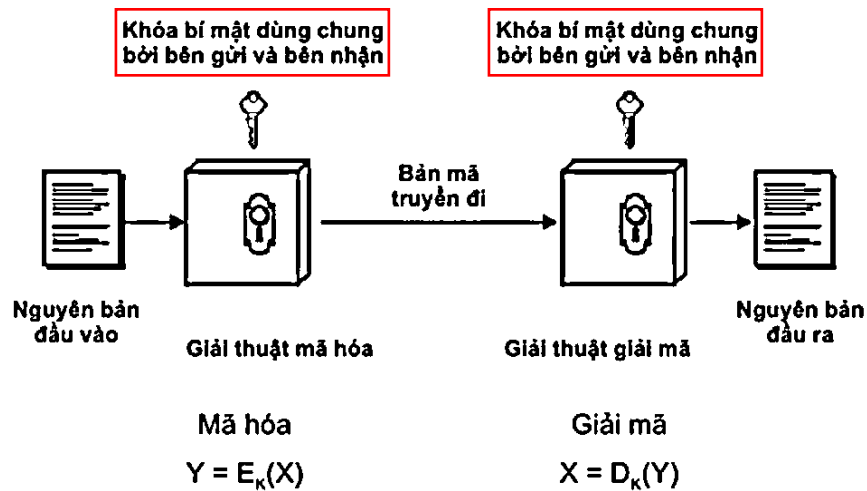


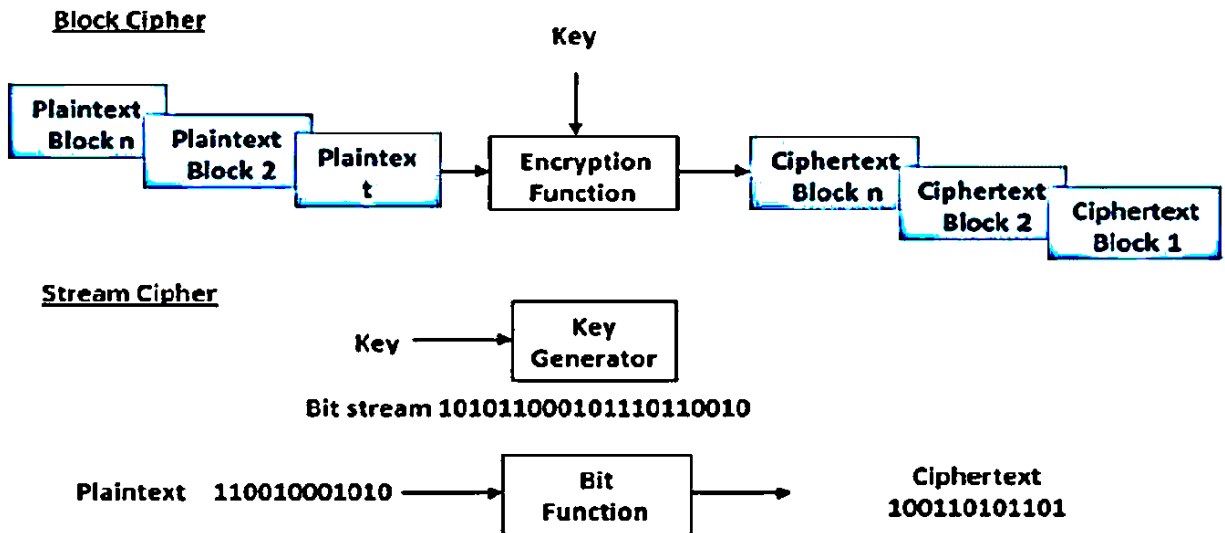
MÃ HOÁ HIỆN ĐẠI – MÃ HÓA ĐỐI XỨNG (P.1)

- Mã hóa khóa đối xứng là một loại sơ đồ hệ thống mã hóa trong đó **một khóa giống nhau** sẽ được dùng cho cả mã hóa và giải mã dữ liệu.

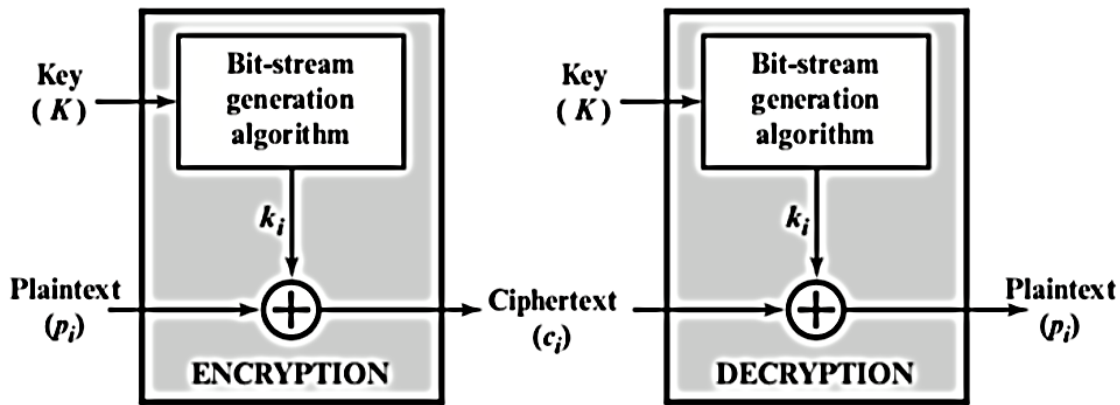


- Phân loại mã hóa đối xứng:**
 - Dựa trên cách các chuỗi nhị phân được xử lý, một sơ đồ mã hóa đối xứng được phân loại thành 2 loại chính:
 - Mã hóa Khối (Block Ciphers)**
 - Bên người gửi:**
 - Xử lý bản rõ theo **từng khối bit có kích thước cố định** (ví dụ: 64 bit, 128 bit, tùy thuộc vào thuật toán).
 - Nếu bản rõ không đủ độ dài để lấp đầy khối, nó sẽ được đệm (padding)** để đạt bội số của kích thước khối.
 - Một khối bit bản rõ được chọn, sau đó **thực hiện các thao tác (thay thế, hoán vị, v.v.) để tạo ra một khối bit bản mã**.
 - Các khối bản mã sau khi mã hóa được ghép lại thành một bản mã hoàn chỉnh.
 - Bên người nhận:**
 - Bản mã được chia lại thành các khối** có kích thước bằng nhau (tương ứng với kích thước khối ban đầu).
 - Từng khối được giải mã bằng khóa và chế độ hoạt động tương ứng.**
 - Các phần đã giải mã được ghép lại, loại bỏ đệm (nếu có), để tái tạo thông điệp gốc (bản rõ).
 - Số bit trong một khối là cố định.**
 - Ví dụ phổ biến: AES, DES.
 - Mã hóa Dòng (Stream Ciphers)**
 - Bên người gửi:**
 - Xử lý bản rõ **từng bit hoặc từng byte một** (liên tục như một dòng dữ liệu).

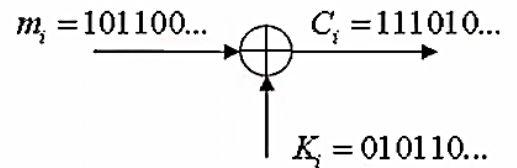
- Thuật toán tạo ra một **keystream** (chuỗi khóa ngẫu nhiên) có độ dài bằng thông điệp, sau đó kết hợp từng bit/byte của bản rõ với keystream (thường bằng phép XOR) để tạo bản mã.
- **Bên người nhận:**
 - Bản mã được giải mã bằng cách tạo lại keystream (dùng cùng khóa bí mật) và kết hợp nó với bản mã (thường bằng phép XOR) để khôi phục bản rõ.
 - Các phần giải mã được ghép lại một cách tự nhiên vì quá trình diễn ra liên tục, không cần chia thành các khối riêng lẻ.
- Về mặt kỹ thuật, có thể xem mã hóa Dòng như một trường hợp đặc biệt của mã hóa Khối với kích thước khối là 1 bit.



I. MÃ HÓA DÒNG

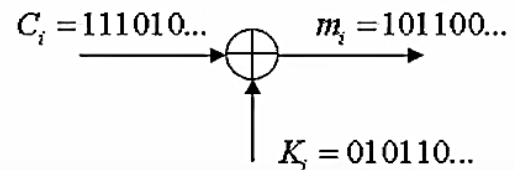


- mã hóa thông tin từng bit một cho mỗi chu kỳ mã hóa.
- trong số các hoạt động có bit chỉ có hai phép đảo ngược là tổng theo modulo 2 và XOR.



- **Mã hóa:** $c_i = m_i \oplus k_i$ hay $c_i = (m_i + k_i) \bmod 2$

- m_i là bit thứ i của bản rõ.
- k_i là bit thứ i của chuỗi khóa.
- c_i là bit thứ i của bản mã.
- \oplus là phép XOR (modulo 2).



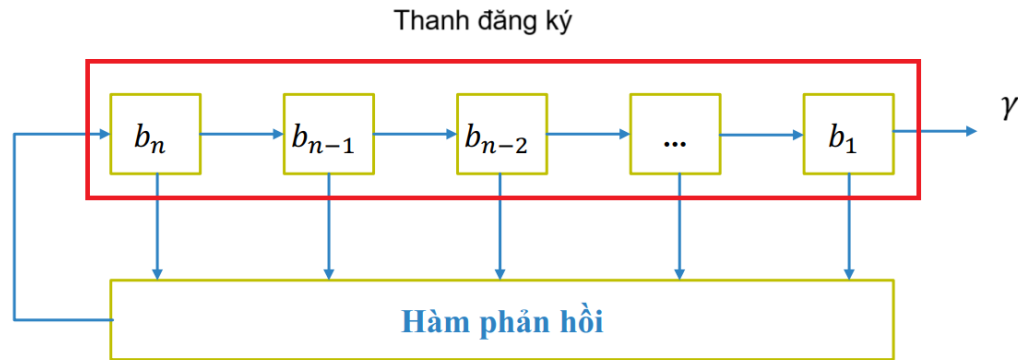
- Giải mã: $m_i = c_i \oplus k_i$ hay $m_i = (c_i + k_i) \bmod 2$

- Vì XOR là phép toán đảo ngược được với chính nó, áp dụng lại cùng k_i sẽ khôi phục m_i .

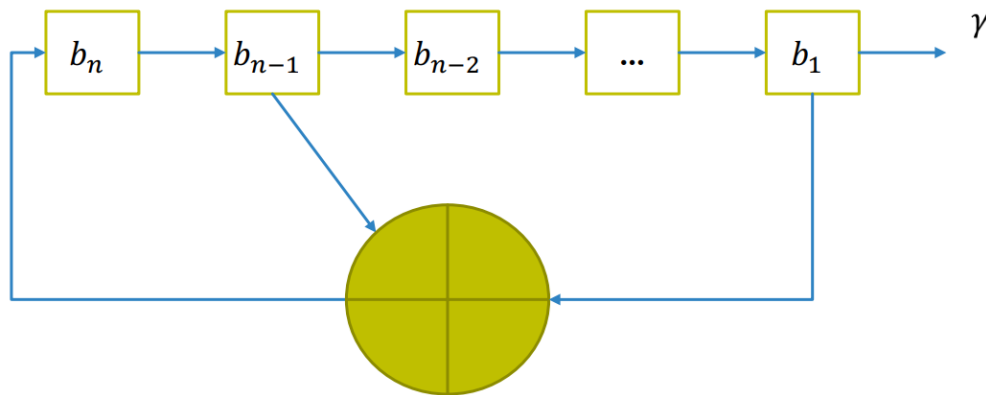
- Nếu biết c_i và m_i , ta có thể suy ra $k_i = c_i \oplus m_i$.

- **Dòng khóa, thường được ký hiệu là γ (gamma)** trong mật mã dòng, là yếu tố cốt lõi quyết định độ an toàn.
- **Nếu γ yếu** (ví dụ: chỉ toàn số 0), thì $c_i = m_i \oplus 0 = m_i$, **nghĩa là bản mã sẽ giống hệt bản rõ**, không có bảo mật.
- Do đó, chất lượng của trình tạo dòng khóa (keystream generator) là yếu tố then chốt. Một dòng khóa tốt cần:
 - **Ngẫu nhiên:** Không có mẫu hình nào có thể dự đoán.
 - **Dài đủ:** Ít nhất bằng độ dài của bản rõ.
 - **Không lặp lại:** Tránh tái sử dụng cùng một γ cho nhiều thông điệp (điều này làm lộ thông tin nếu kẻ tấn công có nhiều bản mã).

- Phương pháp tạo dòng khóa (keystream) γ trong mã hóa dòng: sử dụng **thanh ghi dịch chuyển phản hồi tuyến tính (LFSR - Linear Feedback Shift Register)**:
 - Thanh đăng ký dịch chuyển là một chuỗi các bit, được **dịch chuyển từ phải sang trái** trong mỗi chu kỳ mã hóa.

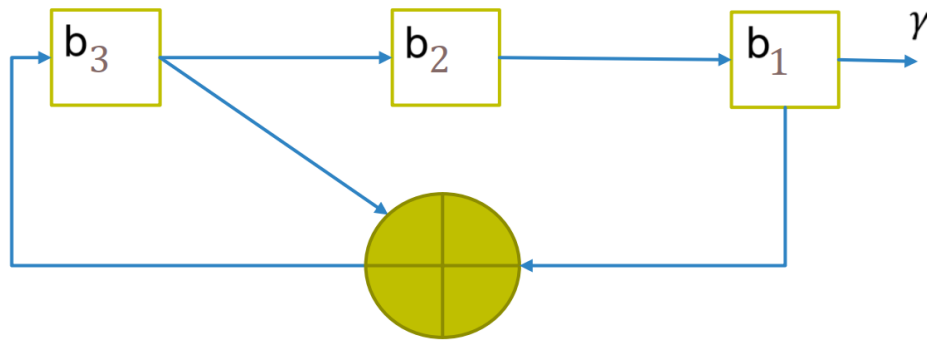


- Mỗi chu kỳ, **bit ngoài cùng bên phải (b_1)** được sử dụng làm đầu ra của dòng khóa γ , **đồng thời bit này cũng được đưa vào một hàm phản hồi (feedback function) để tính toán giá trị bit mới sẽ được thêm vào bên trái (b_n)**.
- $b_i = 0$ or 1 được gọi là các trạng thái của các bit
- **Khóa mã hóa ban đầu (initial key)** được sử dụng để khởi tạo các bit trong thanh đăng ký.
- **Hàm phản hồi** quyết định giá trị bit mới được thêm vào thanh đăng ký. Ở đây sử dụng **hàm phản hồi tuyến tính (dựa trên phép XOR) \rightarrow LSFR**.
- Đầu ra của thanh đăng ký (các bit γ) được sử dụng để mã hóa bản rõ bằng phép XOR.



- Đây là một chuỗi tuần hoàn, nghĩa là **sau một số chu kỳ nhất định, chuỗi bit trong thanh đăng ký sẽ lặp lại**.
- Để đảm bảo an toàn, chu kỳ này cần đủ dài để tránh lặp lại trong quá trình mã hóa một thông điệp.

○ Ví dụ:



- Giá trị ban đầu được nhập là 010 và giá trị đầu ra ở bảng dưới:

Số chu kỳ	Giá trị bit của thanh đăng ký			Bit gamma
	b_3	b_2	b_1	
Khởi đầu	0	1	0	-
1	$0 = b_3 \oplus b_1$	0 (dịch bit tại b_3 sang b_2)	1 (dịch bit tại b_2 sang b_1)	0 (dịch bit tại b_1 sang γ)
2	1	0	0	1
3	1	1	0	0
4	1	1	1	0
5	0	1	1	1
6	1	0	1	1
7	0	1	0	1
8	0	0	1	0

- Chuỗi γ từ bảng: 0, 1, 0, 0, 1, 1, 1.
- Bảng này cho thấy trạng thái của LFSR được lặp lại sau 7 chu kỳ (**trạng thái ban đầu trùng với trạng thái của nó ở chu kỳ thứ 7**).
- Lặp lại trạng thái của LFSR có nghĩa là gamma sẽ được lặp lại định kỳ.
- **Sự lặp lại gamma** làm **giảm sức mạnh bảo mật** của các mật mã hiện tại, cho phép nhà phân tích mật mã phân tích bản mã thu được bằng cách mã hóa trên cùng một gamma.
- Đối với **LFSR có độ dài n bit, chu kỳ tối đa là chu kỳ $2^n - 1$** (trạng thái khi tất cả các bit bằng không là không thể chấp nhận được).
 - Một LFSR với n bit có thể biểu diễn 2^n trạng thái khác nhau (từ 00...0 đến 11...1).
 - Tuy nhiên, trạng thái toàn 0 ($b_n, b_{n-1}, \dots, b_1 = 0$) sẽ dẫn đến chuỗi γ toàn 0, vì hàm phản hồi tuyến tính (dựa trên phép XOR) sẽ luôn cho ra 0 nếu tất cả các bit đầu vào là 0. Do đó, trạng thái này không được sử dụng trong chu kỳ chính.
 - Vì vậy, chu kỳ tối đa là $2^n - 1$, đạt được khi đa thức phản hồi là **nguyên thủy** (primitive polynomial). Một đa thức nguyên thủy đảm bảo rằng chuỗi trạng thái sẽ đi qua tất cả $2^n - 1$ trạng thái không chứa toàn 0 trước khi lặp lại.

- Cấu trúc của LFSR được mô tả bởi phương trình phản hồi tuyến tính (liên quan đến trường G_{2^n})

$$b_n \cdot x^n + b_{n-1} \cdot x^{n-1} + b_{n-2} \cdot x^{n-2} + \dots + b_2 \cdot x^2 + b_1 \cdot x + 1$$

- Nếu $b_i = 1$, bit b_i được lấy để XOR (được sử dụng trong hàm phản hồi).
- Nếu $b_i = 0$, bit b_i không được sử dụng.
- **LFSR không an toàn do tính tuyến tính của nó**, và có thể bị tấn công bằng thuật toán Berlekamp-Massey.
 - Đây là một thuật toán hiệu quả để xác định đa thức phản hồi của LFSR từ một chuỗi bit đầu ra γ .
 - Thuật toán này cần ít nhất $2n$ bit của chuỗi γ để xác định đa thức phản hồi có độ dài n .
 - Ví dụ: Với $n = 3$, cần ít nhất $2 \times 3 = 6$ bit của γ . Trong ví dụ trước, chuỗi $\gamma = 0, 1, 0, 0, 1, 1, 1$, chỉ cần 6 bit đầu $(0, 1, 0, 0, 1, 1)$ là đủ để xác định đa thức phản hồi $x^3 + x + 1$.
 - Sau khi xác định đa thức, kẻ tấn công có thể tái tạo toàn bộ chuỗi γ , phá vỡ tính bảo mật của mã hóa dòng.

II. MÃ HOÁ KHỐI:

1. Khái niệm và sơ lược phép mã hóa khối

- chuyển đổi mật mã trên các **khối văn bản** thuần túy có **kích thước cố định (32, 64, 128, 256 bit)**.
- Mật mã khối lấy một khối bit bản rõ và tạo ra một khối bit bản mã, thường có cùng kích thước. **Kích thước của khối được cố định trong thuật toán**. Việc **lựa chọn kích thước khối không ảnh hưởng trực tiếp đến độ mạnh của lược đồ mã hóa**.
- **Độ mạnh của mật mã phụ thuộc vào độ dài của khóa**.
- **Kích thước khối**: Trên lý thuyết bất kỳ kích thước nào của khối đều được chấp nhận, nhưng có các quy tắc được ghi nhớ trong khi chọn kích thước khối:
 - **Tránh kích thước khối quá nhỏ**: **Giả sử kích thước khối là n bit**, số lượng khối bản rõ có thể có là 2^n .
 - Nếu kích thước khối quá nhỏ, & kẻ tấn công phát hiện ra **các khối bản rõ tương ứng với một số khối bản mã đã được mã hóa bằng cùng một khóa** đã gửi trước đó, thì kẻ tấn công có thể khởi chạy một kiểu “tấn công từ điển” bằng cách **xây dựng một từ điển gồm các cặp bản rõ/mã được gửi bằng khóa mã hóa đó**.
 - Kích thước khối lớn hơn khiến cuộc tấn công trở nên khó khăn hơn vì từ điển cần phải nhiều hơn.
 - **Không nên để kích thước khối quá lớn**: Với kích thước khối quá lớn, mật mã sẽ trở nên hoạt động kém hiệu quả.
 - Nếu bản rõ không đủ để lấp đầy kích thước một khối, chúng cần được đệm trước khi được mã hóa. Ví dụ: Với kích thước khối 256 bit, một thông điệp ngắn (ví dụ: 10 byte) sẽ cần đệm thêm 22 byte, gây lãng phí tài nguyên.
 - Kích thước khối lớn cũng làm **tăng độ phức tạp tính toán** trong mỗi vòng mã hóa, dẫn đến **thời gian xử lý lâu hơn và tiêu tốn nhiều tài nguyên hơn**.
 - **Bội số của 8 bit**: Kích thước khối nên là bội số của 8 vì nó dễ thực hiện vì hầu hết bộ xử lý máy tính xử lý dữ liệu theo bội số của 8 bit, ví dụ: **64 bit (8 byte), 128 bit (16 byte), 256 bit (32 byte)**.
 - Nếu kích thước khối không phải là bội số của 8 bit (ví dụ: 100 bit), việc xử lý dữ liệu sẽ phức tạp hơn, đòi hỏi các **thao tác căn chỉnh (alignment)** và **có thể làm giảm hiệu suất**.
- Một mật mã khối có thể được biểu diễn dưới dạng mật mã thay thế trên một bảng chữ cái rất lớn, ví dụ, **có 2^{64} ký tự cho một khối 64 bit** (tương ứng với tất cả các giá trị có thể của một khối 64 bit).
- Tuy nhiên, **việc lưu trữ một bảng thay thế cho 2^{64} ký tự là không khả thi** (cần 2^{70} bit bộ nhớ cho một khóa, tức là khoảng 10^{21} byte, một con số khổng lồ). Do đó, sự tương ứng giữa đầu vào và đầu ra được mô tả bằng một thuật toán thay vì bảng tra cứu.

- **Thuật toán mã hóa khối:** Một **khối văn bản đơn thuần X** được **chuyển đổi** thành **một khối mật mã Y** có cùng kích thước **bằng cách sử dụng một số Khóa mã hóa**, có Khóa được thể hiện là:

$$Y = \text{Encrypt}(X, \text{Key})$$

- Quy trình **giải mã** thực hiện **chuyển đổi** ngược lại bằng cách sử dụng cùng một khóa:

$$X = \text{Decrypt}(Y, \text{Key})$$

- Các phép **phép chuyển đổi** này thường bao gồm:

- **Xáo trộn bit (Diffusion):**

- **Các bit trong khối bản rõ được "xáo trộn" với nhau** để đảm bảo rằng một thay đổi nhỏ trong bản rõ (ví dụ: 1 bit) sẽ dẫn đến thay đổi lớn trong bản mã (hiệu ứng "lan tỏa").
- Điều này làm tăng tính bảo mật, khiến kẻ tấn công khó phân tích mối quan hệ giữa bản rõ và bản mã.

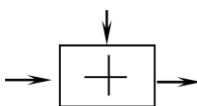
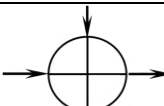
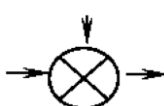
- **Kết hợp với khóa (Confusion):**

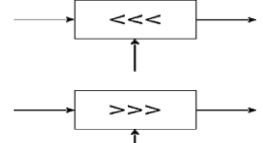
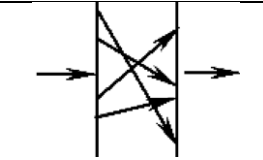
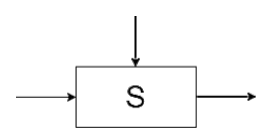
- **Các bit của khối bản rõ được kết hợp với các bit của khóa mã hóa Key**, thường thông qua các phép toán logic hoặc số học.
- Điều này đảm bảo rằng bản mã phụ thuộc mạnh vào khóa, và không thể giải mã nếu không biết khóa.

- **Xử lý dữ liệu dưới dạng số nguyên:**

- **Mỗi khối được diễn giải như một số nguyên trong phạm vi $[0, \dots, 2^k - 1]$** , với k là kích thước khối (ví dụ: k = 64 cho khối 64 bit).
- Các phép toán số học hoặc logic được áp dụng trên các số nguyên này để tạo ra bản mã.

- Để đảm bảo khả năng giải mã (tính đảo ngược), các phép biến đổi trong mã hóa khối phải sử dụng các phép toán có thể đảo ngược.
- Nếu một phép toán không đảo ngược được (ví dụ: mất thông tin trong quá trình biến đổi), thì không thể khôi phục bản rõ từ bản mã.
- Các phép toán đảo ngược được sử dụng thường xuyên nhất trong mật mã khối hiện đại được liệt kê:

Tên	Ký hiệu	Công thức chuyển đổi	Chuyển đổi ngược lại
Phép cộng (Tổng theo modulo 2)		$X' = X + V$ (có thể xem đây là phép XOR vì đều thực hiện trong modulo 2)	Phép trừ
Tổng theo modulo 2		$X' = X \oplus V$	Tự động trả về (Automatic return)
Nhân theo modulo 2^{N+1} (N là kích thước khối)		$X' = (X \cdot V) \bmod (2^{N+1})$ <i>Đây là phép nhân trong trường hữu hạn $Z/(2^{N+1})$</i>	Nhân tử chung có thể tìm được bằng thuật toán Euclid Tìm V^{-1} sao cho $(V \cdot V^{-1}) \bmod (2^{N+1}) = 1$, sau đó: $X = (X' \cdot V^{-1}) \bmod (2^{N+1})$

Dịch chuyển theo chu kỳ sang phải/trái		$X' = X \text{ ROR } V$ hoặc $X' = X \text{ ROL } V$	Dịch chuyển tuần hoàn theo hướng ngược lại
Hoán vị bit		Sắp xếp lại vị trí các bit trong khối theo một quy tắc cố định.	Hoán vị ngược lại (theo quy tắc ngược).
Bảng thay thế		$X' = \text{SBox}(X)$ S-box là một bảng tra cứu ánh xạ mỗi giá trị đầu vào sang một giá trị đầu ra duy nhất.	Thay thế ngược lại, sử dụng bảng S-box ngược.

• **Phép nhân và thay thế (S-box) đặc biệt quan trọng vì chúng mang lại tính phi tuyến:**

◦ **Tính phi tuyến:**

- Các phép toán tuyến tính (như XOR, dịch chuyển, hoán vị) dễ bị tấn công bằng phân tích tuyến tính, vì kẻ tấn công có thể xây dựng các phương trình tuyến tính để dự đoán đầu ra.
- Phép nhân (trong trường hữu hạn) và S-box phá vỡ tính tuyến tính, làm cho các phương trình trở nên phức tạp và khó giải.

◦ **Chống phân tích tuyến tính:**

- Phân tích tuyến tính (linear cryptanalysis) tìm kiếm các mối quan hệ tuyến tính giữa bản rõ, bản mã, và khóa.
- S-box được thiết kế để giảm thiểu xác suất của các mối quan hệ tuyến tính, làm cho các cuộc tấn công này kém hiệu quả.
- Ví dụ: S-box trong AES được xây dựng dựa trên phép nghịch đảo trong $GF(2^8)$, kết hợp với một phép biến đổi affine, để tối ưu hóa tính phi tuyến.

- Một trong những nguyên tắc chính của việc xây dựng cấu trúc của các thuật toán mã hoá hiện đại là **nguyên tắc lặp lại**. Ý tưởng của nó là lặp đi lặp lại, bao gồm nhiều chu kỳ (hoặc vòng), xử lý một khối văn bản thuần túy bằng cách sử dụng một khóa đặc biệt của vòng trên mỗi chu kỳ, khóa này được tính toán trên cơ sở khóa mã hóa.

- **Số lượng chu kỳ có thể thay đổi** vì lý do kháng mật mã và hiệu quả triển khai thuật toán: **tăng số chu kỳ dẫn đến tăng độ mạnh của mật mã, nhưng làm tăng thời gian mã hóa và tiêu tốn tài nguyên máy tính**. Các cấu trúc tuần hoàn như vậy thường được gọi là mạng lưới và hầu hết các mật mã khối hiện đại được xây dựng bằng cách sử dụng kiến trúc mạng lưới

- Các thuật toán mã hóa khối hiện đại thường được xây dựng dưới dạng **mạng lưới (network)**, chẳng hạn:

- **Mạng thay thế-hoán vị (Substitution-Permutation Network - SPN):** Sử dụng S-box (thay thế) và hoán vị để tạo tính phi tuyến và lan tỏa. Ví dụ: AES.

- **Mạng Feistel (Feistel Network):** Chia khối thành hai nửa, áp dụng hàm vòng (round function) lên một nửa và kết hợp với nửa còn lại. Ví dụ: DES.

○ **Ưu điểm của kiến trúc mạng lưới**

- **Tính lan tỏa (Diffusion):** Đảm bảo rằng **một thay đổi nhỏ trong bản rõ (1 bit) ảnh hưởng đến nhiều bit trong bản mã.**
- **Tính nhiễu (Confusion):** **Tạo mối quan hệ phức tạp giữa bản rõ, bản mã, và khóa,** thông qua các phép toán phi tuyến như S-box.
- **Tính lặp lại:** Nhiều vòng biến đổi làm **tăng độ phức tạp**, khiến việc phân tích trở nên khó khăn.

*** Sự khác biệt giữa Mật mã Khối và mật mã Dòng:**

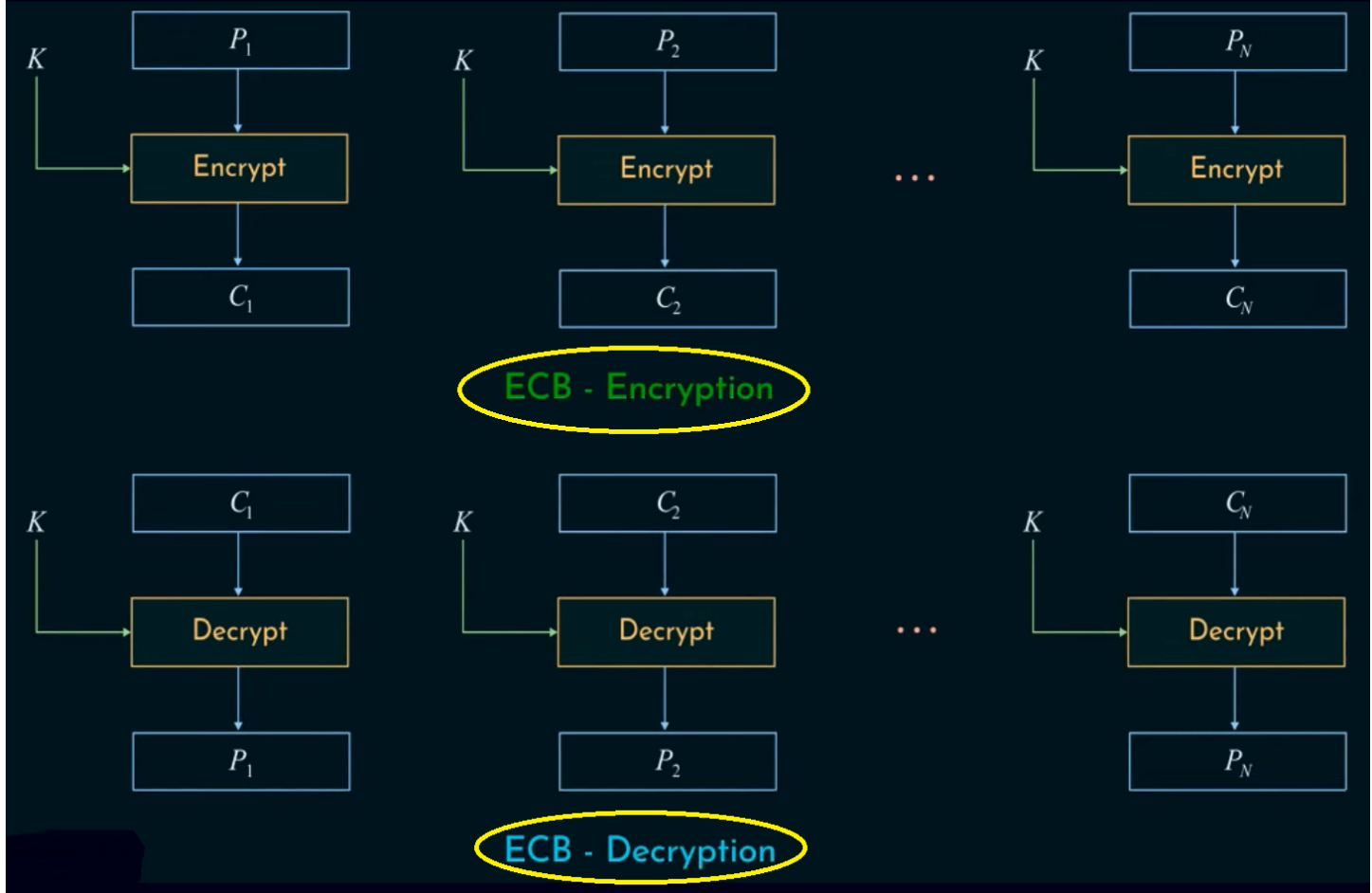
Mật mã Khối	Mật mã Dòng
Chuyển đổi bản rõ thành bản mã bằng cách lấy từng khối của bản rõ	Chuyển đổi bản rõ thành bản mã bằng cách lấy 1 byte bản rõ tại một thời điểm
Sử dụng 64 bit hoặc nhiều hơn	Sử dụng 8 bit (1 byte)
Mật mã khối không phức tạp	Mật mã dòng phức tạp hơn
Mật mã khối sử dụng sự xáo trộn và khuếch tán	Mật mã dòng chỉ sử dụng sự xáo trộn
Bản rõ được mã hóa ngược rất khó	Bản rõ được mã hóa ngược dễ dàng
Các chế độ thuật toán được sử dụng trong mật mã khối là ECB và CBC	Trong mật mã dòng là CFB và OFB
Mật mã khối hoạt động trên các kỹ thuật chuyển vị như kỹ thuật rail-fence, kỹ thuật chuyển vị cột, ...	Mật mã dòng hoạt động trên các kỹ thuật thay thế.
Mật mã khối chậm hơn so với mật mã dòng	Mật mã dòng nhANH hơn mật mã khối

2. Các chế độ hoạt động của mã hóa Khối:

a/ Electronic Code Book (ECB – cơ chế bảng tra điện tử)

- Chế độ này được gọi là **chế độ thay thế đơn giản**.
- Là chế độ đơn giản nhất trong các chế độ hoạt động của mã hóa khối.
- Thực hiện mã hóa trực tiếp từng khối bản rõ đầu vào và đầu ra ở dạng các khối bản mã được mã hóa.
- Giả sử một thuật toán có quy định kích thước mỗi khối là bội của n bit. Nếu **một thông điệp có kích thước lớn hơn n bit**, thì **nó có thể được chia thành nhiều khối có kích thước n bit**, và nếu tồn tại khối $< n$ bit thì ta đệm thêm vào phía sau khối đó 1 chuỗi bit để kích thước của nó là n bit (Chẳng hạn thêm 1 chuỗi 0 đệm phía sau khối).
- **Kích thước n phụ thuộc vào thuật toán mã hóa**.
- Mỗi khối của bản rõ sẽ được mã hóa **độc lập** với nhau, và cùng sử dụng 1 key, và các khối bản mã sau khi mã hóa cũng sẽ **độc lập** nhau, không có sự liên kết giữa các khối.
 - Điều này có nghĩa là nếu hai khối bản rõ giống nhau ($P_i = P_j$), thì các khối bản mã tương ứng cũng giống nhau ($C_i = C_j$).
- Chế độ **Electronic Codebook (ECB)** trong mã hóa khối được gọi như vậy vì cách hoạt động của nó giống với một **"sổ mã (codebook)"** truyền thống trong mật mã học, nhưng được thực hiện dưới dạng điện tử (electronic).
 - Một **codebook** (sổ mã) là một bảng tra cứu (lookup table) được sử dụng để mã hóa và giải mã thông điệp. Sổ mã **chứa các cặp ánh xạ giữa bản rõ (plaintext) và bản mã (ciphertext)**.
 - **Đặc điểm của sổ mã:**
 - **Mỗi bản rõ được ánh xạ cố định sang một bản mã duy nhất.**
 - **Các ánh xạ là độc lập với nhau**, không có sự phụ thuộc giữa các phần của thông điệp.
 - Thay vì sử dụng một sổ mã vật lý (như trong mật mã cổ điển), **ECB sử dụng thuật toán mã hóa khối (block cipher) để thực hiện ánh xạ này một cách điện tử**, thông qua tính toán.

Bản mã được phân thành các khối P_1, P_2, \dots, P_n



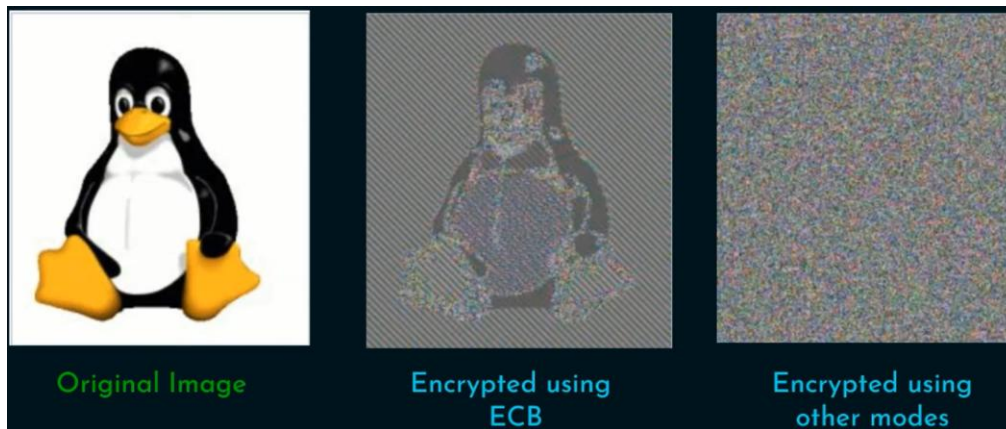
- **Pros**

- Simplest mode.

- Ideal for **short** amount of data. Ex: secure transfer of AES or DES key
 - Independent - Can encrypt any block.
 - Fast - Parallelism.

- **Cons**

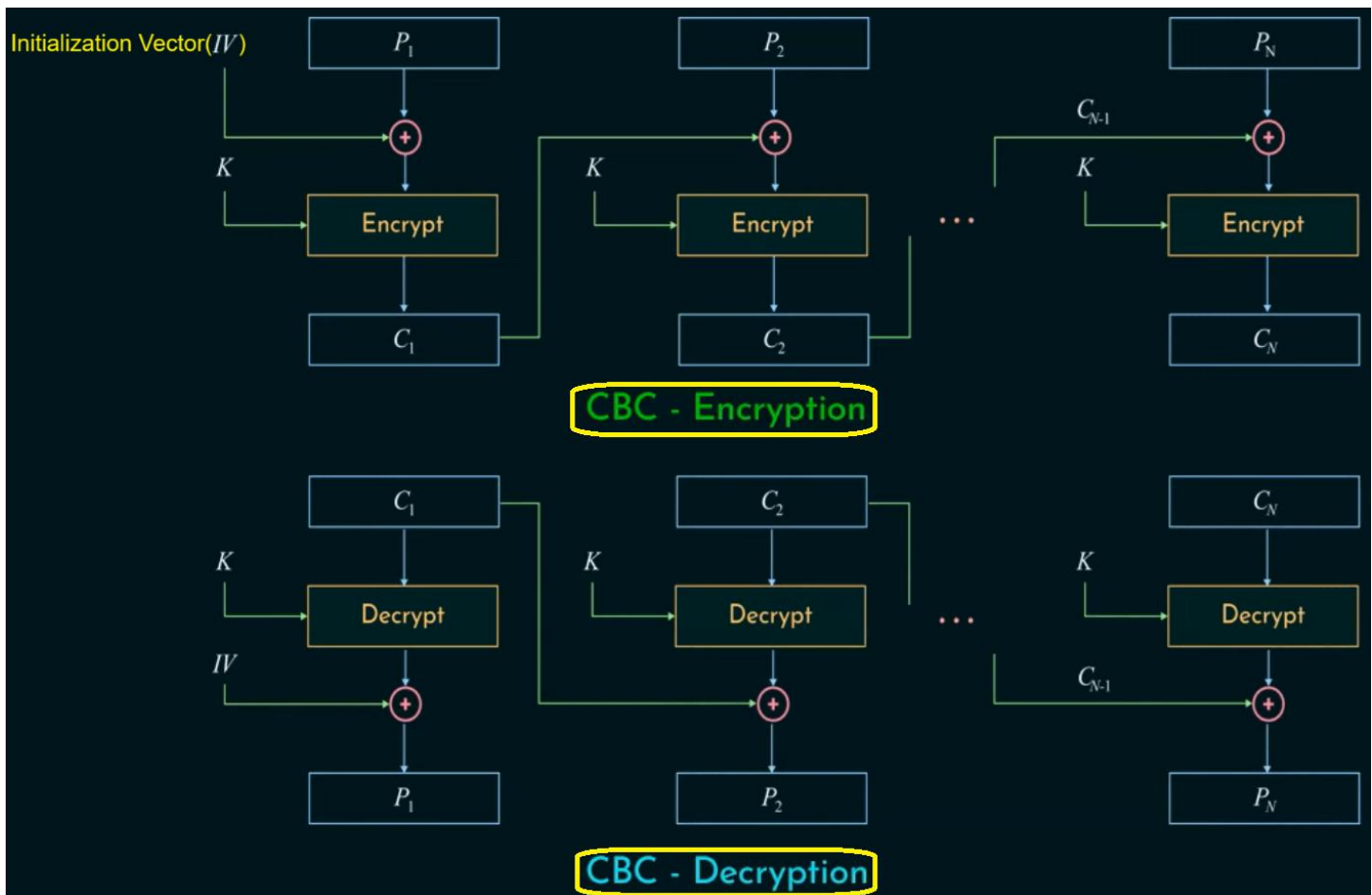
- Not secure for lengthy messages.
 - Trong thông điệp có các khối có mẫu lặp lại → Kẻ tấn công khi nhận được các mẫu bản mã giống nhau sẽ biết được chắc chắn **trong bản rõ cũng có các mẫu giống nhau (mang tính quy luật)**, làm giảm tính bảo mật, & là 1 cơ sở để kẻ tấn công có thể suy luận các vấn đề khác.
 - Attacker can exploit the regularities (quy luật) of the message.



→ Kẻ tấn công vẫn nhìn thấy được đó là con chim cánh cụt thông qua bản mã thực hiện bởi ECB.

b/ Cipher Block Chaining (CBC – cơ chế mã móc xích)

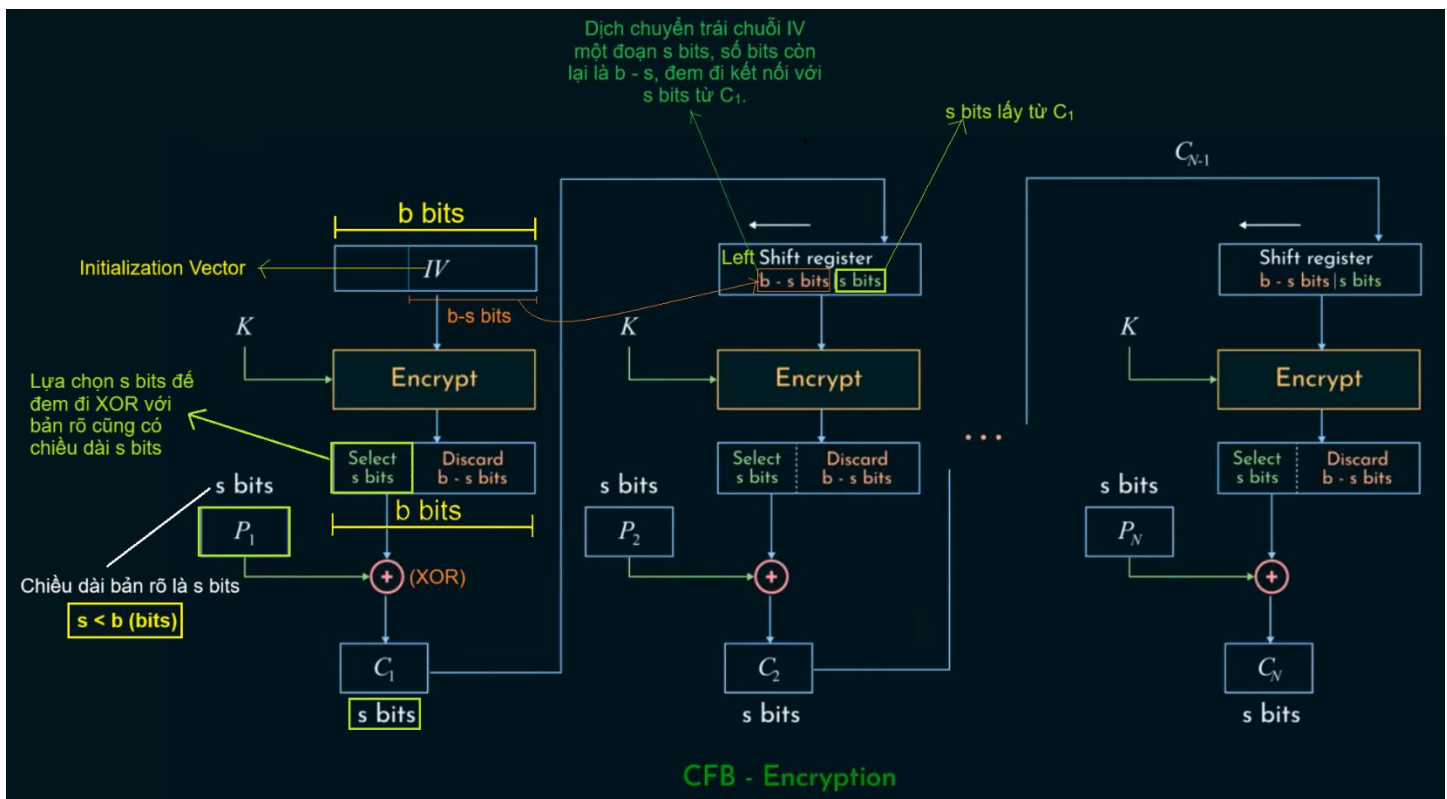
- CBC là một cải tiến được thực hiện trên ECB, do ECB dễ bị phá vỡ.
- Sử dụng một Key chung để mã hóa
- Khối trước móc vào khối sau & thực hiện mã hóa → Chaining
- **Ý tưởng chính:**
 - Dữ liệu được chia thành các khối bản rõ (plaintext) có kích thước cố định.
 - Mỗi khối bản rõ được **XOR** với **khối bản mã trước đó** trước khi mã hóa, tạo ra sự phụ thuộc giữa các khối (chaining).
 - **Khối đầu tiên** được **XOR** với một **vector khởi tạo (Initialization Vector - IV)** để bắt đầu chuỗi.
 - Kết quả là các khối bản mã không còn giống nhau ngay cả khi các khối bản rõ giống nhau, giúp che giấu mẫu dữ liệu.



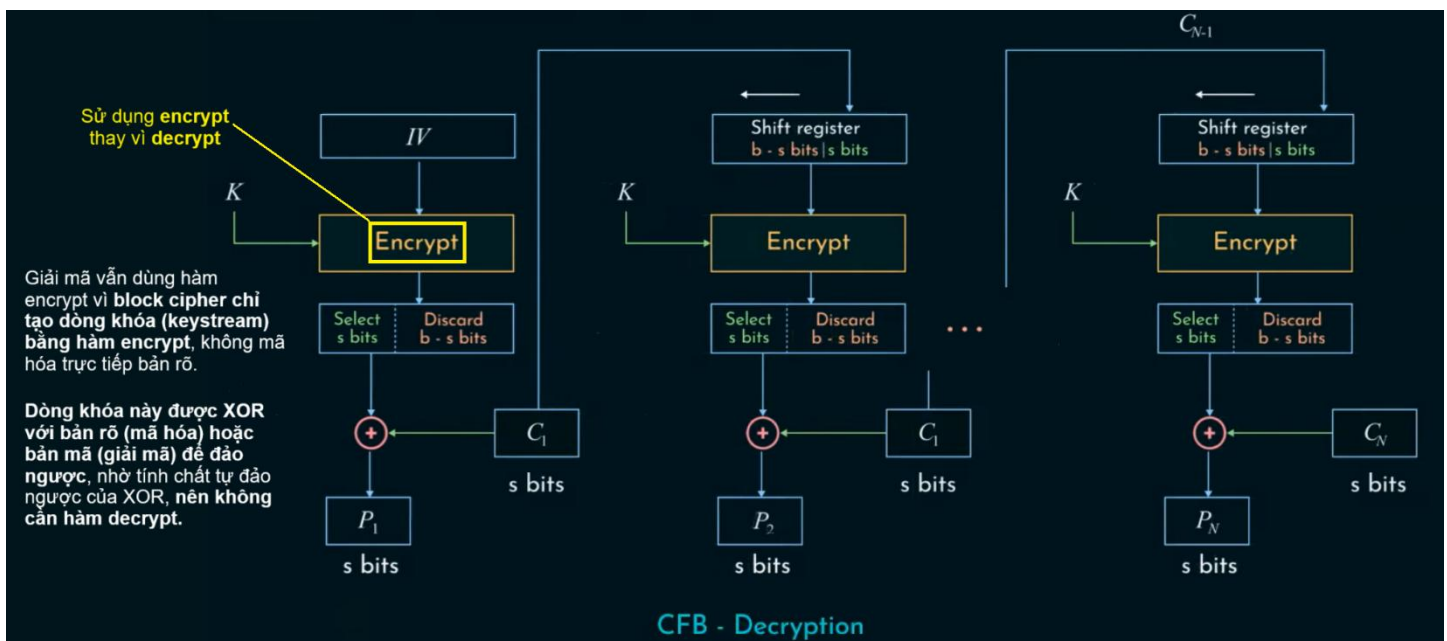
- **Vector khởi tạo có độ dài bằng với kích thước của mỗi khối.**
- Được dùng trong mã hóa CSDL, tập dữ liệu lớn...
- Pros
 - Appropriate mode for encrypting messages of **length greater than 'b' bits**.
 - **Confidentiality + Authentication** (tin tưởng rằng thông điệp chỉ đến từ đúng nguồn tin cậy, vì chỉ có họ và người nhận mới biết được chuỗi vector khởi tạo)
- Cons
 - Slow - **No parallelism**.
 - Cannot encrypt any block since we need the ciphertext of previous block.
 - **Initialization Vector (IV) which must be known to sender & receiver.**

c/ Cipher Feedback Mode (CFB)

- Trong chế độ này, **mật mã được cung cấp dưới dạng phản hồi** cho khối mã hóa tiếp theo với một thông số kỹ thuật mới



- C_1 hoạt động như một phản hồi (feedback), vì nó được sử dụng để sinh ra bản mã hóa tiếp theo C_2 .

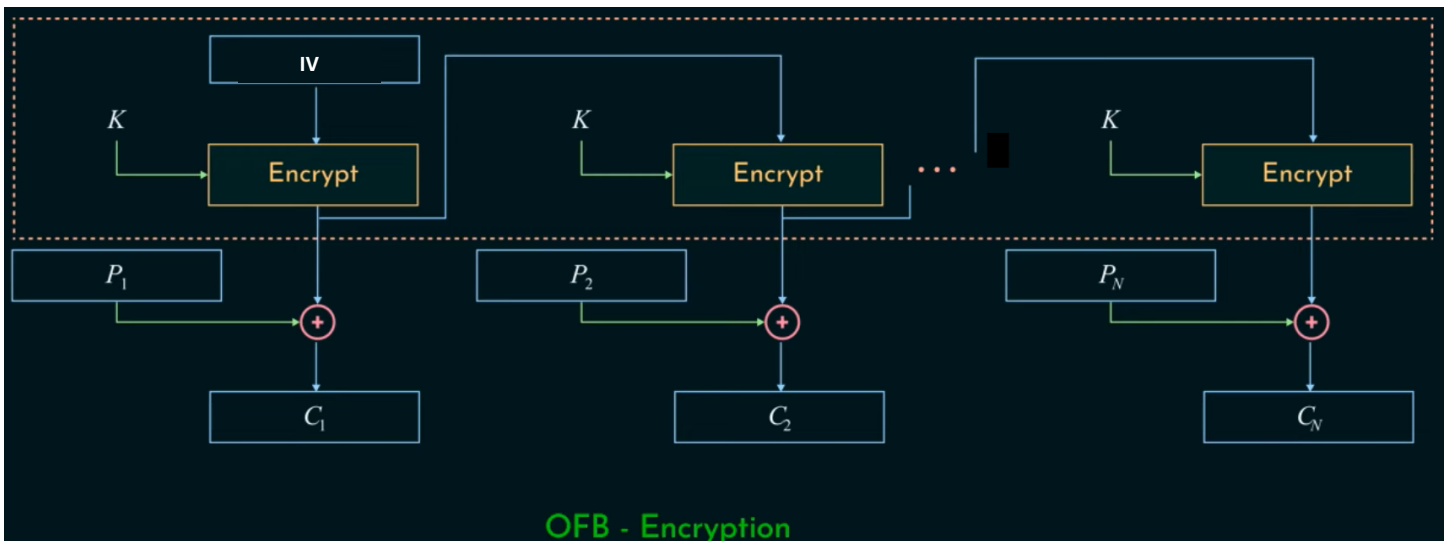


- **Pros**
 - Can operate in **real time**.
 - hoạt động như mã hóa dòng (stream cipher), nghĩa là **chúng tạo dòng khóa (keystream) và XOR với dữ liệu đầu vào từng bit hoặc byte một**. Điều này cho phép mã hóa và giải mã dữ liệu ngay khi nhận được, mà không cần chờ toàn bộ khối dữ liệu (như trong ECB hoặc CBC).
 - **Need of padding is eliminated.**

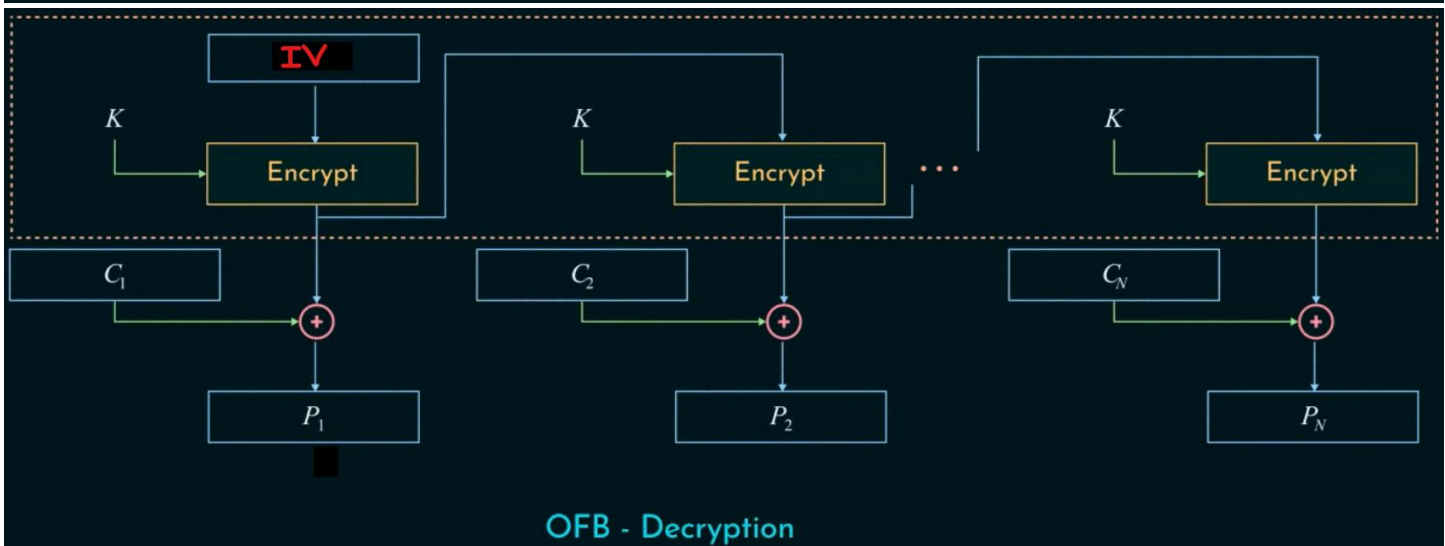
- Encryption function does decryption as well.
- Length of PT = Length of CT.
- Cons
 - Chances of wastage of transmission capacity.
 - Nếu bản rõ rất ngắn (chỉ 1 byte), nhưng IV là 128 bit, thì tổng dữ liệu truyền đi (128 bit IV + 8 bit bản mã) sẽ lớn hơn nhiều so với bản rõ ban đầu, gây lãng phí băng thông.
 - Not a typical **stream cipher**.
 - dựa trên mã hóa khối (block cipher) để tạo dòng khóa, nên hiệu suất có thể thấp hơn so với các mã hóa dòng chuyên dụng.

d/ Output Feedback Mode (OFB – chế độ mã hóa phản hồi đầu ra)

- Chế độ này tuân theo qui trình **gần giống** với chế độ CFB.
- Khác với CFB ở chỗ, **C₁ không đóng vai trò là feedback nữa**, và bản rõ P₁ được thực hiện XOR với **t toàn bộ bits của IV** sau khi đưa vào thuật toán mã hóa, **thay vì chỉ s bits như CFB**.



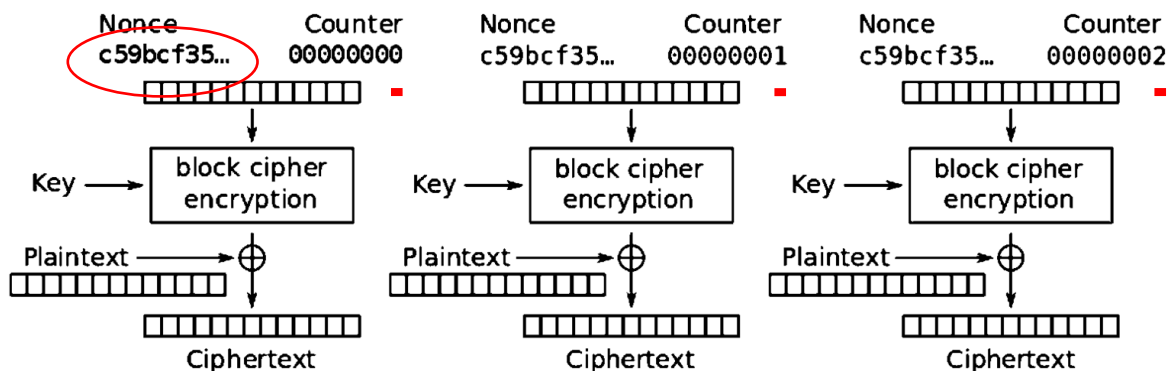
OFB - Encryption



OFB - Decryption

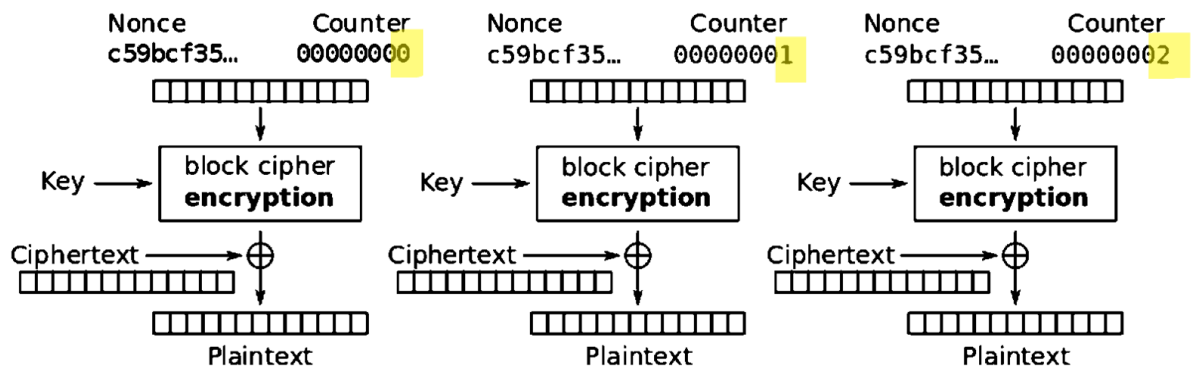
e/ Counter Mode (CTR)

- Chế độ này triển khai mật mã khối **dựa trên một bộ đếm đơn giản**.
- Chế độ này tương tự như OFB với điểm khác biệt là nó sử dụng bộ đếm hoặc số thứ tự làm đầu vào cho thuật toán.
- Mỗi khi **một giá trị khởi tạo ngược** được mã hóa và được cung cấp làm đầu vào cho XOR với văn bản gốc dẫn đến khối mã hóa.
- **Chế độ CTR độc lập với việc sử dụng phản hồi** và do đó **có thể được thực hiện song song**
- Chuyển block cipher thành một dạng mã hóa dòng (stream cipher) bằng cách **sử dụng một bộ đếm (counter)** kết hợp với một **giá trị khởi tạo (nonce - number used once)** để tạo dòng khóa (keystream).
- **Quá trình giải mã tương tự**, sử dụng cùng bộ đếm để tạo lại dòng khóa và XOR với bản mã để khôi phục bản rõ.
- **Dữ liệu đầu vào (bản rõ) được chia thành các khối có kích thước cố định**. Nếu dữ liệu không đủ dài, nó sẽ được **đệm (padding)** để đạt kích thước khối.
- **Different counter value for each plaintext.**
- **Counter value is initialized, given to the first plain text block, and incremented after 1 encryption → different cipher text.**



Counter (CTR) mode encryption

- **Chiều dài Nonce:** Thường từ 64 bit đến 96 bit (thường 96 bit với AES), tùy thuộc vào yêu cầu ngẫu nhiên và bảo mật.
- **Chiều dài Counter:** Phần còn lại của kích thước khối (thường 32 bit hoặc 64 bit), đủ để xử lý lượng dữ liệu trong một Nonce.
- **Tổng chiều dài:** Phải bằng kích thước khối của block cipher (ví dụ: 128 bit với AES).

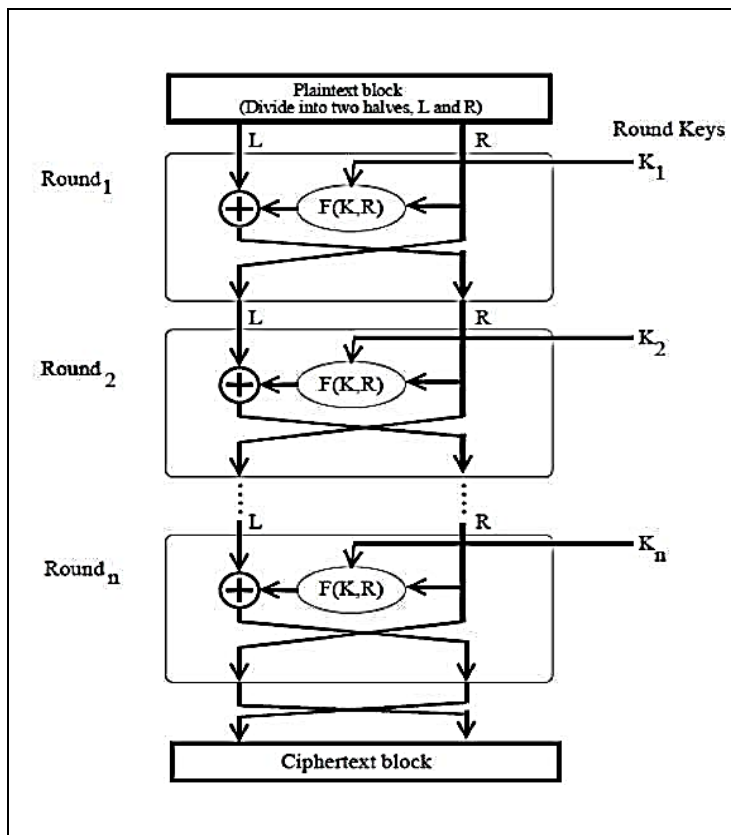


Counter (CTR) mode decryption

3. Khái niệm cấu trúc Feistel

- **Cấu trúc Feistel** là **một mô hình thiết kế (design framework)** dùng để xây dựng các mật mã khối đối xứng (symmetric block ciphers).
- Cấu trúc Feistel **không phải là một thuật toán mã hóa cụ thể, mà là một khuôn mẫu (template)** để xây dựng các thuật toán mã hóa khối.
- Ông đã phát triển cấu trúc này khi nghiên cứu và thiết kế **DES (Data Encryption Standard)**.
- Dựa trên nguyên tắc **khuếch tán (diffusion)** và **xáo trộn (confusion)**:
 - **Khuếch tán (Diffusion)**
 - Một thay đổi nhỏ trong bản rõ (plaintext) hoặc khóa (key) sẽ ảnh hưởng đến nhiều bit trong bản mã (ciphertext) → **Hiệu ứng tuyết lở (Avalanche Effect)**:
 - **Mục đích:** Che giấu các thuộc tính thống kê của bản rõ (như tần suất xuất hiện của các ký tự) và làm cho việc phân tích thống kê để tìm khóa trở nên khó khăn.
 - **Ví dụ:** Trong DES, nếu thay đổi 1 bit trong bản rõ, sau 16 vòng, khoảng 32 bit trong bản mã 64 bit sẽ thay đổi.
 - **Xáo trộn (Confusion)**
 - làm cho **mối quan hệ giữa khóa và bản mã trở nên phức tạp nhất** có thể.
 - **Mục đích:** Ngăn chặn kẻ tấn công sử dụng phân tích thống kê để suy ra thông tin về khóa từ bản mã.
 - **Kết hợp khuếch tán và xáo trộn:**
 - Sự kết hợp này là **nền tảng của các hệ thống mật mã an toàn**, được Claude Shannon đề xuất trong lý thuyết mật mã hiện đại.
 - Khuếch tán làm lan tỏa sự thay đổi, còn xáo trộn làm mờ mối quan hệ giữa khóa và bản mã. Khi kết hợp, chúng tạo ra một hệ thống mật mã mạnh, khó bị phá vỡ bởi các cuộc tấn công thống kê.
- Mật mã khối được xây dựng trong cấu trúc mật mã Feistel. **Mật mã khối có số vòng (Round) và Khóa (Key) cụ thể để tạo bản mã.** Để xác định mức độ phức tạp của một thuật toán, một vài nguyên tắc thiết kế sẽ được xem xét như sau:
 - **Số vòng (round):**
 - Là **số lần lặp lại của quá trình xử lý** trong thuật toán mã hóa. Mỗi vòng thực hiện các bước *thay thế (substitution)* và *hoán vị (permutation)* để *khuếch tán (diffusion)* và *xáo trộn (confusion)* dữ liệu.
 - **Ví dụ:**
 - DES sử dụng 16 vòng.
 - AES (dù không dùng cấu trúc Feistel, mà dùng SPN - Substitution-Permutation Network) sử dụng 10 vòng cho khóa 128 bit, 12 vòng cho khóa 192 bit, và 14 vòng cho khóa 256 bit.
 - Số lượng vòng được sử dụng phụ thuộc vào độ bảo mật mong muốn từ hệ thống. **Số vòng nhiều hơn cung cấp hệ thống an toàn hơn.** (vì mỗi vòng làm tăng mức độ khuếch tán và xáo trộn, khiến việc phân tích bản mã để tìm khóa trở nên khó hơn).

- Với số vòng đủ lớn, một thay đổi nhỏ trong bản rõ hoặc khóa sẽ lan tỏa (diffuse) ra toàn bộ bản mã, gọi là **hiệu ứng tuyết lở (avalanche effect)**.
- Nhưng đồng thời, **nhiều vòng hơn thì quá trình mã hóa và giải mã chậm và không hiệu quả.**
- Do đó, số lượng vòng trong các hệ thống phụ thuộc vào sự đánh đổi hiệu quả an ninh.
- **Thiết kế hàm F:** Phần cốt lõi của cấu trúc mật mã khối Feistel là Hàm vòng (Round Function), thực hiện các **phép toán để tạo xáo trộn và khuếch tán** trong mỗi vòng.
 - Độ phức tạp của phân tích mã (cryptanalysis) có thể bắt nguồn từ Hàm vòng tức là **độ phức tạp đối với hàm round ngày càng tăng thì sẽ làm gia tăng độ khó của hệ thống mật mã.**
 - Để tăng độ phức tạp của round function, **hiệu ứng tuyết lở cũng nằm trong hàm vòng**, có nghĩa là bất kỳ 1 sự thay đổi nhỏ đơn lẻ nào trong bản rõ cũng sẽ tạo ra một đầu ra không chính xác.
- **Thuật toán lịch trình khóa:**
 - Là quá trình **sinh ra các khóa con (subkeys)** từ khóa chính (master key) để sử dụng trong mỗi vòng của cấu trúc Feistel, giúp **tăng độ phức tạp của phân tích mật mã.**
 - Ví dụ:
 - Trong DES, từ khóa chính 64 bit (thực tế 56 bit sau khi bỏ bit kiểm tra), thuật toán sinh ra 16 khóa con, mỗi khóa con 48 bit, cho 16 vòng.
 - Mỗi khóa con được tạo bằng cách dịch chuyển (shift) và hoán vị (permutation) từ khóa chính.
 - Mỗi vòng dùng một khóa con khác nhau, làm tăng độ phức tạp của thuật toán và khiến việc phân tích để tìm khóa chính trở nên khó hơn.
 - Nếu tất cả các vòng dùng cùng một khóa, kẻ tấn công có thể dễ dàng tìm ra mối quan hệ giữa các vòng.
 - **Hiệu ứng tuyết lở làm cho việc lấy khóa con phức tạp hơn.**
 - Trong cấu trúc Feistel, giải mã sử dụng cùng thuật toán như mã hóa, **nhưng với thứ tự khóa con đảo ngược.**
 - Hiệu ứng tuyết lở trong lịch trình khóa có thể gây khó khăn nếu không cẩn thận, **vì một sai sót nhỏ trong việc tạo khóa con (hoặc trong quá trình giải mã) sẽ dẫn đến kết quả không chính xác.**
 - Do đó, cần đảm bảo rằng thuật toán lịch trình khóa được triển khai chính xác để tạo ra các khóa con đúng cho cả mã hóa và giải mã.
- **Mã hóa Feistel không phải là một lược đồ cụ thể của mật mã khối.** Nó là một mô hình thiết kế mà từ đó nhiều mật mã khối dựa trên mô hình này được tạo ra.
- **Quá trình mã hóa:** sử dụng cấu trúc Feistel bao gồm nhiều vòng xử lý bản rõ, mỗi vòng bao gồm một số bước “thay thế” theo sau là một bước hoán vị.
 - **Bất kỳ một hệ thống mã hóa nào cũng đều có 2 bước:** 1. Sinh khóa, 2. Xử lý mã hóa



• Bước 1: Sinh khóa

- Từ khóa ban đầu (master key), tùy vào mỗi thuật toán cụ thể mà có các thuật toán sinh khóa khác nhau để tạo thành tập hợp các khóa con (K_1, K_2, \dots, K_n) , dùng để sử dụng mã hóa theo mỗi vòng (round).
- Khóa con có độ dài bé hơn hoặc bằng khóa trạng thái (Khóa ban đầu).

• Bước 2: Xử lý mã hóa

- a/ Khối (bit) bản rõ được chia làm 2 nửa: L (nửa bên trái), R (nửa bên phải)
- b/ Cấu trúc Feistel thực hiện n vòng (thường từ 8 đến 16 vòng, ví dụ DES có 16 vòng). Trong mỗi vòng:

▪ $L_{i+1} = R_i$

- Trong vòng thứ i, nửa bên phải R_i của dữ liệu ở vòng hiện tại sẽ trở thành nửa bên trái L_{i+1} của vòng tiếp theo mà không thay đổi gì.
- Đây là một bước sao chép trực tiếp,

▪ Thực hiện hàm $F(K_i, R_i)$

- Hàm F thường bao gồm các bước sau (lấy ví dụ từ DES):
 - **Mở rộng (Expansion):** R_i được mở rộng tùy theo thuật toán mã hóa (ví dụ: từ 32 bit lên 48 bit trong DES).
 - **XOR với khóa con:** XOR R_i (đã mở rộng) với K_i .
 - **Thay thế (Substitution):** Dùng S-boxes (substitution boxes) để thay thế các bit theo cách phi tuyến tính, tạo xáo trộn.
 - **Hoán vị (Permutation):** Sắp xếp lại các bit để tạo khuếch tán.
- Kết quả của $F(K_i, R_i)$ là một chuỗi bit có độ dài bằng R_i , sẵn sàng để XOR với L_i .

▪ $R_{i+1} = L_i \oplus F(K_i, R_i)$.

- Nửa bên trái L_i của vòng hiện tại được XOR với kết quả của hàm $F(K_i, R_i)$, và kết quả trở thành nửa bên phải R_{i+1} của vòng tiếp theo.
- Điều này giúp lan tỏa (diffuse) sự thay đổi từ R_i (qua hàm F) sang R_{i+1} , và tiếp tục lan tỏa qua các vòng sau.

• Bước 3: Vòng cuối cùng thực hiện bước hoán vị đổi vị trí L và R cho nhau:

- $L_{n+1} = R_n$.
- Thực hiện hàm $F(K_n, R_n)$.
- $R_{n+1} = L_n \oplus F(K_n, R_n)$.

- $L_{n+1} = R_{n+1}$.
- $R_{n+1} = L_{n+1}$.
- **Kết quả:** $L_{n+1} || R_{n+1}$ là khối bản mã (ciphertext).
- Giải mã là quá trình **ngược lại** quá trình mã hóa.
- Nếu không hoán đổi L và R ở vòng cuối của mã hóa, quá trình giải mã sẽ không thể khôi phục đúng bản rõ khi dùng cùng thuật toán. Hoán đổi cuối cùng đảm bảo tính đối xứng giữa mã hóa và giải mã.
- **Ưu điểm của mạng Feistel:**
 - Đơn giản hóa việc triển khai phần cứng trên cơ sở mã hoá hiện đại.
 - Đơn giản hóa việc triển khai phần mềm do thực tế là một phần quan trọng của các chức năng được hỗ trợ ở cấp độ phần cứng trong các máy tính hiện đại (ví dụ: tổng theo modulo 2, v.v.)
 - Nghiên cứu tốt các thuật toán dựa trên mạng Feistel
- **Nhược điểm:** chỉ một nửa khối đầu vào được mã hóa trong một vòng.