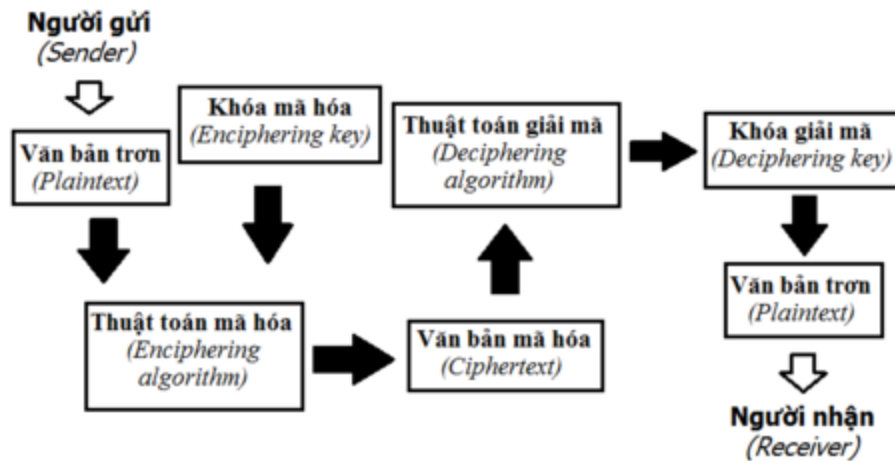
- Với các chính phủ: bảo vệ truyền tin mật trong quân sự và ngoại giao, bảo vệ thông tin các lĩnh vực tầm cỡ lợi ích quốc gia.
- Trong các hoạt động kinh tế: bảo vệ các thông tin nhạy cảm trong giao dịch như hồ sơ pháp lý hay y tế, các giao dịch tài chính hay các đánh giá tín dụng...
- Với các cá nhân: bảo vệ các thông tin nhạy cảm, riêng tư trong liên lạc với thế giới qua các giao dịch sử dụng máy tính và/hoặc kết nối mạng

Vậy mã hóa, giải mã là gì?

Mật mã hóa là quá trình chuyển đổi các thông tin thông thường (văn bản thường hay *văn bản rõ* hay văn bản tron) thành dạng không đọc trực tiếp được, là *văn bản mã hóa*. Giải mật mã hay giải mã là quá trình ngược lại, phục hồi lại văn bản thường từ văn bản mã. Mật mã là thuật toán để mật mã hóa và giải mật mã. Hoạt động chính xác của mật mã thông thường được kiểm soát bởi các khóa — một đoạn thông tin bí mật nào đó cho phép tùy biến cách thức tạo ra văn bản mã.

Trong một hệ thống mật mã khái quát sẽ có các thành phần sau:

- *Văn bản rõ* (plaintext): thông điệp nguyên gốc chưa được mã hóa.
- *Văn bản mã hóa* (ciphertext): thông điệp đã được mã hóa.
- *Thuật toán mã hóa* (enciphering algorithm) là các giao thức hoặc hướng dẫn có tác dụng chuyển đổi văn bản rõ thành văn bản mã hóa.



Mô hình mã hóa, giải mã trong quá trình truyền, nhận thông tin mật

- *Khóa mã hóa* (enciphering key) là một hoặc nhiều đối tượng (thường là các con số hay là các hướng dẫn quan trọng nào đó) được dùng trong việc mã hóa văn bản rõ.
- *Thuật toán giải mã* (deciphering algorithm) là các giao thức hoặc hướng dẫn có tác dụng chuyển đổi văn bản mã hóa trở về văn bản rõ.
- *Khóa giải mã* (deciphering key) là một hoặc nhiều đối tượng (thường là các con số hay là các hướng dẫn quan trọng nào đó) được dùng trong việc giải mã văn bản bị mã hóa.

1.1.2 Đánh giá tính bảo mật của các hệ mật mã

Các thuật toán, hệ thống mật mã được biết đến trên thế giới là không ít. Làm sao để ta có thể đánh giá được tính an toàn, hay tính bảo mật của mỗi một hệ mã đặt ra? Trên cơ sở nào chúng ta có thể thiết lập niềm tin nhiều hoặc không nhiều vào một hệ mã nào đó?

Ta có thể kết luận một hệ mã mật là không an toàn (*insecure*), bằng việc chỉ ra cách phá nó trong một mô hình tấn công phổ biến, trong đó ta chỉ rõ được các mục tiêu về an toàn bảo mật (*security*) không được đảm bảo đúng. Tuy nhiên để kết luận rằng một hệ mã là an toàn cao thì công việc phức tạp hơn nhiều. Thông thường, người ta phải đánh giá hệ mật mã này trong nhiều mô hình tấn công khác nhau, với tính thách thức tăng dần.



Mục tiêu của **thăm mã** (phá mã) là tìm những điểm yếu hoặc không an toàn trong phương thức mật mã hóa. Thăm mã có thể được thực hiện bởi những kẻ tấn công ác ý, nhằm làm hỏng hệ thống; hoặc bởi những người thiết kế ra hệ thống (hoặc những người khác) với ý định đánh giá độ an toàn của hệ thống.

Có rất nhiều loại hình tấn công thăm mã, và chúng có thể được phân loại theo nhiều cách khác nhau. Một trong những đặc điểm liên quan là những người tấn công có thể biết và làm những gì để hiểu được thông tin bí mật. Ví dụ, những người thăm mã chỉ truy cập được bản mã hay không? hay anh ta có biết hay đoán được một phần nào đó của bản rõ? hoặc thậm chí: Anh ta có chọn lựa các bản rõ ngẫu nhiên để mật mã hóa? Các kịch bản này tương ứng với *tấn công bản mã*, *tấn công biết bản rõ* và *tấn công chọn lựa bản rõ*.

Trong khi công việc thăm mã thuần túy sử dụng các điểm yếu trong các thuật toán mật mã hóa, những cuộc tấn công khác lại dựa trên sự thi hành, được biết đến như là các tấn công kênh bên. Nếu người thăm mã biết lượng thời gian mà thuật toán cần để mã hóa một lượng bản rõ nào đó, anh ta có thể sử dụng phương thức tấn công thời gian để phá mật mã. Người tấn công cũng có thể nghiên cứu các mẫu và độ dài của thông điệp để rút ra các thông tin hữu ích cho việc phá mã; điều này được biết đến như là thăm mã lưu thông.

1.2 Lịch sử

Thời kỳ tiền khoa học: Tính từ thượng cổ cho đến 1949. Trong thời kỳ này, khoa mật mã học được coi là một ngành mang nhiều tính thủ công, nghệ thuật hơn là tính khoa học. Các hệ mật mã được phát minh và sử dụng trong thời kỳ này được gọi là các hệ mật mã cổ điển. Sau đây ta làm quen với hai ví dụ hệ mã rất nổi tiếng của thời kỳ này.

1. Một phép mã hoá (cipher) trong thời kỳ này là của Xe-da (Caesar's cipher), cách đây 2000 năm: các chữ cái được thay thế bằng các chữ cái cách chúng 3 vị trí về bên phải trong bản alphabet:



DASEAR → FDHVDU

2. Vernam cipher (1926): người ta đem thực hiện phép XOR văn bản gốc (plaintext) với một chuỗi nhị phân ngẫu nhiên có độ dài bằng độ dài của văn bản gốc (chuỗi này là chính là khoá của phép mã hoá). Trong cipher loại này, khoá chỉ được dùng đúng một lần duy nhất. Vernam tin rằng cipher của ông là không thể phá được nhưng không thể chứng minh được.

Kỷ nguyên mật mã được coi là ngành khoa học: được đánh dấu bởi bài báo nổi tiếng của Claude Shannon “*Communication theory of secrecy systems*”, được công bố năm 1949. Công trình này dựa trên một bài báo trước đó của ông mà trong đó ông cũng đã khai sáng ra ngành khoa học quan trọng khác, lý thuyết thông tin (*information theory*). Bài báo năm 1949 của Shannon đã nền móng cho việc áp dụng công cụ toán, cụ thể là xác suất, trong xây dựng mô hình và đánh giá tính mật của các hệ mã mật.

Các nghiên cứu rộng rãi có tính học thuật về mật mã hóa hiện đại là tương đối gần đây — nó chỉ được bắt đầu trong cộng đồng mở kể từ những năm thập niên 1970 với các chi tiết kỹ thuật của **DES** (viết tắt trong tiếng Anh của *Data Encryption Standard* tức Tiêu chuẩn Mật mã hóa Dữ liệu) và sự phát minh ra **RSA**. Kể từ đó, mật mã hóa đã trở thành công cụ được sử dụng rộng rãi trong liên lạc và bảo mật máy tính.

Sự bùng nổ thực sự trong lý thuyết về mật mã (Cryptology) chỉ bắt đầu từ bài báo của hai nhà bác học Diffie và Hellman, “*New directions in cryptography*”, được công bố vào năm 1976. Trong đó, các ông này đã chứng tỏ rằng trong truyền tin bí mật, không nhất thiết là cả hai bên đều phải nắm khoá bí mật (tức bên gửi phải làm cách nào đó chuyển được khoá mật cho bên nhận). Hơn nữa họ đã lần đầu tiên giới thiệu khái niệm về chữ ký điện tử (*digital signature*).



CHƯƠNG II: MÃ HÓA ĐỐI XỨNG

2.1 Giới thiệu

2.1.1 Hệ mã hóa đối xứng

Loại hệ thống này còn gọi là hệ mật mã khóa bí mật (Secret Key Cryptosystem). Trong mô hình của hệ thống này, khóa của hai thuật toán sinh mã và giải mã là giống nhau và bí mật đối với tất cả những người khác; nói cách khác, hai bên gửi và nhận tin chia sẻ chung một khóa bí mật duy nhất. Vai trò của hai phía tham gia là giống nhau và có thể đánh đổi vai trò, gửi và nhận tin, cho nên hệ thống được gọi là “*mã hóa đối xứng*” (symmetric-key algorithms). Chúng ta sẽ sử dụng ký hiệu viết tắt theo tiếng Anh là **SKC**.

Khóa đối xứng có thể nhóm thành *mật mã khối* và *mật mã luồng*. Mật mã luồng mật mã hóa 1 bit tại một thời điểm, ngược lại với mật mã khối là phương thức cho phép thực hiện trên một nhóm các bit ("khối") với độ dài nào đó trong một lần. Phụ thuộc vào phương thức thực hiện, mật mã khối có thể được thực hiện như là mật mã luồng tự đồng bộ (chế độ CFB). Tương tự, mật mã luồng có thể làm để nó hoạt động trên các khối riêng rẽ của văn bản thường tại một thời điểm. Vì thế, ở đây tồn tại sự đối ngẫu giữa hai cách thức này. Các mật mã khối như **DES**, **IDEA** và **AES**, và mật mã luồng như **RC4**, là những loại mật mã khóa đối xứng nổi tiếng nhất.

Ưu và nhược điểm?

Các thuật toán đối xứng nói chung đòi hỏi công suất tính toán ít hơn các thuật toán khóa bất đối xứng (*asymmetric key algorithms*). Trên thực tế, một thuật toán khóa bất đối xứng có khối lượng tính toán nhiều hơn gấp hàng trăm, hàng ngàn lần một thuật toán khóa đối xứng (symmetric key algorithm) có chất lượng tương đương.

Bảng dưới đây liệt kê một số ví dụ về thời gian phá mã trung bình tương ứng với kích thước của khóa.



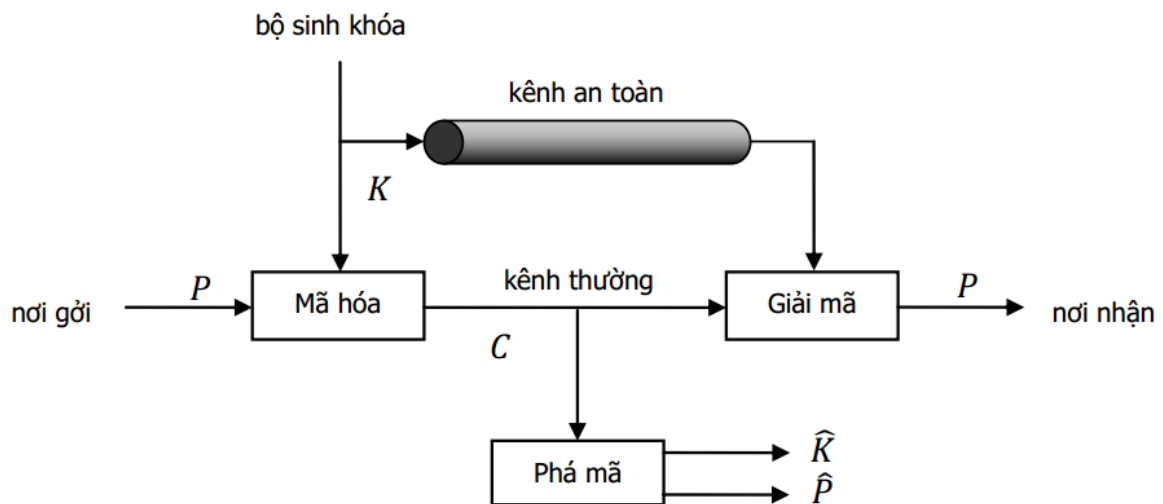
Kích thước khóa (bít)	Số lượng khóa	Thời gian thực hiện (tốc độ thử: 10^3 khóa/giây)	Thời gian thực hiện (tốc độ thử: 10^9 khóa/giây)
32	$2^{32} \approx 4.3 \times 10^9$	35.8 phút	2.15 mili giây
56	$2^{56} \approx 7.2 \times 10^{16}$	1142 năm	10.01 giờ
128	$2^{128} \approx 3.4 \times 10^{38}$	5.4×10^{24} năm	5.4×10^{18} năm
168	$2^{168} \approx 3.7 \times 10^{50}$	5.9×10^{36} năm	5.9×10^{30} năm
hoán vị 26 ký tự	$26! \approx 4 \times 10^{26}$	6.4×10^{12} năm	6.4×10^6 năm

Hệ thống mật mã khóa bí mật đối xứng có những nhược điểm lớn trên phương diện quản lý và lưu trữ, đặc biệt bộc lộ rõ trong thế giới hiện đại khi liên lạc qua Internet đã rất phát triển. Nếu như trong thế giới trước kia liên lạc mật mã chỉ hạn chế trong lĩnh vực quân sự hoặc ngoại giao thì ngày nay các đối tác doanh nghiệp khi giao dịch qua Internet đều mong muốn bảo mật các thông tin quan trọng. Với hệ thống khóa bí mật, số lượng khóa bí mật mà mỗi công ty hay cá nhân cần thiết lập với các đối tác khác có thể khá lớn và do đó rất khó quản lý lưu trữ an toàn các thông tin khóa riêng biệt này.

Một khó khăn đặc thù khác nữa là vấn đề xác lập và phân phối khóa bí mật này giữa hai bên, thường là đang ở xa nhau và chỉ có thể liên lạc với nhau qua một kênh truyền tin thông thường, không đảm bảo tránh được nghe trộm. Với hai người ở xa cách nhau và thậm chí chưa từng biết nhau từ trước thì làm sao có thể có thể thiết lập được một bí mật chung (tức là khóa) nếu không có một kênh bí mật từ trước (mà điều này đồng nghĩa với tồn tại khóa bí mật chung)? Có vẻ như chẳng có cách nào ngoài sử dụng “thần giao cách cảm” để hai người nay có thể trao đổi, thiết lập một thông tin bí mật chung?

Đây là một thách thức lớn đối với hệ thống mật mã khóa đối xứng. Tuy nhiên sẽ thấy câu hỏi này có thể được trả lời bằng giao thức mật mã thiết lập khóa.

2.1.2 Mô hình mã hóa đối xứng



Mô hình gồm 5 yếu tố:

- Bản rõ P (plaintext)
- Thuật toán mã hóa E (encrypt algorithm)
- Khóa bí mật K (secret key)
- Bản mã C (ciphertext)
- Thuật toán giải mã D (decrypt algorithm)

Trong đó: $C = E(P, K)$

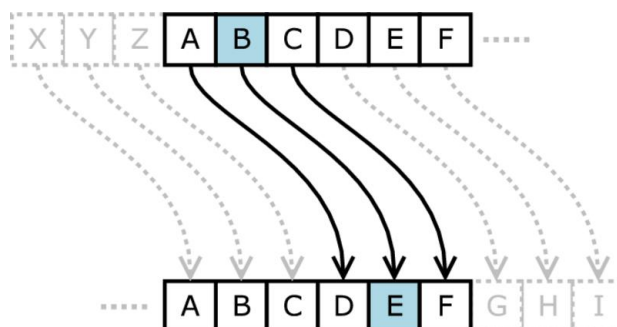
$P = D(C, K)$

Thuật toán mã hóa và giải mã sử dụng chung một khóa, thuật toán giải mã là phép toán ngược của thuật toán mã hóa.

2.2 Một số hệ mã đối xứng

2.2.1 Caesar cipher

2.2.1.1 Giới thiệu





Thế kỷ thứ 3 trước công nguyên, nhà quân sự người La Mã Julius Ceasar đã nghĩ ra phương pháp mã hóa một bản tin như sau: thay thế mỗi chữ trong bản tin bằng chữ đứng sau nó k vị trí trong bảng chữ cái. Giả sử chọn $k = 3$, ta có bảng chuyển đổi như sau:

Chữ ban đầu: 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
 Chữ thay thế: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
 (sau Z sẽ vòng lại là A, do đó $x \rightarrow A$, $y \rightarrow B$ và $z \rightarrow C$)

Giả sử có bản rõ : HOM NAY TROI DEP KINH Y

Bản mã sẽ là : KRP QDB WURL GHS NLQK B

Thay vì gửi trực tiếp bản rõ cho các cấp dưới, Caesar gửi bản mã. Khi cấp dưới nhận được bản mã, tiến hành giải mã theo quy trình ngược lại để có được bản rõ. Như vậy nếu đối thủ của Caesar có lấy được bản mã, thì cũng không hiểu được ý nghĩa của bản mã.

Chúng ta hãy gán cho mỗi chữ cái một con số nguyên từ 0 đến 25:

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Phương pháp Ceasar được biểu diễn như sau: với mỗi chữ cái **p** thay bằng chữ mã hóa **C**, trong đó:

$$C = (p + k) \bmod 26 \quad (\text{trong đó mod là phép chia lấy số dư})$$

Và quá trình giải mã đơn giản là:

$$p = (C - k) \bmod 26$$

k được gọi là khóa. Dĩ nhiên là Caesar và cấp dưới phải cùng dùng chung một giá trị khóa **k**, nếu không bản tin giải mã sẽ không giống bản rõ ban đầu.



2.2.1.2 Thảm mã Caesar

Phương pháp mã hóa Caesar là 1 phương pháp mã hóa cổ điển, vô cùng đơn giản. Ngày nay phương pháp mã hóa của Ceasar không được xem là an toàn. Giả sử đối thủ của Ceasar có được bản mã *KRP QDB WURL GHS NLQK B* và biết được phương pháp mã hóa và giải mã là phép cộng trừ modulo 26. Đối thủ có thể thử tất cả 25 trường hợp của khóa k. Sau khi thử, dễ dàng tìm được khóa đã sử dụng.

```
k = 19 : DKI JWU PNKE ZAL GEJD U
k = 20 : ELJ KXV QOLF ABM HFKE V
k = 21 : FMK LYW RPMG BCN IGLF W
k = 22 : GNL MZX SQNH CDO JHMG X
k = 23 : HOM NAY TROI DEP KINH Y
k = 24 : IPN OBZ USPJ EFQ LJOI Z
```

2.2.1.3 Chương trình ví dụ

Chương trình cài đặt thuật toán Caesar cơ bản, sử dụng ngôn ngữ *python* :

```
abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

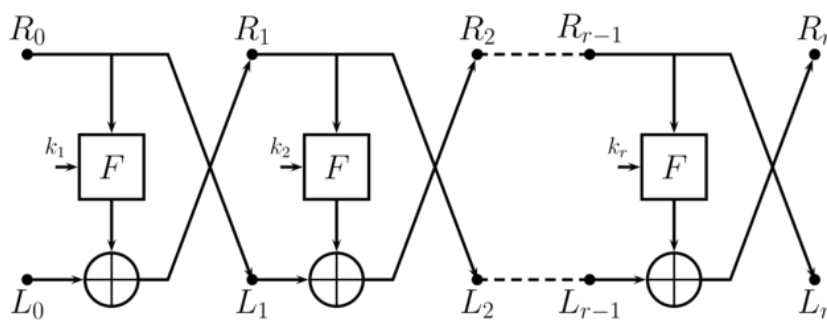
def caesar(plaintext, key):
    ciphertext = ""
    for letter in plaintext:
        if letter != " ":
            pos = abc.index(letter) + key
            pos %= 26
            ciphertext += abc[pos]
        else:
            ciphertext += " "
    return ciphertext

plainText = "KRP QDB WURL GHS NLQK B"
for key in range(1, 26):
    print("k = ", key, ": ", caesar(plainText, key))
```


hóa cùng một lúc. Trong mật mã khối, các tham số quan trọng là kích thước (độ dài khối) và kích thước khóa.

➤ *Kích thước khối* phải đủ lớn để chống lại các loại tấn công phá hoại bằng phương pháp thống kê. Tuy nhiên cần lưu ý rằng kích thước khối lớn sẽ làm thời gian trễ lớn.

➤ *Không gian khóa* phải đủ lớn (tức là chiều dài khóa phải đủ lớn) để chống lại tìm kiếm vét cạn. Tuy nhiên mật khác, khóa cần phải đủ ngắn để việc làm khóa, phân phối và lưu trữ được hiệu quả.



Một cách phổ biến, các hệ mã khối thường được thiết kế theo cấu trúc nhiều vòng lặp với mỗi vòng lặp lại gọi thực hiện một hàm f cơ sở (nhưng với các tham số khác nhau). Theo đó, đầu vào của một vòng lặp là đầu ra của vòng lặp trước và một khóa con phát sinh từ khóa đầy đủ dựa trên một thuật toán lập lịch khóa (key scheduler), hay cũng gọi là thuật toán sinh khóa con.

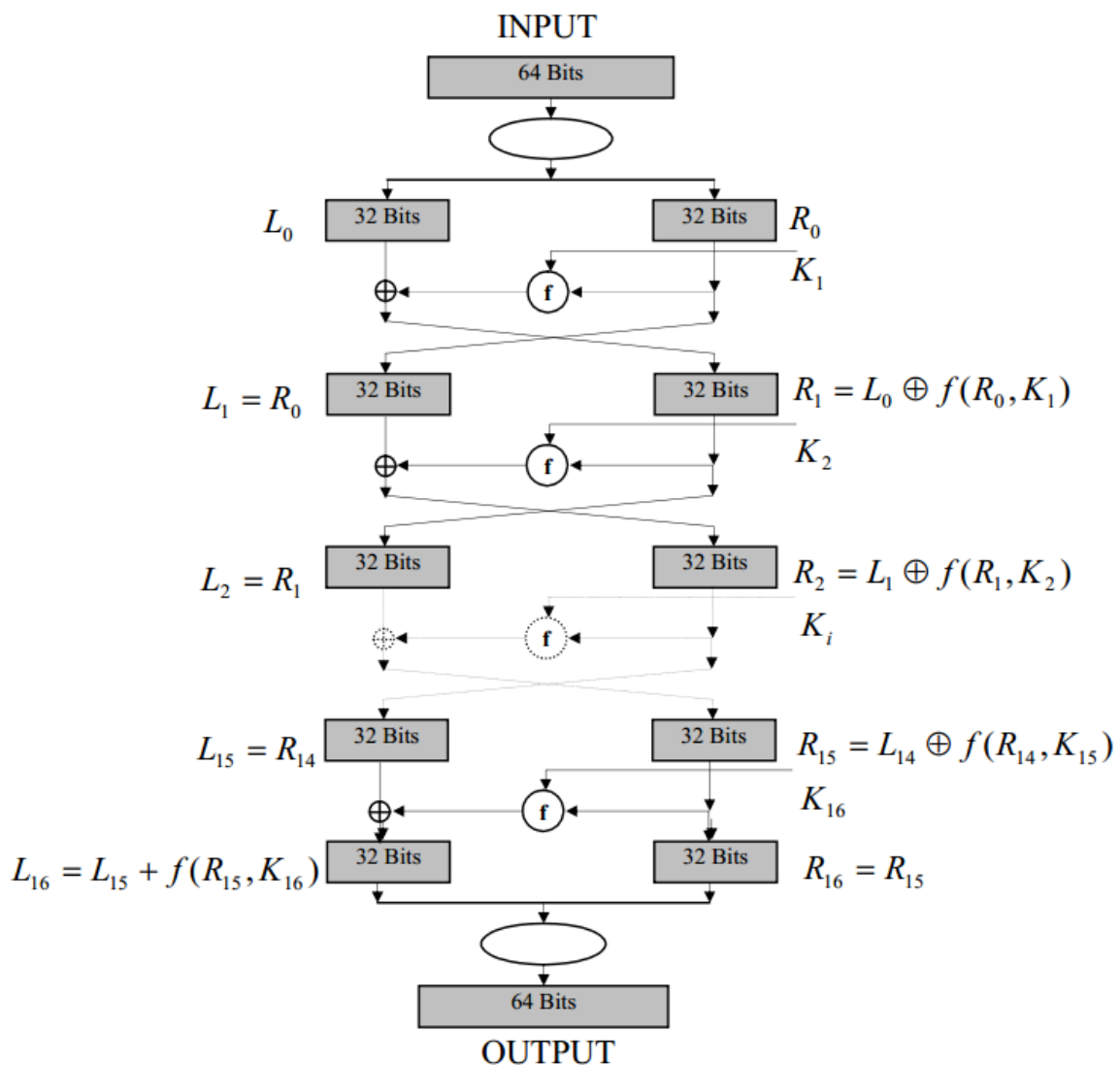
Chúng ta sẽ tìm hiểu một hệ mã khối điển hình, đó là chuẩn mật mã DES (*Data Encryption Standard*); chuẩn này ra đời vào năm 1977 và đã thống trị ứng dụng mật mã suốt 2 thập kỷ sau đó. Tuy nhiên chuẩn mật mã này đã trở nên lạc hậu, kém an toàn và được thay thế bởi chuẩn mới AES (*Advanced Encryption Standard*).

2.2.3.2 Lịch sử của DES

Năm 1973, Cục quản lý các chuẩn quốc gia của Mỹ đã có văn bản cổ động cho việc tạo lập các hệ mật mã chuẩn ở cơ quan đăng ký liên bang của Mỹ. Điều này đã dẫn đến sự công bố vào năm 1977 của cục An ninh Quốc gia Mỹ (NSA) về

Data Encryption Standard, viết tắt là DES. Thực chất, DES được phát triển bởi IBM như là sự sửa đổi của một hệ mã trước kia được biết với cái tên *Lucipher*. Trong khoảng 2 thập kỷ tiếp theo, DES là hệ mã được dùng rộng rãi nhất và cũng là gây ra nhiều nghi ngờ, tranh cãi trong lĩnh vực này: xung quanh các nguyên tắc thiết kế đảm bảo tính mật, chiều dài khóa tương đối ngắn và khả năng NSA còn che giấu cửa sau (backdoor) để có thể bẻ khóa, phá mã ít tốn kém hơn thông thường.

2.2.3.3 Thuật toán và lưu đồ hoạt động



16 vòng lặp của DES cùng gọi thực hiện f nhưng với các tham số khóa khác nhau. Tất cả 16 khóa khác nhau này, được gọi là **khóa con**, cùng sinh ra từ khóa chính của DES bằng một *thuật toán sinh khóa con*.



Thuật toán giải mã DES được xây dựng giống hệt như thuật toán sinh mã nhưng có các khóa con được sử dụng theo thứ tự ngược lại, tức là dùng khóa K16 cho vòng lặp 1, khóa K15 cho vòng lặp 2 ...

2.2.3.4 Độ an toàn của DES

Ta hãy xem xét tính an toàn của DES trước một vài phương pháp tấn công phá mã.

➤ Tấn công vét cạn khóa (Brute Force Attack)

Vì khóa của mã DES có chiều dài là 56 bit nên để tiến hành *brute-force attack*, cần kiểm tra 2^{56} khóa khác nhau. Hiện nay với những thiết bị phổ dụng, thời gian gian để thử khóa là rất lớn nên việc phá mã là không khả thi. Tuy nhiên vào năm 1998, tổ chức *Electronic Frontier Foundation* (EFF) thông báo đã xây dựng được một thiết bị phá mã DES gồm nhiều máy tính chạy song song, trị giá khoảng 250.000\$. Thời gian thử khóa là 3 ngày.

Hiện nay mã DES vẫn còn được sử dụng trong thương mại, tuy nhiên người ta đã bắt đầu áp dụng những phương pháp mã hóa khác có chiều dài khóa lớn hơn (128 bit hay 256 bit).

➤ Phá mã DES theo phương pháp vi sai (differential cryptanalysis)

Năm 1990 *Biham* và *Shamir* đã giới thiệu phương pháp phá mã vi sai. Phương pháp vi sai tìm khóa ít tốn thời gian hơn brute-force. Tuy nhiên phương pháp phá mã này lại đòi hỏi phải có 2^{47} cặp bản rõ - bản mã được lựa chọn (*chosen-plaintext*). Vì vậy phương pháp này là bất khả thi dù rằng số lần thử có thể ít hơn phương pháp brute-force.

➤ Phá mã DES theo phương pháp thử tuyến tính (linear cryptanalysis)

Năm 1997 *Matsui* đưa ra phương pháp phá mã tuyến tính. Trong phương pháp này, cần phải biết trước 2^{43} cặp bản rõ-bản mã (*known-plaintext*). Tuy nhiên

2^{43} cũng là một con số lớn nên phá mã tuyến tính cũng không phải là một phương pháp khả thi.

2.2.4 Triple DES & AES

Triple DES

Một trong những cách để khắc phục yếu điểm kích thước khóa ngắn của mã hóa DES là sử dụng mã hóa DES nhiều lần với các khóa khác nhau cho cùng một bản tin. Đơn giản nhất là dùng DES **hai lần** với **hai khóa khác nhau**, cách thức này được gọi là *Double DES*:

$$C = E(E(P, K_1), K_2)$$

Điều này giống như là Double DES dùng một khóa có kích thước là 112 byte, chỉ có một hạn chế là tốc độ chậm hơn DES vì phải dùng DES hai lần. Tuy nhiên người ta đã tìm được một phương pháp tấn công Double DES có tên gọi là gặp-nhau-ở-giữa (meet-in-the-middle). Đây là một phương pháp tấn công *chosen-plaintext*.

Vì vậy người ta chọn dùng DES ba lần với ba khóa khác nhau, cách thức này được gọi là **Triple DES**:

$$C = E(E(E(P, K_1), K_2), K_3)$$

Chiều dài khóa là 168 bit sẽ gây phức tạp hơn nhiều cho việc phá mã bằng phương pháp tấn công gặp-nhau-ở-giữa. Trong thực tế người ta chỉ dùng Triple DES với hai khóa K_1, K_2 mà vẫn đảm bảo độ an toàn cần thiết. Công thức như sau:

$$C = E(D(E(P, K_1), K_2), K_1)$$

Nguyên nhân của việc dùng EDE thay cho EEE là nếu với $K_1 = K_2 = K$ thì

$$C = E(D(E(P, K), K), K) = E(P, K)$$

Nghĩa là Triple DES suy giảm thành một DES đơn.



Advanced Encryption Standard (AES)

Vào những năm 1990, nhận thấy nguy cơ của mã hóa DES là kích thước khóa ngắn, có thể bị phá mã trong tương lai gần, Cục tiêu chuẩn quốc gia Hoa Kỳ đã kêu gọi xây dựng một phương pháp mã hóa mới. Cuối cùng một thuật toán có tên là Rijndael được chọn và đổi tên thành Advanced Encryption Standard hay AES. Có thể nói rằng mã hóa AES với khóa có kích thước 256 bit là “an toàn mãi mãi” bất kể những tiến bộ trong ngành kỹ thuật máy tính.

AES được xây dựng trên nguyên lý thiết kế lưới giao hoán – thay thế (*substitution-permutation network*). Đây là một hệ mã có tốc độ tốt trong cả cài đặt phần mềm cũng như phần cứng. Khác với DES, AES không theo mẫu thiết kế mạng Feistel. Thay vào đó các thao tác cơ bản được thực hiện trên các khối ma trận dữ liệu 4×4 (bytes), được gọi là các trạng thái (state). Số vòng lặp của AES là một tham số xác định trên cơ sở kích thước khóa: 10 vòng lặp cho khóa 128 bit, 12 cho 192 bit, 14 cho 256 bit.

Độ an toàn của AES làm cho AES được sử dụng ngày càng nhiều và trong tương lai sẽ chiếm vai trò của DES và Triple DES.

CHƯƠNG III: MÃ HÓA BẤT ĐỐI XỨNG

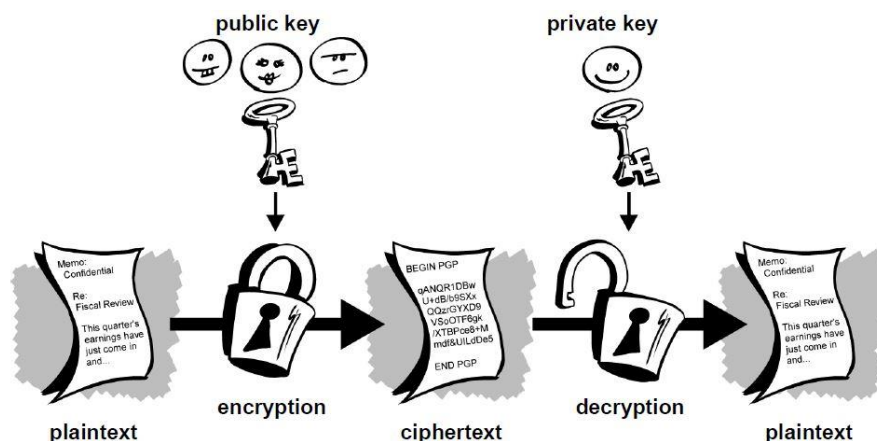
3.1 Giới thiệu

3.1.1 Mã hóa bất đối xứng

Mã hóa đối xứng dù rằng đã phát triển từ cổ điển đến hiện đại, vẫn tồn tại hai điểm yếu sau:

- Vấn đề trao đổi khóa giữa người gửi và người nhận: Cần phải có một kênh an toàn để trao đổi khóa sao cho khóa phải được giữ bí mật chỉ có người gửi và người nhận biết. Điều này tỏ ra không hợp lý khi mà ngày nay, khối lượng thông tin luân chuyển trên khắp thế giới là rất lớn. Việc thiết lập một kênh an toàn như vậy sẽ tốn kém về mặt chi phí và chậm trễ về mặt thời gian.
- Tính bí mật của khóa: không có cơ sở quy trách nhiệm nếu khóa bị tiết lộ.

Ý tưởng về các hệ thống mật mã loại này mới chỉ ra đời vào giữa những năm bảy mươi của thế kỷ 20. Khác cơ bản với mã hóa đối xứng, trong mô hình mới này 2 khóa của thuật toán sinh mã và giải mã là khác nhau và từ thông tin khóa sinh mã, mặc dù trên lý thuyết là có thể tìm được khóa giải mã (có thể thử vét cạn) nhưng khả năng thực tế của việc này là hầu như bằng không (bất khả thi về khối lượng tính toán).



Ý tưởng mới này cho phép mỗi thực thể cá nhân công ty chỉ cần tạo ra cho mình một cặp khóa, với hai thành phần:



- Thành phần *khóa công khai*, có thể đăng ký phổ biến rộng khắp, dùng để sinh mã hoặc để xác thực chữ ký điện tử.
- Thành phần *khóa bí mật*, chỉ dành riêng cho bản thân, dùng để giải mã hoặc tạo ra chữ ký điện tử.

Chỉ với cặp khóa này, thực thể chủ có thể giao dịch bảo mật với quảng đại xã hội, trong đó việc quản lý và lưu trữ có thể được tổ chức chặt chẽ mà việc phải tự nhớ thông tin mật là tối thiểu (giống như việc chỉ nhớ 1 mật khẩu hay một số PIN tài khoản ngân hàng).

Thông thường, các kỹ thuật mật mã hóa khóa công khai đòi hỏi khối lượng tính toán nhiều hơn các kỹ thuật mã hóa khóa đối xứng nhưng những lợi ích mà chúng mang lại khiến cho chúng được áp dụng trong nhiều ứng dụng.

Có nhiều phương pháp mã hóa thuộc loại mã hóa khóa công khai. Đó là các phương pháp *Knapsack*, *RSA*, *Elgamal*, và *phương pháp đường cong elliptic ECC*... Mỗi phương pháp có cách thức ứng dụng hàm một chiều khác nhau.

3.1.2 Lịch sử

Thuật toán mật mã hóa khóa công khai được thiết kế đầu tiên bởi *James H. Ellis*, *Clifford Cocks*, và *Malcolm Williamson* tại GCHQ (Anh) vào đầu thập kỷ 1970. Thuật toán sau này được phát triển và biết đến dưới tên *Diffie-Hellman*, và là một trường hợp đặc biệt của RSA. Tuy nhiên những thông tin này chỉ được tiết lộ vào năm 1997. Trao đổi khóa Diffie-Hellman là phương pháp có thể áp dụng trên thực tế đầu tiên để phân phối khóa bí mật thông qua một kênh thông tin không an toàn.

Thuật toán đầu tiên cũng được *Rivest*, *Shamir* và *Adleman* tìm ra vào năm 1977 tại MIT. Công trình này được công bố vào năm 1978 và thuật toán được đặt tên là **RSA**. RSA sử dụng phép toán tính hàm mũ môđun (môđun được tính bằng tích số của 2 số nguyên tố lớn) để mã hóa và giải mã cũng như tạo chữ ký số.

Kể từ thập kỷ 1970, đã có rất nhiều thuật toán mã hóa, tạo chữ ký số, thỏa thuận khóa... được phát triển. Các thuật toán như *ElGamal* (mật mã) do *Netscape* phát triển hay *DSA* do *NSA* và *NIST* cũng dựa trên các bài toán lôgarit rời rạc tương tự như *RSA*. Vào giữa thập kỷ 1980, *Neal Koblitz* bắt đầu cho một dòng thuật toán mới: *mật mã đường cong elliptic* và cũng tạo ra nhiều thuật toán tương tự. Mặc dù cơ sở toán học của dòng thuật toán này phức tạp hơn nhưng lại giúp làm giảm khối lượng tính toán đặc biệt khi khóa có độ dài lớn.

3.1.3 Độ an toàn

Về khía cạnh an toàn, các thuật toán mật mã hóa khóa bất đối xứng cũng không khác nhiều với các thuật toán mã hóa khóa đối xứng. Có những thuật toán được dùng rộng rãi, có thuật toán chủ yếu trên lý thuyết; có thuật toán vẫn được xem là an toàn, có thuật toán đã bị phá vỡ... Cũng cần lưu ý là những thuật toán được dùng rộng rãi không phải lúc nào cũng đảm bảo an toàn. Một số thuật toán có những chứng minh về độ an toàn với những tiêu chuẩn khác nhau. Nhiều chứng minh gắn việc phá vỡ thuật toán với những bài toán nổi tiếng vẫn được cho là không có lời giải trong thời gian đa thức. Nhìn chung, chưa có thuật toán nào được chứng minh là an toàn tuyệt đối. Vì vậy, cũng giống như tất cả các thuật toán mật mã nói chung, các thuật toán mã hóa khóa công khai cần phải được sử dụng một cách thận trọng.

Các thuật toán khóa công khai thông thường dựa trên các vấn đề toán học với độ khó **NP**. Ví dụ *RSA*, dựa trên độ khó (ước đoán) của bài toán *phân tích ra thừa số nguyên tố*. Vì lý do thuận tiện, các hệ thống mật mã hóa lai ghép được sử dụng trong thực tế; khóa được trao đổi thông qua mật mã khóa công khai, và phần còn lại của thông tin được mật mã hóa bằng cách sử dụng thuật toán khóa đối xứng (điều này về cơ bản là nhanh hơn).



3.1.4 Ứng dụng

Ứng dụng rõ ràng nhất của mật mã hóa khóa công khai là *bảo mật*: một văn bản được mã hóa bằng khóa công khai của một người sử dụng thì chỉ có thể giải mã với khóa bí mật của người đó.

Mật mã hóa bất đối xứng cũng cung cấp cơ chế cho *chữ ký số*, là cách thức để xác minh với độ bảo mật cao rằng thông điệp mà người nhận đã nhận được là chính xác được gửi đi từ phía người gửi mà họ yêu cầu. Sử dụng hợp thức các thiết kế có chất lượng cao và các bổ sung khác tạo ra khả năng có được độ an toàn cao, làm cho chữ ký điện tử vượt qua phần lớn các chữ ký thật cẩn thận nhất về mức độ thực của nó (khó bị giả mạo hơn). Các ví dụ về các giao thức chữ ký số hóa bao gồm **DSA** và chữ ký **ElGamal**. Các chữ ký số hóa là trung tâm trong các hoạt động của hạ tầng khóa công cộng (*PKI*) và rất nhiều hệ thống an ninh mạng (ví dụ *Kerberos*, phần lớn các mạng riêng ảo (*VPN*)...).

Mật mã khóa bất đối xứng cũng cung cấp nền tảng cho các kỹ thuật khóa thỏa thuận xác thực mật khẩu (*PAKA*) và không kỹ năng kiểm chứng mật khẩu (*ZKPP*). Điều này là quan trọng khi xét theo phương diện của các chứng minh lý thuyết và kinh nghiệm rằng việc xác thực chỉ bằng mật khẩu sẽ không đảm bảo an toàn trên mạng chỉ với khóa mật mã đối xứng và các hàm băm.

3.2 Một số hệ mã hóa bất đối xứng

3.2.1 Mã hóa RSA

3.2.1.1 Giới thiệu

Phương pháp RSA là một phương pháp mã hóa khóa công khai. RSA được xây dựng bởi các tác giả *Ron Rivest*, *Adi Shamir* và *Len Adleman* tại học viện MIT vào năm 1977, và ngày nay đang được sử dụng rộng rãi. Về mặt tổng quát RSA là một phương pháp mã hóa theo khối.

Cơ sở thuật toán RSA dựa trên tính khó của bài toán phân tích các số lớn ra thừa số nguyên tố: không tồn tại thuật toán thời gian đa thức (theo độ dài của biểu diễn nhị phân của số đó) cho bài toán này. Chẳng hạn, việc phân tích một



hợp số là tích của 2 số nguyên tố lớn hàng trăm chữ số sẽ mất hàng ngàn năm tính toán với một máy PC trung bình có CPU khoảng trên 2Ghz.

3.2.1.2 Thuật toán

Để thực hiện mã hóa và giải mã, RSA dùng phép lũy thừa modulo của lý thuyết số. Các bước thực hiện như sau:

- 1) Chọn hai số nguyên tố lớn p và q và tính $N = pq$. Cần chọn p và q sao cho:
 $M < 2^{i-1} < N < 2^i$. Với $i = 1024$ thì N là một số nguyên dài khoảng 309 chữ số.
- 2) Tính $n = (p - 1)(q - 1)$
- 3) Tìm một số e sao cho e nguyên tố cùng nhau với n
- 4) Tìm một số d sao cho $e \cdot d \equiv 1 \pmod{n}$ (d là nghịch đảo của e trong phép modulo n)
- 5) Hủy bỏ n , p và q . Chọn khóa công khai K_U là cặp (e, N) , khóa riêng K_R là cặp (d, N)
- 6) Việc mã hóa thực hiện theo công thức: $C = E(M, K_U) = M^e \pmod{N}$
- 7) Việc giải mã thực hiện theo công thức: $\bar{M} = D(M, K_R) = M^d \pmod{N}$

Bản rõ M có kích thước $i-1$ bit, bản mã C có kích thước i bit.

Để minh họa ta sẽ thực hiện một ví dụ về mã hóa RSA với kích thước khóa là 6 bit.

- 1) Chọn $p = 11$ và $q = 3$, do đó $N = pq = 33$
- 2) $n = (p-1)(q-1) = 20$
- 3) Chọn $e = 3$ nguyên tố cùng nhau với n
- 4) Tính nghịch đảo của e trong phép modulo n được $d = 7$ ($3 \times 7 = 21$)
- 5) Khóa công khai $K_U = (e, N) = (3, 33)$. Khóa bí mật $K_R = (d, N) = (7, 33)$



6) Mã hóa bản rõ $M = 15$:

$$C = E(M, K_U) = M^e \bmod N = 15^3 \bmod 33 = 9$$

7) Giải mã bản mã $C = 9$:

$$\bar{M} = D(M, K_R) = M^d \bmod N = 9^7 \bmod 33 = 15$$

3.2.1.3 Đánh giá RSA

So sánh với DES thì RSA

+ Có tốc độ chậm hơn rất nhiều. Thường thì, RSA chậm ít nhất là 100 lần khi cài đặt bằng phần mềm, và có thể chậm hơn từ 1000 đến 10,000 lần khi cài đặt bằng phần cứng (còn tùy cách cài đặt)

+ Kích thước của khoá mật lớn hơn rất nhiều.

Nếu như p và q cần biểu diễn cỡ 300 bits thì n cần 600 bits. Phép nâng lên lũy thừa là khá chậm so với n lớn, đặc biệt là nếu sử dụng phần mềm (chương trình).

Vấn đề đi tìm số nguyên tố lớn

Một thuật toán để tạo ra tất cả các số nguyên tố là không tồn tại, tuy nhiên có những thuật toán khá hiệu quả để kiểm tra xem một số cho trước có phải là nguyên tố hay không (bài toán kiểm tra tính nguyên tố).

Những thuật toán tắt định để kiểm tra tính nguyên tố là khá tốn thời gian và đòi hỏi được thực hiện trên máy tính có tốc độ cao. Tuy nhiên người ta cũng còn sử dụng các thuật toán xác suất, có khả năng ‘đoán’ rất nhanh xem một số có phải nguyên tố không. Các thuật toán xác suất này không đưa ra quyết định đúng tuyệt đối, nhưng cũng gần như tuyệt đối; tức là xác suất báo sai có thể làm nhỏ tùy ý, chỉ phụ thuộc vào thời gian bỏ ra.

Đánh giá về an toàn của thuật toán RSA



Sự an toàn của thành phần khoá mật (private key) phụ thuộc vào tính khó của việc phân tích thành thừa số nguyên tố các số lớn.

Sau đây ta sẽ xem xét một số các tấn công phương pháp RSA.

1) Vết cạn khóa: cách tấn công này thử tất cả các khóa d có thể có để tìm ra bản giải mã có ý nghĩa, tương tự như cách thử khóa K của mã hóa đối xứng. Với N lớn, việc tấn công là bất khả thi.

2) Phân tích N thành thừa số nguyên tố $N = pq$: Chúng ta đã nói rằng việc phân tích phải là bất khả thi thì mới là hàm một chiều, là nguyên tắc hoạt động của RSA. Tuy nhiên, nhiều thuật toán phân tích mới đã được đề xuất, cùng với tốc độ xử lý của máy tính ngày càng nhanh, đã làm cho việc phân tích N không còn quá khó khăn như trước đây. Người ta cho rằng kích thước của N phải khoảng **1024 bit** (309 chữ số) thì mới bảo đảm an toàn thật sự.

3) Một số dạng tấn công có điều kiện quan trọng: đối với một số hệ cài đặt rơi vào một số *điều kiện đặc biệt* có thể trở nên kém an toàn với người sử dụng.

+ *Common modulus attack*: Khi một nhóm user sử dụng các khóa công khai $Z=(e,n)$ khác nhau ở thành phần e nhưng giống nhau ở modul đồng dư n .

+ *Low exponent attack*: Tấn công này xảy ra với điều kiện là giá trị e đã được chọn nhỏ (e mà nhỏ thì thuật toán mã hoá trong truyền tin mật cũng như kiểm định chữ ký sẽ nhanh hơn).

+ *Low decryption attack*: Nếu thành phần khóa mật d mà đủ nhỏ thì có thể bị kẻ thù tìm thấy được.

3.2.1.4 Cài đặt

Trong phạm vi đề tài em đã xây dựng chương trình cài đặt thuật toán mã hóa RSA. Chương trình viết bằng ngôn ngữ C# trên nền .Net Framework, có giao diện thân thiện, trực quan, dễ sử dụng.

- Tự xây dựng thư viện cấu trúc dữ liệu biểu diễn số lớn **BigNum** và các phép toán cơ bản kèm theo: Lưu số lớn dưới dạng mảng các số nguyên nhỏ, **xử lý dịch bit** để tăng tốc độ cho các phép tính (+, -, *, /, module).

- Giai đoạn sinh số nguyên tố lớn p, q: Sinh ngẫu nhiên số lớn rồi kiểm tra tính nguyên tố bằng thuật toán Miller-Rabin.

```
bool CheckPrime(BigNum num)
{
    foreach (var item in lowPrimes)
    {
        if (num == item)
            return true;
        if ((num % new BigNum(item)).GetSize == 0)
            return false;
    }
    for (int i = 0; i < 5; i++)    // Miller Rabin test 5 times
    {
        BigNum a = RandomNumber(num);
        if (MillerRabin(a, num - 1, num) != 1)
            return false;
    }
    return true;
}
/// <summary>
/// Miller Rabin Primality Test
/// return a^x % m == 1?
/// </summary>
/// <param name="a"></param>
/// <param name="x"></param>
/// <param name="m"></param>
/// <returns></returns>
BigNum MillerRabin(BigNum a, BigNum x, BigNum m)
{
    if ((x.num[0] & 1) != 1)    // x is even?
    {
        BigNum res = MillerRabin(a, x >> 1, m);    // Recursive
        if (res == 1 || res == m - 1)
            return new BigNum(1);
        else
            return (res * res) % m;
    }
    else    // x is odd
        return Module(a, x, m);
}
```

- Xây dựng các hàm tính module, Euclid mở rộng, ước chung lớn nhất...

```
public static BigNum GCD(BigNum num1, BigNum num2)
{
    BigNum tmp = num1 % num2;
    num1 = num2;
```

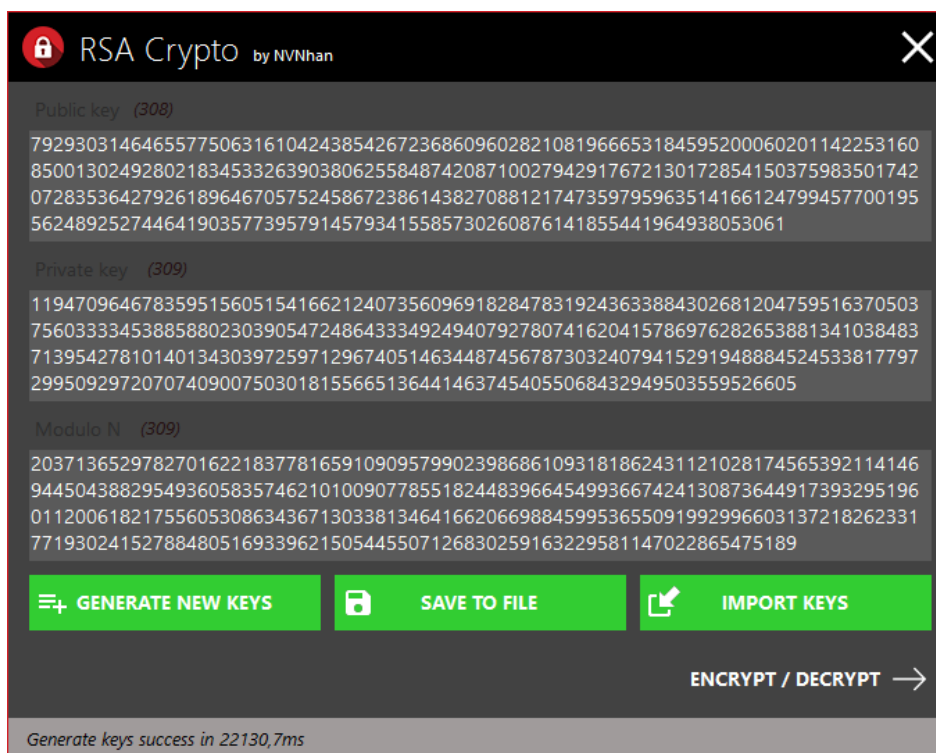
```

        num2 = tmp;
        if (num2.GetSize == 0)
            return num1;
        return GCD(num1, num2);
    }
    /// <summary>
    /// Calculate  $a^x \bmod m$ 
    /// </summary>
    /// <param name="a"></param>
    /// <param name="x"></param>
    /// <param name="m"></param>
    /// <returns></returns>
    public static BigNum Module(BigNum a, BigNum x, BigNum m)
    {
        if (x.GetSize == 0)
            return new BigNum(1);
        else if ((x.num[0] & 1) != 1) //  $a^{2c} \bmod n = (a^c \bmod n)^2 \bmod n$ 
        {
            var uAux = Module(a, x >> 1, m);
            return (uAux * uAux) % m;
        }
        else return (Module(a, x - new BigNum(1), m) * a) % m; //  $a^{c+1} \bmod n = (a^c \bmod n) \cdot a \bmod n$ 
    }
    /// <summary>
    /// Modular Inverse
    /// Extend Modulo
    /// </summary>
    void CalculatePrivateKey()
    {
        BigNum u0 = new BigNum(1), u1 = new BigNum(0);
        BigNum b = o, a = PublicKey;
        BigNum tmp = new BigNum(), tmp1 = new BigNum();
        BigNum tmpu = new BigNum();
        while (b.GetSize > 0)
        {
            tmp = a / b;
            tmp1 = a % b; a = b; b = tmp1; // gcd (a, b)
            tmpu = u0 - tmp * u1; u0 = u1; u1 = tmpu; // Recursive
        }
        if (u0.IsNegative)
            u0 = u0 + o;
        PrivateKey = u0;
        if (PrivateKey == 1)
            CalculatePrivateKey();
    }
}

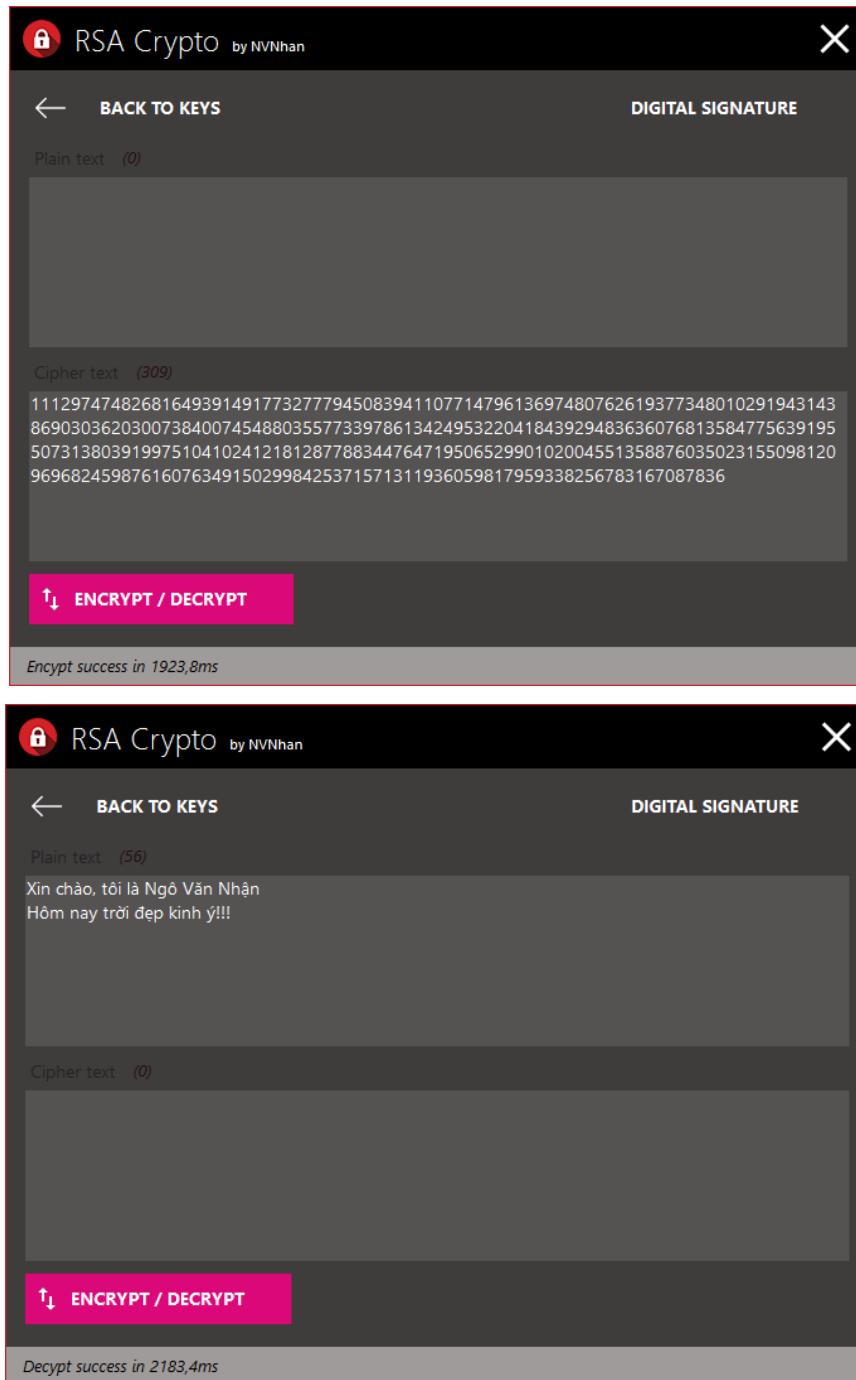
```

- Chuyển các thông điệp cần mã hóa thành số lớn tương ứng và ngược lại.

Giao diện chương trình



Giao diện sinh khóa của chương trình



Giao diện mã hóa & giải mã

Hiệu năng thực tế

Vì do hạn chế thời gian làm tiểu luận cũng như chưa được tối ưu nên phần sinh khóa của chương trình khi chạy ở máy tính cá nhân (*Dell Vostro 3560 i3 4GB Win 10 64bit*) là mất khoảng 20-25s với khóa 1024 bit (~309 chữ số); mã hóa và giải mã 1 khối (128 bit) mất khoảng 1-2s.



3.2.2 Mã hóa ElGamal

3.2.2.1 Giới thiệu

Hệ mật ElGamal được đề xuất vào năm 1984 trên cơ sở bài toán logarith rời rạc. Sau đó, các chuẩn chữ ký số DSS của Mỹ và GOST R34.10-94 của Liên bang Nga đã được phát triển trên cơ sở thuật toán chữ ký số của hệ mật này, còn thuật toán mật mã khóa công khai ElGamal đã được sử dụng bởi Cơ quan An ninh Quốc gia Mỹ - NSA (*National Security Agency*).

Hệ mã hóa với khóa công khai ElGamal được đề xuất năm 1985, dựa vào độ phức tạp của bài toán lôgarit rời rạc.

3.2.2.2 Thuật toán

Quá trình sinh khóa:

1) Chọn số P là số nguyên tố; Q, k' ngẫu nhiên sao cho $Q < P-1, k' < P-2$

2) Tính $D = Q^{k'} \bmod P$

=> Khóa công khai: (P, Q, D)

Khóa bí mật: k'

Quá trình mã hóa:

3) Chọn K ngẫu nhiên sao cho: $1 < K < P-2$

4) Tính $C_1 = Q^K \bmod P, C_2 = (D^K \cdot m) \bmod P$

=> Bản mã: $C = (C_1, C_2)$

Quá trình giải mã:

5) Tính $R = C_1^{P-1-k'} \bmod P, M = (R \cdot C_2) \bmod P$

Ví dụ:

1) Chọn $P = 13, Q = 10, k' = 9$

2) $D = Q^{k'} \bmod P = 10^9 \bmod 13 = 12$



3) Chọn $K = 7, m = 11$

$$4) C_1 = Q^K \bmod P = 10^7 \bmod 13 = 10$$

$$C_2 = (D^K \cdot m) \bmod P = (12^7 \cdot 11) \bmod 13 = 2$$

$$5) R = C_1^{P-1-k'} \bmod P = 10^{13-1-9} \bmod 13 = 12$$

$$M = (R \cdot C_2) \bmod P = (12 \cdot 2) \bmod 13 = 11$$

3.2.2.3 Cài đặt

Chương trình cài đặt thuật toán mã hóa ElGamal, sử dụng các thư viện, hàm kế thừa đã được xây dựng ở chương trình mã hóa RSA. Ở đây là 1 đoạn của chương trình, mô phỏng hàm mã hóa và giải mã Elgamal:

```
void Encrypt(string plain)
{
    var byteArr = Encoding.UTF8.GetBytes(plain);
    string res1 = "", res2 = "";
    int len = byteArr.Length;
    int blocksize = 50;
    int byteToCopy = blocksize;

    byte[] tmp;
    BigNum bitmp;

    BigNum tmpk = rsa.RandomNumber(numP - 2);

    for (int i = 0, j = 1; i < len; i += blocksize, j++)
    {
        if (len - i < blocksize)
            byteToCopy = len - i;
        else
            byteToCopy = blocksize;
        tmp = new byte[byteToCopy];
        Array.Copy(byteArr, i, tmp, 0, byteToCopy);
        bitmp = new BigNum(tmp);
        ////
        res1 += RSACrypto.Classes.RSA.Module(numQ, tmpk, numP).ToString()
+ ";";
        res2 += ((RSACrypto.Classes.RSA.Module(numD, tmpk, numP) * bitmp)
% numP).ToString() + ";";
        /////
    }
    txtCipher1.Text = res1;
    txtCipher2.Text = res2;
}

void Decrypt(string cipher1, string cipher2)
{
    string[] blocks1 = Regex.Split(cipher1, ";");
    string[] blocks2 = Regex.Split(cipher2, ";");
```



```

List<byte> lstbytes = new List<byte>();

for (int i = 0; i < blocks1.Length; i++)
{
    if (blocks1[i] == "" || blocks2[i] == "")
        break;
    var c1 = new BigNum(blocks1[i]);
    var c2 = new BigNum(blocks2[i]);
    ///////
    var r = Classes.RSA.Module(c1, numP - 1 - numK, numP);
    var m = (r * c2) % numP;
    lstbytes.AddRange(m.ToArray());
    ///////
}
txtPlain.Text = Encoding.UTF8.GetString(lstbytes.ToArray());
}

```

Giao diện chương trình

The screenshot displays the 'Elgamal Crypto' application window. It features a 'Gen keys' button at the top. Below it, four text boxes labeled P, Q, D, and K' contain large prime numbers. A message 'Okay, let's rock' is shown. The 'Plain' text box contains 'Xin chào, tôi là Ngô Văn Nhận'. Below this, a signature '1fa7c0964a611462580de5f7e9339477' is displayed. The 'Encrypt' button is highlighted. Below the encryption button, two text boxes labeled 'Cipher 1' and 'Cipher 2' contain the resulting ciphertexts. The 'Decrypt', 'Sign message', and 'Check signature' buttons are also visible.



TÀI LIỆU THAM KHẢO

1. William Stallings - Prentice Hall, *Cryptography and Network Security Principles and Practices* - 4th Edition, 2005
2. TS. Nguyễn Khanh Văn, Viện CNTT – TT, ĐHBKHN – *Giáo trình An toàn & Bảo mật thông tin*, 2012
3. Trần Minh Văn, Khoa CNTT, Trường đại học Nha Trang – *bài giảng An toàn và bảo mật thông tin*, 2008
4. TS. Nguyễn Đại Thọ, Trường đại học Công nghệ, ĐHQGHN – *bài giảng An toàn và An ninh mạng*
5. Wikipedia và các nguồn tài liệu khác.