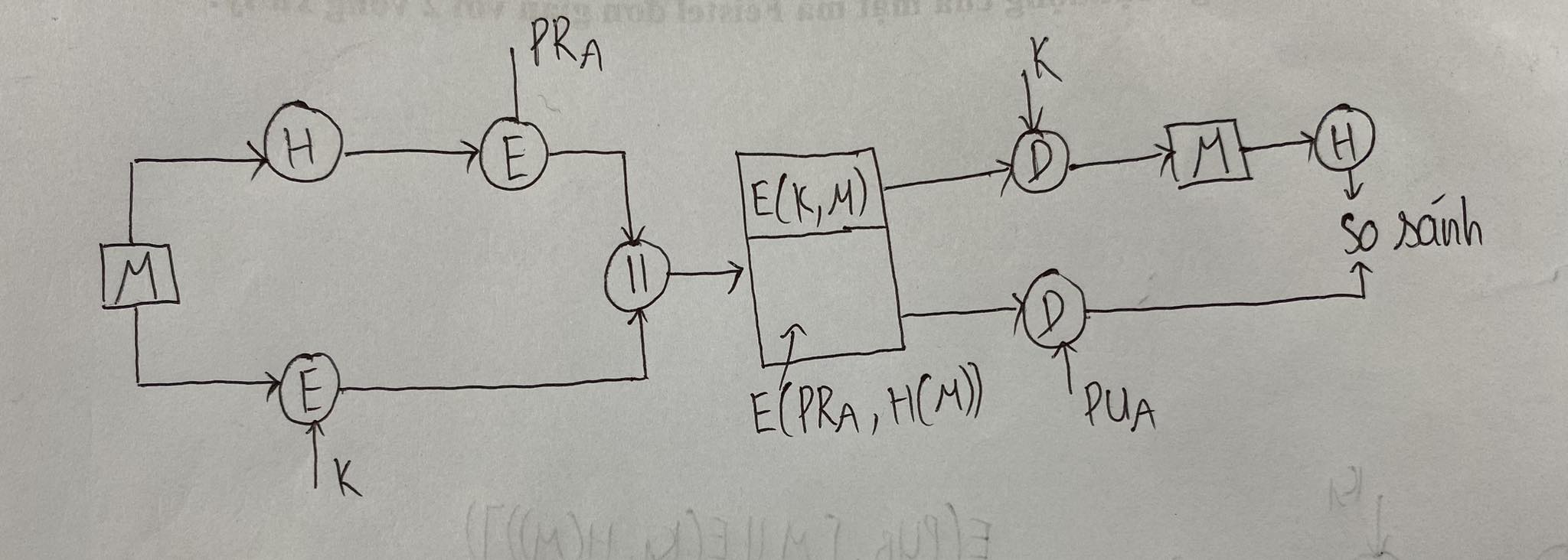
# Câu 1. Cho công thức dữ liệu => Vẽ sơ đồ và giải thích

|  |  |
| --- | --- |
| E(K,?)  Mã hóa |  |
| D(K,?)  Giải mã |  |
| E(PRA, ?) |  |
| D(PUA, ?) |  |
| A || B |  |
|  |  |

|  |  |
| --- | --- |
| Mật mã bất đối xứng | Khóa riêng (PRa): Khóa riêng của người gửi   * Xác thực nguồn gốc thông điệp   Khóa công khai người nhận (PUb)   * Bảo mật thông điệp trên đường truyền * Tốc độ chậm hơn mật mã đối xứng |
| Mật mã đối xứng | E(K): Có tác dụng bảo mật   * Tốc độ nhanh hơn mật mã bất đối xứng |
| Mã MAC | Sử dụng mật khóa K (C(K, ?),…)   * Mã chứng thực thông điệp |
| Mã Hash | H(…) |

Câu 1:

## E(K, M) || E(PRA, H( M))



**Bên gửi :**

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

Phần 1: Mã hóa thông điệp M với khóa K bởi hàm mã hóa E thu được E(K,M)

Phần 2: Mã hóa thông điệp M với hàm Hash thu được H(M) rồi đưa H(M) qua hàm mã hóa E với khóa riêng của người gửi thu được E(PRa, H(M))

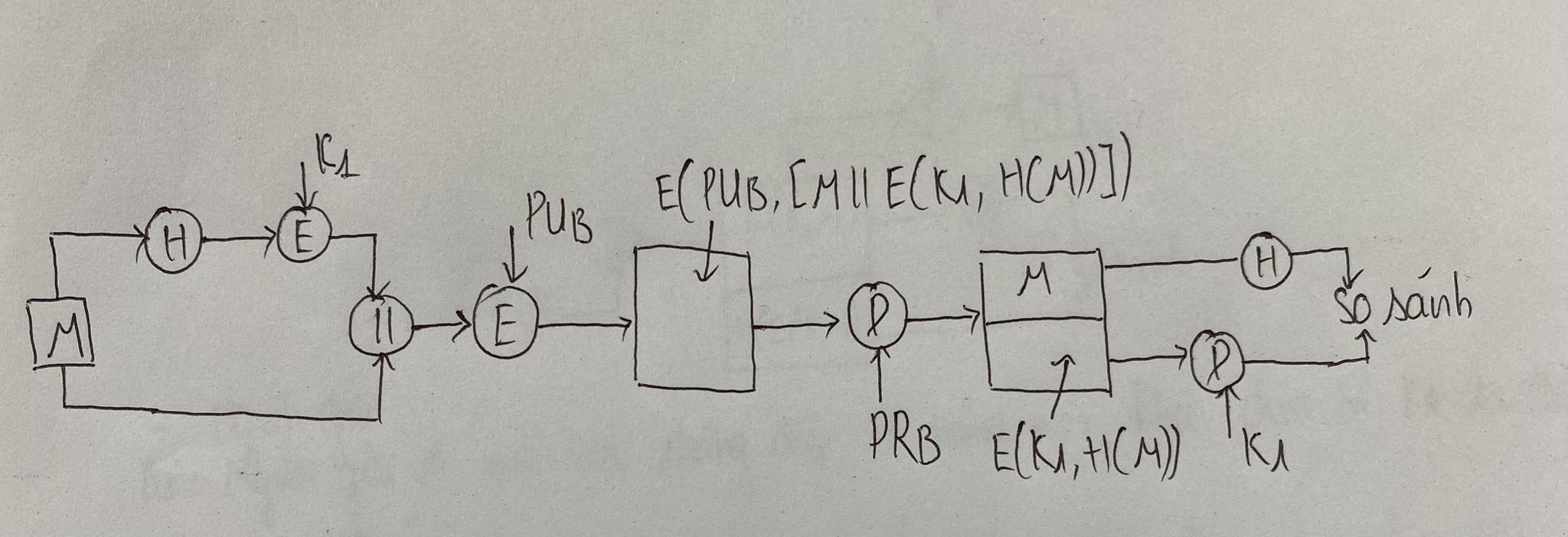
**Bên nhận:**

* + - * B nhận được đoạn mã từ A:
        + Đưa hàm mã hóa E(K, M) đưa qua hàm giải mã D với khóa K được thông điệp M.
        + Mã hóa thông điệp M với hàm hash thu được H(M) (\*)
        + Đưa hàm mã hóa E(PRa, H(M)) qua hàm giải mã D với khóa công khai PUb thu được H(M) (\*\*)
        + So sánh hàm Hash (\*) thu được và hàm Hash (\*\*) trong đoạn mã để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko.

**Tác dụng:** Xác thực nội dung thông điệp có chính xác không, khóa E(K) có tác dụng bảo mật thông điệp, Khóa PRa: Xác thực nguồn gốc thông điệp.

Câu 1:

## E(PUB, [M || E(K1, H(M))])



**Bên gửi :**

* + - * A gửi B đoạn mã
      * Phần 1: Mã hóa thông điệp M với hàm hash thu được H(M) , Mã hóa H(M) với khóa K1 bởi hàm mã hóa E thu được E(K1,H(M)), lấy E(K1,H(M)) ghép nối với thông điệp M được M|| E(K1,H(M)), đưa tất cả mã hóa với khóa công khai PUb bởi hàm mã hóa E thu được E(PUb, [M|| E(K1,H(M))])

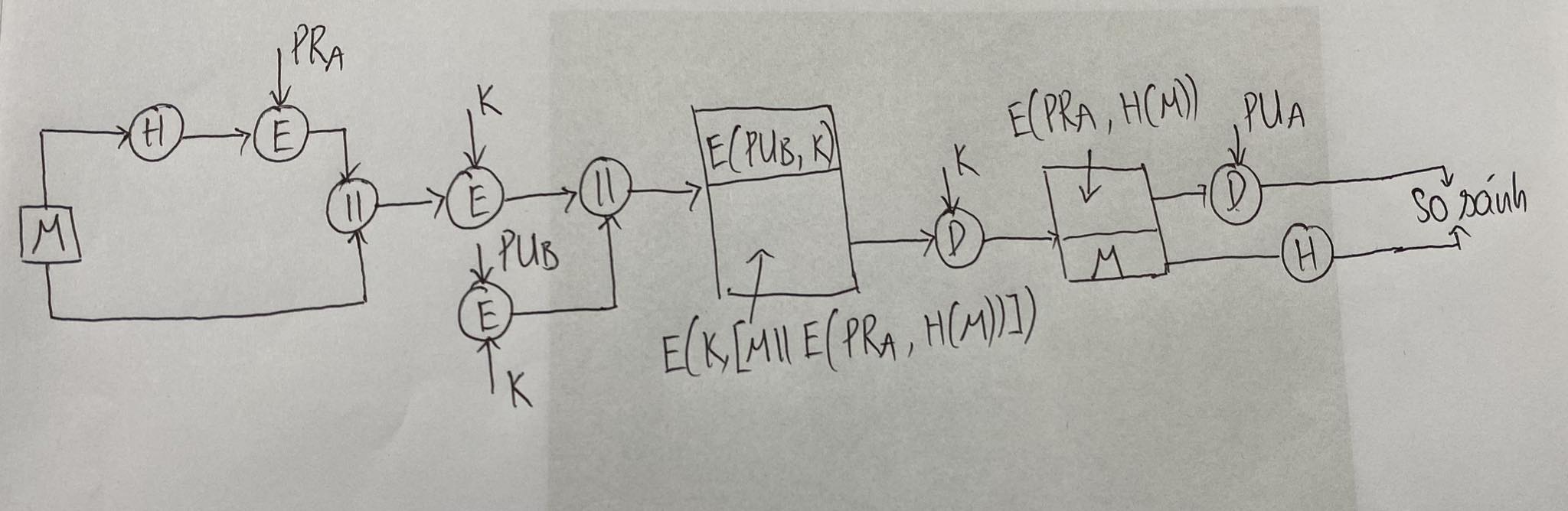
**Bên nhận:**

* + - * B nhận được đoạn mã từ A:
      * Đưa hàm mã hóa E(PUb, [M|| E(K1,H(M))]) đưa qua hàm giải mã D với khóa PRb được thông điệp M|| E(K1,H(M))
        + Mã hóa thông điệp M với hàm hash thu được H(M) (\*)
        + Đưa hàm mã hóa E(K1,H(M)) qua hàm giải mã D với khóa K1 thu được H(M) (\*\*)
        + So sánh hàm Hash (\*) thu được và hàm Hash (\*\*) trong đoạn mã để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko.

**Tác dụng:** Xác thực nội dung thông điệp có chính xác không, PUb: Bảo mật thông điệp trên đường truyền, E(K): Có tác dụng bảo mật

Câu 1:

## E(K, [M || E(PRA, H(M)) ]) || E(PUB, K)



**Bên gửi :**

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

Phần 1: Mã hóa thông điệp M với hàm hash thu được H(M), Mã hóa H(M) với khóa PRa bởi hàm mã hóa E thu được E(PRa,H(M)), lấy mã thu được ghép nối với M ta được

M|| E(PRa,H(M)). Mã hóa M|| E(PRa,H(M)) với khóa K bởi hàm mã hóa E thu được E(K, M|| E(PRa,H(M)))

Phần 2: Khóa K đưa qua khóa PUb của hàm E thu được E(PUb, K))

**Bên nhận:**

* + - * B nhận được đoạn mã từ A:

+ Giải mã hàm E(K, M|| E(PRa,H(M))) qua hàm giải mã D với khóa K ta thu được M|| E(PRa,H(M))

+ Mã hóa thông điệp M với hàm hash thu được H(M) (\*)

+ Giải mã hàm E(PRa,H(M)) qua hàm giải mã D với khóa công khai PUb thu được H(M) (\*\*)

* So sánh hàm Hash (\*) thu được và hàm Hash (\*\*) trong đoạn mã để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko.

**Tác dụng:** Xác thực nội dung thông điệp có chính xác không

Khóa PRa: Xác thực nguồn gốc thông điệp, khóa PUb: Bảo mật thông điệp trên đường truyền, E(K): Có tác dụng bảo mật, khóa PRb: xác thực nguồn gốc thông điệp.

## Câu 18: A->B: E(K,M)||H(M)||E(PUB, K)

**Bên gửi :**

* + - * A gửi B đoạn mã gồm 3 phần ghép với nhau:

Phần 1: Mã hóa thông điệp M với khóa K bởi hàm mã hóa E thu được E(K,M)

Phần 2: Đưa thông điệp M qua hàm Hash thu được H(M)

Phần 3: Mã hóa khóa K với khóa công khai PUB thu được E(PUB, K)

**Bên nhận:**

* + - * B nhận được đoạn mã từ A :

Đầu tiên giải mã E(PUB, K) với hàm giải mã với khóa riêng PRB thu được khóa K, dùng khóa K giải mã E(K, M) thu được thông điệp M. Sau đo đưa M qua hàm Hash, kết quả nhận được so sánh với hàm hash trong đoạn mã để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko.

* Tác dụng:
* Xác thực nội dung thông điệp có chính xác hay ko
* Có tác dụng giữ bí mật thông điệp

## Câu 19 A->B: E(K,M)||H(E(K, M))

**Bên gửi :**

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

Phần 1: Mã hóa thông điệp M với khóa K bởi hàm mã hóa E thu được E(K,M)

Phần 2: Mã hóa thông điệp M với khóa K bởi hàm mã hóa E thu được E(K,M) rồi đưa toàn bộ E(K,M) qua hàm Hash thu được H(E(K,M))

**Bên nhận:**

* + - * B nhận được đoạn mã từ A:
        + Đưa hàm mã hóa E(K, M) đưa qua hàm hash thu được mã hask H(E(K, M)). So sánh hàm Hash thu được và hàm Hash trong đoạn mã để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko.

**Tác dụng:** Xác thực nội dung thông điệp có chính xác không (Bảo mật nội dung thông điệp)

## Câu 20 A->B: E(K2, [M||C(K1, M)])||E(PUB, [K1||K2])

**Bên gửi :**

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

Phần 1: Thông điệp M được ghép mã MAC bởi hàm C với mật khóa K1 thu được C(K1, M) sau đó được ghép với thông điệp M và toàn bộ đoạn mã được mã hóa bởi hàm E với khóa K2 thu được E(K2, [M||C(K1, M)])

Phần 2: Hàm mã hóa E gồm khóa K1 ghép với khóa K2 và mã hóa bởi khóa công khai PUB

**Bên nhận:**

* + - * B nhận được đoạn mã từ A:

Tách ra làm 2 phần:

+ Giải mã hàm E(PUB, [K1||K2]) bằng khóa riêng PRB thu được 2 khóa K1 và K2

+ Giải mã hàm E(K2, [M||C(K1, M)]) biết khóa K2 thu được thông điệp M và hàm C(K1, M) (1)

+ Mã hóa hàm MAC với khóa K1 và thông điệp M thu được hàm C(K1, M) (2)

So sánh 2 hàm MAC (1)&(2) để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko

**Tác dụng:** Xác thực nội dung thông điệp có chính xác không – Có tác dụng giữ bí mật

## A->B: E(K1, M)||C(K2, E(K1, M)) || E(PUB, [K1||K2])

**Bên gửi :**

* + - * A gửi B đoạn mã gồm 3 phần ghép với nhau:

Phần 1: Thông điệp M được mã hóa bởi hàm E với khóa K1

Phần 2: Thông điệp M được mã hóa bởi hàm E với khóa K1 sau đó đưa qua hàm MAC với khóa K2 thu được C(K2,E(K1, M))

Phần 3: Hàm mã hóa E gồm khóa K1 ghép với khóa K2 và mã hóa bởi khóa công khai PUB

**Bên nhận:**

* + - * B nhận được đoạn mã từ A:

Tách ra làm 3 phần:

+ Giải mã hàm E(PUB, [K1||K2]) bằng khóa riêng PRB thu được 2 khóa K1 và K2

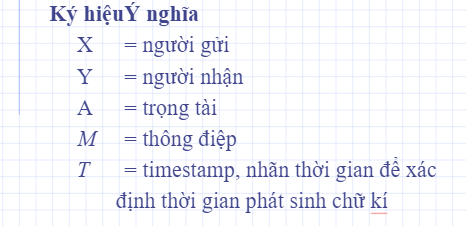
+ Giải mã hàm E(K1, M) biết khóa K1 thu được thông điệp M

+ Đưa hàm E(K1, M) qua hàm MAC với khóa K2 thu được hàm MAC C(K2, E(K1, M))(1)

So sánh 2 hàm MAC (1)&(2) để xác thực nội dung thông điệp có bị thay đổi trên đường truyền hay ko

**Tác dụng:** Xác thực nội dung thông điệp có chính xác không – Có tác dụng giữ bí mật

(PRa: Xác thực nguồn gốc thông điệp , PUb: Bảo mật thông điệp trên đường truyền)

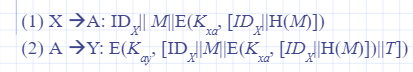


IDX: Tên/ Định danh của X

Kxa: Khóa mà X và A biết

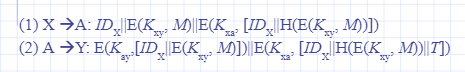
Kay: Khóa mà A và Y biết

### Mã hóa đối xứng, trọng tài thấy nội dung thông điệp



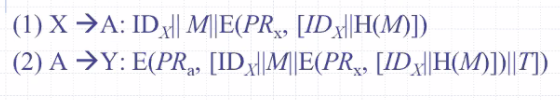
* Bên gửi (1):
  + X gửi đến A đoạn mã gồm 3 phần ghép với nhau:
    - Phần 1: IDX
    - Phần 2: thông điệp M
    - Phần 3: Thông điệp M được đưa qua hàm Hash và **ghép** với IDX **và** mã hóa với khóa Kxa bởi hàm mã hóa E
  + A nhận được đoạn mã:
    - A tạm tin người gửi thông điệp là X bởi IDx
    - A nhìn thấy thông điệp M, đưa thông điệp M qua hàm Hash thu được mã H(M) (1)
    - Tiến hành giải mã hàm E với khóa Kxa thu đc: IDx và H(M) (2)
    - So sánh 2 IDx biết thông điệp do X gửi
    - So sánh 2 thông điệp Hash (1) và (2) để xác thực nội dung thông điệp không bị thay đổi.
* A xác định người gửi đúng từ X và nội dung không bị thay đổi, A ký tên và gắn thêm nhãn thời gian T, sau đó đem toàn bộ đoạn mã mã hóa thêm lần nữa bằng khóa Kay chuyển đoạn mã tới Y.
* Bên nhận (2) :
  + Y nhận được đoạn mã từ A: Tiến hành giải mã bằng khóa Kay , từ đó biết được:
    - Người gửi đoạn mã là A (Kay)
    - Người gửi thông điệp là X (IDx)
    - Thời gian phát sinh chữ ký là T
    - Nội dung thông điệp là M
* Tác dụng:
  + Xác thực nội dung thông điệp
  + Xác thực nguồn gốc người gửi
* Nhận xét:
  + Cả người gửi và người nhận đều phải tin tưởng tuyệt đối vào trọng tài A
  + Nếu A không công tâm, anh ta có thể sửa đổi thông điệp, hoặc thông đồng với X để phủ nhận thông điệp đã ký, hoặc thông đồng với Y để giả mạo chữ ký của X

### Mã hóa đối xứng, trọng tài không thấy nội dung thông điệp



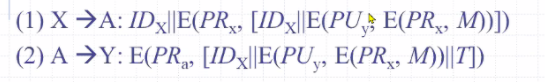
* Bên gửi (1):
  + X gửi đến A đoạn mã gồm 3 phần ghép với nhau:
    - Phần 1: IDX
    - Phần 2: hàm mã hóa E mã hóa thông điệp M với khóa Kxy E(Kxy,M)
    - Phần 3: Thông điệp M được mã hóa với khóa Kxy bởi hàm mã hóa E **sau đó** đưa qua hàm Hash và **ghép** với IDX **và** mã hóa với khóa Kxa bởi hàm mã hóa E
  + A nhận được đoạn mã:
    - A tạm tin người gửi thông điệp là X bởi IDx
    - Tiến hành giải mã hàm E với khóa Kxa thu đc: IDx và H(E(Kxy, M))
    - So sánh IDx ở ngoài và IDx vừa thu được xác định người gửi là X.
    - Hàm mã hóa E(Kxy,M) được đưa qua hàm Hash, kết quả thu được so sánh với hàm Hash vừa giải mã, xác thực thông điệp không bị thay đổi trên đường truyền.
* A xác định người gửi đúng từ X và nội dung không bị thay đổi, A ký tên và gắn thêm nhãn thời gian T, sau đó đem toàn bộ đoạn mã mã hóa thêm lần nữa bằng khóa Kay chuyển đoạn mã tới Y.
* Bên nhận (2) :
  + Y nhận được đoạn mã từ A: Tiến hành giải mã bằng khóa Kay , từ đó biết được:
    - Người gửi đoạn mã là A
    - Người gửi thông điệp là X (IDx)
    - Thời gian phát sinh chữ ký là T
    - Nội dung thông điệp là M (Sau khi giải mã bằng khóa Kxy)
* Tác dụng:
  + Trọng tài A không thể đọc và giả mạo thông điệp M
  + Bảo mật hơn vì trọng tài A không đọc được nội dung của thông điệp M do được mã hóa bởi khóa Kxy
* Nhận xét:
  + Cả người gửi và người nhận đều phải tin tưởng tuyệt đối vào trọng tài A
  + A không có khả năng sửa đổi thông điệp, nhưng vẫn có thể thông đồng với X để phủ nhận thông điệp đã ký, hoặc thông đồng với Y để giả mạo chữ ký của X

### Mã hóa khóa công khai, trọng tài thấy nội dung thông điệp



* Bên gửi (1):
  + X gửi A đoạn mã gồm 3 phần ghéo với nhau:
    - Phần 1: IDX
    - Phần 2: thông điệp M
    - Phần 3: Thông điệp M được đưa qua hàm Hash được **ghép** với IDX rồi đem mã hóa bằng khóa riêng của X là PRx với hàm mã hóa E.
  + A nhận được đoạn mã từ X:
    - A tạm tin người gửi thông điệp là X bởi IDx
    - A nhìn thấy thông điệp M
    - A biết chắc rằng người gửi cho mình đúng là X sau khi giải mã hàm E qua hàm giải mã với khóa công khai của X (PUx), thu được IDx và hàm Hash H(M) (1)
      * So sánh 2 IDx bên trong và ngoài => xác nhận người gửi là X
      * Thông điệp M được đưa qua hàm Hash, kết quả thu được so sánh với hàm Hash vừa giải mã để xác thực nội dung thông điệp M không bị thay đổi trên đường truyền.
* A xác nhận các nội dung thông điệp nhận được từ X là đúng, tiến hành ký tên và đính kèm thời gian ký tên là T, sau đó đem toàn bộ đoạn mã trên mã hóa bằng khóa riêng của A là PRa rồi gửi đến Y.
* Bên nhận(2) :
  + Y sau khi nhận được đoạn mã từ A:
    - Xác nhận đoạn mã từ A vì đoạn mã được giải mã bằng mã công khai của A là PUa
    - Sau đó biết người gửi thông điệp là X
    - Nội dung thông điệp là M chưa xác minh được đưa qua hàm Hash thu được H(M) (1)
    - Thời gian phát sinh chữ ký là T
    - Giải mã tiếp bằng khóa công khai PUx: Thu được IDX và H(M) (2)
      * So sánh 2 IDx bên trong và ngoài
      * So sánh 2 hàm Hash (1) và (2) => Xác nhận thông điệp M
* Y tự kiểm tra được nội dung thông điệp mà X gửi cho A
* Tác dụng:
  + Xác thực nội dung thông điệp
  + Xác thực nguồn gốc và tính toàn vẹn của thông điệp
* Nhận xét: A biết nội dung thông điệp, nhưng không thể sửa đổi thông điệp hay làm giả chữ ký

### Mã hóa khóa công khai, trọng tài không thấy nội dung thông điệp



* Bên gửi (1):
  + X gửi A đoạn mã gồm: IDx ghép với hàm mã hóa E có thông điệp M được mã hóa bởi khóa riêng PRx sau đó được mã hóa bằng khóa công khai PUy rồi đem đoạn mã ghép với IDx sau đó đem toàn bộ đoạn mã mã hóa bởi khóa riêng PRx.
  + A nhận được đoạn mã từ X:
    - Biết người gửi là X(IDx), khi nhìn thấy PRx thì biết được nguồn gốc thông điệp là từ X
    - Giải mã bằng khóa công khai PUx thu được:
      * IDx, so sánh với IDx bên ngoài =>tin rằng người gửi là X
      * Hàm mã hóa khóa công khai PUy =>Không thể giải mã được vì không có khóa riêng của Y
* A xác nhận được người gửi là X, sau đó gắn thêm nhãn thời gian T vào thông điệp mã hóa toàn bộ đoạn mã bằng khóa riêng PRa.
* Bên nhận(2):
  + Y nhận được đoạn mã từ A: Giải mã bằng khóa công khai PUa thu được:
    - IDx => Người gửi thông điệp là X
    - Thời gian ký tên là T
    - Giải mã bằng khóa riêng PRy sau đó giải mã bằng khóa công khai PUx thu được thông điệp M.
* Tác dụng:
  + Xác thực nguồn gốc và tính toàn vẹn của thông điệp
  + Xác thực nội dung
  + Bảo mật
* Nhận xét: A không biết nội dung thông điệp, không thể sửa đổi thông điệp hay làm giả chữ ký

# Câu 2

## So sánh

### So sánh Virus và Worm?

|  |  |
| --- | --- |
| Virus | Worm |
| Virus là một đoạn mã chương trình có thể lây nhiễm tới các chương trình khác (bằng cách gắn bản sao của nó vào các chương trình mà nó tìm thấy) | Worm là một chương trình có thể tự nhân bản và gửi các bản sao của nó từ máy tính này đến máy tính khác qua các kết nối mạng. |

### Thuật toán Vernam thực hiện ntn?

### Hàm Hash ?

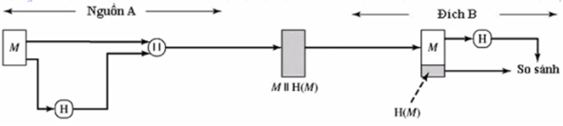
* Không thể tính ngược để thu được M từ H(M)
* Hàm Hash và mật mã đối xứng mục đích: Đảm bảo rằng thông điệp M đến từ A và không bị thay đổi trên đường truyền(do lỗi đường truyền hoặc bị tấn công)
* Còn lại trong Giáo trình Chương 3

### Khóa riêng, khóa công khai?

* PRa: Khi mã hóa bằng khóa riêng thì lấy khóa công khai (PUa) để giải mã => Tác dụng là xác thực nguồn gốc thông điệp.
* PUa: Khi mã hóa bằng khóa công khai thì lấy khóa riêng (PRa) để giải mã => Tác dụng bảo mật.

### Hàm Hash và mật mã đối xứng?

#### Hình 1



Mục đích: Biểu đồ Giúp kiểm tra thông điệp có bị lỗi trên đường truyền

**Bên gửi:**

- Thông điệp M được đưa qua hàm Hash H thu được mã hash của M: H(M)

- Thông điệp M ghép với mã hash H(M) thu được M||H(M)

**Bên nhận:**

- Đoạn mã nhận được thu được (M || H(M)) => Tách ra làm 2 phần

P1: Thông điệp M đưa qua hàm hash H thu được mã H(M)(1)

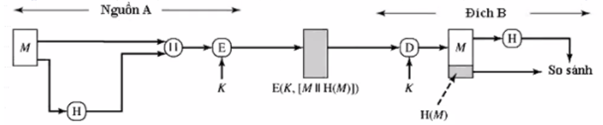
P2: Mã H(M)

Bên nhận so sánh 2 mã Hash(1) & (2) xem thông điệp nhận được có chính xác hay không?

**Tác dụng của hệ thống:**

- Xác thực nội dung thông điệp (phát hiện thay đổi) có chính xác hay không?(So sánh 2 mã Hash)

#### Hình 2



Mục đích: Đảm bảo rằng thông điệp M đến từ A và không bị thay đổi trên đường truyền (do lỗi đường truyền hoặc bị tấn công)

**Bên gửi:**

- Thông điệp M được đưa qua hàm Hash H thu được mã hash của M (H(M))

- Thông điệp M ghép với mã hash H(M) thu được M||H(M)

- Toàn bộ đoạn mã M||H(M) được mã hóa bởi hàm E với khóa K

**Bên nhận:**

- Đoạn mã nhận được đưa qua hàm giải mã với khóa K thu được (M || H(M))

=> Tách ra làm 2 phần

P1: Thông điệp M đưa qua hàm hash H thu được mã H(M)(1)

P2: Mã H(M)

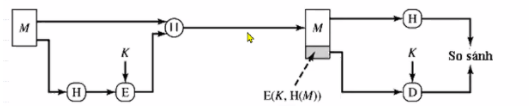
Bên nhận so sánh 2 mã Hash(1) & (2) xem thông điệp nhận được có chính xác hay không?

**Tác dụng của hệ thống:**

- Xác thực nội dung thông điệp có chính xác hay không? (So sánh 2 mã Hash)

- Có khả năng giữ bí mật nội dung thông điệp

#### Hình 3



Mục đích: Đảm bảo rằng thông điệp M đến từ A và không bị thay đổi trên đường truyền (do lỗi đường truyền hoặc bị tấn công)

**Bên gửi:**

- Thông điệp M được đưa qua hàm Hash H thu được mã hash của M (H(M))

- Toàn bộ mã H(M) được mã hóa bởi hàm E với khóa K

- Thông điệp M được ghép với hàm E(K, H(M))

**Bên nhận:**

Đưa thông điệp M qua hàm Hash thu được mã H(M)(1)

- Hàm mã hóa E(K, H(M)) được đưa qua hàm giải mã với khóa K thu được mã H(M) (2)

- So sánh 2 mã Hash(1) & (2) để biết thông điệp M có chính xác hay không?

**Tác dụng của hệ thống:**

- Xác thực nội dung thông điệp có chính xác hay không? (So sánh 2 mã Hash)

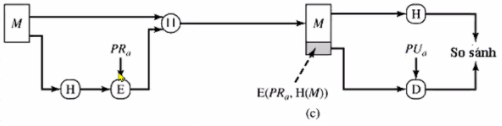
- Có khả năng giữ bí mật nội dung thông điệp

**Nhận xét:**

- Giảm được khối lượng tính toán so với hình 1

- Không có khả năng giữ bí mật nội dung thông điệp

#### Hình 4



Mục đích: Đảm bảo rằng thông điệp M đến từ A và không bị thay đổi trên đường truyền

**Bên gửi:**

- Thông điệp M được đưa qua hàm Hash H thu được mã hash của M (H(M))

- Mã H(M) được mã hóa bằng hàm E với khóa riêng PRa

- Thông điệp M được ghép với hàm E(PRa,H(M))

**Bên nhận:**

- Đưa thông điệp M qua hàm Hash thu được mã H(M) (1)

- Hàm mã hóa E(PRa,H(M)) được đưa qua hàm giải mã với khóa công khai PUa thu được mã H(M) (2)

- So sánh 2 mã Hash (1) và (2) để biết thông điệp M có chính xác hay không?

**Tác dụng của hệ thống:**

- Xác thực nội dung thông điệp M có chính xác hay không (So sánh 2 mã Hash (1), (2))

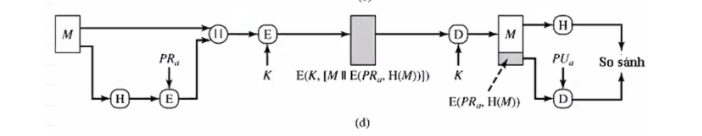
- Xác thực nguồn gốc thông điệp có phải do A gửi đến hay không (PRa) (chỉ có nguồn gốc bất đối xứng)

- Tuy nhiên ko có khả năng giữ bí mật nội dung thông điệp

- Đây chính là cơ sở của chữ ký số.

**Nhận xét:** Dùng mật mã bất đối xứng: a dùng mã hóa riêng của mình tức là đã tạo tiền đề chữ ký số (a muốn phủ nhận cũng ko đc)

#### Hình 5



**Bên gửi:**

- Thông điệp M được đưa qua hàm Hash thu được mã H(M), mã H(M) được mã hóa bằng hàm E với khóa riêng PRa thu được E(PRa, H(M))

- Thông điệp M ghép với hàm mã hóa E(PRa, H(M)) thu được (M||E(PRa,H(M)))

- Đoạn mã (M || E(PRa, H(M))) được mã hóa bằng hàm E với khóa K.

**Bên nhận:**

- Đoạn mã  nhận được đưa qua hàm giải mã với khóa K thu được (M || E(PRa, H(M)))

=> Tách ra 2 phần

- P1: Thông điệp M đưa qua hàm hash thu được mã H(M) (1)

- P2: Hàm mã hóa E(PRa, H(M)) được đưa qua hàm giải mã với khóa công khai PUa thu được H(M) (2)

- Bên nhận so sánh 2 mã Hash (1) & (2)  xem thông điệp nhận được có chính xác hay không?

**Tác dụng của hệ thống:**

- Xác thực nội dung thông điệp có chính xác hay không ? (so sánh 2 mã Hash)

- Xác thực nguồn gốc thông điệp có phải do A gửi không (PRa)

- Bảo mật nội dung thông điệp(Nếu có ăn cắp thì cũng ko đọc được vì nó đã được mã hóa)

**Nhận xét:** - Là 1 sự nâng cấp của hình trên

## Mã hóa -Giải mã bằng tay,…

## Ưu nhược điểm…

### Ưu nhược điểm của các kỹ thuật mã hóa cổ điển

Các kĩ thuật mã hóa cổ điển gồm:

* Kỹ thuật thay thế: Thuật toán mã hoá sẽ thay thế mỗi kí tự trong bản rõ bằng một kí tự khác
  + Đơn giản, dễ thực hiện
  + Độ an toàn không cao, dễ bị bẻ khóa
* Kỹ thuật chuyển dịch – hoán vị: Các kí tự của plaintext sẽ được hoán đổi vị trí cho nhau để tạo thành ciphertext
  + Khó bẻ khóa hơn kỹ thuật thay thế
  + Che dấu được một phần thông tin về tần suất xuất hiện các kĩ tự trong mật mã
  + Mật mã hoán vị thuần túy rất dễ nhận ra bởi nó giữ nguyên tần suất xuất hiện ký tự đơn (và làm thay đổi tần suất của các cặp, các bộ kí tự của plaintext)

Để tăng độ phức tạp, người ta có thể tiến hành đổi chỗ nhiều lần, hoặc kết hợp với thuật toán mã hóa khác.

|  |  |
| --- | --- |
| Kỹ thuật thay thế | Kỹ thuật hoán vị |
| Có tác dụng làm *xáo trộn* thông tin thống kê của plaintext, làm phức tạp hóa mối quan hệ thống kê giữa ciphertext và mật khoá, nhằm ngăn cản nỗ lực tìm khoá. | Có tác dụng làm *khuếch tán* thông tin thống kê của plaintext, pha loãng các cấu trúc thống kê của plaintext ra một phạm vi rộng hơn, làm phức tạp hóa mối quan hệ thống kê giữa ciphertext và plaintext. |

### Phân biệt Caesar, Affine, Monoalphabetic, Ponoalphabetic

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tổng quát | Ưu điểm | Nhược điểm |
| Caesar | Mỗi kí tự trong bảng chữ cái được thay thế bởi một kí tự khác cùng bảng, cách sau nó ba vị trí. | Mật mã Caesar đơn giản, dễ thực hiện | Độ an toàn không cao, dễ bị bẻ khoá bởi tấn công Brute-force do số lượng khoá quá ít (chỉ có 25 khoá) |
| Affine | Kí tự P ban đầu được thay thế bởi kí tự C | Mật mã Affine có độ phức tạp lớn hơn mật mã Caesar tổng quát, số lượng khoá cũng nhiều hơn | Độ an toàn chưa cao, dễ bị phá bởi tấn công Brute-force do số lượng khoá chưa nhiều (chỉ có 312 khoá) |
| Mono alphabetic | - Mỗi kí tự trong bảng chữ cái được thay thế bởi một kí tự bất kì khác cùng bảng.  - Chỉ sử dụng một bảng mã (mỗi kí tự plain text được thay thế bởi một kí tự cố định), nên không giấu được tần suất xuất hiện các kí tự. | Mật mã Monoalphabetic có số lượng khoá rất lớn, khó bẻ khoá bằng phương pháp Brute -force |  |
| Poly alphabetic | Sử dụng nhiều bảng mã khác nhau (mỗi kí tự plain text có thể được thay thế bởi nhiều kí tự khác nhau, dựa trên các khoá thay thế khác nhau) | Tuy nhiên vẫn có thể bẻ khoá mật mã này dựa trên các thống kê về các đặc điểm tự nhiên của ngôn ngữ  -Mật mã Monoalphabetic dễ bẻ vì không che giấu được tần xuất suất hiện của các kí tự trong văn bản. |  |

### Kỹ thuật rain-fence

|  |  |
| --- | --- |
| Tên | Kỹ thuật rain-fence |
| Tổng quát | Plaintext được viết dịch xuống tuần tự theo các đường chéo rồi đọc trình tự theo các hàng. |
| Ưu |  |
| Nhược | Mật mã hoán vị thuần túy rất dễ nhận ra bởi nó giữ nguyên tần suất xuất hiện ký tự đơn (và làm thay đổi tần suất của các cặp, các bộ kí tự của plaintext)   Để tăng độ phức tạp, người ta có thể tiến hành đổi chỗ nhiều lần, hoặc kết hợp với các thuật toán mã hoá khác. |

### Mã hóa đa ký tự

### Phân biệt Mật mã Playfair, Mật mã Hill

|  |  |  |
| --- | --- | --- |
| Tên mã | Mật mã Playfair | Mật mã Hill |
| Tổng quát | -Mật mã Playfair sẽ thay thế từng cặp 2 kí tự trong bản rõ bởi 2 kí tự tương ứng trong ma trận khoá 5 x 5.  -Lần lượt viết từng kí tự của khóa vào ma trận, từ trái sang phải, từ trên xuống dưới, bỏ các kí tự trùng lặp   -Viết các ký tự còn lại trong bảng chữ cái vào ma trận theo thứ tự, I và J được coi như một ký tự  -Mỗi ký tự trong cặp plaintext sẽ được mã hoá bằng ký tự nằm cùng hàng với nó, nhưng cùng cột với ký tự kia   -Nếu cặp ký tự plaintext rơi vào cùng một hàng của ma trận thì mỗi ký tự được thay thế bởi ký tự bên phải nó.  -Nếu ký tự plaintext rơi vào cột cuối cùng, thì ciphertext của nó là ký tự cùng hàng ở cột đầu tiên.  -Nếu cặp ký tự plaintext rơi vào chung một cột của ma trận thì mỗi ký tự được thay thế bởi ký tự ngay sát dưới.  - Nếu ký tự plaintext rơi vào hàng cuối cùng, thì ciphertext của nó là ký tự cùng cột, ở hàng đầu tiên.  -Nếu hai kí tự trong plaintext giống nhau thì chúng sẽ được được cách ly bằng một ký tự đại diện, chẳng hạn là **x**. | -Mật mã Hill sẽ thay thế từng nhóm *m* kí tự trong plaintext bởi *m* kí tự ciphertext   -*m* kí tự ciphertext được xác định bởi hệ *m* phương trình tuyến tính  *C*m = (*k*m1*P*1 + km2*P*2 + …+ *k*mm*P*m) mod 26 |
| Ưu | Mật mã Playfair có không gian khoá lớn tương tự mật mã Monoalphabetic nên khó bẻ được khoá bằng phương pháp Brute -force  Mật mã Playfair có khả năng che giấu một phần thông tin về tần suất xuất hiện các chữ cái, nhờ thực hiện mã hoá từng cặp hai kí tự | Độ an toàn của mật mã Hill sẽ càng lớn khi sử dụng ma trận K càng lớn   Mật mã Hill có khả năng che dấu hoàn toàn tần suất xuất hiện các kí tự đơn   Mật mã Hill rất mạnh khi chống lại tấn công chỉ biết ciphertext |
| Nhược |  | nhưng nó lại dễ dàng bị bẻ gãy với một tấn công biết plaintext, do có thể dễ dàng xác định ma trận K từ các cặp P-C đã biết. |

### Phân biệt Mật mã Vigenere, Mật mã Vernman

|  |  |  |
| --- | --- | --- |
| Tên | Mật mã Vigenere | Vernman |
| Tổng quát | Mật mã Vigenère sẽ thay thế từng nhóm *m* kí tự trong plaintext bởi *m* kí tự ciphertext  *m* kí tự ciphertext được xác định bởi hệ *m* phương trình  *C*m = (*P*m + *k*m) mod 26  *P*m = (*C*m - *k*m) mod 26 | Plaintext được biểu diễn dưới dạng một chuỗi bít nhị phân   Khoá *K* cũng được biểu diễn dưới dạng một chuỗi bít nhị phân (càng dài càng tốt, càng ngẫu nhiên càng tốt)   Ciphertext được sinh ra bởi phép XOR giữa plaintext với khóa *K* |
| Ưu | Như vậy, nếu *k1= k2 = … = km* thì mật mã Vigenère sẽ trở thành mật mã Caesar tổng quát.   Khi *k1≠ k2 ≠ … ≠ km*: một kí tự plaintext có thể được thay thế bởi nhiều kí tự khác nhau (ứng với các *k* khác nhau), nhờ vậy có thể che giấu được tần suất xuất hiện các kí tự. | Với khoá *K* đủ dài và ngẫu nhiên, các thông tin mang tính thống kê của ngôn ngữ có thể được che giấu hoàn toàn  Sự ra đời của mật mã hệ nhị phân là tiền đề cho sự ra đời của các mật mã hiện đại. |
| Nhược | Mật mã Vigenère vẫn không giấu được hoàn toàn tần suất xuất hiện các kí tự, và vẫn có thể bị phân tích  Giải pháp khắc phục là sử dụng hệ thống biểu diễn thông tin không mang tính thống kê của ngôn ngữ - đó là hệ nhị phân | Nếu *K* ngắn thì sẽ phải sử dụng *K* lặp đi lặp lại, làm giảm tính ngẫu nhiên, và có thể làm lộ một phần thông tin về thống kê tần suất.   Tuy nhiên, việc sinh ra được một khoá *K* dài và thực sự ngẫu nhiên như vậy sẽ đòi hỏi nhiều công sức. |

### Phân biệt Virus, Worm

|  |  |
| --- | --- |
| Virus | Worm |
| Virus là một đoạn mã chương trình có thể lây nhiễm tới các chương trình khác (bằng cách gắn bản sao của nó vào các chương trình mà nó tìm thấy) | Worm là một chương trình có thể tự nhân bản và gửi các bản sao của nó từ máy tính này đến máy tính khác qua các kết nối mạng. |

### Phân biệt Virus Boot, Virus File thi hành, Virus Macro

|  |  |  |
| --- | --- | --- |
| Virus Boot | Virus File thi hành | Virus Macro |
| Loại virus này lây lan vào đoạn mã khởi động trên Boot sector hay Master Boot Record của đĩa   Khi máy tính khởi động, virus boot được kích hoạt. Nó sẽ thường trú trong bộ nhớ, chờ để lây vào một ổ đĩa mới   Nếu đem ổ đĩa nhiễm virus lắp sang một máy tính khác, rồi khởi động máy từ ổ đĩa đó, máy tính sẽ bị nhiễm virus. | Loại virus này lây nhiễm vào các file nhị phân thi hành được (.EXE, .COM, .DLL, .BIN, .SYS…)   Đoạn mã virus có thể được gắn vào đầu file, cuối file, hoặc giữa file   Khi file được chạy, virus sẽ được kích hoạt, nó sẽ tìm cách lây vào các file khác trong máy | Loại virus này lây nhiễm vào các macro trong các file tài liệu của MicroSoft Office (Word, Excel…)   Một số macro có khả năng tự khi hành khi mở file, cất file… Virus Macro thường nằm trong các macro tự động đó.   Khi file được mở, virus sẽ được kích hoạt, sau đó nó tìm cách lây vào các file tài liệu khác. |

### Phân biệt mật mã cổ điển và mật mã hiện đại

|  |  |  |
| --- | --- | --- |
| Tên | Mật mã cổ điển | Mật mã hiện đại |
| Đối tượng | những ký tự (có thể là ký tự chữ viết, ký tự số hoặc các ký tự tượng hình) | các bit thông tin |
| Công cụ thực hiện | thủ công, cơ học hay các máy điện toán đơn giản | các máy tính điện tử tốc độ cao. |
| Ứng dụng | khá hạn hẹp và chủ yếu tập trung vào quân sự | miền ứng dụng rộng hơn trong các giao dịch điện tử của đông đảo người dùng. |
| yếu tố thời gian | có từ lâu đời | Mấy chục năm gần đây |

## Là gì ?

### Giao thức SSL là gì? – Giáo trình Chương 4

* SSL một tiêu chuẩn của công nghệ bảo mật, tạo ra một liên kết được mã hóa giữa máy chủ web và trình duyệt.
* Liên kết này đảm bảo tất cả các dữ liệu trao đổi giữa máy chủ web và trình duyệt luôn được bảo mật và an toàn.
* SSL được sử dụng rộng rãi trên hàng triệu Website ở khắp thế giới, giúp bảo vệ dữ liệu an toàn trên môi trường internet
* SSL cung cấp:
  + Sự bảo mật: Các dữ liệu truyền tải giữa máy chủ và trình duyệt sẽ được mã hoá, đảm bảo tính riêng tư và toàn vẹn.
  + Khả năng chứng thực: Mỗi chứng chỉ số SSL được tạo ra cho một Website duy nhất, khẳng định độ tin cậy của Website đó.
  + Một cơ quan uy tín sẽ xác thực danh tính và độ tin cậy của Website trước khi cấp chứng chỉ SSL cho Website.

### SET là gì?

* SET (Secure Electronic Transaction – Giao dịch điện tử an toàn) được thiết kế để bảo vệ các giao dịch bằng thẻ tín dụng qua Internet
* SET là một tập các giao thức an ninh và các dạng thức cho phép người dùng thanh toán bằng thẻ tín dụng trong một mạng mở (như Internet) một cách an toàn.
* SET cung cấp 3 dịch vụ:
  + Cung cấp kênh liên lạc an toàn giữa các thực thể góp mặt trong một giao dịch.
  + Cung cấp tính tin cậy bằng cách dùng các chứng chỉ số X.509v3.
  + Bảo đảm tính riêng tư, do các thông tin chỉ được cung cấp cho các thực thể khi tham gia giao dịch và vào lúc cần thiết.

### Dịch vụ chứng thực X.509 Giáo trình Chương 4

### Các thành phần tham gia giao dịch điện tử & Các bước thực hiện giao dịch Giáo trình Chương 4

### Chữ ký kép Giáo trình Chương 4

### Khi mua hàng người bán gửi cho người mua thông tin?

* Khách hàng chọn các món hàng cần mua rồi gửi danh sách hàng hoá cho người bán.Ngưới bán gửi lại khách một đơn hàng có chứa danh mục hàng hóa đã chọn, số lượng, đơn giá và tổng số tiền.
* Người bán gửi một bản sao chứng chỉ , nhờ vậy khách hàng có thể xác minh được rằng mình đang giao dịch với một cửa hàng có thực và hợp lệ.
* Người bán gửi xác nhận đơn hàng cho khách hàng.
* Người bán cung cấp hàng hóa hoặc dịch vụ. Người bán chuyên chở hàng hóa hoặc áp dụng dịch vụ cho khách hàng.

### Các kỹ thuật lấy mật khẩu Giáo trình Chương 5

# Câu 3

## Đưa ra giải pháp An ninh

## Lập trình

|  |  |  |
| --- | --- | --- |
|  | | #include <iostream>  #include <string>  using namespace std;  int KT\_So(char *c*){    return c-'a';  }  char So\_KT(int *n*){    return 'a'+n;  }  int main()  {    Lệnh xử lý  } |
| **Đề** | | **Lệnh xử lý** | | |
| -Lập trình nhập một chuỗi ký tự từ bàn phím, cộng mỗi phần tử của chuỗi với K(K là một số nguyên nhập từ bàn phím). Hiện chuỗi mới ra màn hình -Lập trình khôi phục lại chuỗi ban đầu. | | string s;    cout<<"Nhap chuoi ki tu: "; getline(cin,s);    int k;    cout<<"Nhap khoa:"; cin>>k;    //  Ma hoa    for(int i = 0; i <s.size(); i++) s[i] = s[i]+k;    cout<<"Chuoi ma hoa: "<<s;    //Giai ma    for(int i = 0; i <s.size(); i++) s[i] = s[i]-k;    cout<<"\nChuoi giai hoa: "<<s; | | |
| a) Lập trình nhập một chuỗi ký tự từ bàn phím, mã hóa chuỗi bằng thuật toán CAESAR tổng quát với khóa K nhập từ bàn phím. Hiện chuỗi mới. -Lập trình giải mã để khôi phục lại chuỗi ban đầu. | | // Thuat toan CAESAR tong quat    string P,C; int K;    cout<<"Nhap chuoi plaintext: "; cin>>P;    cout<<"Nhap khoa K="; cin>>K;    for(int i = 0; i <P.size(); i++){      int m = (KT\_So(P[i]) + K)%26;      C += So\_KT(m);    }    cout<<"Chuoi ma hoa: "<<C;    //  Giai ma    P = "";    for(int i = 0; i <C.size(); i++){      int m = (KT\_So(C[i]) – K + 26)%26;      P += So\_KT(m);    }    cout<<"\nChuoi giai ma: "<<P; | | |
| b) Lập trình bẻ khóa mật mã Caesar bằng phương pháp Brute-force.  -Đầu vào chương trình là chuỗi ký tự cipher text thu được từ câu a | | *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(){  //B2 be khoa bang phuong phap Brute-force  for(int i = 0; i <26; i++){   string Giaima = Giai\_ma(C,i);   string Mahoa = Ma\_hoa(Giaima, i);   int Khoa = i;   cout<<"\nKhoa:"<<Khoa<<"-> "<< Giai\_ma(C,Khoa);  } | | |
| Lập trình nhập một chuỗi ký tự từ bàn phím, mã hóa chuỗi bằng thuật toán Affine với cặp số {a, b} nhập từ bàn phím. Hiện chuỗi mới ra màn hình.  Lập trình giải mã Affine để khôi phục lại chuỗi ban đầu:  - Đầu vào chương trình là cặp số {a, b} và chuỗi ký tự cipher text từ (a).  - Đầu ra chương trình là chuỗi ký tự plain text. | | int main()  {    string P,C;    cout<<"<-----Thuat toan Affine----->\n";    cout<<"\nNhap chuoi plaintext: ";cin>>P;    int a=0, b;    cout<<"Cap so ma Affine\n";    while (a%2 == 0 || a == 13){      cout<<"a ko chẵn va a != 13";      cout<<"a = ";       cin>>a;    }    cout<<"b = ";         cin>>b;    //  Ma hoa    for(int i = 0; i <P.size(); i++){      int m = (a\*KT\_So(P[i])+b)%26;      C += So\_KT(m);    }    cout<<"Chuoi ma hoa: "<<C;    //giai ma    P ="";    int a1;//Nghich dao cua a    for(int i = 1; i <26; i+=2){      if(i\*a%26==1){        a1 = i;   break;      }    }    for(int i = 0; i <C.size(); i++){      int m = a1 \*(KT\_So(C[i]) - b + 26)%26;      P += So\_KT(m);    }    cout<<"\nGiai ma: "<<P;  } | | |

|  |  |
| --- | --- |
| a) Bẻ khóa Affine Lập trình bẻ khóa mật mã Afine bằng phương pháp Brute-force:  - Đầu vào chương trình là chuỗi ký tự cipher text thu được từ (a).  - Hãy xác định cặp số {a, b} đã sử dụng và nội dung của plain text ban đầu. | *string* Giai\_ma(*string* *C*){    int a, b,dem=1;    string P;    for(int a = 1; a <26; a+=2){     if(a!=13){      for(int b = 0; b <26; b++){       //Giai ma      P ="";       int a1;//Nghich dao cua a      for(int i = 1; i <26; i+=2)         if(i\*a%26==1){ a1 = i;  break;}  for(int i = 0; i <C.size(); i++){        int m = a1 \* (KT\_So(C[i]) - b + 26) %26;          P += So\_KT(m);       }      cout<<a<<","<<b<<": "<<dem++<<": "<<P<<endl;      }    }    }  return P;  }  int main()  {    string P;string C = "VHCFIVPCVHSFIOSF";    cout<<"<-Be khoa Thuat toan Affine->\n";    Giai\_ma(C);  } |

|  |  |
| --- | --- |
| a) Lập trình nhập một chuỗi ký tự từ bàn phím, mã hóa chuỗi bằng thuật toán Monoalphabetic với khóa K nhập từ bàn phím (Khóa K là một chuỗi gồm 26 chữ cái có trật tự bất kỳ). Hiện chuỗi mới ra màn hình.  b) Lập trình giải mã Monoalphabetic để khôi phục lại chuỗi ban đầu:  - Đầu vào chương trình là khóa K và chuỗi ký tự cipher text từ (a).  - Đầu ra chương trình là chuỗi ký tự plain text | int main()  {    string P,C,K,B;    B = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";    cout<<"<----Thuat toan Monoalphabetic---->\n";    cout<<"\nNhap chuoi plaintext: "; cin>>P;    C = P;    cout<<"\nNhap chuoi K: "; 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;    //Giai ma    for(int i = 0; i<C.size();i++)      for(int j = 0; j < K.size() ;j++)        if(C[i] == K[j]){ P[i] = B[j];  break;}    cout<<"\nGiai ma: "<<P;  } |

|  |  |
| --- | --- |
| a) Mã hóa tay  -Chọn một từ khóa bất kỳ rồi xây dựng ma trận khóa Playfair  -Chọn một plaintext bất kỳ, áp dụng ma trận khóa trên để tạo ra ciphertext  -Thử giải mã ciphertext rồi so sánh với plaintext ban đầu | VD: DAIHOCTHUYLOI  Khoa K: CONGNGHE |

Ma trận khóa K:

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

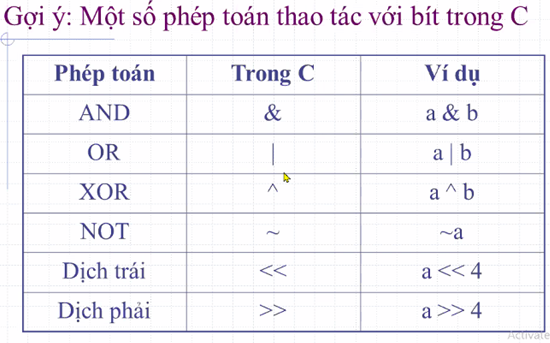
P = DAIHOCTHUYLOI

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | DA | IH | OC | TH | UY | LO | IX |
| C | EB | MO | GA | YD | VU | IG | LV |

|  |  |
| --- | --- |
| b) Code  - Chọn một từ khóa bất kỳ rồi xây dựng ma trận khóa Playfair  - Chọn một plaintext bất kỳ, áp dụng ma trận khóa trên để tạo ra ciphertext  - Thử giải mã ciphertext rồi so sánh với plaintext ban đầu | int main(){    string P;  cin>>P;    //  Buoc 1 + 2    string B = "ABCDEFGHIKLMNOPQRSTUVWXYZ";    P = P + B;    for(int i = 0; i< P.length(); i++){      for(int j = i+1; j< P.length(); j++){        if(P[j] == P[i]){ P.erase(j,1);   j--;}      }    }    //  buoc 3    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[j+ 5\*i];/        matran[i][j] = a[dem++];      }    }    //  In ma tran    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;      }    return 0;  } |
| Bài 7: Mật mã Hill  - Với plaintext là “paymoremoney” và sử dụng với mật khóa K nói trên, hãy xác định ciphertext.  - Giải mã ciphertext thu đươc bằng cách nhân nó với K-1, rồi so sánh kết quả với plaintext ban đầu. | Trong giáo trình – Chương 2 https://lh4.googleusercontent.com/8yFMwePBt0fCNSEhp_wgQSALFxYpucm2T_DbSgksh0MlXxmHfGcNkR257meggZM1yuDzDvwUSt-sSvYXzNwnYTlgkKXD79J0y9daELLbQotXQ5o9AFQCjtVcv-2JQLmcgwmxXN4I=s0 |
| a) Mã hóa tay  - Chọn một plaintext và một khóa K bất kỳ  - Áp dụng thuật toán Vigenere để xác định cirphertext | VD: Trong giáo trinh – Chương 2 |

|  |  |
| --- | --- |
| Bài 8: Mã hóa Vigenere  b) Code  - Lập trình nhập chuỗi ký tự plaintext từ bàn phím, mã hóa chuỗi bằng thuật toán Vigenere với khóa K (là một chuỗi ký tự) nhập từ bàn phím. Hiện chuỗi mới ra màn hình.  - Lập trình giải mã để khôi phục lại chuỗi ban đầu. | #include <iostream>  #include <string>  using namespace std;  string P, K;  int KT\_So(char *c*){  return c-'a';}  char So\_KT(int *n*){  return 'a'+n;}  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;    //  Thuat toan Vigennere tong quat    cout<<"Nhap chuoi plaintext: "; cin>>P;    cout<<"Nhap khoa K="; cin>>K;    check\_k();    cout<<K;  //  Ma\_hoa(P,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;    //  Giai ma    P = "";    for(int i = 0; i <C.size(); i++){      int m = (KT\_So(C[i]) - KT\_So(K[i]) + 26)%26;      P += So\_KT(m);    }    cout<<"\nChuoi giai ma: "<<P;  } |

|  |  |
| --- | --- |
| Bài 9: Vernman  a) Mã hóa tay  -Giải sử plaintext là “HA NOI” (các ký tự trong chuỗi có thể được biểu diễn dưới dạng nhị phân bảng mã chuẩn ASCII), khóa K là một dãy nhị phân 8 bit như sau:  K = 10010011  - Hãy mã hóa chuỗi ban đầu bằng phương pháp Vernman  - Giải mã chuỗi thu được rồi so sánh với chuỗi ban đầu  b) Code  - Nhập một chuỗi plaintext từ bàn phím  - Nhập một khóa K(dài 8 bit)  - Mã hóa chuỗi ban đầu bằng cách XOR các ký tự của nó với K  - Giải mã ciphertext thu được, rồi so sánh với chuỗi ban đầu. | int main()  {    string P, C, K;    cout<<"<----Thuat toan Vermman ---->\n";    cout<<"\nPlaintext: "; getline(cin,P);  //  P ="HA NOI";    C = P;    cout<<"Key: ";getline(cin,K);    //Chuyen chu sang bang ma ASCII    for(int i= 0; i<P.size();i++){      int j;      j = i%(K.size());      C[i]=P[i]^K[j];    }    //Ma hoa    cout<<"Chuoi ma hoa: "<<C;    //Giai ma    for(int i= 0; i<C.size();i++){      int j;      j = i%(K.size());      P[i]=C[i]^K[j];    }    cout<<"\nGiai ma: "<<P;  } |

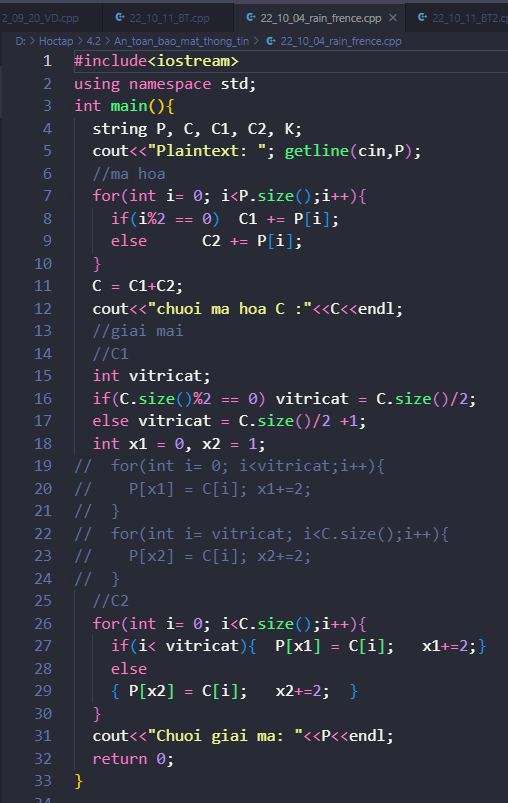


Bài 10: Mã hóa Rain-fence

- Nhập một chuỗi plaintext

- Mã hóa chuỗi bằng kỹ thuật rain-fence, hiện ciphertext ra màn hình

- Giải mã ciphertext



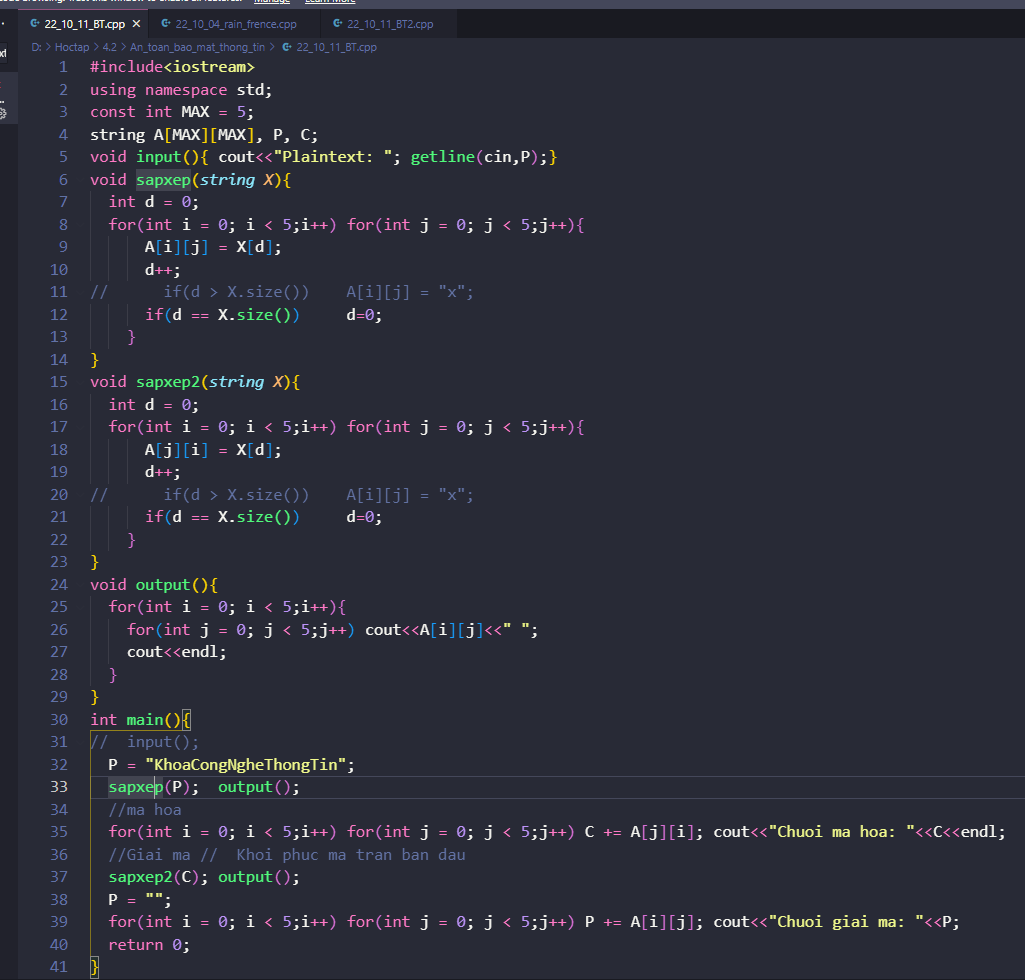
Bài 11: Code đổi hàng cột

- Nhập một chuỗi plaintext(dài không quá 25 ký tự)

- Sắp xếp các ký tự của chuỗi vào một ma trận 5x5(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.

Bài 12: Mật mã Feistel

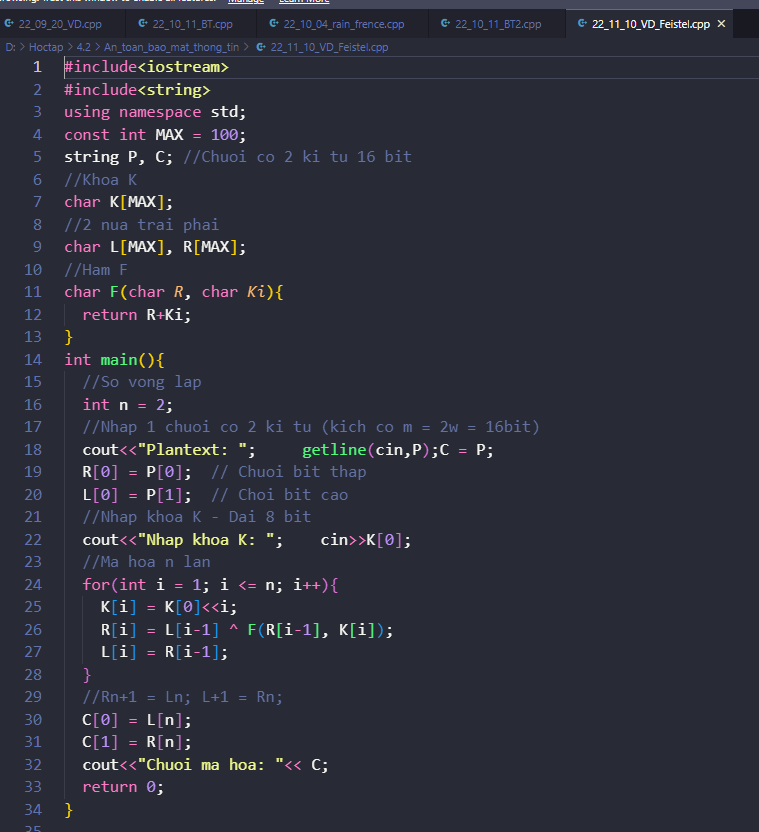
Lập trình mô phỏng hoạt động của mật mã Feistel đơn giản với 2 vòng xử lý:

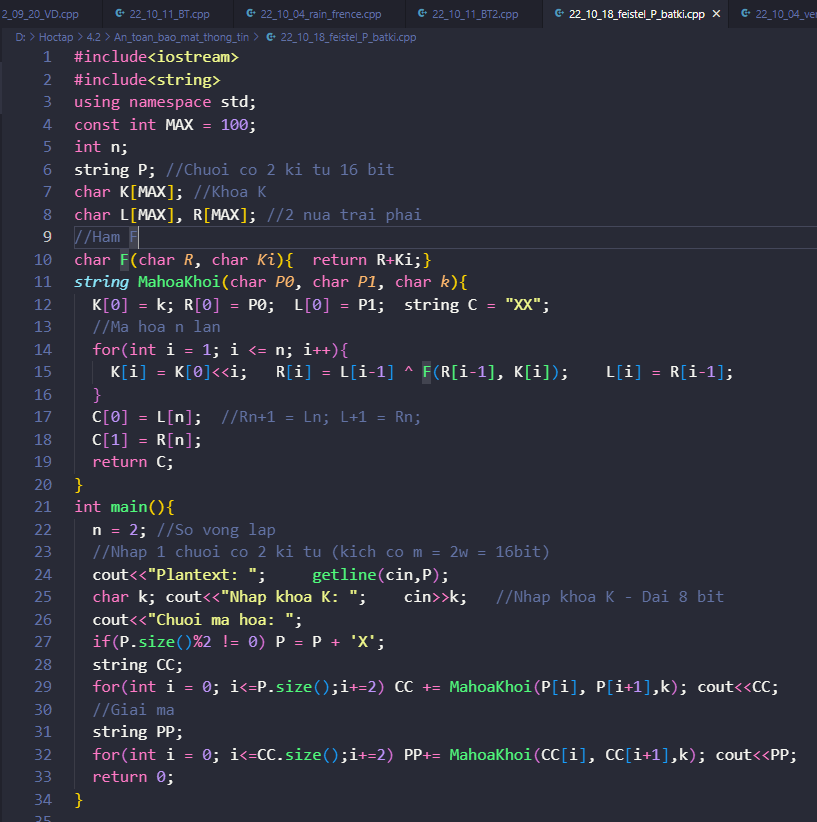
- Nhập một khối plaintext từ bàn phím, khối có độ dài m = 2w = 16 bít.

- 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.

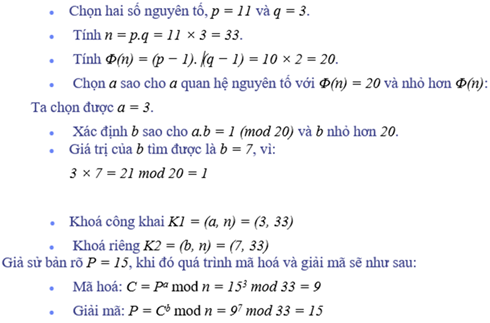




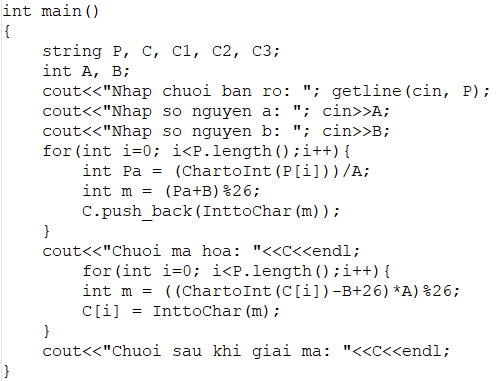
Bài 13: RSA

- Chọn một cặp số nguyên tố p, q

- Thực hiện các bước tạo khóa, mã hóa và giải mã theo thuật toán RSA



Ôn



Câu 1: Mật mã Vigenere

Cho chuỗi Plaintext sau : KHONGBENHLATIENSUONGTUYETTRAN

a) Hãy mã hóa chuỗi nói trên bằng mật mã Vigenere, với khóa K là họ tên của sinh viên, m là độ dài họ tên

b) Giải mã ciphertext thu được và so sánh plaintext ban đầu.

**Mai:**

A.Mã hóa

-Khóa K: HOANGTHIMAI => m = 11

-Plaintext: KHONGBENHLATIENSUONG TUYETTRAN

-Do m = 11, ta sẽ tách thành plaintext thành từng nhóm 11 ký tự:

KHONGBENHLA/TIENSUONGTU/YETTRAN

-Viết theo dạng số là:

10 7 14 13 6 1 4 13 7 11 0/19 8 4 13 18 20 14 13 6 19 20/24 4 19 19 17 0 13

-Từ khóa HOANGTHIMAI tương ứng với:

K = (7, 14, 0, 13, 6, 19, 7, 8, 12, 0, 8)

-Cộng từng nhóm 11 ký tự của plaintext với K ta có:

10   7   14 13 6  1   4  13  7 11 0/19 8   4   13 18 20 14 13   6  19 20/24  4   19 19 17 0   13

7    14   0  13 6  19 7   8  12  0  8/7 14  0  13  6  19   7   8   12   0   8/  7  14  0  13  6  19   7

17  21  14  0 12 20 11 21 19 11 8/0 22 4   0   24 13  21 21 18 19  2/   5   18  19 6  23  19 20

-Ciphertext tương ứng là:

RVOAMULVTLIAWEAYNVVSTCFSTGXTU

Giải mã

-Khóa K: HOANGTHIMAI => m =11

-Ciphertext: RVOAMULVTLIAWEAYNVVSTCFSTGXTU

-Do m = 11, ta sẽ tách Ciphertext thành từng nhóm 11 ký tự:

RVOAMULVTLI/AWEAYNVVSTC/FSTGXTU

-Viết theo dạng số là:

17 21 14 0 12 20 11 21 19 11 8/0 22 4 0 24 13 21 21 18 19 2/5 18 19 6 23 19 20

- Trừ từng nhóm 16 ký tự của Ciphertext với *K* ta có:

17   21 14  0  12 20 11 21 19 11 8/0 22  4   0  24 13  21 21  18 19   2/   5  18  19   6   23 19  20

7    14   0  13 6  19  7   8  12  0  8/7 14  0  13  6  19   7   8   12   0   8/   7  14    0    13   6  19   7

10   7   14 13 6  1   4   13  7  11 0/19  8  4  13 18 20 14 13   6   19  20/ 24  4  19    19   17  0   13

             - Plaintext : KHONGBENHLATIENSUONGTUYETTRAN

**Câu 1: Mật mã Verman (Giải mã bằng tay)**

-Giải sử plaintext là “TIEN VE HA NOI 1945” (các ký tự trong chuỗi có thể được biểu diễn dưới dạng nhị phân bảng mã chuẩn ASCII), khóa K là Họ và Tên sinh viên dưới dạng mã nhị phân. Hãy mã hóa chuỗi ban đầu bằng phương pháp Vernman

-Giải mã chuỗi thu được rồi so sánh với chuỗi ban đầu.

Khóa K = HOANG THI MAI = 48 4F 41 4E 47 20 54 48 49 20 4D 41 49

=   0100 1000  0100 1111 0100 0001 0100 1110 0100 0111 0010 0000 0101 0100 0100 1000 0100 1001 0010 0000 0100 1101 0100 0001 0100 1001

“TIEN VE HA NOI 1945” =54 49 45 4E 20 56 45 20 48 41 20 4E 4F 49 20 31 39 34 35

=0101 0100 0100 1001 0100 0101 0100 1110 0010 0000 0101 0110 0100 0101 0010 0000 0100 1000 0100 0001 0010 0000 0100 1110 0100 1111 0100 1001 0011 0001 0011 1001 0011 0100 0011 0101

-Mã hóa:

Plaintext = 0101 0100 0100 1001 0100 0101 0100 1110 0010 0000 0101 0110 0100 0101

0010 0000 0100 1000 0100 0001 0010 0000 0100 1110 0100 1111 0100 1001 0010 0000

0011 0001 0011 1001 0011 0100 0011 0101

Khóa K   = 0100 1000  0100 1111 0100 0001 0100 1110 0100 0111 0010 0000 0101 0100

0100 1000 0100 1001 0010 0000 0100 1101 0100 0001 0100 1001

Ciphertext=0001 1100 0000  0110 0000 0100 0000 0000 0110 0111 0111 0110 0001 0001

0110 1000 0000 0001 0110 0001 0110 1101 0000 1111 0000 0110 0000 0001  0110 1111 0111  0000 0111  0111 0111 0011 0001 0101

-Giải mã:

Ciphertext=0001 1100 0000  0110 0000 0100 0000 0000 0110 0111 0111 0110 0001 0001

0110 1000 0000 0001 0110 0001 0110 1101 0000 1111 0000 0110 0000 0001  0110 1111 0111  0000 0111  0111 0111 0011 0001 0101

Khóa K    = 0100 1000  0100 1111 0100 0001 0100 1110 0100 0111 0010 0000 0101 0100

0100 1000 0100 1001 0010 0000 0100 1101 0100 0001 0100 1001

Plaintext =  0101 0100 0100 1001 0100 0101 0100 1110 0010 0000 0101 0110 0100 0101

0010 0000 0100 1000 0100 0001 0010 0000 0100 1110 0100 1111 0100 1001 0010 0000 0011 0001 0011 1001 0011 0100 0011 0101

XOR

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Câu 1: Mật mã Affine**

Mã hóa bằng tay

Cho chuỗi plaintext sau:

KHOACONGNGHETHONGTIN

-Hãy mã hóa chuỗi nói trên bằng mật mã Affine, với khóa K là cặp số {a,b} lấy từ 2 số cuối của mã sinh viên

* a là số hàng chục(nếu a chẵn thì cộng thêm 1)
* b là số hàng đơn vị

-Giải mã ciphertext thu được và so sánh với plaintext ban đầu.

**Mai**

-K = {a,b} = {3, 9},  a-1 = 9

-P = KHOACONGNGHETHONGTIN

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pi | K | H | O | A | C | O | N | G | N | G | H | E | T | H | O | N | G | T | I | N |
|  | 10 | 7 | 14 | 0 | 2 | 14 | 13 | 6 | 13 | 6 | 7 | 4 | 19 | 7 | 14 | 13 | 6 | 19 | 8 | 13 |

-Mã hóa theo công thức:

Ci = E(Pi, {a,b}) = (aPi + b) mod 26 = (3Pi + 9) mod 26

=>Áp dụng công thức trên ta có:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ci | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C18 | C19 | C20 |
|  | 13 | 4 | 25 | 9 | 15 | 25 | 22 | 1 | 22 | 1 | 4 | 21 | 14 | 4 | 25 | 22 | 1 | 14 | 7 | 22 |
|  | N | E | Z | J | P | Z | W | B | W | B | E | V | O | E | Z | W | B | O | H | W |

-Giải mã theo công thức:

*Pi* = *D*(*Ci, {a,b}*) = *a-1*(*Ci* – *b*) mod 26 = 9(*Ci* – *9*) mod 26

=>Áp dụng công thức trên ta có:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ci | N | E | Z | J | P | Z | W | B | W | B | E | V | O | E | Z | W | B | O | H | W |
| Pi | 10 | 7 | 14 | 0 | 2 | 14 | 13 | 6 | 13 | 6 | 7 | 4 | 19 | 7 | 14 | 13 | 6 | 19 | 8 | 13 |
|  | K | H | O | A | C | O | N | G | N | G | H | E | T | H | O | N | G | T | I | N |

=>KL: Giải mã được kết quả trùng với Plaintext ban đầu

**Bài 4: Tính mã Hash**

Nhập một chuỗi ký tự từ bàn phím, chia chuỗi thành từng khối 64 bit (ứng với 8 ký 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 ký tự quy ước.)

#include <iostream> //Ma hash 2

using namespace std;

int main()

{

string M; char H[]={0,0,0,0,0,0,0,0};

cout<<"Nhap chuoi: ";getline(cin, M);

while (M.size()%8 !=0) M=M+'X';

for(int i = 0; i<M.size()/8; i++)

for(int j = 0; j<8; j++)

H[j] = H[j]^M[i\*8+j];

cout<<"Ma hash: ";

for(int i = 0; i<8; i++) cout<<H[i];

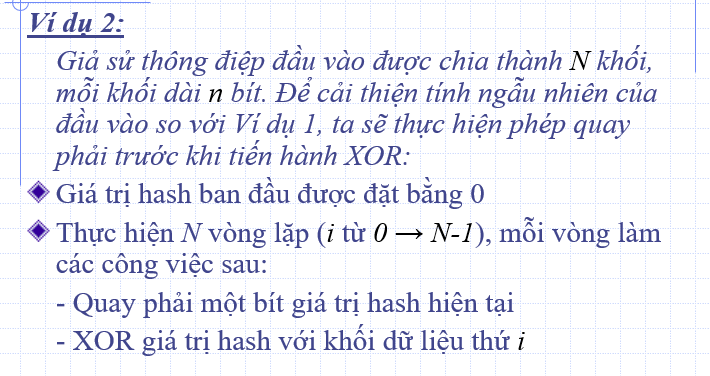
}

Quay phải tức là dịch bit phải

**Phép dịch phải là đẩy ra khỏi dãy bit**

**Phép quay phải chính là phép dịch phải**

**Bit tận cùng là bit 1 thì chúng ta phải đưa bit ngoài cùng**

****

**Ví dụ phép quay phải 1 dãy bit**

#include <iostream> //Ma hash 3

using namespace std;

int main()

{

string M; char H=0; char n;

cout<<"Nhap chuoi: ";getline(cin, M);

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

{

if(H % 2 == 1) n=128; else n = 0;

H=H<<1; H=H+n;

H=H^M[i];

}

cout<<"Ma hash: "<<H;

}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **0** | **1** | **1** | **1** | **1** | **0** | **1** |

**1 0 1 1 1 1 0**

**512 chia làm 8 khúc mỗi khúc 8 bit**

**Trọng tài quyền lực lớn quá sẽ biết hết mọi thứ (biết hết mật mã )có thể làm giả mã hasd và làm giả thông điêp.**

**Giải pháp thay thế bằng mật mã bất đối xứng**

**M -> H(M): Tác dụng xác thực nội dung, nhưng nếu hacker chặn trên đường truyền sửa thông điệp M, sửa luôn H(M) -> 000 không thể tránh hacker được, giải pháp kết hợp mã hóa**

***Lập trình nhập 1 chuỗi kí tự plaintext từ bàn phím. Nhập khóa K (dài 16 bit) từ bàn phím. Chia chuỗi thành từng khối 16 bit(nếu độ dài chuỗi không phải là bội số của 16 bit thì có thể chèn thêm các kí tự quy ước). Mã hóa chuỗi ban đầu bằng cách XOR từng khối của khối với K. Hiện chuỗi sau khi mã hóa ra màn hình. Giai mã và hiện chuỗi sau khi giải mã ra màn hình***

**#include <iostream>**

**#include <string>**

**#include <bitset>**

**const int BLOCK\_SIZE = 16;**

**// Chuyển ký tự sang dạng bitset**

**std::bitset<8> char\_to\_bits(char c) {**

**return std::bitset<8>(c);**

**}**

**// Chuyển dạng bitset sang ký tự**

**char bits\_to\_char(const std::bitset<8>& bits) {**

**return static\_cast<char>(bits.to\_ulong());**

**}**

**// Mã hóa chuỗi bằng cách XOR từng khối với khóa**

**std::string encrypt(const std::string& plaintext, const std::bitset<16>& key) {**

**std::string ciphertext = "";**

**// Chèn kí tự '\0' để làm cho độ dài của chuỗi trở thành bội số của 16**

**std::string padded\_plaintext = plaintext;**

**while (padded\_plaintext.length() % BLOCK\_SIZE != 0) {**

**padded\_plaintext += '\0';**

**}**

**for (size\_t i = 0; i < padded\_plaintext.length(); i += BLOCK\_SIZE) {**

**std::bitset<16> block\_bits(0);**

**// XOR từng ký tự trong khối với khóa**

**for (size\_t j = 0; j < BLOCK\_SIZE; ++j) {**

**block\_bits <<= 8;**

**block\_bits |= char\_to\_bits(padded\_plaintext[i + j]);**

**}**

**block\_bits ^= key;**

**// Chuyển dạng bitset thành chuỗi ký tự**

**for (size\_t j = 0; j < BLOCK\_SIZE; ++j) {**

**ciphertext += bits\_to\_char(block\_bits >> (8 \* (BLOCK\_SIZE - j - 1)));**

**}**

**}**

**return ciphertext;**

**}**

**// Giải mã chuỗi bằng cách XOR từng khối với khóa**

**std::string decrypt(const std::string& ciphertext, const std::bitset<16>& key) {**

**std::string decrypted\_text = "";**

**for (size\_t i = 0; i < ciphertext.length(); i += BLOCK\_SIZE) {**

**std::bitset<16> block\_bits(0);**

**// XOR từng ký tự trong khối với khóa**

**for (size\_t j = 0; j < BLOCK\_SIZE; ++j) {**

**block\_bits <<= 8;**

**block\_bits |= char\_to\_bits(ciphertext[i + j]);**

**}**

**block\_bits ^= key;**

**// Chuyển dạng bitset thành chuỗi ký tự**

**for (size\_t j = 0; j < BLOCK\_SIZE; ++j) {**

**decrypted\_text += bits\_to\_char(block\_bits >> (8 \* (BLOCK\_SIZE - j - 1)));**

**}**

**}**

**return decrypted\_text;**

**}**

**int main() {**

**std::string plaintext;**

**std::bitset<16> key;**

**std::cout << "Nhập chuỗi plaintext: ";**

**std::getline(std::cin, plaintext);**

**std::cout << "Nhập khóa K (dài 16 bit): ";**

**std::cin >> key;**

**// Mã hóa**

**std::string encrypted\_text = encrypt(plaintext, key);**

**std::cout << "Chuỗi sau khi mã hóa: " << encrypted\_text << std::endl;**

**// Giải mã**

**std::string decrypted\_text = decrypt(encrypted\_text, key);**

**std::cout << "Chuỗi sau khi giải mã: " << decrypted\_text << std::endl;**

**return 0;**

**}**

***Lập trình nhập 1 chuỗi kí tự plaintext từ bàn phím (các kí tự nằm trong khoảng từ 'A' đến 'Z'). Nhập 2 số nguyên a,b thuộc Z26 từ bàn phím. Mã hóa chuỗi ban đầu theo công thức C=(P/a) + b***

***Trong đó P là kí tự 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***

***Giai mã và hiện chuỗi sau khi giải mã ra màn hình***

#include <iostream>

#include <string>

const int Z26\_SIZE = 26;

// Chuyển ký tự sang số trong Z26 ('A' -> 0, 'B' -> 1, ..., 'Z' -> 25)

int char\_to\_int(char c) {

return c - 'A';

}

// Chuyển số trong Z26 sang ký tự ('0' -> 'A', '1' -> 'B', ..., '25' -> 'Z')

char int\_to\_char(int num) {

return 'A' + (num % Z26\_SIZE);

}

// Tìm nghịch đảo modulo Z26 của a (a^(-1))

int modular\_inverse(int a) {

for (int i = 1; i < Z26\_SIZE; ++i) {

if ((a \* i) % Z26\_SIZE == 1) {

return i;

}

}

return -1; // Không tìm thấy nghịch đảo

}

// Mã hóa chuỗi theo công thức C = (P/a) + b mod Z26

std::string encrypt(const std::string& plaintext, int a, int b) {

int a\_inverse = modular\_inverse(a);

if (a\_inverse == -1) {

std::cerr << "Không tìm thấy nghịch đảo modulo Z26 cho a." << std::endl;

return "";

}

std::string ciphertext = "";

for (char P : plaintext) {

int P\_value = char\_to\_int(P);

int C\_value = ((P\_value \* a\_inverse) + b) % Z26\_SIZE;

ciphertext += int\_to\_char(C\_value);

}

return ciphertext;

}

// Giải mã chuỗi theo công thức P = (C - b)\*a^(-1) mod Z26

std::string decrypt(const std::string& ciphertext, int a, int b) {

int a\_inverse = modular\_inverse(a);

if (a\_inverse == -1) {

std::cerr << "Không tìm thấy nghịch đảo modulo Z26 cho a." << std::endl;

return "";

}

std::string plaintext = "";

for (char C : ciphertext) {

int C\_value = char\_to\_int(C);

int P\_value = ((C\_value - b + Z26\_SIZE) \* a\_inverse) % Z26\_SIZE;

plaintext += int\_to\_char(P\_value);

}

return plaintext;

}

int main() {

std::string plaintext;

int a, b;

std::cout << "Nhập chuỗi plaintext (chỉ chấp nhận ký tự từ 'A' đến 'Z'): ";

std::cin >> plaintext;

std::cout << "Nhập số nguyên a thuộc Z26: ";

std::cin >> a;

std::cout << "Nhập số nguyên b thuộc Z26: ";

std::cin >> b;

// Mã hóa

std::string encrypted\_text = encrypt(plaintext, a, b);

std::cout << "Chuỗi sau khi mã hóa: " << encrypted\_text << std::endl;

// Giải mã

std::string decrypted\_text = decrypt(encrypted\_text, a, b);

std::cout << "Chuỗi sau khi giải mã: " << decrypted\_text << std::endl;

return 0;

}

***lập trình nhập 1 chuỗi kí tự từ plaintext từ bàn phím (các kí tự nằm trong khoảng từ 'A' đến 'Z' ). Nhập số nguyên k thuộc Z26 từ bàn phím. Mã hóa chuỗi ban đầu theo công thức C = 7\*P + k. trong đó P là kí tự bản rõ, C là kí tự mã hóa, các phép tín được thực hiện trong Z26. Hiện chuỗi sau khi mã hóa ra màn hình. Giải mã và hiện chuỗi sau khi giải mã ra màn hình***

#include <iostream>

#include <string>

const int Z26\_SIZE = 26;

// Chuyển ký tự sang số trong Z26 ('A' -> 0, 'B' -> 1, ..., 'Z' -> 25)

int char\_to\_int(char c) {

return c - 'A';

}

// Chuyển số trong Z26 sang ký tự ('0' -> 'A', '1' -> 'B', ..., '25' -> 'Z')

char int\_to\_char(int num) {

return 'A' + (num % Z26\_SIZE);

}

// Mã hóa chuỗi theo công thức C = (7 \* P + k) mod Z26

std::string encrypt(const std::string& plaintext, int k) {

std::string ciphertext = "";

for (char P : plaintext) {

int P\_value = char\_to\_int(P);

int C\_value = (7 \* P\_value + k) % Z26\_SIZE;

ciphertext += int\_to\_char(C\_value);

}

return ciphertext;

}

// Giải mã chuỗi theo công thức P = (C - k) mod Z26

std::string decrypt(const std::string& ciphertext, int k) {

std::string plaintext = "";

for (char C : ciphertext) {

int C\_value = char\_to\_int(C);

int P\_value = (C\_value - k + Z26\_SIZE) % Z26\_SIZE;

plaintext += int\_to\_char(P\_value);

}

return plaintext;

}

int main() {

std::string plaintext;

int k;

std::cout << "Nhập chuỗi plaintext (chỉ chấp nhận ký tự từ 'A' đến 'Z'): ";

std::cin >> plaintext;

std::cout << "Nhập số nguyên k thuộc Z26: ";

std::cin >> k;

// Mã hóa

std::string encrypted\_text = encrypt(plaintext, k);

std::cout << "Chuỗi sau khi mã hóa: " << encrypted\_text << std::endl;

// Giải mã

std::string decrypted\_text = decrypt(encrypted\_text, k);

std::cout << "Chuỗi sau khi giải mã: " << decrypted\_text << std::endl;

return 0;

}