# Mã hóa dữ liệu

# Mục Lục

# Mở đầu

Thế kỷ XXI thế kỷ công nghệ thông tin, thông tin đã và đang tác động trực tiếp đến mọi mặt hoạt động kinh tế xã hội của hầu hết các quốc gia trên thế giới. Thông tin có một vai trò hết sức quan trọng, bởi vậy chúng ta phải làm sao đảm bảo được tính trong suốt của thông tin nghĩa là thông tin không bị sai lệch, bị thay đổi, bị lộ trong quá trình truyền từ nơi gửi đến nơi nhận.

Với sự phát triển rất nhanh của công nghệ mạng máy tính đặc biệt là mạng INTERNET thì khối lượng thông tin ngày càng chuyển tải nhiều hơn. Những tập đoàn công nghiệp, những công ty đa quốc gia, thị trường chứng khoán  tiến hành xử lý và truyền nhận những thông tin đắt giá, những phiên giao dịch hay mua bán cổ phiếu, trái phiếu đều được tiến hành qua mạng. Giờ đây với sự tăng trưởng nhanh của các siêu thị điện tử, thương mại điện tử thì hàng ngày có một khối lượng tiền rất lớn được lưu chuyển trên mạng toàn cầu INTERNET, vấn đề khó khăn đặt ra là làm sao giữ được thông tin bí mật và giữ cho tiền đến đúng được địa chỉ cần đến.

Bạn sẽ ra sao nếu như bạn gửi thư cho một người bạn nhưng lại bị một kẻ lạ mặt nào đó xem trộm và sửa đổi nội dung bức thư trái với chủ ý của bạn, tệ hại hơn nữa là khi bạn ký một hợp đồng, gửi thông qua mạng và lại bị kẻ xấu sửa đổi những điều khoản trong đó, và sẽ còn nhiều điều tương tự như vậy nữa ... Hậu quả sẽ như thế nào nhỉ ? Bạn bị người khác hiểu nhầm vì nội dung bức thư bị thay đổi, còn hợp đồng bị phá vỡ bởi những điều khoản đã không còn nguyên vẹn. Như vậy là cả tình cảm, tiền bạc của bạn và nói rộng hơn là cả sự nghiệp của bạn đều bị đe dọa nếu như những thông tin mà bạn gửi đi không đảm bảo được tính nguyên vẹn của chúng. ***Mã hoá thông tin*** là một

trong các phương pháp đảm bảo được tính trong suốt của thông tin. Nó có thể giải quyết các vấn rắc rối ở trên giúp bạn, một khi thông tin đã được mã hoá và gửi đi thì kẻ xấu rất khó hoặc không thể giải mã được.

Một số khái niệm cơ bản về mã hoá thông tin, phương pháp mã hoá thông tin RSA và xây dựng một thư viện các hàm mã hoá phục vụ trao đổi thông tin trong mô hình Client/Server.

Chương I Cơ sở toán học

Chương II Mật mã

Chương III Hệ mã hoá RSA.

Chương IV Mô hình Client/Server

Chương V Xây dựng hàm thư viện

# Chương i Cơ sở toán học

Để có những thuật toán mã hoá tốt, chúng ta phải có những kiến thức cơ bản về toán học đáp ứng cho yêu cầu, chương này mô tả những khái niệm cơ bản về lý thuyết thông tin như Entropy, tốc độ của ngôn ngữ, hiểu biết về độ phức tạp của thuật toán, độ an toàn của thuật toán, cùng với những kiến thức toán học: modulo số học, số nguyên tố, định lý phần dư trung hoa, định lý Fermat . . . và các phương pháp kiểm tra xem một số có phải là nguyên tố hay không. Những vấn đề chính sẽ được trình bày trong chương này gồm :

- Lý thuyết thông tin
- Lý thuyết độ phức tạp
- Lý thuyết số học.

## 1.Lý thuyết thông tin

Mô hình lý thuyết thông tin được định nghĩa lần đầu tiên vào năm 1948 bởi Claude Elmwood Shannon. Trong phần này chúng ta chỉ đề cập tới một số chủ đề quan trọng của lý thuyết thông tin.

### *1.1 Entropy*

Lý thuyết thông tin được định nghĩa  là khối lượng thông tin trong một thông báo như là số bít nhỏ nhất cần thiết để mã hoá tất cả những nghĩa có thể của thông báo đó.

Ví dụ, trường ngay_thang trong một cơ sở dữ liệu chứa không quá 3 bít thông tin, bởi vì thông tin tại đây có thể mã hoá với 3 bít.

000 = Sunday

001 = Monday

010 = Tuesday

011 = Wednesday

100 = Thursday

101 = Friday

110 = Saturday

111 is unused

Nếu thông tin này được biểu diễn bởi chuỗi ký tự ASCII tương ứng, nó sẽ chiếm nhiều không gian nhớ hơn, nhưng cũng không chứa nhiều thông tin hơn. Tương tự như trường gioi_tinh của một cơ sở dữ liệu chứa chỉ 1 bít thông tin, nó có thể lưu trữ như một trong hai xâu ký tự ASCII : Nam, Nữ.

Khối lượng thông tin trong một thông báo M là đo bởi **Entropy** của thông báo đó, ký hiệu bởi H(M). Entropy của thông báo gioi_tinh chỉ ra là 1 bít, ký hiệu H(gioi_tinh) = 1, Entropy của thông báo số ngày trong tuần là nhỏ hơn 3bits.

Trong trường hợp tổng quát, **Entropy** của một thông báo là $\log_2 n$, với n là số khả năng có thể.

$$H(M) = \log_2 n$$

## *1.2 Tốc độ của ngôn ngữ. (Rate of Language)*

Đối với một ngôn ngữ, tốc độ của ngôn ngữ là

$$r = H(M)/N$$

trong trường hợp này N là độ dài của thông báo. Tốc độ của tiếng Anh bình thường có một vài giá trị giữa 1.0 bits/chữ cái và 1.5 bits/chữ cái, áp dụng với giá trị N rất lớn.

Tốc độ tuyệt đối của ngôn ngữ là số bits lớn nhất, chúng có thể mã hoá trong mỗi ký tự. Nếu có L ký tự trong một ngôn ngữ, thì tốc độ tuyệt đối

là :

$$R = \log_2 L$$

Đây là số Entropy lớn nhất của mỗi ký tự đơn lẻ. Đối với tiếng Anh gồm 26 chữ cái, tốc độ tuyệt đối là $\log_2 26 = 4.7$bits/chữ cái. Sẽ không có điều gì là ngạc nhiên đối với tất cả mọi người rằng thực tế tốc độ của tiếng Anh nhỏ hơn nhiều so với tốc độ tuyệt đối.

### *1.3 An toàn của hệ thống mã hoá*

Shannon định nghĩa rất rõ ràng, tỉ mỉ các mô hình toán học, điều đó có nghĩa là hệ thống mã hoá là an toàn. Mục đích của người phân tích là phát hiện ra khoá **k**, bản rõ **p**, hoặc cả hai thứ đó. Hơn nữa họ có thể hài lòng với một vài thông tin có khả năng về bản rõ **p** nếu đó là âm thanh số, nếu nó là văn bản tiếng Đức, nếu nó là bảng tính dữ liệu, v. v . . .

Trong hầu hết các lần phân tích mã, người phân tích có một vài thông tin có khả năng về bản rõ **p** trước khi bắt đầu phân tích. Họ có thể biết ngôn ngữ đã được mã hoá. Ngôn ngữ này chắc chắn có sự dư thừa kết hợp với chính ngôn ngữ đó. Nếu nó là một thông báo gửi tới Bob, nó có thể bắt đầu với "Dear Bob". Chắc chắn là "Dear Bob " sẽ là một khả năng có thể hơn là chuỗi không mang ý nghĩa gì chẳng hạn "tm*h&rf". Mục đích của việc thám mã là sửa những tập hợp khả năng có thể có của bản mã với mỗi khả năng có thể của bản rõ.

Có một điều giống như hệ thống mã hoá, chúng đạt được sự bí mật tuyệt đối. Hệ thống mã hoá này trong đó bản mã không mang lại thông tin có thể để tìm lại bản rõ. Shannon phát triển lý thuyết cho rằng, hệ thống mã hoá chỉ an toàn tuyệt đối nếu nếu số khoá có thể ít nhất là nhiều bằng số thông báo có thể. Hiểu theo một nghĩa khác, khoá tối thiểu dài bằng thông báo của chính nó.

Ngoại trừ an toàn tuyệt đối, bản mã mang lại một vài thông tin đúng với bản rõ, điều này là không thể tránh được. Một thuật toán mật mã tốt giữ cho thông tin ở mức nhỏ nhất, một người thám mã tốt khai thác những thông tin này để phát hiện ra bản rõ.

Người phân tích mã sử dụng sự dư thừa tự nhiên của ngôn ngữ để làm giảm số khả năng có thể của bản rõ. Nhiều thông tin dư thừa của ngôn ngữ, sẽ dễ dàng hơn cho sự phân tích mật mã. Chính vì lý do này mà nhiều sự thực hiện mã hoá sử dụng chương trình nén bản rõ để giảm kích thước văn bản trước khi mã hoá chúng. Bởi vậy quá trình nén làm giảm sự dư thừa của thông báo.

Entropy của hệ thống mã hoá là đo kích thước của không gian khoá (keyspace).

$$H(K) = \log_2(\text{number of keys})$$

*1.4 Sự lộn xộn và sự rườm rà. (Confusion and Diffusion)*

Theo nhà khoa học Shannon, có hai kỹ thuật cơ bản để che dấu sự dư thừa thông tin trong thông báo gốc đó là : sự lộn xộn và sự rườm rà.

Kỹ thuật lộn xộn (Confusion) che dấu mối quan hệ giữa bản rõ và bản gốc. Kỹ thuật này làm thất bại sự cố gắng nghiên cứu bản mã tìm kiếm thông tin dư thừa và thống kê mẫu. Phương pháp dễ nhất để thực hiện điều này là thông qua kỹ thuật thay thế. Một hệ mã hoá thay thế đơn giản, chẳng hạn hệ mã dịch vòng Caesar, dựa trên nền tảng của sự thay thế các chữ cái, nghĩa là chữ cái này được thay thế bằng chữ cái khác. Sự tồn tại của một chữ cái trong bản mã, là do việc dịch chuyển đi k vị trí của chữ cái trong bản rõ.

Kỹ thuật rườm rà (Diffusion) làm mất đi sự dư thừa của bản rõ bằng bề rộng của nó vượt quá bản mã (nghĩa là bản mã kích thước nhỏ hơn bản rõ). Một người phân tích tìm kiếm sự dư thừa đó sẽ có một

thời gian rất khó khăn để tìm ra chúng. Cách đơn giản nhất tạo ra sự rườm rà là thông qua việc đổi chỗ (hay còn gọi là hoán vị).

## 2.Lý thuyết độ phức tạp.

Lý thuyết độ phức tạp cung cấp một phương pháp để phân tích độ phức tạp tính toán của thuật toán và các kỹ thuật mã hoá khác nhau. Nó so sánh các thuật toán mã hoá, kỹ thuật và phát hiện ra độ an toàn của các thuật toán đó. *Lý thuyết thông tin đã cho chúng ta biết rằng một thuật toán mã hoá có thể bị bại lộ. Còn lý thuyết độ phức tạp cho biết nếu liệu chúng có thể bị bại lộ trước khi vũ trụ xụp đổ hay không.*

Độ phức tạp thời gian của thuật toán là hàm số với độ dài đầu vào. Thuật toán có độ phức tạp thời gian f(n) đối với mọi n và độ dài đầu vào n, nghĩa là sự thực hiện của thuật toán lớn hơn f(n) bước.

Độ phức tạp thời gian thuật toán phụ thuộc vào mô hình của các thuật toán, số các bước nhỏ hơn nếu các hoạt động được tập chung nhiều trong một bước.

Các lớp của thuật toán, thời gian chạy được chỉ rõ như hàm số mũ của đầu vào là "không có khả năng thực hiện được". Các thuật toán có độ phức tạp giống nhau được phân loại vào trong các lớp tương đương. Ví dụ tất cả các thuật toán có độ phức tạp là $n^3$ được phân vào trong lớp $n^3$ và ký hiệu bởi $O(n^3)$. Có hai lớp tổng quát sẽ được chỉ dẫn là lớp P và lớp NP.

Các thuật toán thuộc lớp P có độ phức tạp là hàm đa thức của đầu vào. Nếu mỗi bước tiếp theo của thuật toán là duy nhất thì thuật toán gọi là đơn định. Tất cả thuật toán thuộc lớp P đơn định có thời gian giới hạn là P_time, điều này cho biết chúng sẽ thực hiện trong thời gian đa thức, tương đương với độ phức tạp đa thức trong độ dài đầu vào.

Thuật toán mà ở bước tiếp theo sự tính toán phải lựa chọn giải pháp từ những giới hạn giá trị của hoạt động gọi là không đơn định. Lý thuyết độ phức tạp sử dụng các máy đặc biệt mô tả đặc điểm bằng cách đưa ra kết luận bởi các chuẩn. Máy Turinglà một máy đặc biệt, máy hoạt động trong thời gian rời rạc, tại một thời điểm nó nằm trong khoảng trạng thái đầy đủ số của tất cả các trạng thái có thể là hữu hạn. Chúng ta có thể định nghĩa hàm độ phức tạp thời gian kết hợp với máy Turing A.

$f_A(n) = \max\{m/A$ kết thúc sau m bước với đầu vào $w = n^3\}$

Chúng ta giả sử rằng A là trạng thái kết thúc đối với tất cả các đầu vào, vấn đề sẽ trở nên khó khăn hơn nếu các trạng thái không nằm trong P . Máy Turing không đơn định hoạt động trong thuật toán NP. Máy Turing không đơn định có thể có một vài trạng thái chính xác. S(w) là trạng thái đo sự thành công ngắn nhất của thuật toán, (Nghĩa là sự tính toán dẫn đến trạng thái cuối cùng)

Hàm số độ phức tạp thời gian của máy Turing không đơn định A được định nghĩa :

$f_A(n)=\max\{1,m/s(w)$ có m bước đối với $w/w=n\}$,

ở mỗi bước máy Turing không đơn định bố trí nhiều bản sao của chính nó như có một vài giải pháp và tính toán độc lập với mọi lời giải.

Các thuật toán thuộc lớp NP là không đơn định và có thể tính toán trên máy Turing không đơn định trong thời gian P.

## 3.Lý thuyết toán học.

### *3.1 Modular số học.*

Về cơ bản $a \equiv b(\text{mod } n)$ nếu $a = b+kn$ trong đó k là một số nguyên. Nếu a và b dương và a nhỏ hơn n, bạn có thể nghĩ rằng a là phần dư

của b khi chia cho n. Nói chung a và b đều là phần dư khi chia cho n. Đôi khi b gọi là thặng dư của a, modulo n, đôi khi a gọi là đồng dư của b, modulo n.

Tập hợp các số nguyên từ 0 đến n-1 còn được gọi là tập hợp thặng dư hoàn toàn modulo n. Điều này có nghĩa là, với mỗi số nguyên a, thì thặng dư modulo n là một số từ 0 đến n-1.

Modulo số học cũng giống như số học bình thường, bao gồm các phép giao hoán, kết hợp và phân phối. Mặt khác giảm mỗi giá trị trung gian trong suốt quá trình tính toán.

$(a+b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$

$(a- b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n$

$(a\times b) \bmod n = ((a \bmod n) \times (b \bmod n)) \bmod n$

$(a\times (b + c)) \bmod n = (((a \times b) \bmod n) + ((a \times c) \bmod n)) \bmod n$

Hệ thống mã hoá sự dụng nhiều sự tính toán modulo n, bởi vì vấn đề này giống như tính toán logarithm rời rạc và diện tích hình vuông là khó khăn. Mặt khác nó làm việc dễ hơn, bởi vì nó bị giới hạn trong tất cả giá trị trung gian và kết quả. Ví dụ : a là một số k bits, n là kết quả trung gian của phép cộng, trừ, nhân sẽ không vượt quá 24 bits. Như vậy chúng ta có thể thực hiện hàm mũ trong modulo số học mà không cần sinh ra kết quả trung gian đồ sộ.

## *3.2 Số nguyên tố.*

Số nguyên tố là một số lớn hơn 1, nhưng chỉ chia hết cho 1 và chính nó, ngoài ra không còn số nào nó có thể chia hết nữa. Số 2 là một số nguyên tố. Do vậy 7, 17, 53, 73, 2521, 2365347734339 cũng là số nguyên tố. Số lượng số nguyên tố là vô tận. Hệ mật mã thường sử dụng số nguyên tố lớn cỡ 512 bits và thậm chí lớn hơn như vậy.

## 3.3 Ước số chung lớn nhất.

Hai số gọi là cặp số nguyên tố khi mà chúng không có thừa số chung nào khác 1, hay nói một cách khác, nếu ước số chung lớn nhất của a và n là bằng 1. Chúng ta có thể viết như sau :

gcd(a,n)=1

Số 15 và 28 là một cặp số nguyên tố, nhưng 15 và 27 thì không phải cặp số nguyên tố do có ước số chung là 1 và 3, dễ dàng thấy 13 và 500 cũng là một cặp số nguyên tố. Một số nguyên tố là một cặp số nguyên tố với tất cả những số khác loại trừ những số là bội số.

Một cách dễ nhất để tính toán ra ước số chung lớn nhất của hai số là nhờ vào thuật toán Euclid. Knuth mô tả thuật toán và một vài mô hình của thuật toán đã được sửa đổi.

Dưới đây là đoạn mã nguồn trong ngôn ngữ C.

```
/* Thuật toán tìm ước số chung lớn nhất của x và y, giả sử x,y>0 */
int  gcd(int x, int y)
{
        int  g;
        if(x<0)
                x=-x;
        if(y<0)
                y=-y    ;
        g=y;
        while(x>0){
                g=x;
                x=y%x;
                y=g;
        }
        return g;
}
```

Thuật toán sau đây có thể sinh ra và trả lại ước số chung lớn nhất của một mảng m số.

```
int multiple gcd ( int m, int *x)
{
        size  t, i ;
        int  g;
        if(m<1)
                return(0);
        g = x[0];
        for(i=1;i<m;++i){
                g=gcd(g,x[i]);
                if(g==1)
                        return 1;
        }
        return g;
}
```

### *3.4 Số nghịch đảo Modulo.*

Số nghịch đảo của 10 là 1/10, bởi vì $10 \times 1/10=1$. Trong số học modulo thì vấn đề nghịch đảo phức tạp hơn.

$$4 \times x \equiv 1 \bmod 7$$

Phương trình trên tương đương với tìm x và k sao cho

$$4x = 7k+1$$

với điều kiện là cả x và k đều là số nguyên.

Vấn đề chung đặt ra tại đây là tìm x sao cho

$$1 = (a \times x) \bmod n$$

có thể viết lại như sau :

$$a^{-1} \equiv x \pmod n$$

Sự thu nhỏ vấn đề Modulo là rất khó giải quyết. Đôi khi nó là một vấn đề, nhưng đôi khi lại không phải vậy.

Ví dụ : nghịch đảo của 5 modulo 14 là 3 bởi

$5 \times 3 = 15 \equiv 1 \pmod{14}$.

Trong trường hợp chung $a^{-1} \equiv x \pmod{n}$ chỉ có duy nhất một giải pháp nếu a và n là một cặp số nguyên tố. Nếu a và n không phải là cặp số nguyên tố, thì $a^{-1} \equiv x \pmod{n}$ không có giải pháp nào. Thuật toán Euclid có thể tính ra được số nghịch đảo của số Modulo n, đôi khi thuật toán này còn gọi là thuật toán Euclid mở rộng. Sau đây thuật toán được mô tả trong ngôn ngữ C.

```
static void Update(int *un,int *vn, int q)
{
        int tn;

        tn = *un-vn*q;
        *un = *vn;
        *vn = tn;
}

int extended euclidian(int u,int v,int u1_out,int u2_out)
{
        int u1=1;
        int u3=u;
        int v1=0;
        int v3=v;
        int q;

        while(v3>0){
                q=u3/v3;
                Update(&u1,&v1,q);
                Update(&u3,&v,q);
        }
```

```
        *u1_out=u1;
        *u2_out=(u3-u1*u)/v;
        return u3;
}
```

## 3.5 Ký hiệu La grăng (Legendre Symboy)

Ký hiệu L(a,p) được định nghĩa khi a là một số nguyên và p là một số nguyên tố lớn hơn 2. Nó nhận ba giá trị 0, 1, -1 :

L(a,p) = 0 nếu a chia hết cho p.

L(a,p) = 1 nếu a là thặng dư bậc 2 mod p.

L(a,p) = -1 nếu a không thặng dư mod p.

Một phương pháp dễ dàng để tính toán ra L(a,p) là :

$L(a,p) = a^{(p-1)/2} \bmod p$

## 3.6 Ký hiệu Jacobi (Jacobi Symboy)

Ký hiệu Jacobi được viết J(a,n), nó là sự khái quát hoá của ký hiệu Lagrăng, nó định nghĩa cho bất kỳ cặp số nguyên a và n. Ký hiệu Jacobi là một chức năng trên tập hợp số thặng dư thấp của ước số n và có thể tính toán theo công thức sau:

- Nếu n là số nguyên tố, thì J(a,n) = 1 với điều kiện a là thặng dư bậc hai modulo n .
- Nếu n là số nguyên tố, thì J(a,n) = -1 với điều kiện a không là thặng dư bậc hai modulo n .
- Nếu n không phải là số nguyên tố thì Jacobi

  $J(a,n)=J(h,p_1) \times J(h,p_2) \times \ldots \times J(h,p_m)$

  với $p_1,p_2 \ldots,p_m$ là các thừa số lớn nhất của n.

Thuật toán này tính ra số Jacobi tuần hoàn theo công thức sau :

1. J(1,k) = 1
2. J(a× b,k) = J(a,k) × J(b,k)
3. J(2,k) =1 Nếu $(k^2-1)/8$ là chia hết

   J(2,k) =-1 trong các trường hợp khác.
4. J(b,a) = J((b mod a),a)
5. Nếu GCD(a,b)=1 :
   a. J(a,b) × J(b,a) = 1 nếu (a-1)(b-1)/4 là chia hết.
   b. J(a,b) × J(b,a) = -1 nếu (a-1)(b-1)/4 là còn dư.

Sau đây là thuật toán trong ngôn ngữ C :

```
int jacobi(int a,int b)
{
        int a1,a2;
        if(a>=b)
                a%=b;
        if(a==0)
                return 0;
        if(a==1)
                return 1;
        if(a==2)
                if(((b*b-1)/8)%2==0)
                        return 1;
                else
                        return -1;
        if(a&b&1)  (cả a và b đều là số dư)
                if(((a-1)*(b-1)/4)%2==0)
                        return +jacobi(b,a);
                else
                        return -jacobi(b,a);
        if(gcd(a,b)==1)
                if(((a-1)*(b-1)/4)%2==0)
                return +jacobi(b,a);
```

```
            else
                return -jacobi(b,a);
        factor2(a,&a1,&a2);
        return jacobi(a1,b) * jacobi(a2,b);
}
```

Nếu p là số nguyên tố có cách tốt hơn để tính số Jacobi như dưới đây :

1. Nếu a=1 thì J(a/p)=1
2. Nếu a là số chai hết, thì $J(a,p)=J(a/2,p) \times (-1)^{(p\hat{}2-1)/8}$
3. Nếu a là số dư khác 1 thì $J(a,p)=J(p \bmod a, a) \times (-1)^{(a-1)\times (p-1)/4}$

## *3.7 Định lý phần dư trung hoa.*

Nếu bạn biết cách tìm thừa số nguyên tố của một số n, thì bạn có thể đã sử dụng, một số điều gọi là định lý phần dư trung hoa để giải quyết trong suốt hệ phương trình. Bản dịch cơ bản của đinh lý này được khám phá bởi toán học Trung Hoa vào thế kỷ thứ nhất.

Giả sử, sự phân tích thừa số của $n=p_1 \times p_2 \times \ldots \times p_t$ thì hệ phương trình

$$(X \bmod p_i) = a_i, \text{ với } i=1,2,\ldots t$$

có duy nhất một cách giải, tại đó x nhỏ hơn n.

Bởi vậy, với a,b tuỳ ý sao cho a < p và b < q (p,q là số nguyên tố) thì tồn tại duy nhất a,x ,khi x nhỏ hơn $p \times q$ thì

$$x \equiv a \pmod p, \text{ và } x \equiv b \pmod q$$

Để tìm ra x đầu tiên sử dụng thuật toán Euclid để tìm u, ví dụ :

$$u \times q \equiv 1 \pmod p$$

Khi đó cần tính toán :

$$x=(((a-b)\times u) \bmod p) \times q + b$$

Dưới đây là đoạn mã định lý phần dư trung hoa trong ngôn ngữ C :

```
Int chinese remainder(size t r, int *m, int *u)
{
```

```
        size t i;
        int modulus;
        int n;
        modulus = 1;
        for ( i=0; i<r:++i )
                modulus *=m[i];
                n=0;
        for ( i=0; i<r:++i )
        {
        n+=u[i]*modexp(modulus/m[i],totient(m[i]),m[i]);
                n%=modulus;
        }
        return n;
}
```

### *3.8 Định lý Fermat.*

Nếu m là số nguyên tố, và a không phải là bội số của m thì định lý Fermat phát biểu :

$$a^{m-1} \equiv 1 (\text{mod } m)$$

## 4. Các phép kiểm tra số nguyên tố.

Hàm một phía là một khái niệm cơ bản của mã hoá công khai, việc nhân hai số nguyên tố được phỏng đoán như là hàm một phía, nó rất dễ dàng nhân các số để tạo ra một số lớn, nhưng rất khó khăn để phân tích số lớn đó ra thành các thừa số là hai số nguyên tố lớn.

Thuật toán mã hoá công khai cần thiết tới những số nguyên tố. Bất kỳ mạng kích thước thế nào cũng cần một số lượng lớn số nguyên tố. Có một vài phương pháp để sinh ra số nguyên tố. Tuy nhiên có một số vấn đề được đặt ra đối với số nguyên tố như sau :

- Nếu mọi người cần đến những số nguyên tố khác nhau, chúng ta sẽ không đạt được điều đó đúng không. Không đúng, bởi vì trong thực tế có tới $10^{150}$ số nguyên tố có độ dài 512 bits hoặc nhỏ hơn.
- Điều gì sẽ xảy ra nếu có hai người ngẫu nhiên chọn cùng một số nguyên tố?. Với sự chọn lựa từ số lượng $10^{150}$ số nguyên tố, điều kỳ quặc này xảy ra là xác xuất nhỏ hơn so với sự tự bốc cháy của máy tính. Vậy nó không có gì là đáng lo ngại cho bạn hết.

## *4.1 Soloway-Strassen*

Soloway và Strassen đã phát triển thuật toán có thể kiểm tra số nguyên tố. Thuật toán này sử dụng hàm Jacobi.

Thuật toán kiểm tra số p là số nguyên tố :

1. Chọn ngẫu nhiên một số a nhỏ hơn p.
2. Nếu ước số chung lớn nhất gcd(a,p) ≠ 1 thì p là hợp số.
3. Tính $j = a^{(p-1)/2} \bmod p$.
4. Tính số Jacobi J(a,p).
5. Nếu j ≠ J(a,p), thì p không phải là số nguyên tố.
6. Nếu j = J(a,p) thì nói p có thể là số nguyên tố với chắc chắn 50%.

Lặp lại các bước này n lần, với những n là giá trị ngẫu nhiên khác nhau của a. Phần dư của hợp số với n phép thử là không quá $2^n$.

Thực tế khi thực hiện chương trình, thuật toán chạy với tốc độ nhanh.

## *4.2 Rabin-Miller*

Thuật toán này được phát triển bởi Rabin, dựa trên một phần ý tưởng của Miller. Thực tế những phiên bản của thuật toán đã được giới thiệu tại NIST. (National Institute of Standards and Technology).

Đầu tiên là chọn ngẫu nhiên một số p để kiểm tra. Tính b, với b là số mũ của 2 chia cho p-1. Tiếp theo tính m tương tự như $n = 1+2^b m$.

Sau đây là thuật toán :

1. Chọn một sô ngẫu nhiên a, và giả sử a nhỏ hơn p.
2. Đặt j=0 và z=$a^m$ mod p.
3. Nếu z=1, hoặc z=p-1 thì p đã qua bước kiểm tra và có thể là số nguyên tố.
4. Nếu j > 0 và z=1 thì p không phải là số nguyên tố.
5. Đặt j = j+1. Nếu j < b và z ≠ p-1 thì đặt z=$z^2$ mod p và trở lại bước 4.
6. Nếu j = b và z ≠ p-1, thì p không phải là số nguyên tố.

## *4.3 Lehmann.*

Một phương pháp đơn giản hơn kiểm tra số nguyên tố được phát triển độc lập bởi Lehmann. Sau đây là thuật toán với số bước lặp là 100.

1. Chọn ngẫu nhiên một số n để kiểm tra.
2. Chắc chắn rằng n không chia hết cho các số nguyên tố nhỏ như 2,3,5,7 và 11.
3. Chọn ngẫu nhiên 100 số $a_1$, $a_2$, . . . , $a_{100}$ giữa 1 và n-1.
4. Tính $a_i^{(n-1)/2}$ (mod n) cho tất cả $a_i = a_1 . . . a_{100}$ . Dừng lại nếu bạn tìm thấy $a_i$ sao cho phép kiểm tra là sai.
5. Nếu $a_i(n-1)/2 = 1$ (mod n) với mọi i, thì n có thể là hợp số.

   Nếu $a_i(n-1)/2 \neq 1$ hoặc -1 (mod n) với i bất kỳ, thì n là hợp số.

   Nếu $a_i(n-1)/2 = 1$ hoặc -1 (mod n) với mọi $i \neq 1$, thì n là số nguyên tố.

## *4.4 Strong Primes.*

Strong Primes thường được sử dụng cho hai số p và q, chúng là hai số nguyên tố với các thuộc tính chắc chắn rằng có thể tìm được thừa số

bằng phương pháp phân tích thừa số. Trong số các thuộc tính đạt được bao gồm

+ Ước số chung lớn nhất của p-1 và q-1 là nhỏ.

+ Hai số p-1 và q-1 nên có thừa số nguyên tố lớn, đạo hàm riêng p' và q'

+ Hai số p'-1 và q'-1 nên có thừa số nguyên tố lớn, đạo hàm riêng p'' và q''

+ Cả (p-1)/2 và (q-1)/2 nên là số nguyên tố.

Trong bất cứ trường hợp nào Strong Primes rất cần thiết là đối tượng trong các buổi tranh luận. Những thuộc tính đã được thiết kế cản trở một vài thuật toán phân tích thừa số. Hơn nữa, những thuật toán phân tích thừa số nhanh nhất có cơ hội tốt để đạt các tiêu chuẩn.

# Chương II **Mật mã**

Trong chương trước chúng ta đã nêu ra các khái niệm cơ bản về lý thuyết thông tin, về độ phức tạp của thuật toán, và những khái niệm cơ bản về toán học cần thiết. Chương này sẽ mô tả một cách tổng quan về mã hoá, bao gồm những khái niệm về mã hoá thông tin, một hệ thống mã hoá bao gồm những thành phần nào, khái niệm protocol, các loại protocol. Mã hoá dòng là gì, mã hoá khối là gì, thế nào là hệ thống mã hoá cổ điển, thế nào là hệ thống mã hoá công khai. Và cuối cùng là bằng những cách nào kẻ địch tấn công hệ thống mã hoá. Những vấn đề sẽ được đề cập trong chương này:

- Khái niệm cơ bản của mã hoá.
- Protocol
- Mã dòng , mã khối (CFB, CBC)
- Các hệ mật mã đối xứng và công khai
- Các cách thám mã

## 1. Khái niệm cơ bản.

-Bản rõ (plaintext or cleartext)

Chứa các xâu ký tự gốc, thông tin trong bản rõ là thông tin cần mã hoá để giữ bí mật.

-Bản mã (ciphertext)

Chứa các ký tự sau khi đã được mã hoá, mà nội dung được giữ bí mật.

-Mật mã học (Crytography)

Là nghệ thuật và khoa học để giữ thông tin được an toàn.

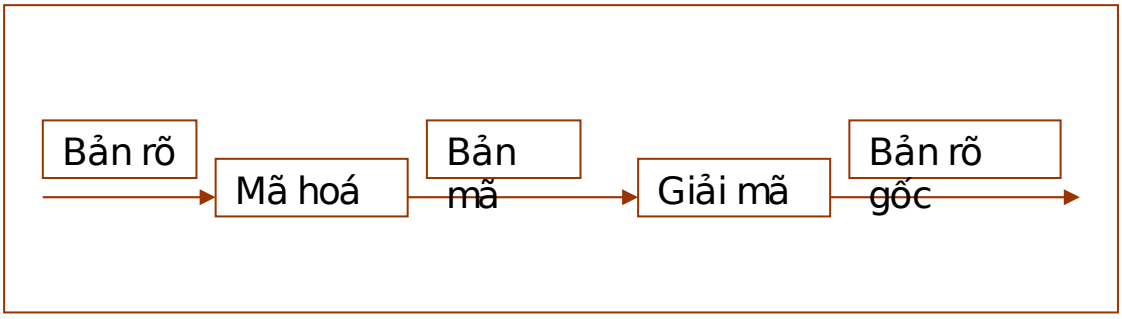
-Sự mã hoá (Encryption)

Quá trình che dấu thông tin bằng phương pháp nào đó để làm ẩn nội dung bên trong gọi là sự mã hoá.

-Sự giải mã (Decryption)

Quá trình biến đổi trả lại bản mã bản thành bản rõ gọi là giải mã.

Quá trình mã hoá và giải mã được thể hiện trong sơ đồ sau:




-Hệ mật mã : là một hệ bao gồm 5 thành phần (P, C, K, E, D) thoả mãn các tính chất sau

P (Plaintext) là tập hợp hữu hạn các bản rõ có thể.

C (Ciphertext) là tập hợp hữu hạn các bản mã có thể.

K (Key) là tập hợp các bản khoá có thể.

E (Encrytion) là tập hợp các qui tắc mã hoá có thể.

D (Decrytion) là tập hợp các qui tắc giải mã có thể.

Chúng ta đã biết một thông báo thường được tổ chức dưới dạng bản rõ. Người gửi sẽ làm nhiệm vụ mã hoá bản rõ, kết quả thu được gọi là bản mã. Bản mã này được gửi đi trên một đường truyền tới người nhận sau khi nhận được bản mã người nhận giải mã nó để tìm hiểu nội dung.

Dễ dàng thấy được công việc trên khi sử dụng định nghĩa hệ mật mã :

$$E_K( P) = C \text{ và } D_K( C ) = P$$

## 2. Protocol

### *2.1 Giới thiệu Protocol*

Trong suốt cả quá trình của hệ thống mật mã là giải quyết các vấn đề, những vấn đề của hệ bao gồm: giải quyết công việc xung quanh sự bí mật, tính không tin cậy và những kẻ bất lương. Bạn có thể học mọi điều về thuật toán cũng như các kỹ thuật, nhưng có một điều rất đáng quan tâm đó là Protocol. ***Protocol là một loạt các bước, bao gồm hai hoặc nhiều người, thiết kế để hoàn thành nhiệm vụ .*** "Một loạt các bước" nghĩa là Protocol thực hiện theo một tuần tự, từ khi bắt đầu cho tới lúc kết thúc. Mỗi bước phải được thực hiện tuần tự và không có bước nào được thực hiện trước khi bước trước đó đã hoàn thành. "Bao gồm hai hay nhiều người" nghĩa là cần ít nhất hai người hoàn thành protocol, một người không thể tạo ra được một Protocol. Và chắc chắn rằng một người có thể thực hiện một loạt các bước để hoàn thành nhiệm vụ, nhưng đó không phải là Protocol. Cuối cùng "thiết kế để hoàn thành nhiệm vụ" nghĩa là mỗi Protocol phải làm một vài điều gì đó.

Protocol có một vài thuộc tính khác như sau :

1. Mọi người cần phải trong một Protocol, phải biết protocol đó và tuân theo tất cả mọi bước trong sự phát triển.
2. Mọi người cần phải trong một Protocol, và phải đồng ý tuân theo nó.
3. Một Protocol phải rõ ràng, mỗi bước phải được định nghĩa tốt và phải không có cơ hội hiểu nhầm.
4. Protocol phải được hoàn thành, phải có những hành động chỉ rõ cho mỗi trường hợp có thể.

## *2.2 Protocol mật mã.*

Protocol mật mã là protocol sử dụng cho hệ thống mật mã. Một nhóm có thể gồm những người bạn bè và những người hoàn toàn tin cậy khác hoặc họ có thể là địch thủ hoặc những người không tin cậy một chút nào hết. Một điều hiển nhiên là protocol mã hoá phải bao gồm một số thuật toán mã hoá, nhưng mục đích chung của protocol là một điều gì đó xa hơn là điều bí mật đơn giản.

## *2.3 Mục đích của Protocol.*

Trong cuộc sống hàng ngày, có rất nhiều nghi thức thân mật cho hầu hết tất cả mọi điều như gọi điện thoại, chơi bài, bầu cử. Không có gì trong số chúng lại không có protocol, chúng tiến triển theo thời gian, mọi người đều biết sử dụng chúng như thế nào và làm việc với chúng. Hơn nữa bây giờ mọi người giao tiếp với nhau qua mạng máy tính thay cho sự gặp mặt thông thường. Máy tính cần thiết một nghi thức chuẩn để làm những việc giống nhau như con người không phải suy nghĩ. Nếu bạn đi từ một địa điểm này tới địa điểm khác, thậm chí từ quốc gia này tới quốc gia khác, bạn thấy một trạm điện thoại công cộng khác hoàn toàn so với cái bạn đã sử dụng, bạn dễ dàng đáp ứng. Nhưng máy tính thì không mềm dẻo như vậy.

Thật ngây thơ khi bạn tin rằng mọi người trên mạng máy tính là chân thật, và cũng thật ngây thơ khi tin tưởng rằng người quản trị mạng, người thiết kế mạng là chân thật. Hầu hết sẽ là chân thật, nhưng nó sẽ là không chân khi bạn cần đến sự an toàn tiếp theo. ***Bằng những protocol chính thức, chúng ta có thể nghiên cứu những cách mà những kẻ không trung thực có thể lừa đảo và phát triển protocol để đánh bại những kẻ lừa đảo đó***. Protocol rất hữa ích bởi vì họ trừu

tượng hoá tiến trình hoàn thành nhiệm vụ từ kỹ thuật, như vậy nhiệm vụ đã được hoàn thành.

Sự giao tiếp giữa hai máy tính giống như một máy tính là IBM PC, máy kia là VAX hoặc loại máy tương tự. Khái niệm trừu tượng này cho phép chúng ta nghiên cứu những đặc tính tốt của protocol mà không bị xa lầy vào sự thực hiện chi tiết. Khi chúng ta tin rằng chúng ta có một protocol tốt, thì chúng ta có thể thực hiện nó trong mọi điều từ một máy tính đến điện thoại, hay đến một lò nướng bánh thông minh.

## *2.4 Truyền thông sử dụng hệ mật mã đối xứng.*

Hai máy thực hiện việc truyền thông an toàn như thế nào ? Chúng sẽ mã hoá sự truyền thông đó, đương nhiên rồi. Để hoàn thành một protocol là phức tạp hơn việc truyền thông. Chúng ta hãy cùng xem xét điều gì sẽ xảy ra nếu máy Client muốn gửi thông báo mã hoá tới cho Server.

1. Client và Server đồng ý sử dụng một hệ mã hóa.
2. Client và Server thống nhất khoá với nhau.
3. Client lấy bản rõ và mã hoá sử dụng thuật toán mã hoá và khoá. Sau đó bản mã đã được tạo ra.
4. Client gửi bản mã tới cho Server.
5. Server giải mã bản mã đó với cùng một thuật toán và khoá, sau đó đọc được bản rõ.

Điều gì sẽ xảy ra đối với kẻ nghe trộm cuộc truyền thông giữa Client và Server trong protocol trên. Nếu như kẻ nghe trộm chỉ nghe được sự truyền đi bản mã trong bước 4, chúng sẽ cố gắng phân tích bản mã. Những kẻ nghe trộm chúng không ngu rốt, chúng biết rằng nếu có thể nghe trộm từ bước 1 đến bước 4 thì chắc chắn sẽ thành công. Chúng sẽ biết được thuật toán và khoá như vậy chúng sẽ biết được nhiều như

Server. Khi mà thông báo được truyền đi trên kênh truyền thông trong bước thứ 4, thì kẻ nghe trộm sẽ giải mã bằng chính những điều đã biết.

Đây là lý do tại sao quản lý khoá lại là vấn đề quan trọng trong hệ thống mã hoá. Một hệ thống mã hoá tốt là mọi sự an toàn phụ thuộc vào khoá và không phụ thuộc vào thuật toán. Với thuật toán đối xứng, Client và Server có thể thực hiện bước 1 là công khai, nhưng phải thực hiện bước 2 bí mật. Khoá phải được giữ bí mật trước, trong khi, và sau protocol, mặt khác thông báo sẽ không giữ an toàn trong thời gian dài.

Tóm lại, hệ mật mã đối xứng có một vài vấn đề như sau :

- Nếu khoá bị tổn thương (do đánh cắp, dự đoán ra, khám phá, hối lộ) thì đối thủ là người có khoá, anh ta có thể giải mã tất cả thông báo với khoá đó. Một điều rất quan trọng là thay đổi khoá tuần tự để giảm thiểu vấn đề này.
- Những khoá phải được thảo luận bí mật. Chúng có thể có giá trị hơn bất kỳ thông báo nào đã được mã hoá, từ sự hiểu biết về khoá có nghĩa là hiểu biết về thông báo.
- Sử dụng khoá riêng biệt cho mỗi cặp người dùng trên mạng vậy thì tổng số khoá tăng lên rất nhanh giống như sự tăng lên của số người dùng. Điều này có thể giải quyết bằng cách giữ số người dùng ở mức nhỏ, nhưng điều này không phải là luôn luôn có thể.

## *2.5 Truyền thông sử dụng hệ mật mã công khai.*

♦ Hàm một phía (one way function)

Khái niệm hàm một phía là trung tâm của hệ mã hoá công khai. Không có một Protocol cho chính nó, hàm một phía là khối xây dựng cơ bản cho hầu hết các mô tả protocol.

Một hàm một phía là hàm mà dễ dàng tính toán ra quan hệ một chiều nhưng rất khó để tính ngược lại. Ví như : biết giả thiết x thì có thể dễ dàng tính ra f(x), nhưng nếu biết f(x) thì rất khó tính ra được x. Trong trường hợp này "khó" có nghĩa là để tính ra được kết quả thì phải mất hàng triệu năm để tính toán, thậm chí tất cả máy tính trên thế giới này đều tính toán công việc đó.

Vậy thì hàm một phía tốt ở những gì ? Chúng ta không thể sử dụng chúng cho sự mã hoá. Một thông báo mã hoá với hàm một phía là không hữu ích, bất kỳ ai cũng không giải mã được. Đối với mã hoá chúng ta cần một vài điều gọi là cửa sập hàm một phía.

Cửa sập hàm một phía là một kiểu đặc biệt của hàm một phía với cửa sập bí mật. Nó dễ dàng tính toán từ một điều kiện này nhưng khó khăn để tính toán từ một điều kiện khác. Nhưng nếu bạn biết điều bí mật, bạn có thể dễ dàng tính toán ra hàm từ điều kiện khác. Ví dụ : tính f(x) dễ dàng từ x, rất khó khăn để tính toán x ra f(x). Hơn nữa có một vài thông tin bí mật, y giống như f(x) và y nó có thể tính toán dễ dàng ra x. Như vậy vấn đề có thể đã được giải quyết.

Hộp thư là một ví dụ rất tuyệt về cửa sập hàm một phía. Bất kỳ ai cũng có thể bỏ thư vào thùng. Bỏ thư vào thùng là một hành động công cộng. Mở thùng thư không phải là hành động công cộng. Nó là khó khăn, bạn sẽ cần đến mỏ hàn để phá hoặc những công cụ khác. Hơn nữa nếu bạn có điều bí mật (chìa khoá), nó thật dễ dàng mở hộp thư. Hệ mã hoá công khai có rất nhiều điều giống như vậy.

♦ Hàm băm một phía.

Hàm băm một phía là một khối xây dựng khác cho nhiều loại protocol. Hàm băm một phía đã từng được sử dụng cho khoa học tính toán trong một thời gian dài. Hàm băm là một hàm toán học hoặc loại khác, nó lấy

chuỗi đầu vào và chuyển đổi thành kích thước cố định cho chuỗi đầu ra.

Hàm băm một phía là một hàm băm nó sử dụng hàm một phía. Nó rất dễ dàng tính toán giá trị băm từ xâu ký tự vào, nhưng rất khó tính ra một chuỗi từ giá trị đơn lẻ đưa vào.

Có hai kiểu chính của hàm băm một phía, hàm băm với khoá và không khoá. Hàm băm một phía không khoá có thể tính toán bởi mọi người giá trị băm là hàm chỉ có đơn độc chuỗi đưa vào. Hàm băm một phía với khoá là hàm cả hai thứ chuỗi vào và khoá, chỉ một vài người có khoá mới có thể tính toán giá trị băm.

♦ Hệ mã hoá sử dụng khoá công khai.

Với những sự mô tả ở trên có thể nghĩ rằng thuật toán đối xứng là an toàn. Khoá là sự kết hợp, một vài người nào đó với sự kết hợp có thể mở sự an toàn này, đưa thêm tài liệu vào, và đóng nó lại. Một người nào đó khác với sự kết hợp có thể mở được và lấy đi tài liệu đó.

Năm 1976 Whitfied và Martin Hellman đã thay đổi vĩnh viễn mô hình của hệ thống mã hoá. Chúng được mô tả là hệ mã hoá sử dụng khoá công khai. Thay cho một khoá như trước, hệ bao gồm hai khoá khác nhau, một khoá là công khai và một khoá kia là khoá bí mật. Bất kỳ ai với khoá công khai cũng có thể mã hoá thông báo nhưng không thể giải mã nó. Chỉ một người với khoá bí mật mới có thể giải mã được.

Trên cơ sở toán học, tiến trình này phụ thuộc vào cửa sập hàm một phía đã được trình bày ở trên. Sự mã hoá là chỉ thị dễ dàng. Lời chỉ dẫn cho sự mã hoá là khoá công khai, bất kỳ ai cũng có thể mã hoá. Sự giải mã là một chỉ thị khó khăn. Nó tạo ra khó khăn đủ để một người sử dụng máy tính Cray phải mất hàng ngàn năm mới có thể giải mã. Sự bí

mật hay cửa sập chính là khoá riêng. Với sự bí mật, sự giải mã sẽ dễ dàng như sự mã hoá.

Chúng ta hãy cùng xem xét khi máy Client gửi thông báo tới Server sử dụng hệ mã hoá công khai.

1. Client và Server nhất trí sử dụng hệ mã hóa công khai.
2. Server gửi cho Client khoá công khai của Server.
3. Client lấy bản rõ và mã hoá sử dụng khoá công khai của Server. Sau đó gửi bản mã tới cho Server.
4. Server giải mã bản mã đó sử dụng khoá riêng của mình.

Chú ý rằng hệ thống mã hoá công khai giải quyết vấn đề chính của hệ mã hoá đối xứng, bằng cách phân phối khoá. Với hệ thống mã hoá đối xứng đã qui ước, Client và Server phải nhất trí với cùng một khoá. Client có thể chọn ngẫu nhiên một khoá, nhưng nó vẫn phải thông báo khoá đó tới Server, điều này gây lãng phí thời gian. Đối với hệ thống mã hoá công khai, thì đây không phải là vấn đề.

## 3. Khoá

### *3.1 Độ dài khoá.*

Độ an toàn của thuật toán mã hoá cổ điển phụ thuộc vào hai điều đó là độ dài của thuật toán và độ dài của khoá. Nhưng độ dài của khoá dễ bị lộ hơn.

Giả sử rằng độ dài của thuật toán là lý tưởng, khó khăn lớn lao này có thể đạt được trong thực hành. Hoàn toàn có nghĩa là không có cách nào bẻ gãy được hệ thống mã hoá trừ khi cố gắng thử với mỗi khoá. Nếu khoá dài 8 bits thì có $2^8 = 256$ khoá có thể. Nếu khoá dài 56 bits, thì có $2^{56}$ khoá có thể. Giả sử rằng siêu máy tính có thể thực hiện 1 triệu phép tính một giây, nó cũng sẽ cần tới 2000 năm để tìm ra khoá thích hợp.

Nếu khoá dài 64 bits, thì với máy tính tương tự cũng cần tới xấp xỉ 600,000 năm để tìm ra khoá trong số $2^{64}$ khoá có thể. Nếu khoá dài 128 bits, nó cần tới $10^{25}$ năm , trong khi vũ trụ của chúng ta chỉ tồn tại cỡ $10^{10}$ năm. Như vậy với $10^{25}$ năm có thể là đủ dài.

Trước khi bạn gửi đi phát minh hệ mã hoá với 8 Kbyte độ dài khoá, bạn nên nhớ rằng một nửa khác cũng không kém phần quan trọng đó là thuật toán phải an toàn nghĩa là không có cách nào bẻ gãy trừ khi tìm được khoá thích hợp. Điều này không dễ dàng nhìn thấy được, hệ thống mã hoá nó như một nghệ thuật huyền ảo.

Một điểm quan trọng khác là độ an toàn của hệ thống mã hoá nên phụ thuộc vào khoá, không nên phụ thuộc vào chi tiết của thuật toán. Nếu độ dài của hệ thống mã hoá mới tin rằng trong thực tế kẻ tấn công không thể biết nội dung bên trong của thuật toán. Nếu bạn tin rằng giữ bí mật nội dung của thuật toán, tận dụng độ an toàn của hệ thống hơn là phân tích những lý thuyết sở hữu chung thì bạn đã nhầm. Và thật ngây thơ hơn khi nghĩ rằng một ai đó không thể gỡ tung mã nguồn của bạn hoặc đảo ngược lại thuật toán.

Giả sử rằng một vài kẻ thám mã có thể biết hết tất cả chi tiết về thuật toán của bạn. Giả sử rằng họ có rất nhiều bản mã, như họ mong muốn. Giả sử họ có một khối lượng bản rõ tấn công với rất nhiều dữ liệu cần thiết. Thậm chí giả sử rằng  họ có thể lựa chọn bản rõ tấn công. Nếu như hệ thống mã hoá của có thể dư thừa độ an toàn trong tất cả mọi mặt, thì bạn đã có đủ độ an toàn bạn cần.

*Tóm lại câu hỏi đặt ra trong mục này là :* ***Khoá nên dài bao nhiêu.***

Trả lời câu hỏi này phụ thuộc vào chính những ứng dụng cụ thể của bạn. Dữ liệu cần an toàn của bạn dài bao nhiêu ? Dữ liệu của bạn trị

giá bao nhiêu ? ... Thậm chí bạn có thể chỉ chỉ rõ những an toàn cần thiết theo cách sau.

> Độ dài khoá phải là một trong $2^{32}$ khoá để tương ứng với nó là kẻ tấn công phải trả 100.000.000 $ để bẻ gãy hệ thống.

### *3.2 Quản lý khoá công khai.*

Trong thực tế, quản lý khoá là vấn đề khó nhất của an toàn hệ mã hoá. Để thiết kế an toàn thuật toán mã hoá và protocol là một việc là không phải là dễ dàng nhưng để tạo và lưu trữ khoá bí mật là một điều khó hơn. Kẻ thám mã thường tấn công cả hai hệ mã hoá đối xứng và công khai thông qua hệ quản lý khoá của chúng.

Đối với hệ mã hoá công khai việc quản lý khoá dễ hơn đối với hệ mã hoá đối xứng, nhưng nó có một vấn đề riêng duy nhất. Mối người chỉ có một khoá công khai, bất kể số người ở trên mạng là bao nhiêu. Nếu Eva muốn gửi thông báo đến cho Bob, thì cô ấy cần có khoá công khai của Bob. Có một vài phương pháp mà Eva có thể lấy khoá công khai của Bob :

- Eva có thể lấy nó từ Bob.
- Eva có thể lấy từ trung tâm cơ sở dữ liệu.
- Eva có thể lấy từ cơ sở dữ liệu riêng của cô ấy.

**Chứng nhận khoá công khai :**

Chứng nhận khoá công khai là xác định khoá thuộc về một ai đó, được quản lý bởi một người đáng tin cậy. Chứng nhận để sử dụng vào việc cản trở sự cống gắng thay thế một khoá này bằng một khoá khác. Chứng nhận của Bob, trong sơ sở dữ liệu khoá công khai, lưu trữ nhiều thông tin hơn chứ không chỉ là khoá công khai. Nó lưu trữ thông tin về Bob như tên, địa chỉ, ... và nó được viết bởi ai đó mà Eva tin tưởng,

người đó thường gọi là CA(certifying authority). Bằng cách xác nhận cả khoá và thông tin về Bob. CA xác nhận thông tin về Bob là đúng và khoá công khai thuộc quyền sở hữu của Bob. Eva kiểm tra lại các dấu hiệu và sau đó cô ấy có thể sử dụng khoá công khai, sự an toàn cho Bob và không một ai khác biết. Chứng nhận đóng một vai trò rất quan trọng trong protocol của khoá công khai.

**Quản lý khoá phân phối :**

Trong một vài trường hợp, trung tâm quản lý khoá có thể không làm việc. Có lẽ không có một CA (certifying authority) nào mà Eva và Bob tin tưởng. Có lẽ họ chỉ tin tưởng bạn bè thân thiết hoặc họ không tin tưởng bất cứ ai.

Quản lý khoá phân phối, sử dụng trong những chương trình miền công khai, giải quyết vấn đề này với người giới thiệu (introducers). Người giới thiệu là một trong những người dùng khác của hệ thống anh ta là người nhận ra khoá công khai của bạn anh ta.

Ví dụ :

Khi Bob sinh ra khoá công khai, anh ta đưa bản copy cho bạn anh ấy là Bin và Dave. Họ đều biết Bob, vì vậy họ có khoá của Bob và đưa cho các dấu hiệu của anh ta. Bây giờ Bob đưa ra khoá công khai của anh ta cho người lạ, giả sử đó là Eva, Bob đưa ra khoá cùng với các dấu hiệu của hai người giới thiệu. Mặt khác nếu Eva đã biết Bin hoặc Dave, khi đó cô ta có lý do tin rằng khoá của Bob là đúng. Nếu Eva không biết Bin hoặc Dave thì cô ấy không có lý do tin tưởng khoá của Bob là đúng.

Theo thời gian, Bob sẽ tập hợp được nhiều người giới thiệu như vậy khoá của anh ta sẽ được biết đến rộng rãi hơn. Lợi ích của kỹ thuật này là không cần tới trung tâm phân phối khoá, mọi người đều có sự tín nhiệm, khi mà Eva nhận khoá công khai của Bob, sẽ không có sự bảo

đảm nào rằng cô ấy sẽ biết bất kỳ điều gì của người giới thiệu và hơn nữa không có sự đảm bảo nào là cô ấy sẽ tin vào sự đúng đắn của khoá.

## 4. Mã dòng, mã khối (CFB, CBC)

### *4.1 Mô hình mã hoá khối.*

Mã hoá sử dụng các thuật toán khối gọi đó là mã hoá khối, thông thường kích thước của khối là 64 bits. Một số thuật toán mã hoá khối sẽ được trình bày sau đây.

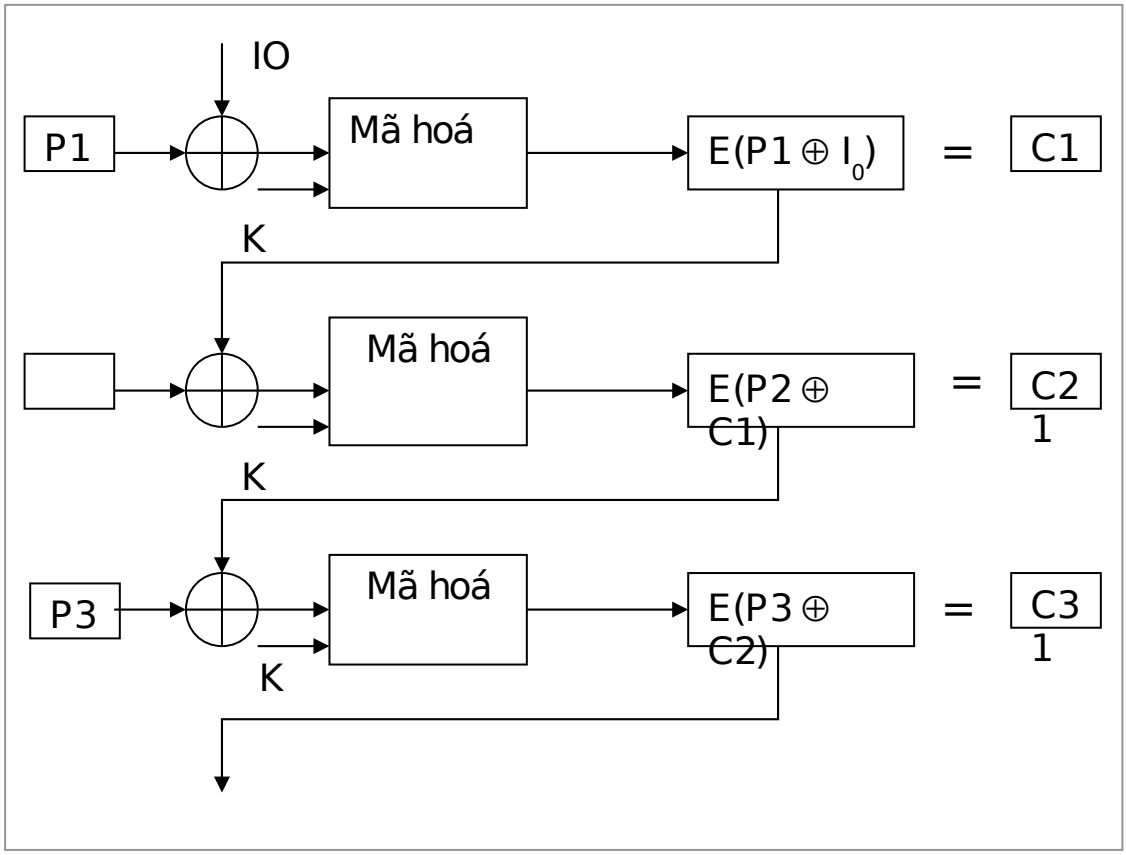
#### 4.1.1 Mô hình dây truyền khối mã hoá.

Dây truyền sử dụng kỹ thuật thông tin phản hồi, bởi vì kết quả của khối mã hoá trước lại đưa vào khối mã hoá hiện thời. Nói một cách khác khối trước đó sử dụng để sửa đổi sự mã hoá của khối tiếp theo. Mỗi khối mã hoá không phụ thuộc hoàn toàn vào khối của bản rõ.

Trong dây truyền khối mã hoá (Cipher Block Chaining Mode), bản rõ đã được XOR với khối mã hoá kế trước đó trước khi nó được mã hoá. Hình

#### 4.1.1 thể hiện các bước trong dây truyền khối mã hoá.

Sau khi khối bản rõ được mã hoá, kết quả của sự mã hoá được lưu trữ trong thanh ghi thông tin phản hồi. Trước khi khối tiếp theo của bản rõ được mã hoá, nó sẽ XOR với thanh ghi thông tin phản hồi để trở thành đầu vào cho tuyến mã hoá tiếp theo. Kết quả của sự mã hoá tiếp tục được lưu trữ trong thanh ghi thông tin phản hồi, và tiếp tục XOR với khối bản rõ tiếp theo, tiếp tục như vậy cho tới kết thúc  thông báo. Sự mã hoá của mỗi khối phụ thuộc vào tất cả các khối trước đó.




Hình 4.1.1 Sơ đồ mô hình dây chuyền khối mã hoá .

Sự giải mã là cân đối rõ ràng. Một khối mã hoá giải mã bình thường và mặt khác được cất giữ trong thanh ghi thông tin phản hồi. Sau khi khối tiếp theo được giải mã nó XOR với kết quả của thanh ghi phản hồi. Như vậy khối mã hoá tiếp theo được lưa trữ trong thanh ghi thông tin phản hồi, tiếp tục như vậy cho tới khi kết thúc thông báo.

Công thức toán học của quá trình trên như sau :

$C_i = E_K(P_i \text{ XOR } C_{i-1})$

$P_i = C_{i-1} \text{ XOR } D_K(C_i)$

### 4.1.2 Mô hình mã hoá với thông tin phản hồi.

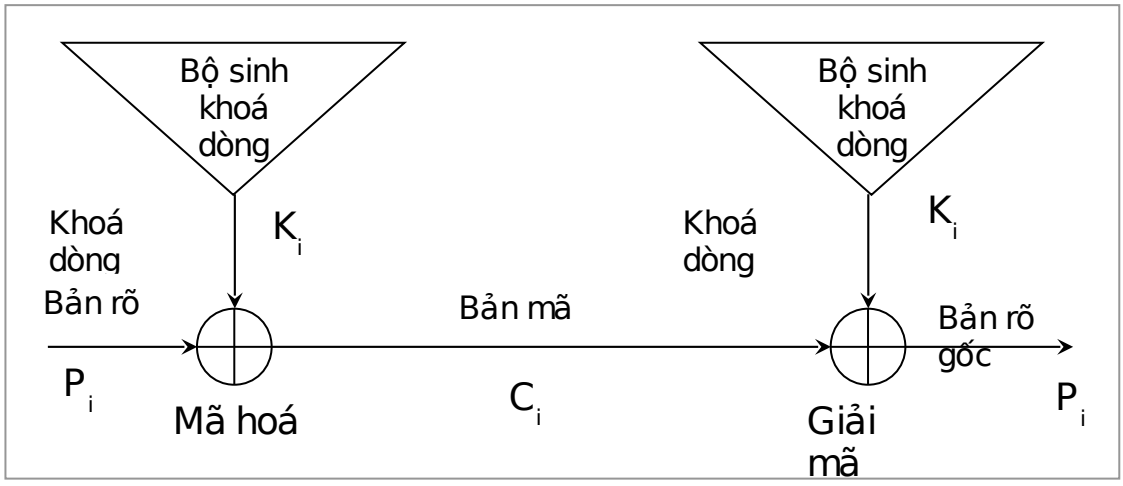
Trong mô hình dây truyền khối mã hoá(CBC_Cipher Block Chaining Mode), sự mã hóa không thể bắt đầu cho tới khi hoàn thành nhận được một khối dữ liệu. Đây thực sự là vấn đề trong một vài mạng ứng

dụng. Ví dụ, trong môi trường mạng an toàn, một thiết bị đầu cuối phải truyền mỗi ký tự tới máy trạm như nó đã được đưa vào. Khi dữ liệu phải xử lý như một khúc kích thước byte, thì mô hình dây truyền khối mã hoá là không thoả đáng.

Tại mô hình CFB dữ liệu là được mã hóa trong một đơn vị nhỏ hơn là kích thước của khối. Ví dụ sẽ mã hoá một ký tự ASCII tại một thời điểm (còn gọi là mô hình 8 bits CFB) nhưng không có gì là bất khả kháng về số 8. Bạn có thể mã hoá 1 bit dữ liệu tại một thời điểm, sử dụng thuật toán 1 bit CFB.

## *4.2 Mô hình mã hoá dòng.*

Mã hóa dòng là thuật toán, chuyển đổi bản rõ sang bản mã là 1 bit tại mỗi thời điểm. Sự thực hiện đơn giản nhất của mã hoá dòng được thể hiện trong hình 4.2




Hình 4.2 Mã hoá dòng.

Bộ sinh khoá dòng là đầu ra một dòng các bits : $k_1$, $k_2$, $k_3$, . . . $k_i$. Đây là khoá dòng đã được XOR với một dòng bits của bản rõ, $p_1$, $p_2$, $p_3$, . . $p_i$, để đưa ra dòng bits mã hoá.

$$c_i = p_i \text{ XOR } k_i$$

Tại điểm kết thúc của sự giải mã, các bits mã hoá được XOR với khoá dòng để trả lại các bits bản rõ.

$$p_i = c_i \text{ XOR } k_i$$

Từ lúc $p_i$ XOR $k_i$ XOR $k_i$ = $p_i$ là một công việc tỉ mỉ.

Độ an toàn của hệ thống phụ thuộc hoàn toàn vào bên trong bộ sinh khoá dòng. Nếu đầu ra bộ sinh khoá dòng vô tận bằng 0, thì khi đó bản rõ bằng bản mã và cả quá trình hoạt động sẽ là vô dụng. Nếu bộ sinh khoá dòng sinh ra sự lặp lại 16 bits mẫu, thì thuật toán sẽ là đơn giản với độ an toàn không đáng kể.

Nếu bộ sinh khoá dòng là vô tận của dòng ngẫu nhiên các bits, bạn sẽ có một vùng đệm (one time-pad) và độ an toàn tuyệt đối.

Thực tế mã hoá dòng nó nằm đâu đó giữa XOR đơn giản và một vùng đệm. Bộ sinh khoá dòng sinh ra một dòng bits ngẫu nhiên, thực tế điều này quyết định thuật toán có thể hoàn thiện tại thời điểm giải mã. Đầu ra của bộ sinh khoá dòng là ngẫu nhiên, như vậy người phân tích mã sẽ khó khăn hơn khi bẻ gãy khoá. Như bạn đã đoán ra được rằng, tạo một bộ sinh khoá dòng mà sản phẩm đầu ra ngẫu nhiên là một vấn đề không dễ dàng.

## 5. Các hệ mật mã đối xứng và công khai

### *5.1 Hệ mật mã đối xứng*

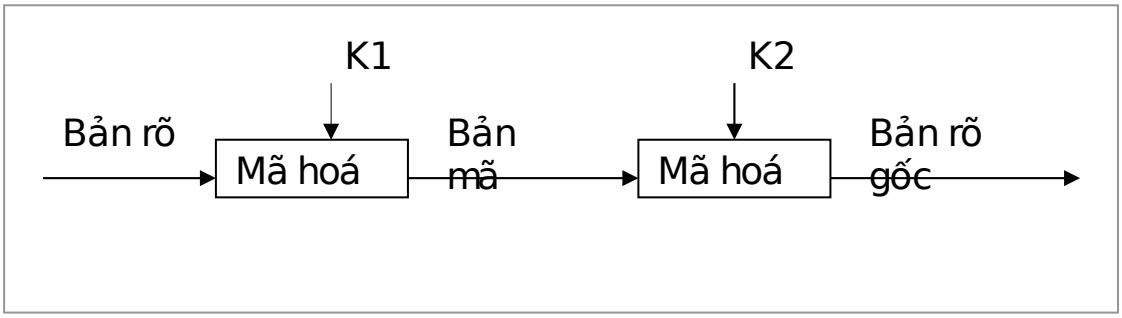
Thuật toán đối xứng hay còn gọi thuật toán mã hoá cổ điển là thuật toán mà tại đó khoá mã hoá có thể tính toán ra được từ khoá giải mã. Trong rất nhiều trường hợp, khoá mã hoá và khoá giải mã là giống

nhau. Thuật toán này còn có nhiều tên gọi khác như thuật toán khoá bí mật, thuật toán khoá đơn giản, thuật toán một khoá. Thuật toán này yêu cầu người gửi và người nhận phải thoả thuận một khoá trước khi thông báo được gửi đi, và khoá này phải được cất giữ bí mật. Độ an toàn của thuật toán này vẫn phụ thuộc và khoá, nếu để lộ ra khoá này nghĩa là bất kỳ người nào cũng có thể mã hoá và giải mã thông báo trong hệ thống mã hoá.

Sự mã hoá và giải mã của thuật toán đối xứng biểu thị bởi :

$E_K( P ) = C$

$D_K( C ) = P$




Hình 5.1 Mã hoá và giải mã với khoá đối xứng .

Trong hình vẽ trên thì :

K1có thể trùng K2, hoặc

K1 có thể tính toán từ K2, hoặc

K2 có thể tính toán từ K1.

## *Một số nhược điểm của hệ mã hoá cổ điển*

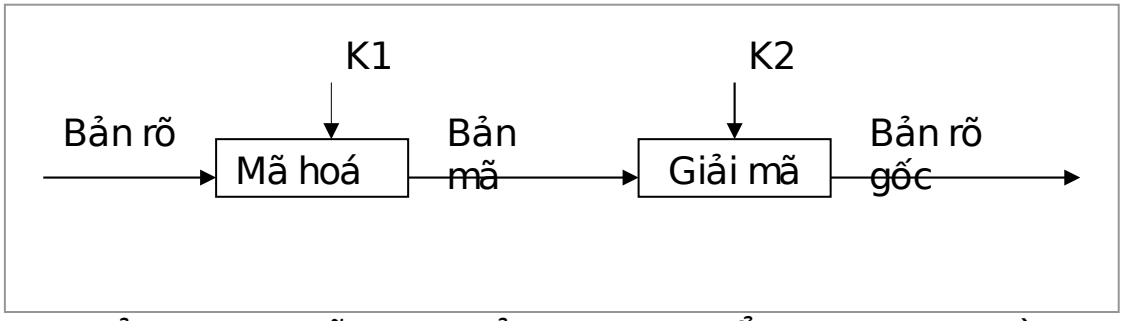
- Các phương mã hoá cổ điển đòi hỏi người mã hoá và người giải mã phải cùng chung một khoá. Khi đó khoá phải được giữ bí mật tuyệt đối, do vậy ta dễ dàng xác định một khoá nếu biết khoá kia.

- Hệ mã hoá đối xứng không bảo vệ được sự an toàn nếu có xác suất cao khoá người gửi bị lộ. Trong hệ khoá phải được gửi đi trên kênh an toàn nếu kẻ địch tấn công trên kênh này có thể phát hiện ra khoá.
- Vấn đề quản lý và phân phối khoá là khó khăn và phức tạp khi sử dụng hệ mã hoá cổ điển. Người gửi và người nhận luôn luôn thông nhất với nhau về vấn đề khoá. Việc thay đổi khoá là rất khó và dễ bị lộ.
- Khuynh hướng cung cấp khoá dài mà nó phải được thay đổi thường xuyên cho mọi người trong khi vẫn duy trì cả tính an toàn lẫn hiệu quả chi phí sẽ cản trở rất nhiều tới việc phát triển hệ mật mã cổ điển.

## *5.2 Hệ mật mã công khai*

Vào những năm 1970 Diffie và Hellman đã phát minh ra một hệ mã hoá mới được gọi là **hệ mã hoá công khai** hay **hệ mã hoá phi đối xứng.**

Thuật toán mã hoá công khai là khác biệt so với thuật toán đối xứng. Chúng được thiết kế sao cho **khoá** sử dụng vào việc mã hoá là khác so




với **khoá** giải mã. Hơn nữa khoá giải mã không thể tính toán được từ khoá mã hoá. Chúng được gọi với tên hệ thống mã hoá công khai bởi vì khoá để mã hoá có thể công khai, một người bất kỳ có thể sử dụng khoá công khai để mã hoá thông báo, nhưng chỉ một vài người có đúng khoá giải mã thì mới có khả năng giải mã. Trong nhiều hệ thống, khoá mã hoá gọi là khoá công khai (public key), khoá giải mã thường được gọi là khoá riêng (private key).

Hình 5.2 Mã hoá và giải mã với hai khoá .

Trong hình vẽ trên thì :

K1 không thể trùng K2, hoặc

K2 không thể tính toán từ K1.

Đặc trưng nổi bật của hệ mã hoá công khai là cả khoá công khai(public key) và bản tin mã hoá (ciphertext) đều có thể gửi đi trên một kênh thông tin không an toàn.

***Diffie và Hellman đã xác đinh rõ các điều kiện của một hệ mã hoá công khai như sau :***

1. Việc tính toán ra cặp khoá công khai $K_B$ và bí mật $k_B$ dựa trên cơ sở các điều kiện ban đầu phải được thực hiện một cách dễ dàng, nghĩa là thực hiện trong thời gian đa thức.

2. Người gửi A có được khoá công khai của người nhận B và có bản tin P cần gửi đi thì có thể dễ dàng tạo ra được bản mã C.

$$C = EKB (P) = EB (P)$$

Công việc này cũng trong thời gian đa thức.

3. Người nhận B khi nhận được bản tin mã hóa C với khoá bí mật $k_B$ thì có thể giải mã bản tin trong thời gian đa thức.

$$P = DkB (C) = DB[EB(M)]$$

4. Nếu kẻ địch biết khoá công khai $K_B$ cố gắng tính toán khoá bí mật thì khi đó chúng phải đương đầu với trường hợp nan giải, trường hợp này đòi hỏi nhiều yêu cầu không khả thi về thời gian.
5. Nếu kẻ địch biết được cặp ($K_B$,C) và cố gắng tính toán ra bản rõ P thì giải quyết bài toán khó với số phép thử là vô cùng lớn, do đó không khả thi.

## 6. Các cách thám mã

Có sáu phương pháp chung để phân tích tấn công, dưới đây là danh sách theo thứ tự khả năng của từng phương pháp. Mỗi phương pháp trong số chúng giả sử rằng kẻ thám mã hoàn toàn có hiểu biết về thuật toán mã hoá được sử dụng.

1. **Chỉ có bản mã.** Trong trường hợp này, người phân tích chỉ có một vài bản tin của bản mã, tất cả trong số chúng đều đã được mã hoá và cùng sử dụng chung một thuật toán. Công việc của người phân tích là tìm lại được bản rõ của nhiều bản mã có thể hoặc tốt hơn nữa là suy luận ra được khoá sử dụng mã hoá, và sử dụng để giải mã những bản mã khác với cùng khoá này.

Giả thiết : $C_1 = E_k(P_1)$, $C_2 = E_k(P_2)$, . . .$C_i = E_k(P_i)$

Suy luận : Mỗi $P_1, P_2$, . . $P_i$, k hoặc thuật toán kết luận $P_{i+1}$ từ

$C_{i+1} = E_k(P_{i+1})$

2. **Biết bản rõ.** Người phân tích không chỉ truy cập được một vài bản mã mặt khác còn biết được bản rõ. Công việc là suy luận ra khoá để sử dụng giải mã hoặc thuật toán giải mã để giải mã cho bất kỳ bản mã nào khác với cùng khoá như vậy.

   Giả thiết : $P_1$, $C_1 = E_k(P_1)$, P2, C2= $E_k(P_2)$, . . . Pi, $C_i = E_k(P_i)$

   Suy luận : Mỗi k hoặc thuật toán kết luận $P_{i+1}$ từ $C_{i+1} = E_k(P_{i+1})$

3. **Lựa chọn bản rõ.** Người phân tích không chỉ truy cập được bản mã và kết hợp bản rõ cho một vài bản tin, nhưng mặt khác lựa chọn bản rõ đã mã hoá. Phương pháp này tỏ ra có khả năng hơn phương pháp **biết bản rõ** bởi vì người phân tích có thể chọn cụ thể khối bản rõ cho mã hoá, một điều khác có thể là sản lượng thông tin về khoá nhiều hơn.

   Giả thiết : $P_1$, $C_1 = E_k(P_1)$, P2, C2= $E_k(P_2)$, . . . Pi, $C_i = E_k(P_i)$

   tại đây người phân tích chọn $P_1, P_2$,. . . $P_i$

   Suy luận : Mỗi k hoặc thuật toán kết luận $P_{i+1}$ từ $C_{i+1} = E_k(P_{i+1})$

4. **Mô phỏng lựa chọn bản rõ.** Đây là trường hợp đặc biệt của lựa chọn bản rõ. Không chỉ có thể lựa chọn bản rõ đã mã hoá, nhưng họ còn có thể sửa đổi sự lựa chọn cơ bản kết quả của sự mã hoá lần trước. Trong trường lựa chọn bản mã người phân tích có thể đã chọn một khối lớn bản rõ đã mã hoá, nhưng trong trường hợp này có thể chọn một khối nhỏ hơn và chọn căn cứ khác trên kết quả của lần đầu tiên.

5. **Lựa chọn bản mã.** Người phân tích có thể chọn bản mã khác nhau đã được mã hoá và truy cập bản rõ đã giải mã. Trong ví dụ khi một người phân tích có một hộp chứng cớ xáo chộn không thể tự động giải mã, công việc là suy luận ra khoá.

   Giả thiết : $C_1$, $P_1 = D_k(C_1)$, C2, P2= $D_k(C_2)$, . . . Ci, $P_i = D_k(C_i)$ tại Suy luận : k

6. **Lựa chọn khoá.** Đây không phải là một cách tấn công khi mà bạn đã có khoá. Nó không phải là thực hành thám mã mà chỉ là sự giải mã thông thường, bạn chỉ cần lựa chọn khoá cho phù hợp với bản mã.

Một điểm đáng chú ý khác là đa số các kỹ thuật thám mã đều dùng phương pháp thống kê tần suất xuất hiện của các từ, các ký tự trong bản mã. Sau đó thực hiện việc thử thay thế với các chữ cái có tần suất xuất hiện tương đồng trong ngôn ngữ tự nhiên. Tại đây chúng ta chỉ xem xét đối với ngôn ngữ thông dụng nhất hiện nay đó là tiếng Anh. Việc thống kê tần suất xuất hiện của các ký tự trong trường hợp này được tiến hành dựa trên các bài báo, sách, tạp chí và các văn bản cùng với một số loại khác ...

Sau đây là bảng thống kê tần suất xuất hiện của 26 chữ cái trong bảng chữ cái tiếng Anh theo tài liệu của Beker và Piper.

| Ký tự | Xác Suất | Ký tự | Xác suất | Ký tự | Xác suất |
|---|---|---|---|---|---|
| A | 0.082 | J | 0.002 | S | 0.063 |
| B | 0.015 | K | 0.008 | T | 0.091 |
| C | 0.028 | L | 0.040 | U | 0.028 |
| D | 0.043 | M | 0.024 | V | 0.010 |
| E | 0.127 | N | 0.067 | W | 0.023 |
| F | 0.022 | O | 0.075 | X | 0.001 |
| G | 0.020 | P | 0.019 | Y | 0.020 |
| H | 0.061 | Q | 0.001 | Z | 0.001 |

| | | | | | |
|---|---|---|---|---|---|
| I | 0.070 | R | 0.060 | | |

Cùng với việc thống kê các tần xuất của các ký tự trong tiếng Anh, việc thống kê tần suất xuất hiện thường xuyên của các dãy gồm 2 hoặc 3 ký tự liên tiếp nhau cũng có một vai trò quan trọng trong công việc thám mã. Sysu Deck đưa ra 30 bộ đôi xuất hiện thường xuyên của tiếng Anh được sắp theo thứ tự giảm dần như sau :

Tính hữu dụng của các phép thống kê ký tự và các dãy ký tự được người phân tích mã khai thác triệt để trong những lần thám mã. Khi thực hiện việc thám mã người phân tích thống kê các ký tự trong bản mã, từ đó so sánh với bản thống kê mẫu và đưa ra các ký tự phỏng đoán tương tự. Phương pháp này được sử dụng thường xuyên và đem lại hiệu quả khá cao.

| Cặp chữ | Tần suất | Cặp chữ | Tần suất | Cặp chữ | Tần suất |
|---|---|---|---|---|---|
| TH | 10.00 | ED | 4.12 | OF | 3.38 |
| HE | 9.50 | TE | 4.04 | IT | 3.26 |
| IN | 7.17 | TI | 4.00 | AL | 3.15 |
| ER | 6.65 | OR | 3.98 | AS | 3.00 |
| RE | 5.92 | ST | 3.81 | HA | 3.00 |
| ON | 5.70 | AR | 3.54 | NG | 2.92 |
| AN | 5.63 | ND | 3.52 | CO | 2.80 |
| EN | 4.76 | TO | 3.50 | SE | 2.75 |
| AT | 4.72 | NT | 3.44 | ME | 2.65 |
| ES | 4.24 | IS | 3.43 | DE | 2.65 |

# Chương III **Hệ mã hoá RSA**.

Với đề tài xây dựng thư viện các hàm mã hoá dùng cho việc bảo mật thông tin trao đổi trong mô hình Client/Server, thì cần thiết một phương pháp mã hoá để áp dụng, thuật toán mã hoá công khai RSA đã được lựa chọn cho giải pháp này. Phương pháp này có những ưu điểm, nhược điểm, đặc tính gì đó là phần sẽ trình bày trong chương này

- Khái niệm hệ mật mã RSA
- Phân phối khoá công kkai trong RSA
- Độ an toàn của hệ RSA
- Một số tính chất của hệ RSA

## 1. Khái niệm hệ mật mã RSA

Khái niệm hệ mật mã RSA đã được ra đời năm 1976 bởi các tác giả R.Rivets, A.Shamir, và L.Adleman. Hệ mã hoá này dựa trên cơ sở của hai bài toán :

+ Bài toán Logarithm rời rạc (Discrete logarith)

+ Bài toán phân tích thành thừa số.

Trong hệ mã hoá RSA các bản rõ, các bản mã và các khoá (public key và private key) là thuộc tập số nguyên $Z_N = \{1, \ldots, N-1\}$. Trong đó tập $Z_N$ với $N=p\times q$ là các số nguyên tố khác nhau cùng với phép cộng và phép nhân Modulo N tạo ra modulo số học N.

Khoá mã hoá $E_{KB}$ là cặp số nguyên $(N,K_B)$ và khoá giải mã $D_{kb}$ là cặp số nguyên $(N,k_B)$, các số là rất lớn, số N có thể lên tới hàng trăm chữ số.

Các phương pháp mã hoá và giải mã là rất dễ dàng.

Công việc mã hoá là sự biến đổi bản rõ P (Plaintext) thành bản mã C (Ciphertext) dựa trên cặp khoá công khai $K_B$ và bản rõ P theo công thức sau đây :

$$C = E_{KB}(P) = E_B(P) = P^{KB} \pmod N . \quad (1)$$

Công việc giải mã là sự biến đổi ngược lại bản mã C thành bản rõ P dựa trên cặp khoá bí mật $k_B$, modulo N theo công thức sau :

$$P = D_{kB}(C) = D_B(C) = C^{kB} \pmod N . \quad (2)$$

Dễ thấy rằng, bản rõ ban đầu cần được biến đổi một cách thích hợp thành bản mã, sau đó để có thể tái tạo lại bản rõ ban đầu từ chính bản mã đó :

$$P = D_B(E_B(P)) \quad (3)$$

Thay thế (1) vào (2) ta có :

$$(P^{KB})^{kB} = P \pmod N \quad (4)$$

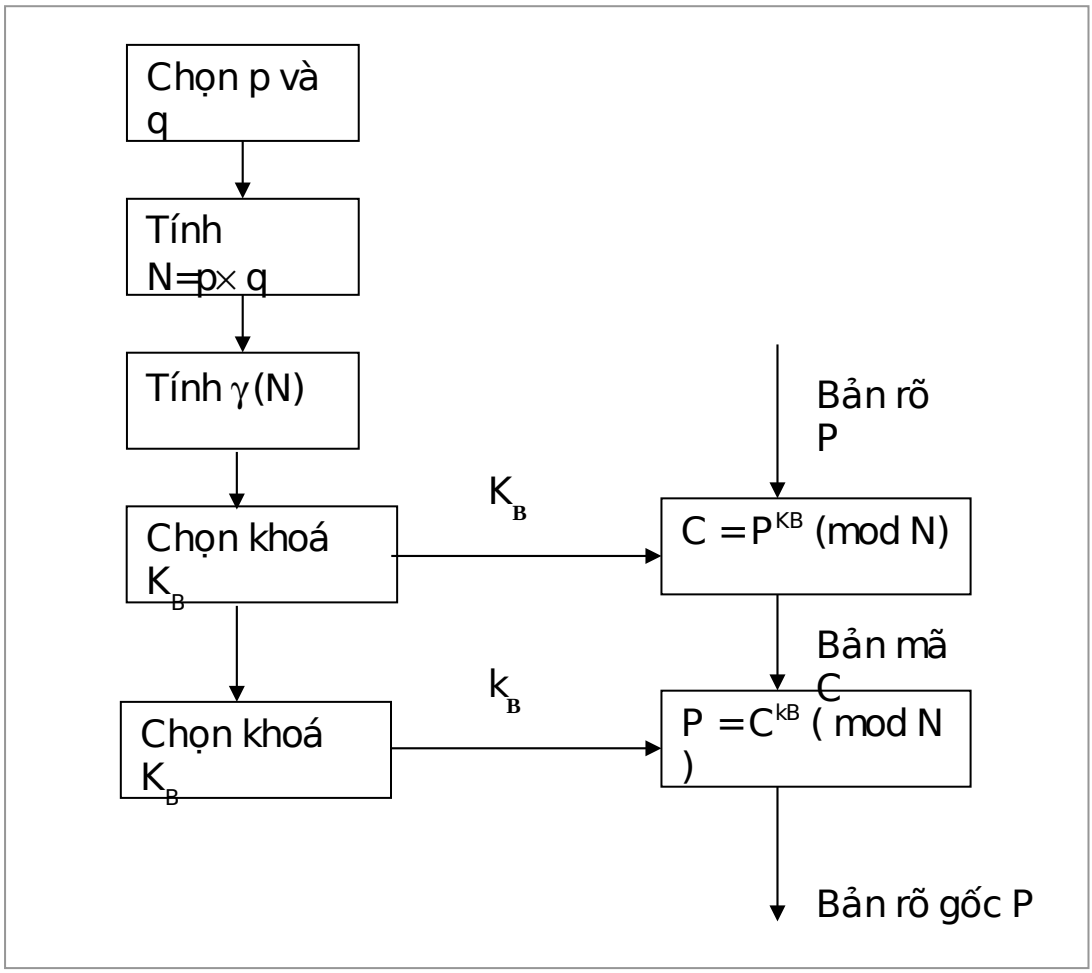
Trong toán học đã chứng minh được rằng, nếu N là số nguyên tố thì công thức (4) sẽ có lời giải khi và chỉ khi $K_B.k_B = 1 \pmod{N-1}$, áp dụng thuật toán ta thấy N=p× q với p, q là số nguyên tố, do vậy (4) sẽ có lời giải khi và chỉ khi :

$$K_B.k_B \equiv 1 \pmod{\gamma(N)} \quad (5)$$

trong đó $\gamma(N) = LCM(p-1,q-1)$ .

LCM (Lest Common Multiple) là bội số chung nhỏ nhất.

Nói một cách khác, đầu tiên người nhận B lựa chọn một khoá công khai $K_B$ một cách ngẫu nhiên. Khi đó khoá bí mật $k_B$ được tính ra bằng công thức (5). Điều này hoàn toàn tính được vì khi B biết được cặp số nguyên tố (p,q) thì sẽ tính được $\gamma(N)$.




Hình 1.1 Sơ đồ các bước thực hiện mã hoá theo thuật toán RSA.

## 2. Độ an toàn của hệ RSA

Một nhận định chung là tất cả các cuộc tấn công giải mã đều mang mục đích không tốt. Trong phần độ an toàn của hệ mã hoá RSA sẽ đề cập đến một vài phương thức tấn công điển hình của kẻ địch nhằm giải mã trong thuật toán này.

Chúng ta xét đến trường hợp khi kẻ địch nào đó biết được modulo N, khoá công khai $K_B$ và bản tin mã hoá C, khi đó kẻ địch sẽ tìm ra bản tin gốc (Plaintext) như thế nào. Để làm được điều đó kẻ địch thường tấn vào hệ thống mật mã bằng hai phương thức sau đây:

- Phương thức thứ nhất :

Trước tiên dựa vào phân tích thừa số modulo N. Tiếp theo sau chúng sẽ tìm cách tính toán ra hai số nguyên tố p và q, và có khả năng thành công khi đó sẽ tính được $\lambda(N)$ và khoá bí mật $k_B$. Ta thấy N cần phải là tích của hai số nguyên tố, vì nếu N là tích của hai số nguyên tố thì thuật toán phân tích thừa số đơn giản cần tối đa $\sqrt{N}$ bước, bởi vì có một số nguyên tố nhỏ hơn $\sqrt{N}$ . Mặt khác, nếu N là tích của n số nguyên tố, thì thuật toán phân tích thừa số đơn giản cần tối đa $N^{1/n}$ bước.

Một thuật toán phân tích thừa số có thể thành phức tạp hơn, cho phép phân tích một số N ra thành thừa số trong O( $\sqrt{P}$ ) bước, trong đó p là số chia nhỏ nhất của N, việc chọn hai số nguyên tố là cho thuật toán tăng hiệu quả.

- Phương thức thứ hai :

Phương thức tấn công thứ hai vào hệ mã hoá RSA là có thể khởi đầu bằng cách giải quyết trường hợp thích hợp của bài toán logarit rời rạc. Trường hợp này kẻ địch đã có trong tay bản mã C và khoá công khai $K_B$ tức là có cặp ($K_B$,C)

Cả hai phương thức tấn công đều cần một số bước cơ bản, đó là :

$$O(\exp \sqrt{\ln N \ln(\ln N)}), \text{ trong đó N là số modulo.}$$

## 3. Một số tính chất của hệ RSA

- ***Trong các hệ mật mã RSA, một bản tin có thể được mã hoá trong thời gian tuyến tính.***

Đối với các bản tin dài, độ dài của các số được dùng cho các khoá có thể được coi như là hằng. Tương tự như vậy, nâng một số lên luỹ thừa được thực hiện trong thời gian hằng, các số không được phép dài hơn một độ dài hằng. Thực ra tham số này che dấu nhiều chi tiết cài đặt có

liên quan đến việc tính toán với các con số dài, chi phí của các phép toán thực sự là một yếu tố ngăn cản sự phổ biến ứng dụng của phương pháp này. Phần quan trọng nhất của việc tính toán có liên quan đến việc mã hoá bản tin. Nhưng chắc chắn là sẽ không có hệ mã hoá nào hết nếu không tính ra được các khoá của chúng là các số lớn.

- ***Các khoá cho hệ mã hoá RSA có thể được tạo ra mà không phải tính toán quá nhiều.***

Một lần nữa, ta lại nói đến các phương pháp kiểm tra số nguyên tố. Mỗi số nguyên tố lớn có thể được phát sinh bằng cách đầu tiên tạo ra một số ngẫu nhiên lớn, sau đó kiểm tra các số kế tiếp cho tới khi tìm được một số nguyên tố. Một phương pháp đơn giản thực hiện một phép tính trên một con số ngẫu nhiên, với xác suất 1/2 sẽ chứng minh rằng số được kiểm tra không phải nguyên tố. Bước cuối cùng là tính p dựa vào thuật toán Euclid.

Như phần trên đã trình bày trong hệ mã hoá công khai thì khoá giải mã (private key) $k_B$ và các thừa số p,q là được giữ bí mật và sự thành công của phương pháp là tuỳ thuộc vào kẻ địch có khả năng tìm ra được giá trị của $k_B$ hay không nếu cho trước N và $K_B$. Rất khó có thể tìm ra được $k_B$ từ $K_B$ cần biết về p và q, như vậy cần phân tích N ra thành thừa số để tính p và q. Nhưng việc phân tích ra thừa số là một việc làm tốn rất nhiều thời gian, với kỹ thuật hiện đại ngày nay thì cần tới hàng triệu năm để phân tích một số có 200 chữ số ra thừa số.

Độ an toàn của thuật toán RSA dựa trên cơ sở những khó khăn của việc xác định các thừa số nguyên tố của một số lớn. Bảng dưới đây cho biết các thời gian dự đoán, giả sử rằng mỗi phép toán thực hiện trong một micro giây.

| Số các chữ số trong số được phân tích | Thời gian phân tích |
|---|---|
| 50 | 4 giờ |
| 75 | 104 giờ |
| 100 | 74 năm |
| 200 | 4.000.000 năm |
| 300 | $5\times 10^{15}$ năm |
| 500 | $4\times 10^{25}$ năm |

# Chương IV Mô hình Client/Server

Trong thực tế, mô hình Client/Server đã trở nên rất phổ biến trong hệ thống mạng điểm tới điểm, và chúng được áp dụng hầu hết cho những máy tính truyền thông ngày nay. Kiến trúc mô hình Client/Server và khi nào cần mã hoá thông tin truyền trong Client/Server là chủ đề sẽ được trình bày trong chương này.

## 1.Mô hình Client/Server

Nói chung, một ứng dụng khởi tạo truyền thông từ điểm tới điểm được gọi là client. Người dùng cuối thường xuyên gọi phần mềm client khi họ cần tới những dịch vụ trên mạng. Mô hình Client/Server cố gắng tổ chức lại các máy PC, trên mạng cụ bộ, để thích hợp với các máy tính lớn mainframe, tăng tính thích ứng, tính hiệu quả của hệ thống. Mặc dù có sự thay đổi rất lớn các quan điểm về mô hình Client/Server, nhưng chúng có một vài đặc tính dưới đây.

- Máy Client là các máy PC hay là các workstations, truy cập vào mạng và sử dụng các tài nguyên trên mạng.
- Giao diện người sử dụng với Client, nói chung sử dụng giao diện người dùng đồ hoạ (GUI), ví như Microsoft Windowns
- Trong hệ thống Client/Server có một vài Client, với mỗi Client sử dụng giao diện riêng của mình. Các Client sử dụng các tài nguyên được chia sẻ bởi Server.
- Server có thể là một workstation lớn, như mainframe, minicomputer, hoặc các thiết bị mạng LAN.
- Client có thể gửi các truy vấn hoặc các lệnh tới Server, nhưng thực hiện tiến trình này không phải là Client.
- Server trả lại kết quả trên màn hình của Client.

- Các loại Server thông thường là : database server, file server, print server, image-processing server, computing server và communication server.
- Server không thể khởi tạo bất kỳ công việc nào, nhưng nó thực hiện các yêu cầu to lớn của Client.
- Nhiệm vụ chia là hai phần : phần mặt trước thực hiện bởi client, và phần mặt sau thực hiện bởi Server.
- Server thực hiện việc chia sẻ File, lưu trữ và tìm ra các thông tin, mạng và quản lý tài liệu, quản lý thư điện tử, bảng thông báo và văn bản video.

## 2. Mã hoá trong mô hình Client/Server.

Trong mô hình Client/Server việc trao đổi thông tin diễn ra thường xuyên nên rất dễ bị kẻ xấu lợi dụng, bởi vậy bảo vệ thông tin trên đường truyền là vô cùng quan trọng, chúng đảm bảo thông tin trên đường truyền là đúng đắn. Tại mô hình này mỗi khi những yêu cầu được gửi từ Client đến Server hoặc khi Server gửi trả lại kết quả cho Client thì những thông tin này đều được mã hoá trong khi truyền.

# Chương V Xây dựng hàm thư viện

Xu hướng trên thế giới hiện nay là phần mềm được bán và phân phối ở dạng các modul phần mềm. Các hình thức của modul phụ thuộc vào các gói phần mềm cụ thể và các ngôn ngữ mà người sử dụng dùng. Ví dụ bạn có thể tạo các thư viện tĩnh với các file có phần mở rộng .LIB hoặc bạn có thể tạo một điều khiển ActiveX với phần mở rộng OCX, hoặc hơn nữa bạn có thể tạo các thư viện liên kết động với các file .DLL .

Các ngôn ngữ lập trình hiện nay có tính modul độc lập rất cao, nghĩa là bạn có thể tạo ra các ứng dụng bằng cách kết hợp nhiều modul phần mềm độc lập nhau thành một ứng dụng cụ thể. Thông thường khi thiết kế một phần mềm ứng dụng thuộc loại phức tạp, bạn sẽ tìm kiếm các modul có thể sử dụng được để giảm chi phí, giảm thời gian thiết kế và tập chung nhiều hơn cho những phần ứng dụng tự bạn viết ra.

Một câu hỏi đặt ra tại đây là vì sao chúng ta lại không tạo ra các hàm thực hiện các công việc chuyên biệt và phân phối nó cho người sử dụng, có một vài lý do sau đây không cho phép thực hiện điều này :

- Người dùng có thể vô tình thay đổi làm xáo trộn các lệnh trong chương trình.
- Bạn không muốn người dùng biết "bí quyết" của bạn mà chỉ muốn họ sử dụng kết quả bạn tạo ra.

Trong chương này của cuốn luận văn trình bày thư viện liên kết động là gì, và chúng thực hiện như thế nào. Thư viện liên kết động DLL (Dynamic Link Library) là một tập tin thư viện chứa các hàm. Người lập trình có thể gọi một tập tin DLL vào trong chương trình của họ và sử dụng các hàm trong DLL đó.

DLL là một thư viện liên kết động với các chương trình sử dụng nó, nghĩa là khi bạn tạo ra tập tin EXE của chương trình mà không cần liên

kết tập tin DLL với chương trình của bạn. Tập tin DLL sẽ được liên kết động với chương trình trong thời gian thi hành chương trình. Bởi vậy khi viết một ứng dụng có sử dụng DLL, bạn phải phân phối tập tin DLL cùng với tập tin EXE của chương trình bạn viết.

## 1.Xây dựng thư viện liên kết động CRYPTO.DLL

Thư viện **crypto.dll** được xây dựng dới đây cung cấp cho các bạn các hàm cần thiết phục vụ cho việc mã hoá thông tin, chúng bao gồm

int enciph(char *, char *) : hàm mã hoá.

int deciph(char *, char *) : hàm giải mã.

- **Hàm Enciph.c**

Các bạn có thể sử dụng hàm này để thực hiện các thao tác mã hoá với xâu kí tự, bằng cách đưa vào một xâu ký tự (bản rõ) ở đầu ra bạn sẽ nhận được một xâu ký tự đã được mã hoá (bản mã). Với bản mã này các bạn có thể yên tâm về nội dụng thông tin sẽ rất khó bị lộ. Hàm thực hiện có sử dụng khoá công khai lấy vào từ File PUBLIC.KEY.

```
//============================
// Ham Enciph.c
#include <stdio.h>
#include <conio.h>
#include <miracl.h>
#include <stdlib.h>
#include <string.h>

/*
#define RSA
*/
int enciph(char  *sin,char *sout)
{  /*  encipher using public key  */
```

```
   big x,ke;
   FILE *ifile;
   int ch,i,leng;
   long seed;
   miracl *mip=mirsys(100,0);
   x=mirvar(0);
   ke=mirvar(0);
   mip->IOBASE=60;

   if ((ifile=fopen("public.key","r"))==NULL)
   {
       return 1;
   }
   cinnum(ke,ifile);
   fclose(ifile);
   seed=123456789;
   irand(seed);
   bigrand(ke,x);
   leng=strlen(sin);
   for(i=0; i <= (leng-1); i++)
   { /* encipher character by character */
#ifdef RSA
       power(x,3,ke,x);
#else
       mad(x,x,x,ke,ke,x);
#endif
       ch=*(sin+i);
       ch^=x[1];          /* XOR with last byte of x */
       sout[i]=ch;
          }
   return 0;
}
```

```
//============================
miracl *mirsys(int nd,mr_small nb)
{  /*  Initialize MIRACL system to   *
   *  use numbers to base nb, and   *
   *  nd digits or (-nd) bytes long */
   int i;
   mr_small b;
   mr_mip=(miracl *)mr_alloc(1,sizeof(miracl));
   mr_mip->depth=0;
   mr_mip->trace[0]=0;
   mr_mip->depth++;
   mr_mip->trace[mr_mip->depth]=25;
if (MIRACL>=MR_IBITS) mr_mip->TOOBIG =(1<<(MR_IBITS-2));
   else
   mr_mip->TOOBIG =(1<<(MIRACL-1));

#ifdef MR_FLASH
   mr_mip->BTS=MIRACL/2;
   if (mr_mip->BTS==MR_IBITS) mr_mip->MSK=(-1);
   else  mr_mip->MSK=(1<<(mr_mip->BTS))-1;
   #endif

#ifdef MR_NO_STANDARD_IO
   mr_mip->ERCON=TRUE;
#else
   mr_mip->ERCON=FALSE;
#endif
   mr_mip->N=0;
   mr_mip->MSBIT=((mr_small)1<<(MIRACL-1));
   mr_mip->OBITS=mr_mip->MSBIT-1;
   mr_mip->user=NULL;
   mr_set_align(0);
```

```
#ifdef MR_NOFULLWIDTH
   if (nb==0)
   {
      mr_berror(MR_ERR_BAD_BASE);
      mr_mip->depth--;
      return mr_mip;
   }
#endif
   if (nb==1 || nb>MAXBASE)
   {
      mr_berror(MR_ERR_BAD_BASE);
      mr_mip->depth--;
      return mr_mip;
   }
   mr_setbase(nb);
   b=mr_mip->base;
   mr_mip->lg2b=0;
   mr_mip->base2=1;
   if (b==0)
   {
      mr_mip->lg2b=MIRACL;
      mr_mip->base2=0;
   }
   else while (b>1)
   {
      b/=2;
      mr_mip->lg2b++;
      mr_mip->base2*=2;
   }
   if (nd>0)
      mr_mip->nib=(nd-1)/mr_mip->pack+1;
```

```
   else
      mr_mip->nib=(mr_mip->lg2b-8*nd-1)/mr_mip->lg2b;
   if (mr_mip->nib<2) mr_mip->nib=2;
#ifdef MR_FLASH
   mr_mip->workprec=mr_mip->nib;
   mr_mip->stprec=mr_mip->nib;
while(mr_mip->stprec>2 && mr_mip->stprec> MR_FLASH/  mr_mip->lg2b)
   mr_mip->stprec=(mr_mip->stprec+1)/2;
   if (mr_mip->stprec<2) mr_mip->stprec=2;
   mr_mip->pi=NULL;
#endif
   mr_mip->check=ON;
   mr_mip->IOBASE=10; mr_mip->ERNUM=0;
   mr_mip->RPOINT=OFF;
   mr_mip->NTRY=6;
   mr_mip->EXACT=TRUE;
   mr_mip->TRACER=OFF;
   mr_mip->INPLEN=0;
   mr_mip->PRIMES=NULL;
   mr_mip->IOBUFF=mr_alloc(MR_IOBSIZ+1,1);
for (i=0;i<NK;i++) mr_mip->ira[i]=0L;
   irand(0L);
mr_mip->nib=2*mr_mip->nib+1;
#ifdef MR_FLASH
      if (mr_mip->nib!=(mr_mip->nib&(mr_mip->MSK)) || mr_mip->nib > mr_mip-
>TOOBIG)
#else
   if(mr_mip->nib!=(mr_mip->nib&(mr_mip->OBITS)) || mr_mip->nib>mr_mip-
            >TOOBIG)
#endif
   {
      mr_berror(MR_ERR_TOO_BIG);
```

```
        mr_mip->nib=(mr_mip->nib-1)/2;
        mr_mip->depth--;
        return mr_mip;
    }
    mr_mip->modulus=NULL;
    mr_mip->A=NULL;
    mr_mip->B=NULL;
    mr_mip->fin=FALSE;
    mr_mip->fout=FALSE;
    mr_mip->active=ON;
    mr_mip->w0=mirvar(0); /* w0 is double length  */
    mr_mip->nib=(mr_mip->nib-1)/2;
#ifdef MR_KCM
    mr_mip->big_ndash=NULL;
    mr_mip->ws=mirvar(0);
#endif
    mr_mip->w1=mirvar(0); /* initialize workspace */
    mr_mip->w2=mirvar(0);
    mr_mip->w3=mirvar(0);
    mr_mip->w4=mirvar(0);
    mr_mip->nib=2*mr_mip->nib+1;
    mr_mip->w5=mirvar(0);
    mr_mip->w6=mirvar(0);
    mr_mip->w7=mirvar(0);
    mr_mip->nib=(mr_mip->nib-1)/2;
    mr_mip->w5d=&(mr_mip->w5[mr_mip->nib+1]);
    mr_mip->w6d=&(mr_mip->w6[mr_mip->nib+1]);
    mr_mip->w7d=&(mr_mip->w7[mr_mip->nib+1]);

    mr_mip->w8=mirvar(0);
    mr_mip->w9=mirvar(0);
    mr_mip->w10=mirvar(0);
```

```
    mr_mip->w11=mirvar(0);
    mr_mip->w12=mirvar(0);
    mr_mip->w13=mirvar(0);
    mr_mip->w14=mirvar(0);
    mr_mip->w15=mirvar(0);
    mr_mip->depth--;
    return mr_mip;
}
//============================
flash mirvar(int iv)
{ /* initialize big/flash number */
    flash x;
    if (mr_mip->ERNUM) return NULL;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=23;
    if (mr_mip->TRACER) mr_track();
    if (!(mr_mip->active))
    {
        mr_berror(MR_ERR_NO_MIRSYS);
        mr_mip->depth--;
        return NULL;
    }
    x=(mr_small *)mr_alloc(mr_mip->nib+1,sizeof(mr_small));
    if (x==NULL)
    {
        mr_berror(MR_ERR_OUT_OF_MEMORY);
        mr_mip->depth--;
        return x;
    }
    convert(iv,x);
    mr_mip->depth--;
    return x;
```

```
}
//==============================
int cinnum(flash x,FILE *filep)
{ /* convert from string to flash x */
   int n;
   if (mr_mip->ERNUM) return 0;
   mr_mip->depth++;
   mr_mip->trace[mr_mip->depth]=14;
   if (mr_mip->TRACER) mr_track();
   mr_mip->infile=filep;
   mr_mip->fin=TRUE;
   n=cinstr(x,NULL);
   mr_mip->fin=FALSE;
   mr_mip->depth--;
   return n;
}
//==============================
void power(flash x,int n,flash w)
{
   copy(x,mr_mip->w8);
   zero(w);
   if (mr_mip->ERNUM || size(mr_mip->w8)==0) return;
   convert(1,w);
   if (n==0) return;
   mr_mip->depth++;
   mr_mip->trace[mr_mip->depth]=51;
   if (mr_mip->TRACER) mr_track();
   if (n<0)
   {
      n=(-n);
      frecip(mr_mip->w8,mr_mip->w8);
   }
```

```
    if (n==1)
    {
        copy(mr_mip->w8,w);
        mr_mip->depth--;
        return;
    }
    forever
    {
        if (n%2!=0) fmul(w,mr_mip->w8,w);
        n/=2;
        if (mr_mip->ERNUM || n==0) break;
        fmul(mr_mip->w8,mr_mip->w8,mr_mip->w8);
    }
    mr_mip->depth--;
}
//============================
void mad(big x,big y,big z,big w,big q,big r)
{
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=24;
    if (mr_mip->TRACER) mr_track();
    mr_mip->check=OFF;
    if (w==r)
    {
        mr_berror(MR_ERR_BAD_PARAMETERS);
        mr_mip->depth--;
        return;
    }
    multiply(x,y,mr_mip->w0);
    if (x!=z && y!=z)add(mr_mip->w0,z,mr_mip->w0);
```

```
   divide(mr_mip->w0,w,q);
   if (q!=r) copy(mr_mip->w0,r);
   mr_mip->check=ON;
   mr_mip->depth--;
}
//============================
```

- **Hàm Deciph.c**

Hàm sử dụng để thực hiện các thao tác giải mã hoá với xâu kí tự đã được mã hoá bằng hàm enciph.c ở trên, bằng cách đa vào một xâu ký tự đã mã hoá (bản mã) ở đầu ra bạn sẽ nhận lại một xâu ký tự ban đầu (bản rõ gốc). Hàm thực hiện có sử dụng khoá bí mật lấy vào từ File PRIVATE.KEY. Hai File PUBLIC.KEY và PRIVATE.KEY chúng cùng được sinh ra do chương trình genkey, chúng có quan hệ mật thiết với nhau và không thể tách rời, nếu có khoá công khai mà không có khoá bí mật thì cũng không thể giải mã được, còn nếu có khoá bí mật mà không có khoá công khai thì cũng chẳng ích lợi gì.

```
//============================
//Deciph.c
#include <stdio.h>
#include <miracl.h>
#include <stdlib.h>
#include <string.h>

int deciph(char *strinputde, char *stroutputde)
{
        /*  decipher using private key  */
   big x,y,ke,p,q,n,a,b,alpha,beta,t;
   FILE *ifile;
   int ch,i,leng;
```

```
    long ipt;
    miracl *mip=mirsys(100,0);
    x=mirvar(0);
    ke=mirvar(0);
    p=mirvar(0);
    q=mirvar(0);
    n=mirvar(0);
    y=mirvar(0);
    alpha=mirvar(0);
    beta=mirvar(0);
    a=mirvar(0);
    b=mirvar(0);
    t=mirvar(0);
    mip->IOBASE=60;
    if ((ifile=fopen("private.key","r"))==NULL)
    {
        return 1;
    }
    cinnum(p,ifile);
    cinnum(q,ifile);
    fclose(ifile);
    multiply(p,q,ke);
    leng=strlen(strinputde);
    cinstr(x,strinputde);
    xgcd(p,q,a,b,t);
    lgconv(leng,n);    /* first recover "one-time pad" */

#ifdef RSA
    decr(p,1,alpha);
    premult(alpha,2,alpha);
    incr(alpha,1,alpha);
    subdiv(alpha,3,alpha);
```

```
#else
   incr(p,1,alpha);
   subdiv(alpha,4,alpha);
#endif
   decr(p,1,y);
   powmod(alpha,n,y,alpha);
#ifdef RSA
   decr(q,1,beta);
   premult(beta,2,beta);
   incr(beta,1,beta);
   subdiv(beta,3,beta);
#else
   incr(q,1,beta);
   subdiv(beta,4,beta);
#endif
   decr(q,1,y);
   powmod(beta,n,y,beta);
   copy(x,y);
   divide(x,p,p);
   divide(y,q,q);
   powmod(x,alpha,p,x);
   powmod(y,beta,q,y);
   mad(x,q,q,ke,ke,t);
   mad(t,b,b,ke,ke,t);
   mad(y,p,p,ke,ke,x);
   mad(x,a,a,ke,ke,x);
   add(x,t,x);
   divide(x,ke,ke);
   if (size(x)<0) add(x,ke,x);

for (i=0;i<leng;i++)
   { /* decipher character by character */
```

```
        ch=*(strinputde+i);
        ch^=x[1];           /* XOR with last byte of x */
        stroutputde[i]=ch;
#ifdef RSA
        power(x,3,ke,x);
#else
        mad(x,x,x,ke,ke,x);
#endif
    }
  return 0;
}
//============================
void multiply(big x,big y,big z)
{  /*  multiply two big numbers: z=x.y  */
    int i,xl,yl,j,ti;
    mr_small carry,sz;
    big w0;
#ifdef MR_NOASM
    mr_large dble;
#endif
    if (mr_mip->ERNUM) return;
    if (y[0]==0 || x[0]==0)
    {
        zero(z);
        return;
    }
    w0=mr_mip->w0;    /* local pointer */
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=5;
    if (mr_mip->TRACER) mr_track();
#ifdef MR_FLASH
    if (mr_notint(x) || mr_notint(y))
```

```
    {
        mr_berror(MR_ERR_INT_OP);
        mr_mip->depth--;
        return;
    }
#endif
    sz=((x[0]&mr_mip->MSBIT)^(y[0]&mr_mip->MSBIT));
    xl=(int)(x[0]&mr_mip->OBITS);
    yl=(int)(y[0]&mr_mip->OBITS);
    zero(w0);
    if (mr_mip->check && xl+yl>mr_mip->nib)
    {
        mr_berror(MR_ERR_OVERFLOW);
        mr_mip->depth--;
        return;
    }

//============================
void mad(big x,big y,big z,big w,big q,big r)
{
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=24;
    if (mr_mip->TRACER) mr_track();
    mr_mip->check=OFF;
    if (w==r)
    {
        mr_berror(MR_ERR_BAD_PARAMETERS);
        mr_mip->depth--;
        return;
    }
    multiply(x,y,mr_mip->w0);
```

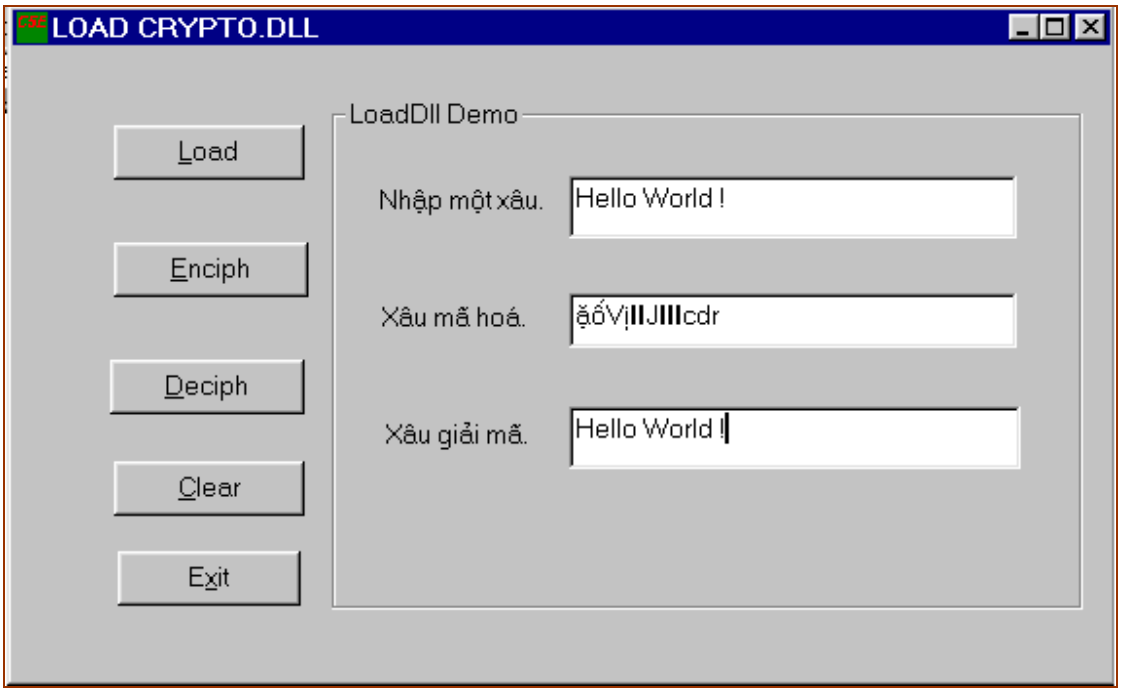
```
    if (x!=z && y!=z)add(mr_mip->w0,z,mr_mip->w0);

    divide(mr_mip->w0,w,q);
    if (q!=r) copy(mr_mip->w0,r);
    mr_mip->check=ON;
    mr_mip->depth--;
}
//============================
int cinstr(flash x,unsigned char *string)
{  /*  input big number in base IOBASE  */
    mr_small newb,oldb,b,lx;
    int ipt;
    if (mr_mip->ERNUM) return 0;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=78;
    if (mr_mip->TRACER) mr_track();
    newb=mr_mip->IOBASE;
    oldb=mr_mip->apbase;
    mr_setbase(newb); /* temporarily change base ... */
    b=mr_mip->base;
    mr_mip->check=OFF;
    ipt=instr(mr_mip->w5,string); /* ... and get number  */
    mr_mip->check=ON;
    lx=(mr_mip->w5[0]&mr_mip->OBITS);
#ifdef MR_FLASH
     if ((int)(lx&mr_mip->MSK)>mr_mip->nib || (int)((lx>>mr_mip->BTS)&mr_mip-
>MSK)>mr_mip->nib)
#else
    if ((int)lx>mr_mip->nib)
#endif
    { /* numerator or denominator too big */
       mr_berror(MR_ERR_OVERFLOW);
```

```
        mr_mip->depth--;
        return 0;
    }
    mr_setbase(oldb);      /* restore original base */
    cbase(mr_mip->w5,b,x);
    mr_mip->depth--;
    return ipt;
}
//============================
void incr(big x,int n,big z)
{  /* add int to big number: z=x+n */
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=7;
    if (mr_mip->TRACER) mr_track();
    convert(n,mr_mip->w0);
    select(x,PLUS,mr_mip->w0,z);
    mr_mip->depth--;
}
//============================
void decr(big x,int n,big z)
{  /* subtract int from big number: z=x-n */
    if (mr_mip->ERNUM) return;
    mr_mip->depth++;
    mr_mip->trace[mr_mip->depth]=8;
    if (mr_mip->TRACER) mr_track();
    convert(n,mr_mip->w0);
    select(x,MINUS,mr_mip->w0,z);
    mr_mip->depth--;
}
```

## 2.Chương trình Demo thư viện CRYPTO.DLL

Phần này xây dựng một ứng dụng đơn giản để Demo thư viện CRYPTO.DLL, chương trình xây dựng nhập vào một xâu rồi mã hoá, giải mã và trả lại kết quả ban đầu.



## *Tài liệu tham khảo :*

BRASSARD, **Modern Cryptology. Lecture Notes in Computer Science**, Vol. 325. Springer-Verlag 1988.

BRUCE SCHNEIER, **APPLIED CRYPTOGRAPHY, Protocol, Algorithms, and Source Code in C**, John Wiley & Sons 1994

COMBA, **Exponentiation Cryptosystems on the IBM PC. IBM**

Phạm Văn ất, **Kỹ thuật lập trình C, cơ sở và nâng cao** Nhà xuất bản giáo dục 1997.

Xuân Nguyệt và Phùng Kim Hoàng, **học Visual C++ 5 trong 21 ngày.** Nhà xuất bản Mũi cà mau 1998.