**MỤC LỤC**

[Chương 2: Mật mã đối xứng 1](#_Toc124178338)

[1. Bảng mã ASCII 1](#_Toc124178339)

[2. Các kỹ thuật mã hoá cổ điển 1](#_Toc124178340)

[1.1. Kỹ thuật thay thế 1](#_Toc124178341)

[a) Mật mã Caesar 1](#_Toc124178342)

[b) Mật mã Affine 2](#_Toc124178343)

[c) Mật mã Monoalphabetic 4](#_Toc124178344)

[d) Mật mã Playfair 5](#_Toc124178345)

[e) Mật mã Hill 8](#_Toc124178346)

[f) Mật mã Vigenere 9](#_Toc124178347)

[g) Mật mã Vernman 9](#_Toc124178348)

[1.2. Kỹ thuật chuyển dịch - hoán vị 11](#_Toc124178349)

[a) Kỹ thuật rain - fence 11](#_Toc124178350)

[b) Kỹ thuật hoán vị (hoán đổi hàng cột của ma trận plain text) 12](#_Toc124178351)

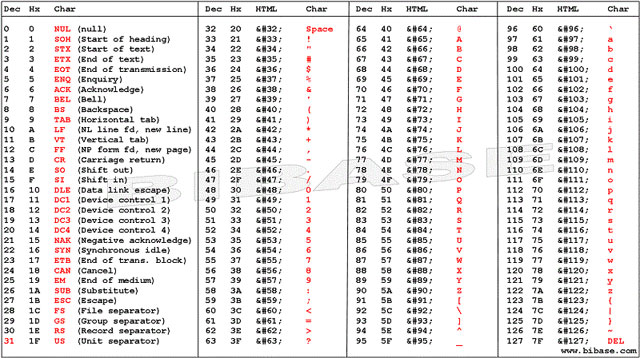
[3. Chuẩn mã hóa dữ liệu DES (Mật mã khối Feistel) 13](#_Toc124178352)

[Chương 3: Mật mã khóa công khai và ứng dụng 16](#_Toc124178353)

[1. Hàm Hash 16](#_Toc124178354)

# Chương 2: Mật mã đối xứng

## Bảng mã ASCII

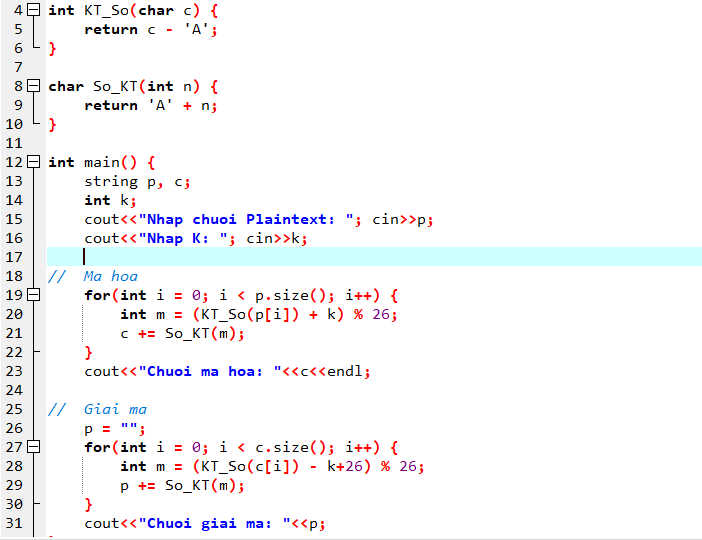


## Các kỹ thuật mã hoá cổ điển

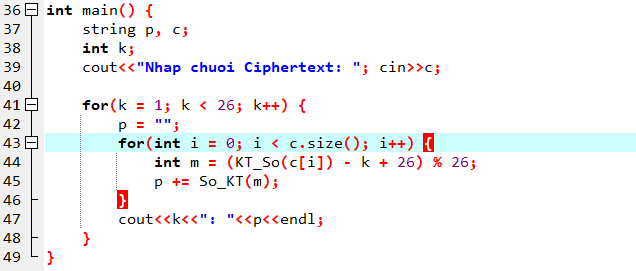
### Kỹ thuật thay thế

#### Mật mã Caesar

* ***Bài tập 1:*** *Lập trình nhập một chuỗi kí tự từ bàn phím, mã hoá chuỗi bằng thuật toán CAESAR tổng quát với khoá K nhập từ bàn phím. Hiện chuỗi mới ra màn hình. Lập trình giải mã để khôi phục lại chuỗi ban đầu.*

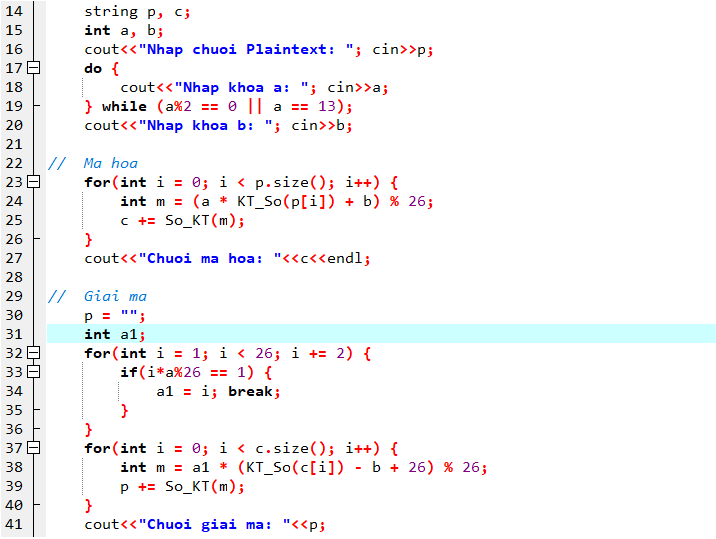


* ***Bài tập 2:*** *Lập trình bẻ khoá mật mã Caesar bằng phương pháp Brute-force. Đầu vào chương trình là chuỗi kí tự cipher text thu được từ Bài tập 1. Hãy xác định khoá K đã sử dụng và nội dung của plain text ban đầu.*

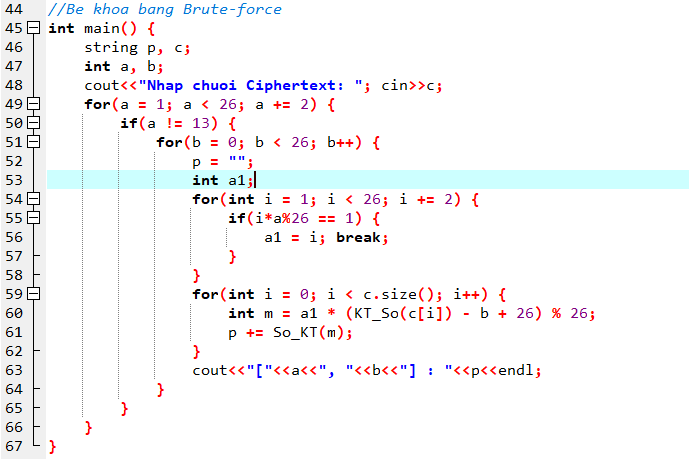


#### Mật mã Affine

* ***Bài tập 1:*** *Lập trình nhập một chuỗi kí tự từ bàn phím, mã hoá chuỗi bằng thuật toán Affine với cặp số {a,b} nhập từ bàn phím. Hiện chuỗi mới ra màn hình.*
* ***Bài tập 2:*** *Lập trình giải mã Affine để khôi phục lại chuỗi ban đầu: Đầu vào chương trình là cặp số {a,b} và chuỗi kí tự cipher text từ Bài tập 1. Đầu ra chương trình là chuỗi kí tự plain text.*

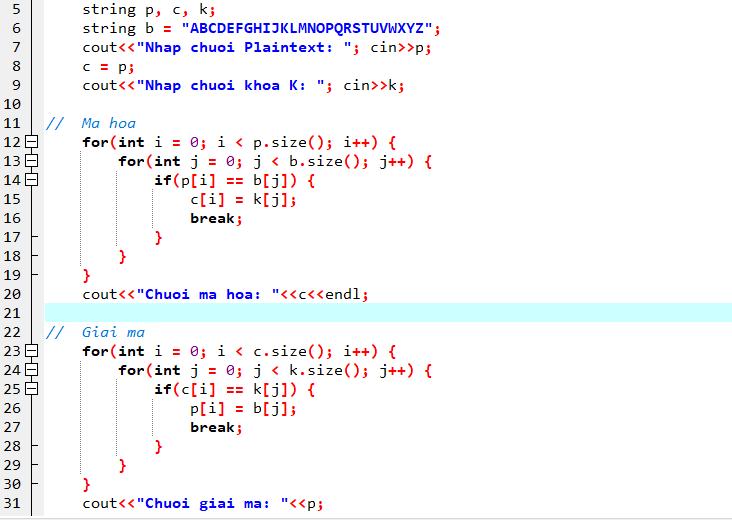
******

* ***Bài tập 3:*** *Lập trình bẻ khoá mật mã Affine bằng phương pháp Brute-force: Đầu vào chương trình là chuỗi kí tự cipher text thu được từ Bài tập 1. Hãy xác định cặp số {a,b} đã sử dụng và nội dung của plain text ban đầu.*

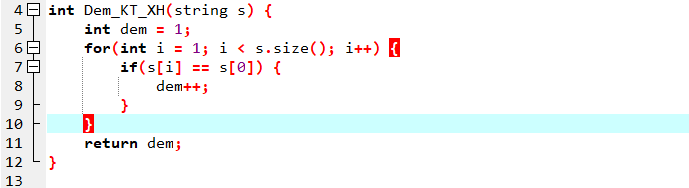
******

#### Mật mã Monoalphabetic

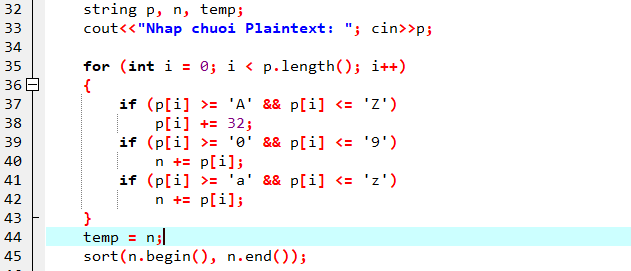
* ***Bài tập 1:*** *Lập trình nhập một chuỗi kí tự từ bàn phím, mã hoá chuỗi bằng thuật toán Monoalphabetic với khoá K nhập từ bàn phím (Khoá K là một chuỗi gồm 26 chữ cái có trật tự bất kì). Hiện chuỗi mới ra màn hình.*
* ***Bài tập 2:*** *Lập trình giải mã Monoalphabetic để khôi phục lại chuỗi ban đầu: Đầu vào chương trình là khoá K và chuỗi kí tự cipher text từ Bài tập 1. Đầu ra chương trình là chuỗi kí tự plain text.*

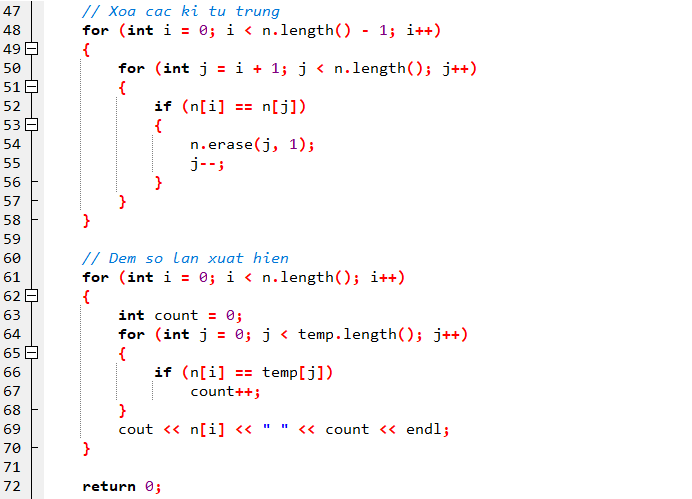
****

* ***Bài tập 1:*** *Lập trình nhập một chuỗi kí tự từ bàn phím. Đếm xem kí tự đầu tiên của chuỗi xuất hiện bao nhiêu lần trong chuỗi?*

**

* ***Bài tập 2:*** *Lập trình tính tần suất xuất hiện của các kí tự trong một đoạn văn bản cho trước.*



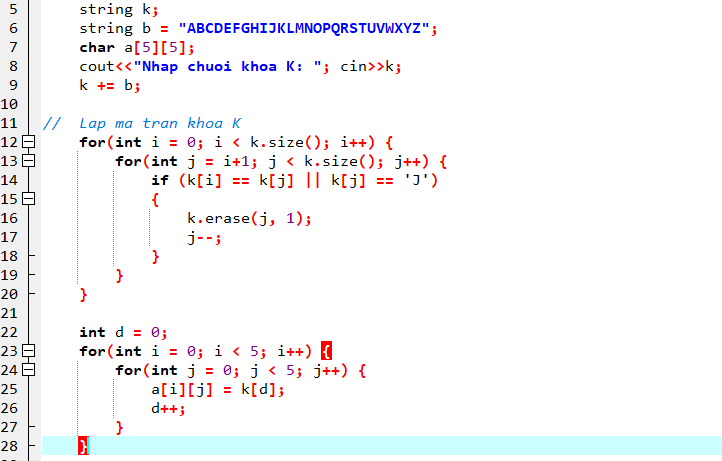


#### Mật mã Playfair

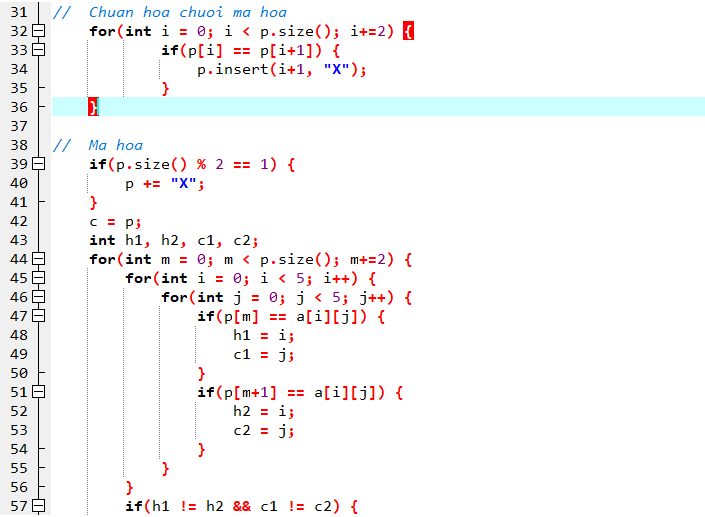
* **Bài tập (làm bằng tay):** *Chọn một từ khoá bất kì rồi xây dựng ma trận khoá Playfair. Chọn một plaintext bất kì, áp dụng ma trận khoá trên để tạo ra ciphertext. Thử giải mã ciphertext rồi so sánh với plaintext ban đầu.*
  + **Ma trận khóa:**

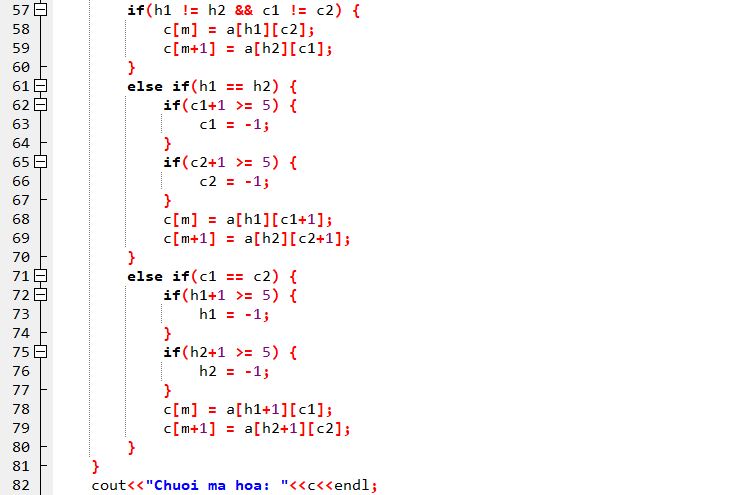
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | A | O | V | N |
| T | B | D | E | F |
| G | H | I/J | K | L |
| M | P | Q | R | S |
| U | W | X | Y | Z |

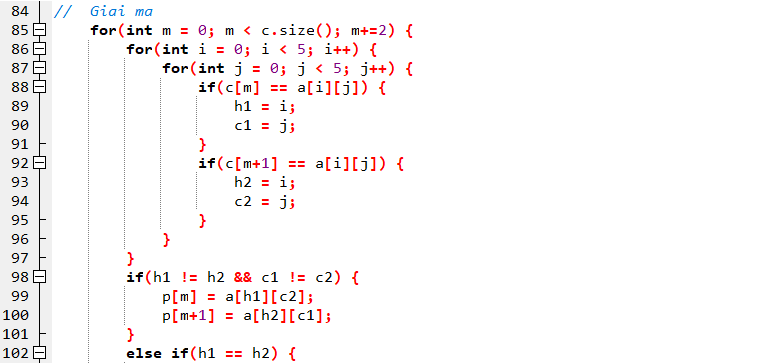
* + **Plaintext:** WE WI LX LM EX ET YO UW
  + **Ciphertext:** YB XH IZ GS DY FB XV WX
  + **Plaintext:** WE WI LX LM EX ET YO UW
* ***Bài tập 1:*** *Nhập một từ khóa bất kì. Hãy loại bỏ các kí tự trùng lặp trong chuỗi. Điền thêm vào chuỗi trên những kí tự còn lại trong bảng chữ cái. Đặt chuỗi thu được vào ma trận khóa 5x5.*

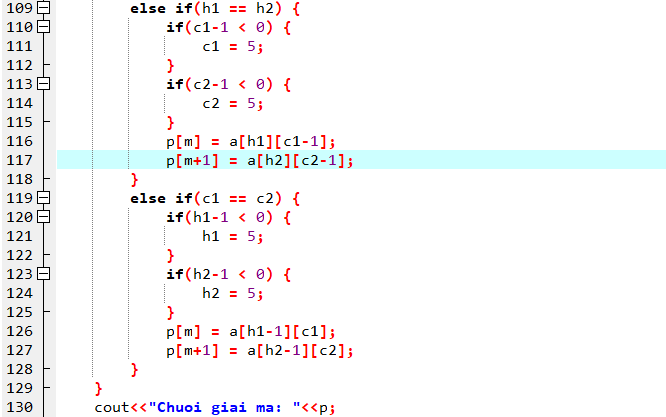
******

* ***Bài tập 2:*** *Lập trình mã hoá và giải mã thông điệp theo thuật toán Playfair.*

**

**

**

**

#### Mật mã Hill

* **Bài tập *(làm bằng tay):*** *Với plaintext là ”paymoremoney” và sử dụng với mật khóa K nói trên, hãy xác định ciphertext. Giải mã ciphertext thu được bằng cách nhân nó với K-1, rồi so sánh kết quả với plaintext ban đầu.*

C1 = (17\*15 + 17\*0 + 5\*24) % 26 = 11 = L

C2 = (21\*15 + 18\*0 + 21\*24) % 26 = 13 = N

C3 = (2\*15 + 2\*0 + 19\*24) % 26 = 18 = S

C4 = (17\*12 + 17\*14 + 5\*17) % 26 = 7 = H

C5 = (21\*12 + 18\*14 + 21\*17) % 26 = 3 = D

C6 = (2\*12 + 2\*14 + 19\*17) % 26 = 11 = L

C7 = (17\*4 + 17\*12 + 5\*14) % 26 = 4 = E

C8 = (21\*4 + 18\*12 + 21\*14) % 26 = 22 = W

C9 = (2\*4 + 2\*12 + 19\*14) % 26 = 12 = M

C10 = (17\*13 + 17\*4 + 5\*24) % 26 = 19 = T

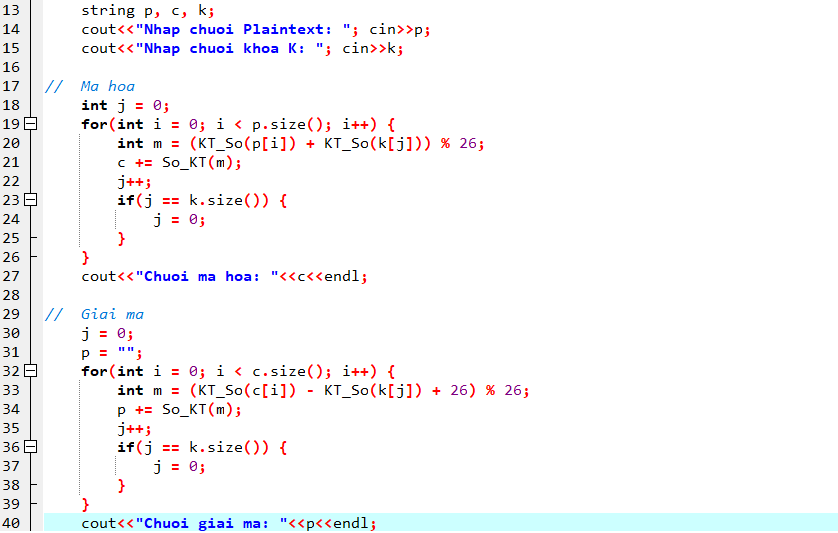
C11 = (21\*13 + 18\*4 + 21\*24) % 26 = 17 = R

C12 = (2\*13 + 2\*4 + 19\*24) % 26 = 22 = W

* Chuỗi Ciphertext: LNSHDLEWMTRW

#### Mật mã Vigenere

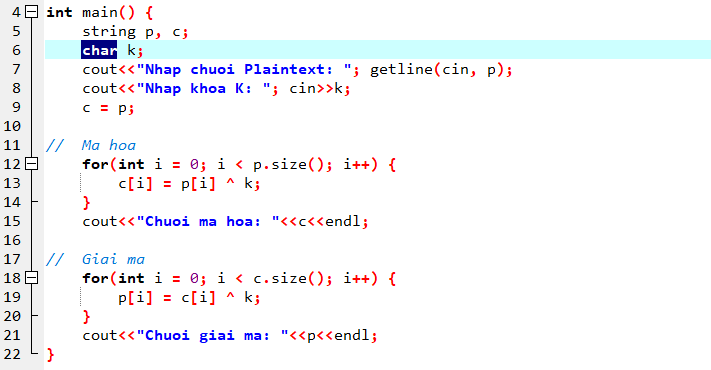
* **Bài tập:** *Lập trình nhập chuỗi kí tự plaintext từ bàn phím, mã hoá chuỗi bằng thuật toán* Vigenère *với khoá K (là một chuỗi kí tự) nhập từ bàn phím. Hiện chuỗi mới ra màn hình. Lập trình giải mã để khôi phục lại chuỗi ban đầu.*

****

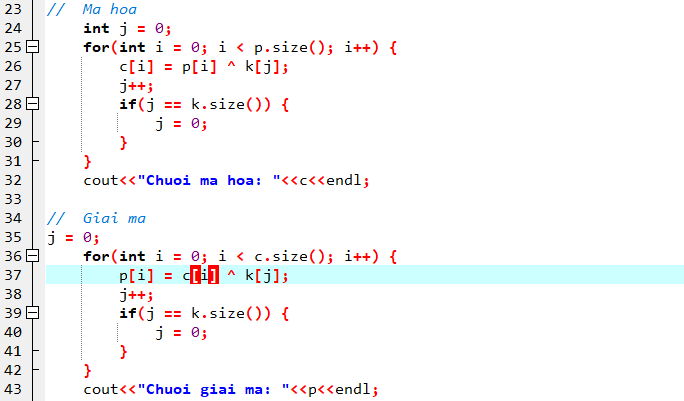
#### Mật mã Vernman

* **Bài tập 1:** Nhập một chuỗi plaintext từ bàn phím. Nhập một khóa *K* (dài 8 bít). Mã hóa chuỗi ban đầu bằng cách XOR các kí tự của nó với *K*. Giải mã ciphertext thu được, rồi so sánh với chuỗi ban đầu.

**Dạng 1:**

****

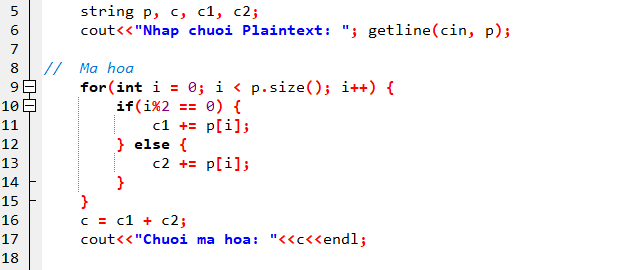
**Dạng 2:**

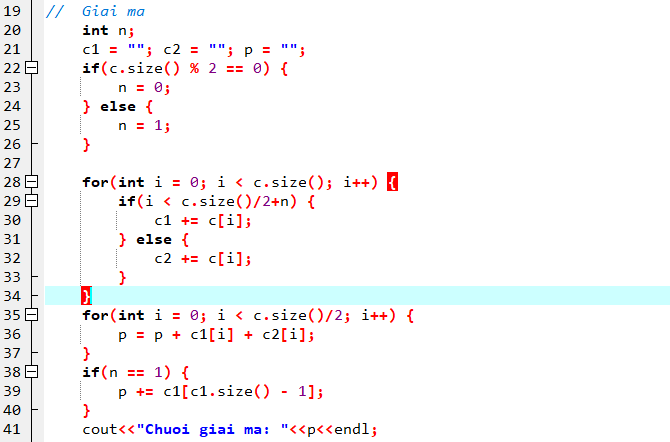
****

### Kỹ thuật chuyển dịch - hoán vị

#### Kỹ thuật rain - fence

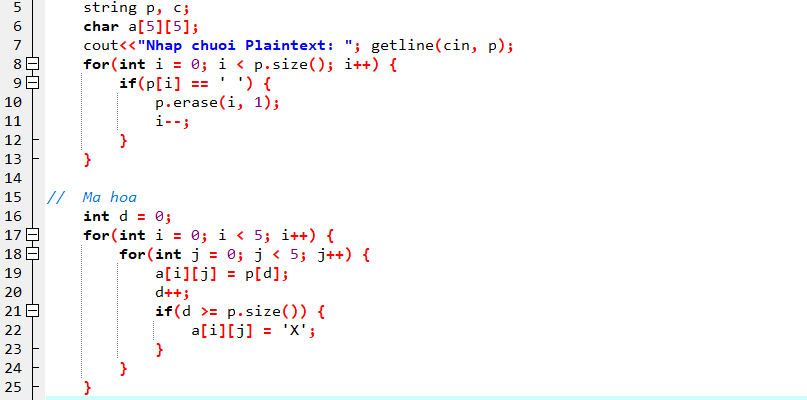
* **Bài tập:** Nhập một chuỗi plaintext. Mã hóa chuỗi bằng kỹ thuật rain-fence, hiện ciphertext ra màn hình. Giải mã ciphertext, so sánh kết quả với chuỗi ban đầu.

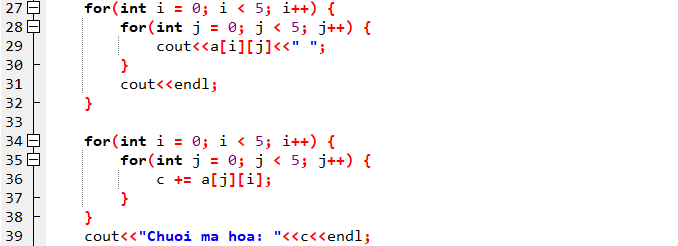
****

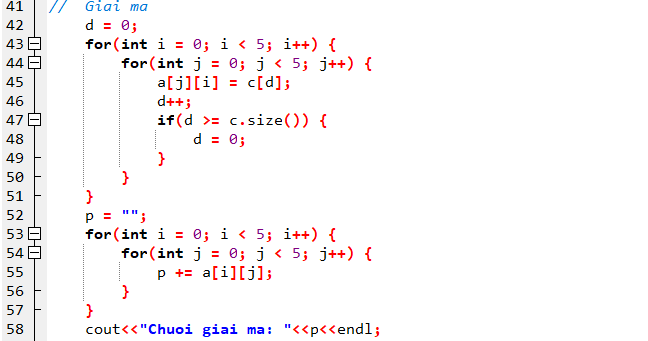
****

#### Kỹ thuật hoán vị (hoán đổi hàng cột của ma trận plain text)

* **Bài tập:** Nhập một chuỗi plaintext (dài không quá 25 kí tự). Sắp xếp các kí tự của chuỗi vào một ma trận 5 x 5 (lần lượt theo từng hàng). Đọc các phần tử của ma trận theo từng cột để tạo thành ciphertext. Giải mã ciphertext, so sánh kết quả với chuỗi ban đầu.

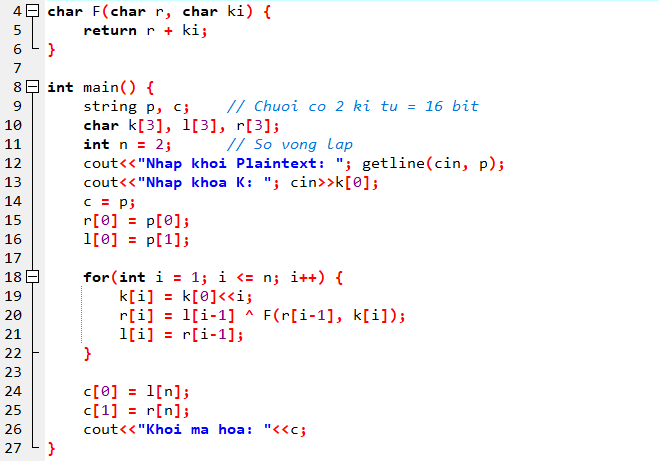
****

****

****

## Chuẩn mã hóa dữ liệu DES (Mật mã khối Feistel)

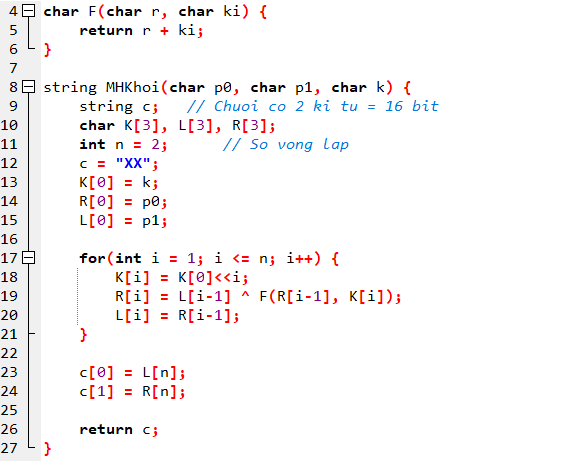
* **Mật mã khối Feistel:**
* **Bài tập 1:** Lập trình mô phỏng hoạt động của mật mã Feistel đơn giản với 2 vòng xử lý:
  + Nhập một khối plaintext từ bàn phím, khối có độ dài m = 2w = 16 bít.
  + Nhập khóa K (dài 8 bít) từ bàn phím. Khóa Ki được sinh ra từ khóa K nhờ phép dịch trái K i lần.
  + Hàm F thực hiện phép cộng giữa Ri-1 với Ki.
  + Hiện khối ciphertext ra màn hình.

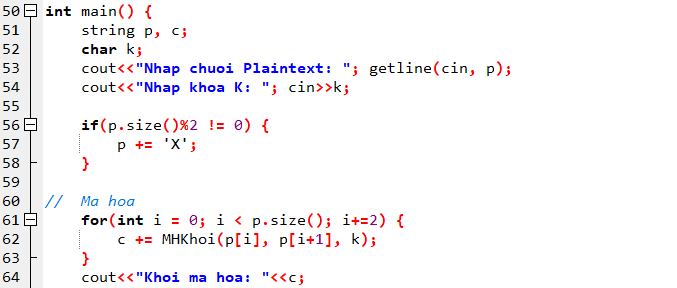
****

* **Bài tập 2: Lập trình mô phỏng hoạt động của mật mã Feistel đơn giản với 2 vòng xử lý:**
  + Nhập chuỗi plaintext từ bàn phím, chia chuỗi thành các khối dài *m* = 2*w* = *16* bít.
  + Nhập khóa *K* (dài *8* bít) từ bàn phím. Khóa *Ki* được sinh ra từ khóa *K* nhờ phép dịch trái *K* i lần.
  + Hàm *F* thực hiện phép cộng giữa *Ri-1* với *Ki*.
  + Hiện chuỗi ciphertext ra màn hình.
  + Giải mã chuỗi ciphertext thu được*(xem hướng dẫn sau).*
    - Công thức tổng quát của mỗi vòng:

**Li-1 = Ri** ⊕ **F(Li,Ki)**

**Ri-1 = Li**

****

****

# Chương 3: Mật mã khóa công khai và ứng dụng

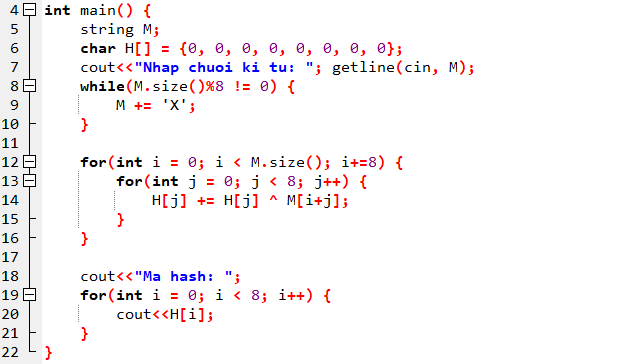
## Hàm Hash

* + ***Ví dụ 1:*** Giả sử thông điệp đầu vào được chia thành *N* khối, mối khối dài *n* bít. Mã hash dài *n* bít được tính bằng cách XOR các bít tương ứng của tất cả các khối:

*Hi* = *bi1* ⊕ *bi2* ⊕ ... ⊕ *biN*

*Trong đó:*

* + *Hi* là bit thứ *i* của mã hash*, 1 ≤ i ≤ n.*
  + *bij* là bit thứ *i* của khối thứ *j*.
* **Bài tập 1:** Nhập một chuỗi kí tự từ bàn phím, chia chuỗi thành từng khối *8* bít (ứng với 1 kí tự) rồi tính mã hash của chuỗi theo phương pháp đã nêu ở Ví dụ 1.
* **Bài tập 2:** Nhập một chuỗi kí tự từ bàn phím, chia chuỗi thành từng khối *64* bít (ứng với *8* kí tự) rồi tính mã hash của chuỗi theo phương pháp đã nêu ở Ví dụ 1 (Nếu độ dài chuỗi không phải là bội số của *8* thì có thể chèn thêm các kí tự quy ước).

****

* ***Ví dụ 2:*** Giả sử thông điệp đầu vào được chia thành N khối, mỗi khối dài n bít. Để cải thiện tính ngẫu nhiên của đầu vào so với Ví dụ 1, ta sẽ thực hiện phép quay phải trước khi tiến hành XOR:
  + Giá trị hash ban đầu được đặt bằng 0
  + Thực hiện N vòng lặp (i từ 0 → N-1), mỗi vòng làm các công việc sau:
    - Quay phải một bít giá trị hash hiện tại (Dịch phải và kiểm tra điều kiện: nếu bit chẵn thì giữ nguyên, bit lẻ thì + 128bit ).
    - XOR giá trị hash với khối dữ liệu thứ i
* **Bài tập:** Nhập một chuỗi kí tự từ bàn phím, chia chuỗi thành từng khối *8* bít (ứng với 1 kí tự) rồi tính mã hash của chuỗi theo phương pháp đã nêu ở Ví dụ 2.

**// Code mật mã Vigenere**

**string p,k;**

**cout<<"P: ";**

**getline(cin,p);**

**cout<<"K: ";**

**getline(cin,k);**

**string c ="";**

**int j =0;**

**for(int i=0;i<p.size();i++)**

**{**

**if(i%k.length()==0)**

**{**

**j = 0;**

**c+= So\_kt((p[i]+k[j])%26);**

**}**

**else{**

**c+= So\_kt((p[i]+k[j])%26);**

**}**

**j++;**

**}**

**cout<<c;**

**// Code rain\_fence ( đọc các phần tử của ma trận theo từng cột để tạo thành ciphertext)**

**string p ;**

**getline(cin,p);**

**char arr[5][5];**

**int x = 0;**

**for (int i = 0; i < 5; i++)**

**{**

**for (int j = 0; j < 5; j++)**

**{**

**arr[i][j]=p[x];**

**x++;**

**if(p.length()<x)**

**arr[i][j]='X';**

**}**

**}**

**for (int i = 0; i < 5; i++)**

**{**

**for (int j = 0; j < 5; j++)**

**{**

**cout<<arr[i][j]<<" ";**

**}**

**cout<<endl;**

**}**

**string c;**

**for (int i = 0; i < 5; i++)**

**{**

**for (int j = 0; j < 5; j++)**

**{**

**c+=arr[j][i];**

**}**

**}**

**// code mật mã affine**

**string C = '';**

**for(int i =0;i<p.Size();i++)**

**{**

**int p = Kt\_so(P[i]);**

**int c = (a\*p+b)%26;**

**C= C+So\_kt(C);**

**}**

**//giai ma**

**int a1;P ="";**

**for(int i=1;i<26;i=i+2)**

**if(i\*a%26==1)**

**{ a1=i;break;}**

**for(int i=0;i<C.size();i++)**

**{**

**int c=Kt\_so(C[i]);**

**int p = a1\*(c-b+26)%26;**

**P=P+So\_kt(p);**

**}**

**// Monoalphabetic**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**string P, C, K, B;**

**B = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";**

**cout<<"Nhap chuoi plaintext: "; getline(cin, P); C= P;**

**cout<<"Nhap chuoi khoa K: "; getline(cin, K);**

**// Ma hoa**

**for(int i = 0; i<P.size(); i++)**

**for (int j = 0; j<B.size(); j++)**

**if(P[i] == B[j])**

**{**

**C[i] = K[j];**

**break;**

**}**

**cout<<"Chuoi ma hoa: "<<C<<endl;;**

**// Giai ma**

**for(int i = 0; i<C.size(); i++)**

**for (int j = 0; j<B.size(); j++)**

**if(C[i] == K[j])**

**{**

**P[i] = B[j];**

**break;**

**}**

**cout<<"Chuoi giai ma: "<<P;**

**}**

**// ma khoi feistel**

**#include<iostream>**

**using namespace std;**

**const int MAX = 100;**

**string P,C;**

**char K[MAX];**

**char L[MAX],R[MAX];**

**char F(char r,char Ki)**

**{**

**return r+Ki;**

**}**

**int main()**

**{**

**int n =2;**

**cout<<"P: ";**

**getline(cin,P);**

**C = P;**

**R[0] = P[0];**

**L[0] = P[1];**

**cout<<"khoa K: ";cin>>K[0];**

**for (int i = 1; i <= n; i++)**

**{**

**K[i] = K[0]<<i;**

**R[i] = L[i-1] ^ F(R[i-1],K[i]);**

**L[i] = R[i-1];**

**}**

**C[0] = L[n];**

**C[1] = R[n];**

**cout<<"C: "<<C;**

**return 0;**

**}**

**// thuat toan vermman**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**string P,C;**

**char K;**

**cout<<"P: ";**

**getline(cin,P);**

**C = P;**

**cout<<"Nhap chuoi K: ";**

**cin>>K;**

**for (int i = 0; i < P.size(); i++)**

**{**

**C[i] = P[i]^K;**

**}**

**cout<<"Chuoi ma hoa: "<<C;**

**for (int i = 0; i < C.size(); i++)**

**{**

**P[i] = C[i]^K;**

**}**

**cout<<"Giai ma: "<<P;**

**}**

**// thuat toan rain fence**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**string P,C,C1,C2,K;**

**cout<<"P: ";**

**getline(cin,P);**

**for (int i = 0; i < P.size(); i++)**

**{**

**if(i%2==0) C1+=P[i];**

**else C2+= P[i];**

**}**

**C = C1+C2;**

**cout<<"Chuoi ma hoa C: "<<C<<endl;**

**int vitricat;**

**if(C.size()%2==0) vitricat = C.size()/2;**

**else vitricat = C.size()/2+1;**

**int x1 = 0,x2=1;**

**for (int i = 0; i < C.size(); i++)**

**{**

**if(i<vitricat)**

**{**

**P[x1] = C[i];**

**x1+=2;**

**}**

**else**

**{**

**P[x2] = C[i];**

**x2+=2;**

**}**

**}**

**cout<<"Chuoi giai ma: "<<P;**

**}**

**// thuat toan ???**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**string a;**

**cin>>a;**

**string B = "ABCDEFGHIKLMNOPQRSTUVWXYZ";**

**a = a+B;**

**for (int i = 0; i < a.length(); i++)**

**{**

**for (int j = i+1; j < a.length(); j++)**

**{**

**if(a[j] == a[i]){**

**a.erase(j,1);**

**j--;**

**}**

**}**

**}**

**char matran[5][5];**

**int dem = 0;**

**for (int i = 0; i < 5; i++)**

**{**

**for (int j = 0; j < 5; j++)**

**{**

**matran[i][j] = a[dem++];**

**}**

**}**

**cout<<"Ma tran khoa: \n";**

**for (int i = 0; i < 5; i++)**

**{**

**for (int j = 0; j < 5; j++)**

**{**

**cout<<matran[i][j]<<" ";**

**cout<<endl;**

**}**

**}**

**}**

**// thuật toán CAESAR tong quat**

**#include<iostream>**

**using namespace std;**

**int KT\_So(char c)**

**{**

**return c-'a';**

**}**

**char So\_Kt(int n)**

**{**

**return 'a'+n;**

**}**

**string Giai\_ma(string C,int i)**

**{**

**string B;**

**for(int j = 0;j<C.size();j++)**

**{**

**int m = (KT\_So(C[j])-i+26)%26;**

**B+= So\_Kt(m);**

**}**

**return B;**

**}**

**string Ma\_Hoa(string C,int i )**

**{**

**string B;**

**for (int j = 0; j < C.size(); j++)**

**{**

**int m = (KT\_So(C[j])+i)%26;**

**B+=So\_Kt(m);**

**}**

**return B;**

**}**

**int main()**

**{**

**string P,C;int K;**

**cout<<"P: ";cin>>P;**

**cout<<"Khoa K: ";cin>>K;**

**for (int i = 0; i < P.size(); i++)**

**{**

**int m = (KT\_So(P[i])+K%26)%26;**

**C+=So\_Kt(m);**

**}**

**cout<<"Chuoi ma hoa: "<<C;**

**P = "";**

**for (int i = 0; i < C.size(); i++)**

**{**

**int m = (KT\_So(C[i])-K%26 + 26)%26;**

**P+=So\_Kt(m);**

**}**

**cout<<"Chuoi giai ma: "<<P;**

**// B2 be khoa bang phuong phap brute-force**

**// string x = "ABCZYXQLA";**

**for (int i = 0; i < 26; i++)**

**{**

**string Giaima = Giai\_ma(C,i);**

**string MaHoa = Ma\_Hoa(Giaima,i);**

**int khoa = i;**

**cout<<"\n Khoa: "<<khoa<<"--->"<<Giai\_ma(C,khoa);**

**}**

**}**

**// thuat toan Vigennere**

**#include<iostream>**

**using namespace std;**

**int KT\_So(char c)**

**{**

**return c-'a';**

**}**

**char So\_Kt(int n)**

**{**

**return 'a'+n;**

**}**

**string P,K;**

**void check\_k()**

**{**

**if(P.size()<=K.size())**

**K.erase(P.size(),K.size());**

**while(P.size()>K.size())**

**{**

**K+=K;**

**K.erase(P.size(),K.size());**

**}**

**}**

**int main()**

**{**

**string C;**

**cout<<"P :";**

**cin>>P;**

**cout<<"K= ";**

**cin>>K;**

**check\_k();**

**cout<<K;**

**for (int i = 0; i < P.size(); i++)**

**{**

**int m = (KT\_So(P[i])+KT\_So(K[i])+26)%26;**

**C+= So\_Kt(m);**

**}**

**cout<<"Chuoi ma hoa: "<<C;**

**P = "";**

**for (int i = 0; i < C.size(); i++)**

**{**

**int m = (KT\_So(P[i])-KT\_So(K[i])+26)%26;**

**P+= So\_Kt(m);**

**}**

**cout<<"Chuoi giai ma: "<<P;**

**}`**

**// thuat toan ???**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**string P,C,K;**

**cout<<"P: ";getline(cin,P);**

**cout<<"K: ";getline(cin,P);**

**for(int i = 0;i<P.size();i++)**

**{**

**int j;**

**j = i%(K.size());**

**cout<<j<<endl;**

**C[i] = P[i]^K[j];**

**}**

**cout<<"chuoi ma hoa: <<C;**

**for(int i = 0;i<C.size();i++)**

**{**

**int j;**

**j = i%(K.size());**

**cout<<j<<endl;**

**P[i] = C[i]^K[j];**

**}**

**cout<<"Chuoi giai ma: "<<P;**

**}**

**//lập trình nhập một chuỗi kí tự P từ bàn phím.Nhập 2 số nguyên a,b thuộc Z26. Mã hóa chuỗi ban đầu theo công thức C= P/a+b**

**// trong do P la bản rõ,C là kí tự mã hóa, các phép tính được thực hiện trong Z26.Hiện chuỗi sau khi mã hóa ra màn hình.Giải mã và hiện chuỗi.**

**#include<iostream>**

**using namespace std;**

**int KT\_So(char c)**

**{**

**return c-'a';**

**}**

**char So\_Kt(int n)**

**{**

**return 'a'+n;**

**}**

**string P,K;**

**int main()**

**{**

**string P,C="";**

**int a,b;**

**cout<<"P: ";**

**getline(cin,P);**

**cout<<"a: ";**

**cin>>a;**

**cout<<"b: ";**

**for (int i = 0; i < P.size(); i++)**

**{**

**int m = ((KT\_So(P[i])/a)+b)%26;**

**C+=So\_Kt(m);**

**}**

**cout<<"Chuoi ma hoa "<<C;**

**P = "";**

**for (int i = 0; i < C.size(); i++)**

**{**

**int m = ((KT\_So(C[i])\*a)-b)%26;**

**P+= So\_Kt(m);**

**}**

**cout<<"Chuoi giai ma "<<P;**

**}**

**KHÓA**

**\* Khóa công khai:**

- Bảo mật: Mã hóa để giữ bí mật nội dung của thông điệp

- Xác thực: Mã hóa để khẳng định nguồn gốc của thông điệp, chủ sở hữu

của thông điệp không phải do kẻ khác giả mạo

- Mỗi người dùng đều có 2 khóa:

+ Khóa K1: Khóa công khai

+ Khóa K2: Khóa riêng

- Bảo mật thông điệp: Mã hóa thông điệp bằng Khóa công khai của người

nhận. Mà chỉ có người nhận có thể dùng khóa riêng của mình để giải mã

được.

- Xác thực: Người gửi mã hóa = khóa riêng của mình, người nhận dùng

khóa công khai để xác thực.

- Khóa riêng ko có tác dụng bảo mật nhưng nó có khả năng khẳng định người

gửi đó không phải kẻ khác giả mạo.

**\* Thuật toán mã hóa RSA**: Mã hóa cho từng khối 1024 bit

Mật mã đối xứng để mã hóa sẽ đảm bảo tốc độ, khóa được bảo vệ

bằng mật mã bất đối xứng sẽ an toàn, không lo lộ khóa.

**\* Các hàm hash và MAC:**

- Hàm hash: Dù đầu vào ntn thì khi tác động đầu ra bởi hàm hash

luôn thu được đầu ra dãy bit có kích thước cố định, = nhau trong tất cả

các TH. Dùng để chứng thực thông điệp về mặt nội dung và nguồn gốc.

- 2 mã hash của 2 thông điệp M là khác nhau thì Độ dài = nhau

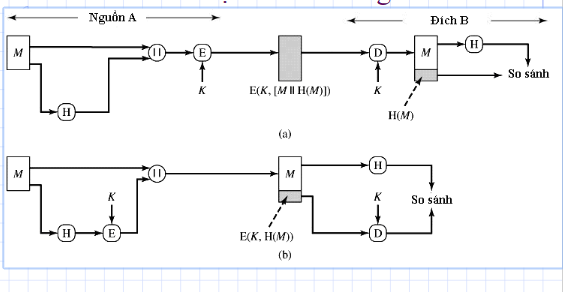
nhưng nội dung khác nhau.

- Không tìm đc thông điệp M qua H(M). Dữ liệu qua Hash đã bị mất mát rồi.

- M||H(M): So sánh mã 2 mã hash để xem nội dung có bị thay đổi trên đường truyền hay không. 2 hàm hash mà khác nhau (mã hash tính được ở bên nhận và mã hash đính kèm) có nghĩa là thông điệp M nhận được, không giống với thông điệp M đã gửi. Lỗi có thể do thời tiết, hacker.

+ Khả năng phát hiện thông điệp M có bị thay đổi không, nhưng chỉ phát hiện sự thay đổi này do khách quan, thời tiết, nhiễu đường truyền thôi, không có khả năng chống hacker. Hacker sửa thông điệp M và lấy mã hash của thông điệp M mới ghép vào.

\* **Hash và mật mã đối xứng:**



Hình A: E(K, [M || H(M)])

Hình B: M || E(K, H(M))

Hình B thông điệp M ko đc mã hóa, chỉ mã hóa mã hash thôi; hacker chặn trên đường truyền vẫn có thể đọc đc nội dung thông điệp M, nhưng không sửa được thông điệp M, vì nếu sửa M sẽ bị phát hiện ngay, do mã hash không sửa được, mã hash đã được mã hóa. Chỉ sửa đc M chứ không sửa đc mã hash nên bị phát hiện.

Hình A: vừa có tác dụng bảo mật thông điệp, vừa có tác dụng xác thực nội dung thông điệp

Hình B: Chỉ có tác dụng xác thực nội dung thông điệp thôi, không có tác dụng bảo mật thông điệp.

Tùy vào từng vào mục đích, ứng dụng mà lựa chọn sơ đồ,

Hình A vừa có tác dụng bảo mật, vừa có tác dụng xác thực nội dung thì có thể dùng làm các dịch vụ cần giữ bí mật: email cá nhân,…

Không cần bảo mật nội dung như báo chí, bức thư ngỏ,… dùng hình B

- Hình A:

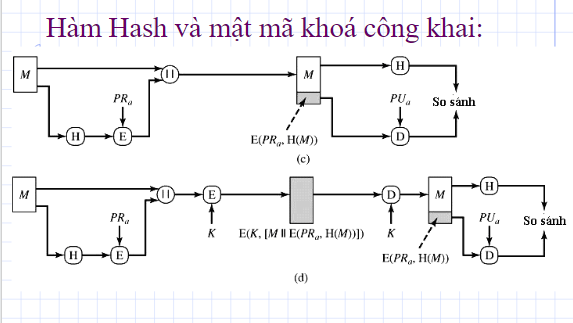
+ Toàn bộ thông điệp phải mã hóa sẽ tốn thời gian mã hóa và chậm tốc độ truyền tin.

- Hình B:

+ Không cần bảo mật nội dung, không mã hóa thông điệp M, không tốn thười gian, không mã tốc độ truyền tin nhanh hơn.

Khi cần bảo mật dùng hình A, không cần bảo mật dùng hình B.

\* **Hàm hash và mật mã khóa công khai:**



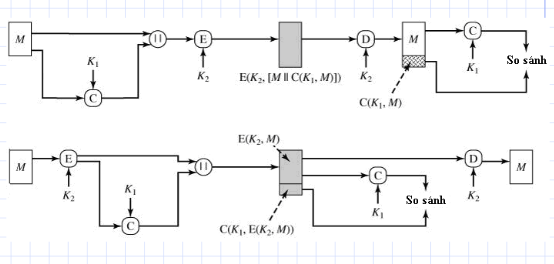
- Hình c và hình b: c cũng ko có tác dụng bảo mật ndung, có khả năng xác thực nội dung thông điệp M. c dùng mật mã bất đxứng: khóa riêng của a xác thực nguồn gốc thông điệp. b xác thực nội dung, không có khả năng xác thực nguồn gốc.

- Khi dùng mật mã bất đối xứng để bảo mật thông điệp, mã hóa = khóa công khai của người nhận, còn để xác thực nguồn gốc thông điệp, mã hóa = khóa riêng của người gửi.

- Hình d: sự phối hợp mật mã đối xứng, bất đối xứng, vừa bảo mật, vừa xác thực nội dung, nguồn gốc thông điệp.

**\* MAC = C(K, M):**

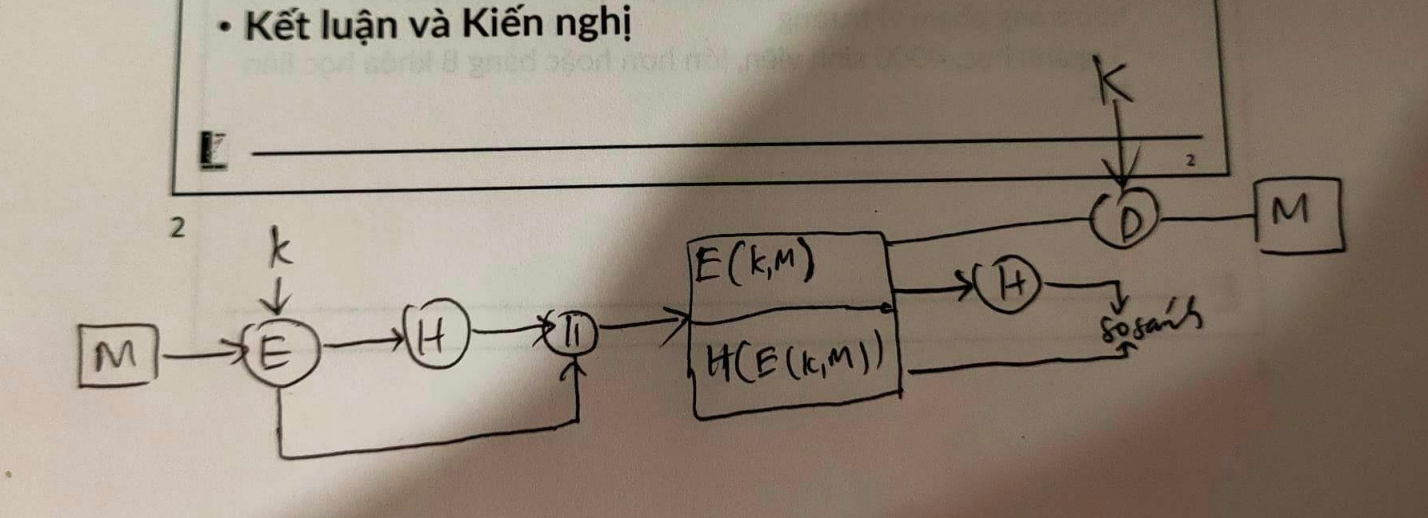
MAC và hash cơ bản giống nhau, nhưng MAC có thêm khóa K. Nên có thể sử dụng độc lập mà không cần mật mã với mục đích xác thực ndung thông điệp.



Hình 1: xác thực nội dung thông điệp, bảo mật thông điệp

Hình 2: xác thực thêm nguồn gốc thông điệp

Câu 1



- **Bên gửi**: A gửi B đoạn mã gồm 2 phần ghép với nhau:

Phần 1: Mã hóa thông điệp M.

Phần 2: Thông điệp M được đưa vào mã hash, Khóa mã hash.

**- Bên nhận:**

+ B nhận được đoạn mã từ A: E(K, M) và H(E(K,M))

Đầu tiên giải mã E(K, M) bởi D(k) ta được thông điệp M.

Sau đó, đưa thông điệp M vào hàm hash thu được H(M). Cuối cùng so sánh mã hash H(E(K, M)) vừa thu được với mã hash được đính kèm ban đầu xem giống nhau hay không. Để chứng thực được nội dung thông điệp đã nhận.

- Ở hình này, bảo mật được nội dung thông điệp, bảo mật mã hash, chứng thực được nội dung thông điệp. Hacker không thể biết được nội dung thông điệp hãy mã hash của thông điệp nên không thể làm giả hay sửa đổi thông điệp được.

**Câu 2**

Các hàm toán học được sử dụng phổ biến nhất trong mật mã khóa công khai là các hàm dựa trên các thuộc tính của các số nguyên tố lớn, chẳng hạn như thuật toán RSA và Mật mã đường cong Elliptic (ECC).

Một đặc điểm cơ bản của các hàm này là chúng dễ dàng tính toán để thực hiện theo một hướng (ví dụ: mã hóa), nhưng cực kỳ khó đảo ngược (ví dụ: giải mã) nếu không biết một số thông tin nhất định, chẳng hạn như khóa riêng.

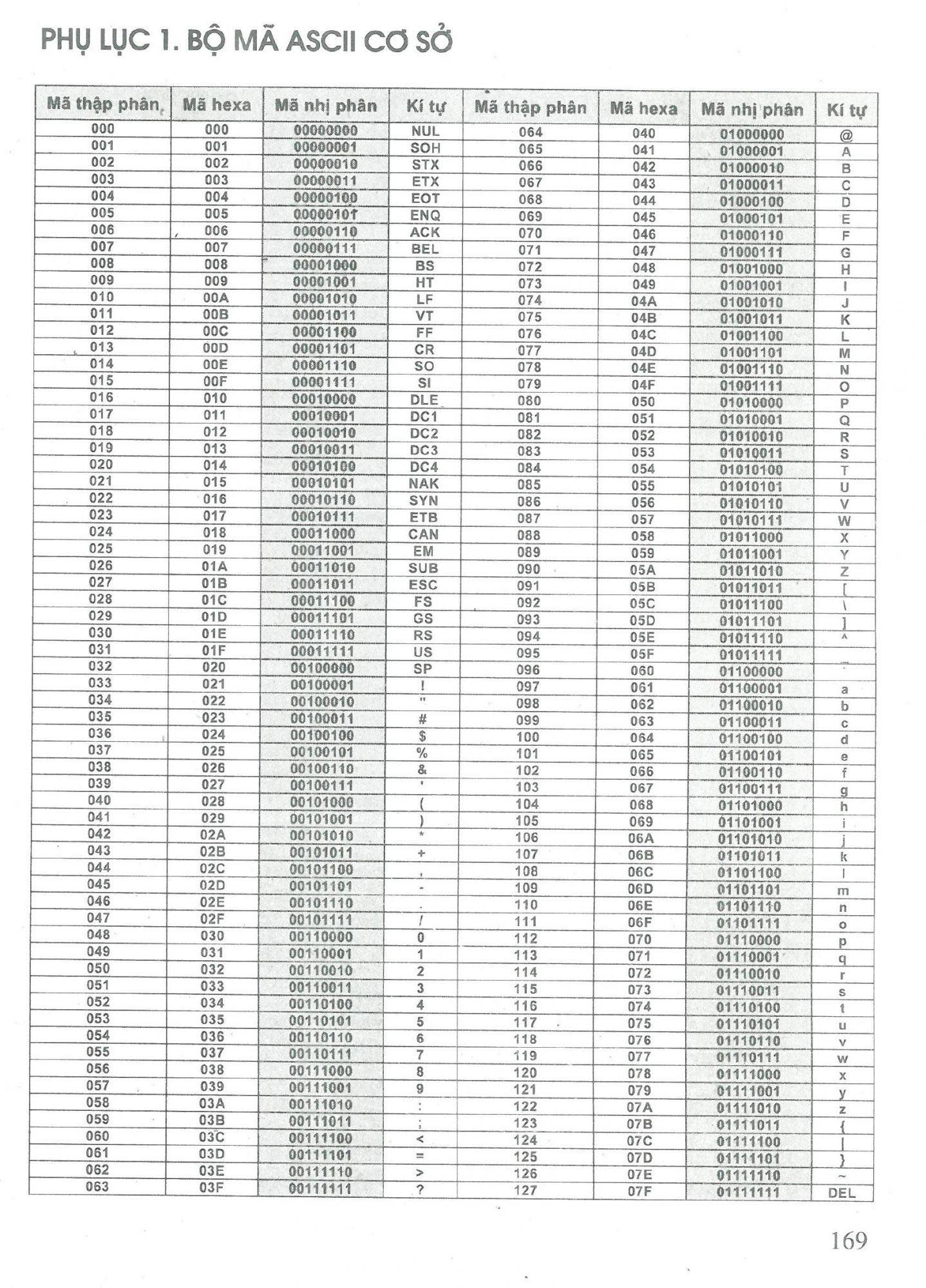
RSA là một ví dụ về chức năng như vậy. Nó dựa trên các tính chất toán học của các số nguyên tố lớn. Để tạo một cặp khóa, hai số nguyên tố lớn được chọn ngẫu nhiên và nhân với nhau để tạo khóa chung. Các yếu tố của khóa công khai này sau đó được sử dụng để lấy khóa riêng, được giữ bí mật bởi chủ sở hữu của cặp khóa. Dữ liệu có thể được mã hóa bằng khóa chung và chỉ có thể được giải mã bằng khóa riêng tương ứng.

- Trong mật mã khóa công khai. Hàm cửa lật một chiều thường được sử dụng.

- Đặc điểm của hàm 1 chiều là tính toàn y = f(x) là dễ nhưng việc tính hàm ngược x = f-1(y) là không thể thực hiện hay là thời gian tiêu tốn để tính được hàm ngược là vô cùng lớn nhưng có cửa lật z để tính x = f-1z(y) là dễ.

- Ví dụ: Hàm f(x) = xa mod n với n = p.q là tích của 2 số nguyên tố lớn là một hàm một chiều vì tính f(x) là khá dễ dàng nhưng tính hàm ngược x = f-1(y) là rất khó và tốn rất nhiều thời gian tuy nhiên nếu biết cửa lật là hai thừa số p.q của n thì sẽ tính được x = f-1(y) là rất dễ dàng.

BÀI 3 (Bài code cuối bên trên)

****