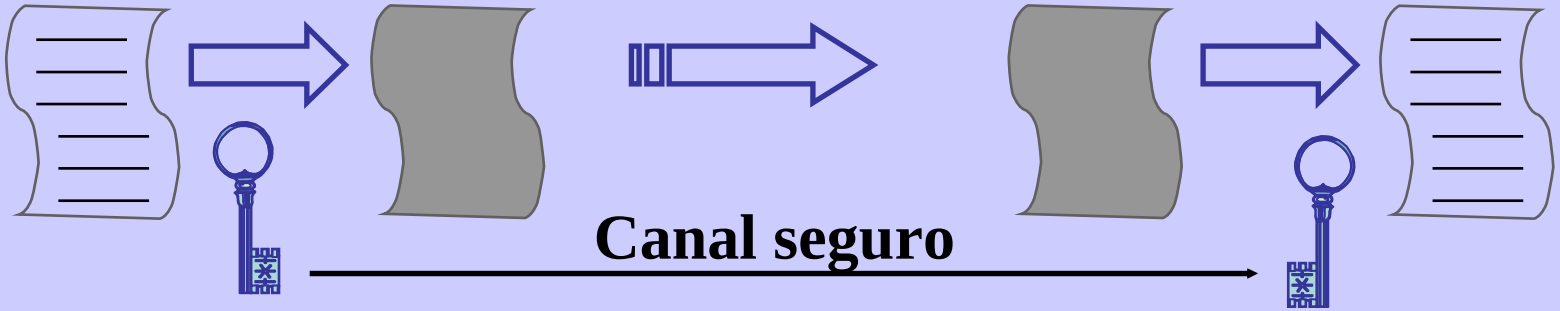
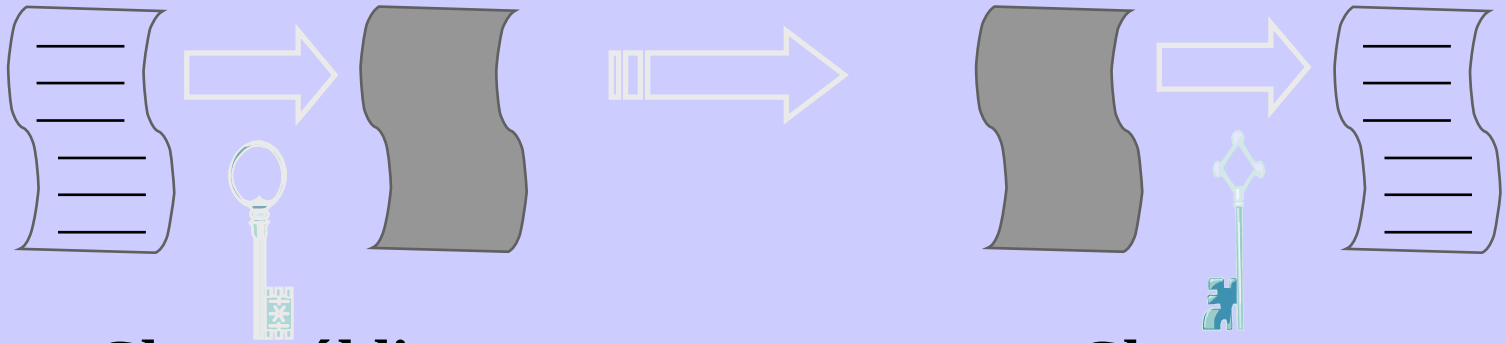
Criptografía Moderna

– Criptosistemas Simétricos o de Clave Secreta:



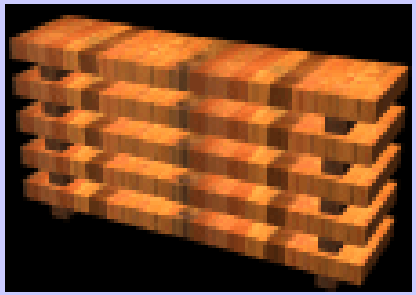
– Criptosistemas Asimétricos o de Clave Pública:



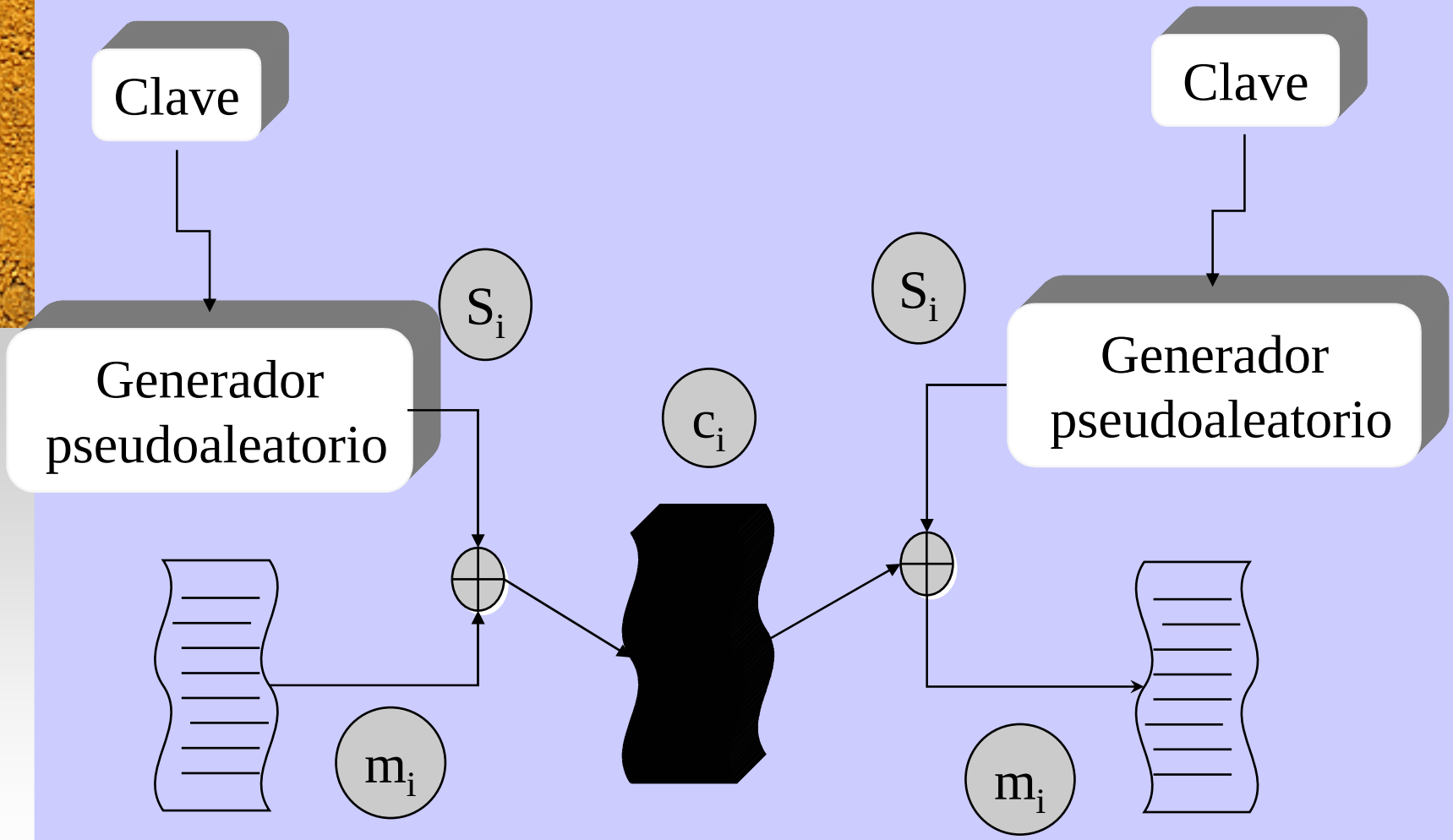
Criptografía Moderna

Criptosistemas modernos

- Clave secreta
  - Cifrado en flujo
  - Cifrado en bloque
- Clave pública

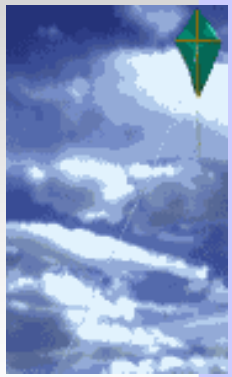


Cifrado en Flujo

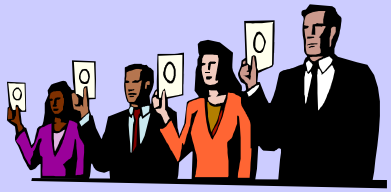


Cifrado en Flujo

- El emisor usa una semilla secreta y un algoritmo determinista (generador pseudoaleatorio) para generar una secuencia binaria.
- Con la secuencia generada y el texto en claro expresado en binario se realiza una xor bit a bit.
- Realizando la misma operación con el texto cifrado y la misma secuencia pseudoaleatoria se recupera el texto en claro.
- **La seguridad del sistema se basa únicamente en las características de la secuencia de clave**



Cifrado en Flujo



Propiedades de las secuencias cifrantes:

- Periodo T muy alto.
- Postulados de pseudoaleatoriedad de Golomb:
  1. Unos y ceros deben aparecer con idéntica frecuencia.
  2. En cada periodo, la mitad de las rachas es de longitud 1, la cuarta parte de longitud 2, la octava de 3, etc. Las rachas de ceros y de unos deben aparecer con idéntica frecuencia para cada longitud.
  3. Autocorrelación AC(k)=(NºCoinє NºDif)/T. El número de coincidencias entre una secuencia y su desplazada k posiciones no debe aportar ninguna información sobre el periodo, para cualquier k no múltiplo de T.
- Gran complejidad lineal: Cantidad de secuencia necesaria para describir el resto

Cifrado en Flujo: Generadores de Secuencia

$$X_{i+1} = a X_i + b \pmod{m}$$

- Congruencial
- Generador Blum Blum Shub

  p, q (= 3 mod 4), n = pq

  X número aleatorio en [1, n− 1] primo con n.

  $s_0 = X^2 \pmod{n}$

  $s_{i+1} = s^2_i \pmod{n}$

  z_i = el bit menos significativo de s_i.

  La secuencia de salida es $z_1 z_2 \ldots z_n$

RC4

- Algoritmo propietario diseñado por Rivest en 1987
- De implementación sencilla y rápida, y orientado a generar secuencias en unidades de un byte
- Permite claves de diferentes longitudes
- El algoritmo no se ha publicado nunca, pero en 1994 en Internet apareció una descripción
- Se usa en SSL y en WEP

Algoritmo RC4

Inicialización:

1. $S_i = i$, i=0 ... 255
2. Rellenar el array K_i i=0 ... 255 repitiendo la semilla de clave secreta
3. f=0
4. Desde i = 0 hasta 255 hacer:

   $f = (f + S_i + K_i) \bmod 256$

   Intercambiar S_i y S_f

Generación de secuencia cifrante:

Inicializar i y f a 0, y calcular cada byte S_t de la secuencia cifrante mediante:

1. $i = (i + 1) \bmod 256$; $f = (f + S_i) \bmod 256$;
2. Intercambiar S_i y S_f
3. $t = (S_i + S_f) \bmod 256$

Ejemplo de RC4

Con semillas de 2 bytes (“de 2 bits”) para generar secuencias de 4 bytes (“de 2 bits”)

Inicialización:

*S[ ]=[**S0**, S1, S2, S3] = [0, 1, 2, 3]*

*K[ ]=[**K0**, K1] = [2, 5]*

f=0

Iteración1:

*f=(f + S0 + K0)mod 4=**2***,

intercambiar *S0 con S2*

Nuevo array *S[ ]=[**S0**, S1, **S2**, S3] = [2, 1, 0,3]*

Iteración 2:

*f=(f + S1 + K1)mod 4=**0***,

intercambiar *S1 con S0*

Nuevo array *S[ ]=[**S0**, **S1**, S2, S3] = [**1**, **2**, 0, 3]*

Ejemplo de RC4

Iteración 3:

*f=(f + S2 + K0)mod 4=**2***,

intercambiar *S2* con *S2*

Nuevo array *S[ ]=[S0, S1, **S2**, S3] = [1, 2, **0**, 3]*

Iteración 4:

*f=(f + S3 + K1)mod 4=**2***,

intercambiar *S3* con *S2*

Nuevo array *S[ ]=[S0, S1, **S2**, **S3**] = [1, 2, **3**, **0**]*

Ejemplo de RC4

Generación de secuencia cifrante:

Iteración 1:

i=0, f=0

i = (i + 1)mod 4 = 1

f =(f + S1)mod 4 = 2,

intercambiar *S1* con *S2*

Nuevo array *S[ ]=[S0, S1, S2, S3] = [1, 3, 2, 0]*

t = (S1 + S2)mod 4 = 1

S1 = 3 (0000 0011)

Iteración 2:

i=1, f=2

i = (i + 1)mod 4 = ***2***

f =(f + S2)mod 4 = ***0,***

intercambiar *S2* con *S0*

Nuevo array *S[ ]=[* ***S0****, S1,* ***S2****, S3] = [* ***2****, 3,* ***1,*** *0]*

t = (S2 + S0)mod 4 = ***3***

S3 *= 0 (0000 0000)*

Ejemplo de RC4

Cifrado

* Para este ejemplo usamos texto original “HI”

H

0100 1000

XOR 0000 0011

0100 1011

I

0100 1001

XOR 0000 0000

0100 1001

Texto original: 0100 1000 0100 1001

Texto cifrado: 0100 1011 0100 1001

Inseguridad de RC4

- Genera secuencias con periodos bastante grandes
- Es inmune a los criptoanálisis diferencial y lineal
- Sin embargo algunos estudios indican que pueden existir claves débiles y que es sensible a estudios analíticos del contenido de la S Caja. De hecho, algunos afirman que en una de cada 256 claves posibles, los bytes que se generan tienen una fuerte correlación con un subconjunto de los bytes de la clave
- Se puede recuperar la clave empleada si la inicialización del algoritmo cumple determinadas premisas muy comunes, y se interceptan el suficiente número de mensajes.

Ataques a RC4

Basado en el uso del comienzo de una secuencia cifrante para deducir información de la clave completa.

Para realizar estos ataques con éxito, durante la primera fase del ataque, los primeros bytes deben ser adivinados correctamente. Hay dos métodos para conseguirlo:

1. Basado en la recuperación de la clave secreta cuando la semilla del algoritmo se deriva de la concatenación de dicha clave secreta y un vector inicial (IV) público y conocido. Permite recuperar la parte secreta de la clave RC4 recopilando un gran número de mensajes y IVs. En este caso hay que buscar IVs débiles, que hagan que no haya información de la clave en la secuencia ya que así sólo es necesario adivinar un byte de la clave (con probabilidad 5%).
2. Basado en *patrones invariantes*, e.d., patrones de la clave que se propagan al estado interno del algoritmo debilitándolo. Así, los primeros bytes generados pueden resultar muy predecibles. En este caso hay que centrarse en la manera en que se distribuyen las claves ya que una clave fácil de recordar suele contener caracteres ASCII, luego comprobando si los bytes de la clave concuerdan con caracteres ASCII cómo letras o símbolos etc. las posibilidades de adivinar la clave correcta aumentan

Ataques a RC4

- ¿Cuántos paquetes cifrados necesitamos recolectar para romper una clave WEP?
  - En general de 5 a 10 millones de paquetes cifrados

- Hay dos herramientas en Internet que implementan totalmente el ataque:
  - Wepcrack: wepcrack.sourceforge.net/
  - Airsnort: airsnort.shmoo.com/

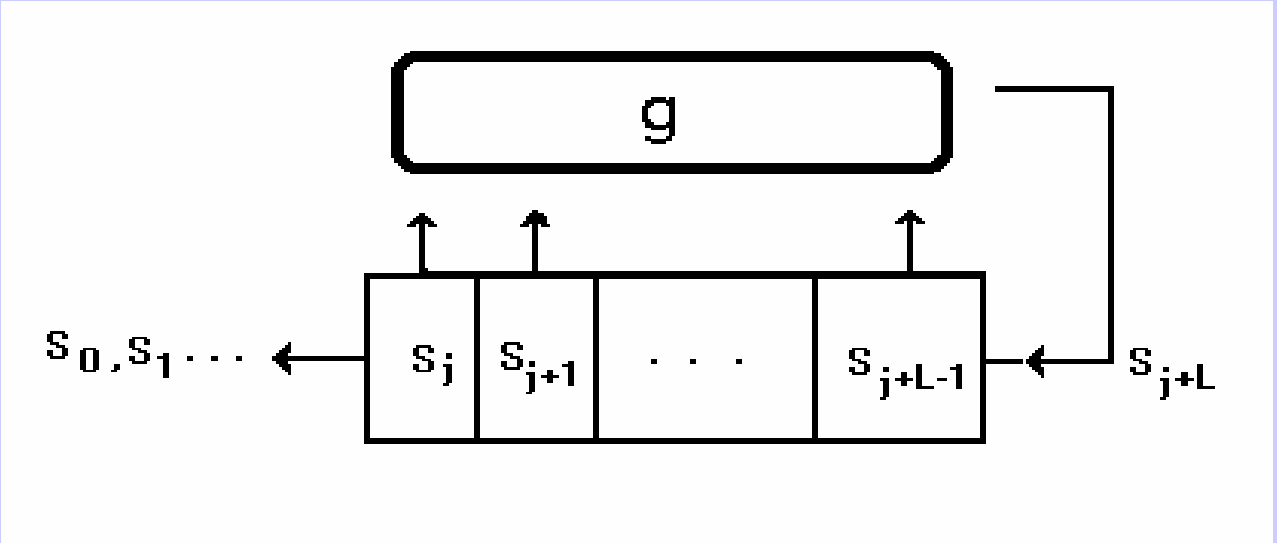
Seguridad de RC4

1. Ya que uno de los problemas reside en la predecibilidad de los primeros bytes RC4, una posibilidad es descartarlos. Es decir, no empezar a utilizar la salida RC4 desde el primer byte.
2. Otro solución consiste en evitar que el atacante obtenga segmentos largos de la clave RC4. Una opción sería reemplazar la concatenación típica entre la clave secreta y los IVs por una función *hash*, aunque esto repercute en la eficiencia.

- En general, resulta preferible descartar los, p.ej. primeros 256 bytes generados, que descartar 64 bytes y calcular un *hash*. Si se descarta el suficiente número de bytes iniciales, se resuelven ambos problemas

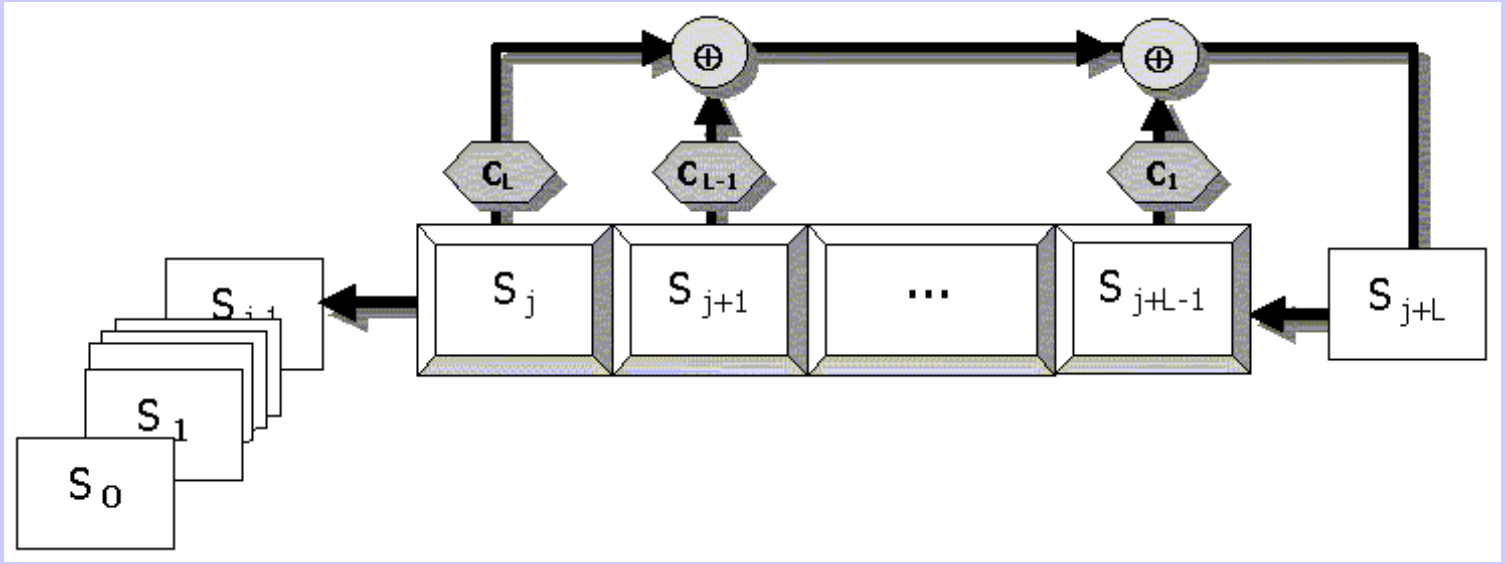
Cifrado en Flujo: Generadores de Secuencia

- Registro de Desplazamiento Realimentado:
  - Linealmente.
  - No Linealmente.



➢**Función de realimentación g no-singular: Secuencia periódica**

Cifrado en Flujo: RDRL



Polinomio de realimentación: $C(x)=1+c_1x+c_2x^2+\ldots+c_Lx^L$

- Factorizable: Periodo depende de semilla
- Irreducible: Periodo no depende de semilla, y divisor de 2^L-1
- Primitivo: periodo máximo 2^L-1 e independiente de la semilla
- Nº de polinomios primitivos de grado L $=\Phi(2^L-1)/L$

1000
0001
0011
0111
1111
1110
1101
1010
0101
1011
0110
1100
1001
0010
0100

PN-Secuencias

- Ejemplo: $C(x)=1+x+x^4$

Postulados de Golomb:

1. 7 ceros, 8 unos
2. 0, 1 :2 veces,

   00,11: 1 vez

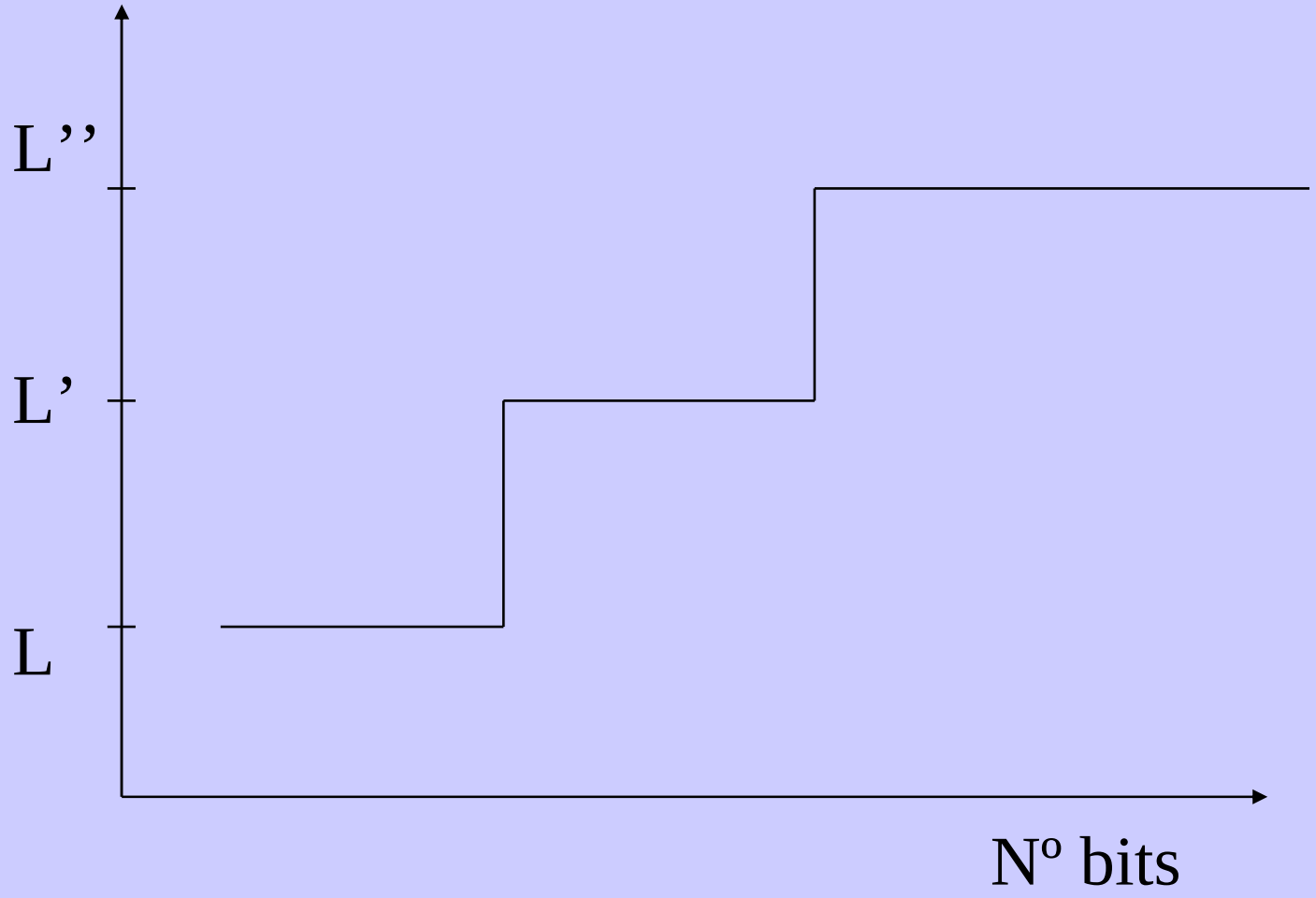
   000: 1 vez

   1111: 1 vez
3. AC(k)=-1/15

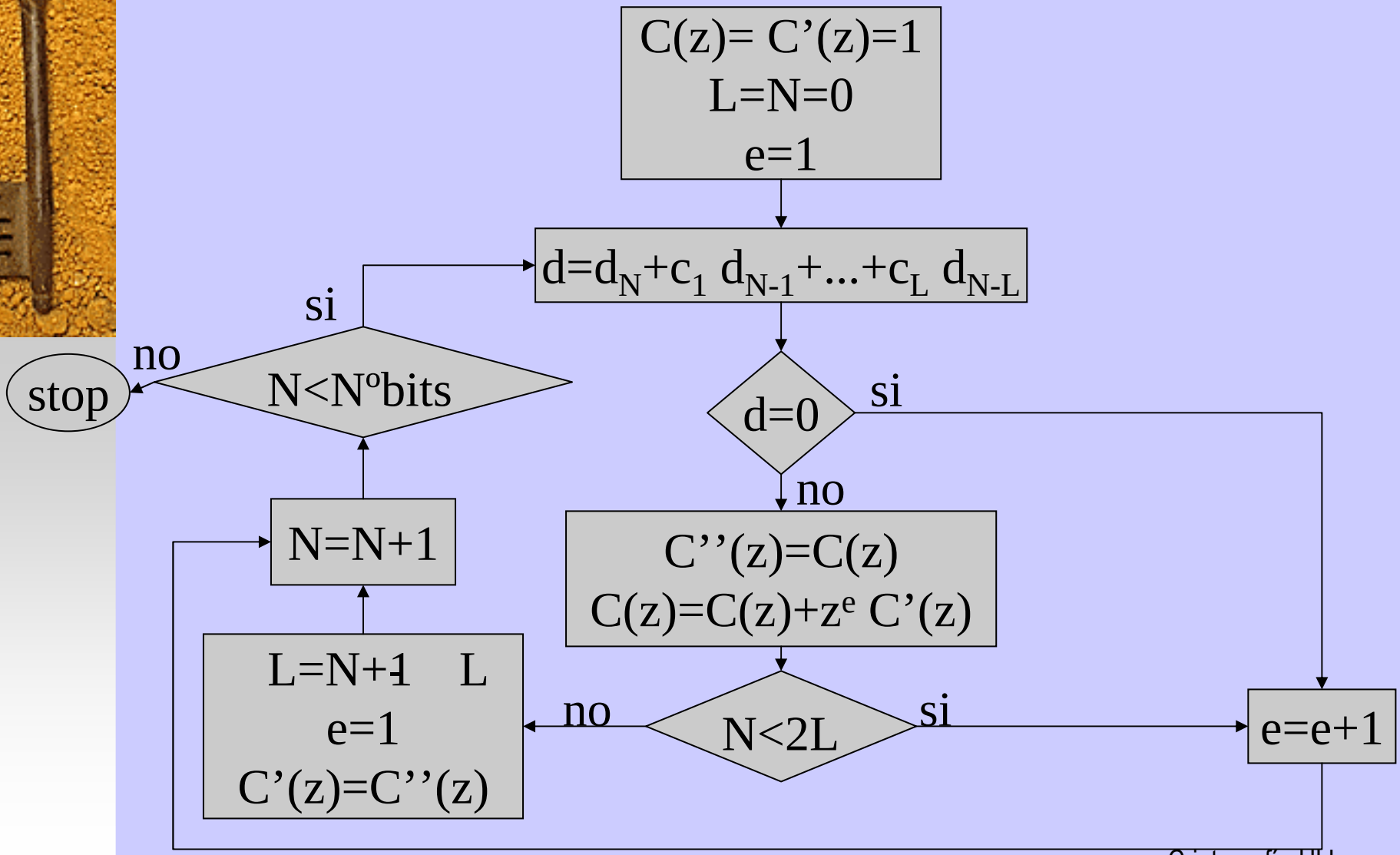
Complejidad Lineal

- Dada una secuencia producida con un RDRL de longitud L bastan 2L bits para resolver el sistema de L ecuaciones con L incógnitas y descubrir los coeficientes de realimentación.
- Cualquier secuencia periódica se puede generar con un RDRL no singular
- Complejidad lineal: Longitud del menor RDRL que genera la secuencia

Complejidad Lineal

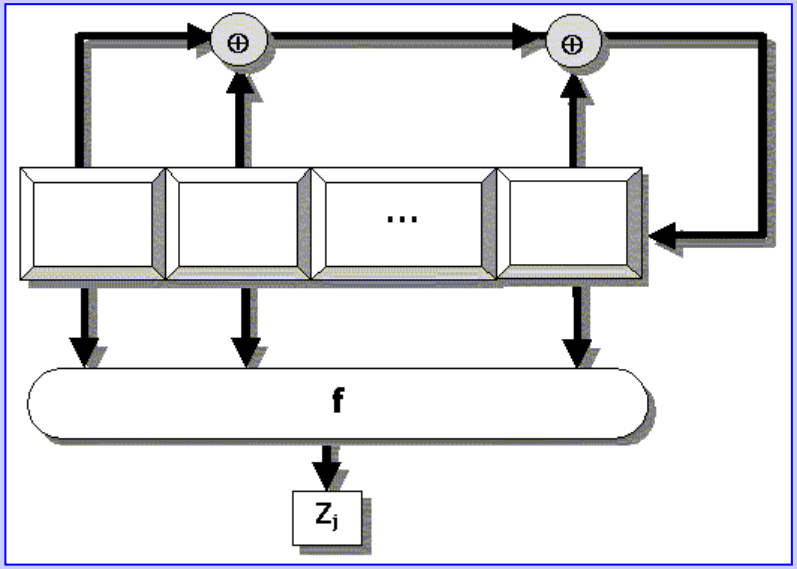


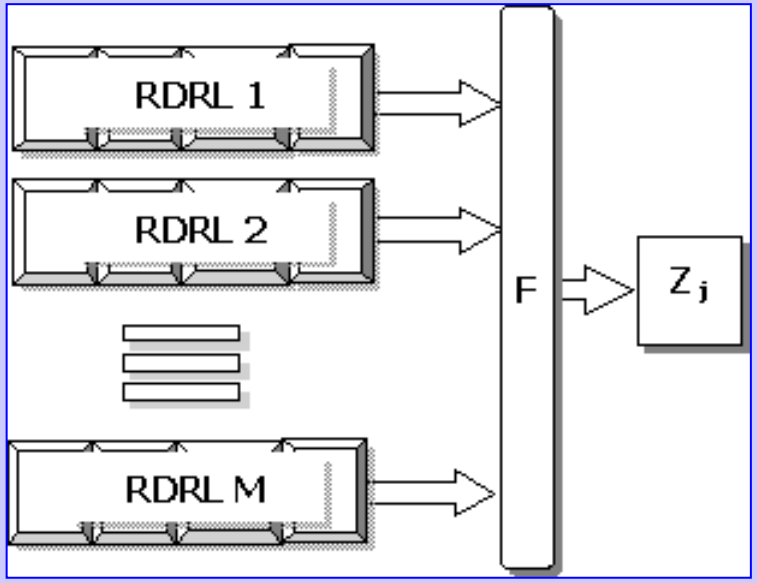
Algoritmo de Berlekamp-Massey



Cifrado en Flujo: Generadores de Secuencia

- Filtrado no lineal
- Combinador no lineal





Filtrado no Lineal

- Complejidad lineal acotada por $\sum_{i=1}^{m} \binom{L}{i}$

(siendo m el orden de la función de filtrado f)

- Principios de diseño:

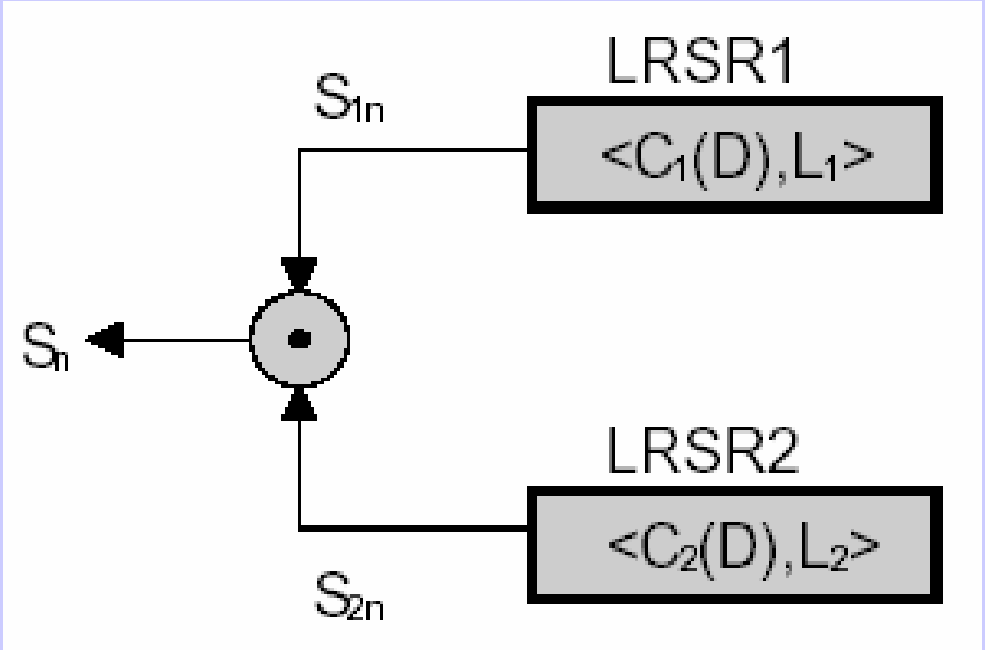
1. Usar registro con polinomio primitivo para lograr periodo máximo
2. Orden de la función del orden L/2
3. Incluir en f término lineal y términos de orden pequeño para lograr buena distribución de 0 y 1
4. Incluir en f términos de cada orden
5. Que la semilla determine algún término de f

Combinadores no Lineales

- If gcd (L_i, L_j) = 1 and $C_1(D)$ ... $C_M(D)$ irreducibles entonces: $CL(S_1,...S_M) = F(L_1,...,L_M)$

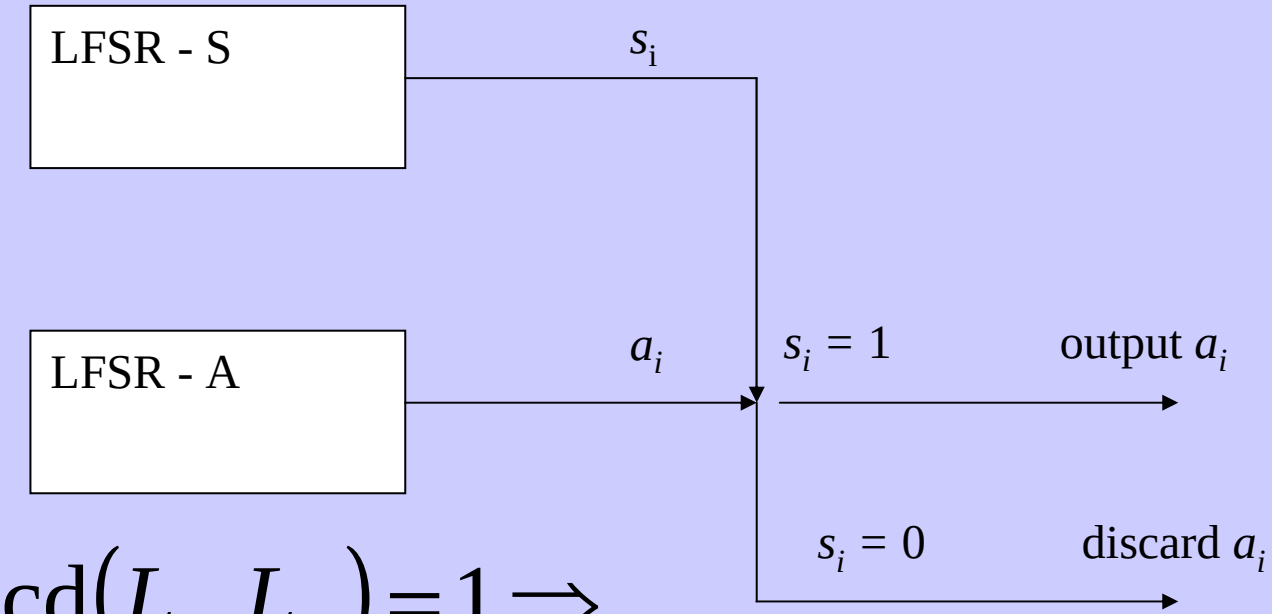
1. Simple combinación de varios registros (con relojes sincronizados)
2. De control paso a paso (un registro controla el reloj de los demás) (Beth-Piper, Gollman, Bilateral)
3. Multi-reloj (los relojes de los registros con independientes) (Massey-Rueppel)

Combinadores no Lineales: Generador de Hadamard



$$if\ \gcd(L_1, L_2) = 1 \Rightarrow \begin{cases} T = mcm(T_1, T_2) \\ CL = L_1 \cdot L_2 \end{cases}$$

Combinadores no Lineales: Generador Shrinking



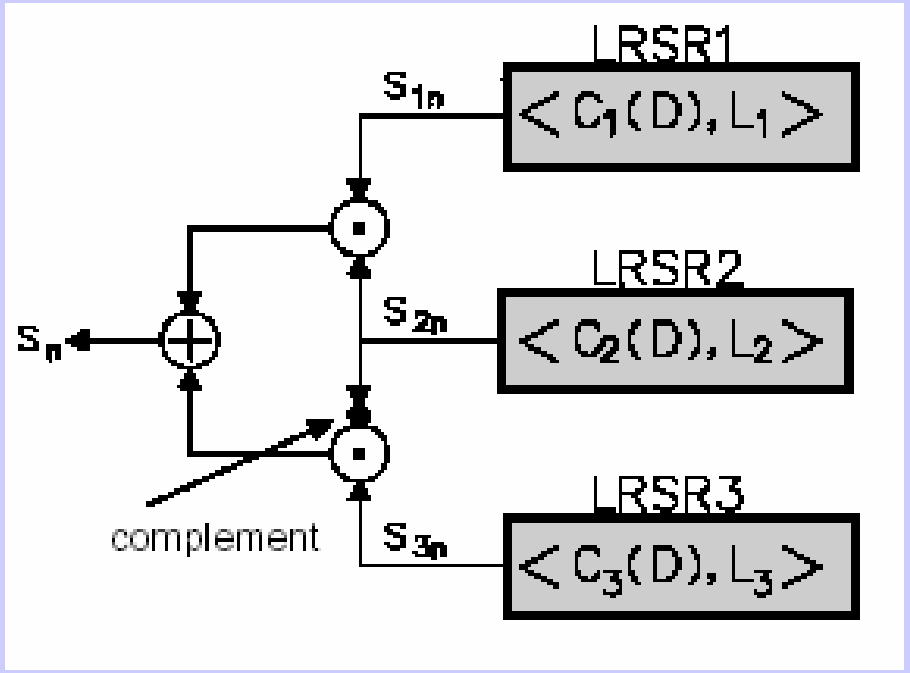
$$if\ \gcd(L_s, L_A) = 1 \Rightarrow$$

$$T = (2^{L_A} - 1) \cdot 2^{L_S - 1}$$

$$L_A \cdot 2^{L_S - 2} < CL \leq L_A \cdot 2^{L_S - 1}$$

Combinadores no Lineales: Generador de Geffe

$$F(x_1, x_2, x_3) = x_1 x_2 \oplus x_2 x_3 \oplus x_3$$



- Si L_i son primos entre sí y los polinomios son primitivos:

$$CL = L_1 L_2 + L_2 L_3 + L_3$$

$$T = (2^{L_1} - 1) \cdot (2^{L_2} - 1) \cdot (2^{L_3} - 1)$$

Combinadores no Lineales: Generador de Geffe

x_1	x_2	x_3	$z = F(x_1,x_2,x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Balanceado (1º y 2º postulados de Golomb), pero no 3º porque

$$P(z = x_1) = 3/4.$$

Combinadores no Lineales: Generador de Beth-Piper

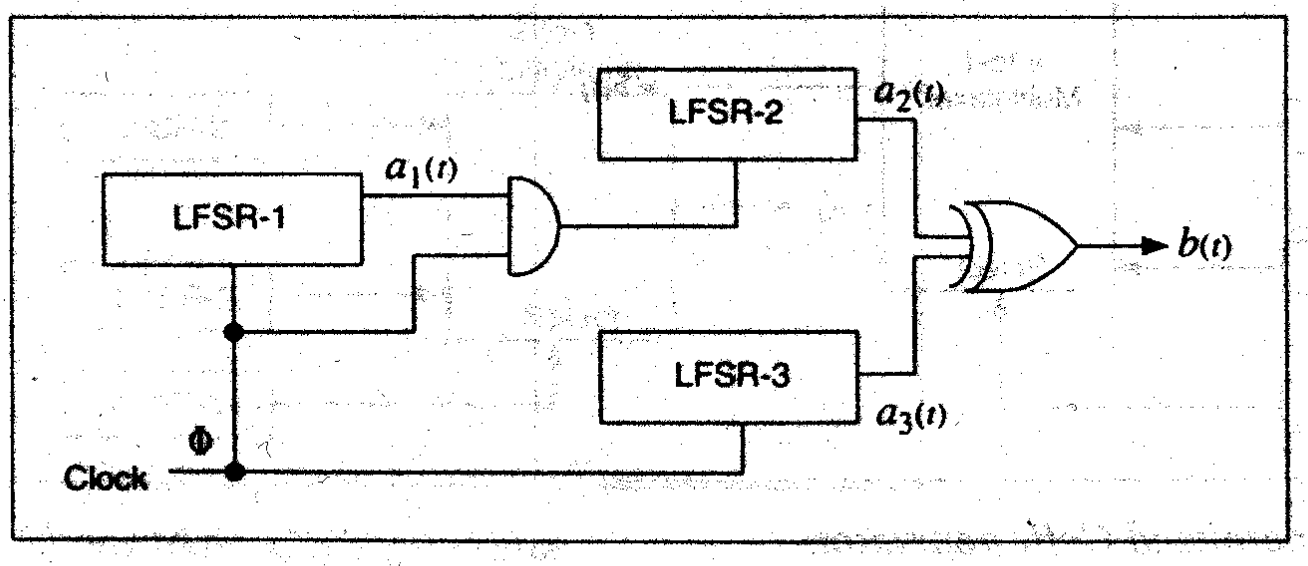


Figure 16.9 Beth-Piper stop-and-go generator.

$$CL = (2^{L_1} - 1) \cdot L_2 + L_3$$

$$T = (2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1)$$

$$P(b(t) + b(t+1) = a_3(t) + a_3(t+1)) = 3/4$$

Combinadores no Lineales: Generador Bilateral

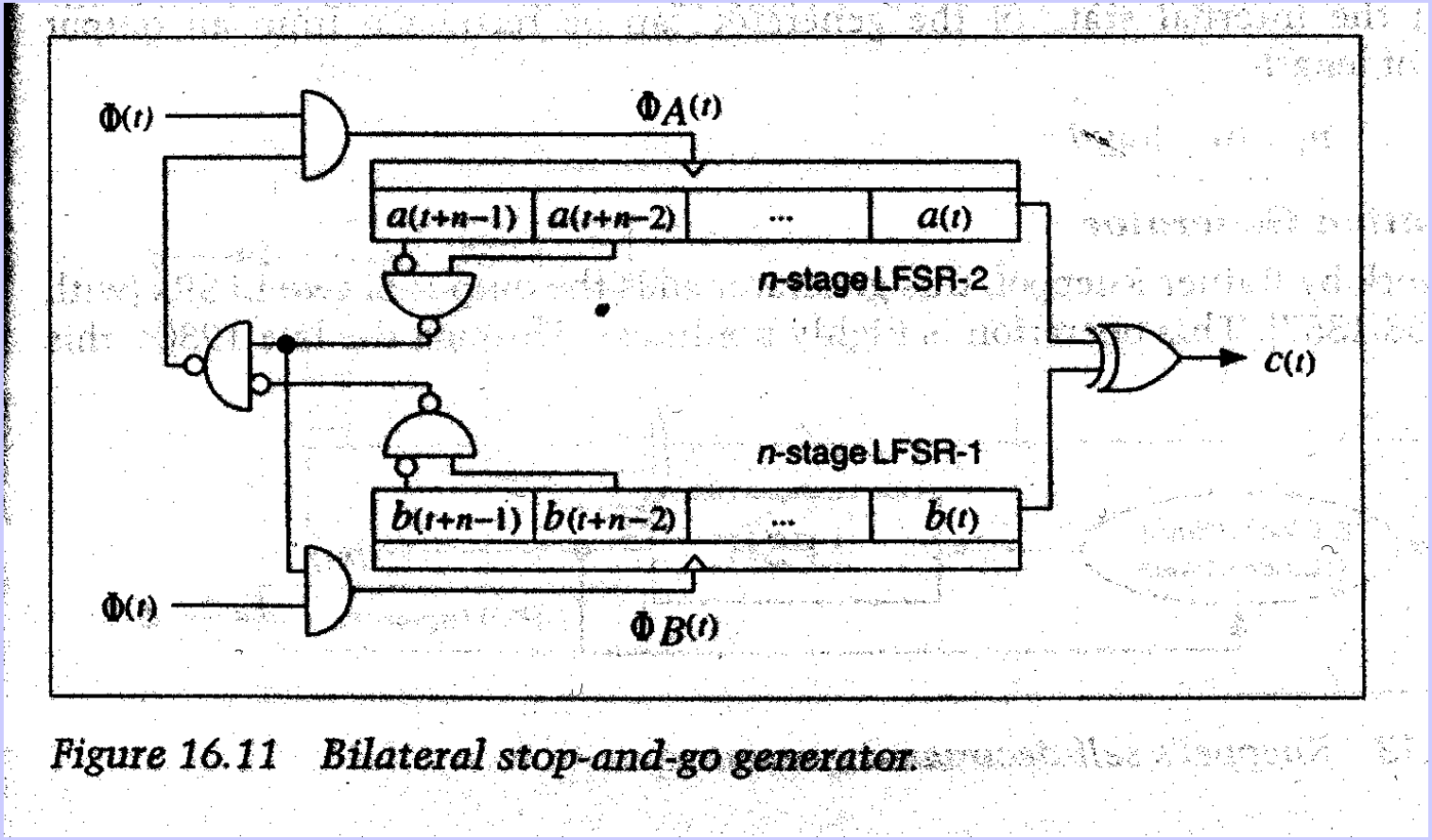
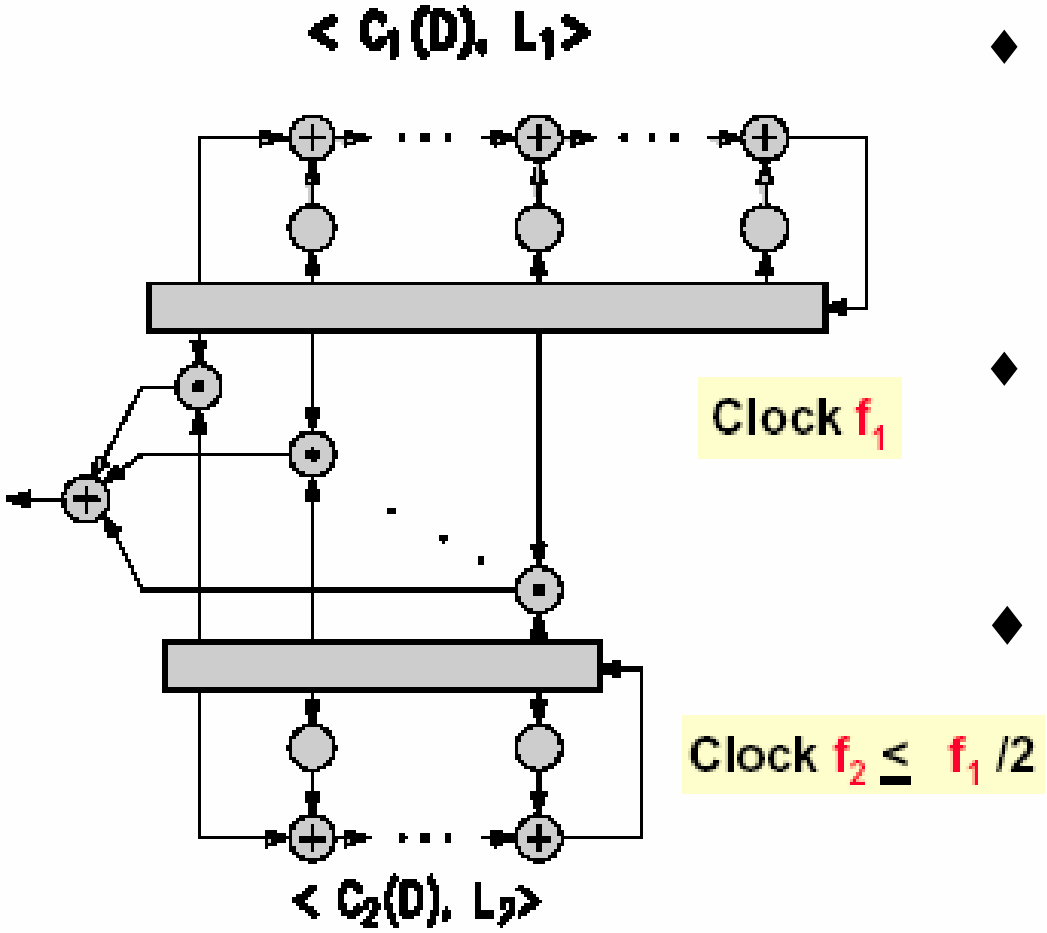


Figure 16.11 Bilateral stop-and-go generator.

$$CL \approx T \approx (5 \bullet 2^{L-2}) - 1$$

Combinadores no Lineales: Generador de Massey-Rueppel



- Combinación que suma los productos internos de cada bit con los dos estados
- Los RDRL pueden tener diferentes tasas de reloj
- Si $\gcd(L_1,L_2)=1$, C_1, C_2 primitivos y $L_2 \leq L_1$:
  - $CL = L_1.L_2$
  - $T = \mathrm{lcm}(T_1, T_2)$

Combinadores no Lineales: Generador en Cascada

- Su ventaja está en su diseño modular y repetitivo, permitiendo la concatenación de un número indefinido de generadores, obteniendo de esta manera enormes periodos y altas complejidades lineales.
- Se recomienda utilizar no menos de quince generadores en cascada.

Combinadores no Lineales: Generador de Gollmann

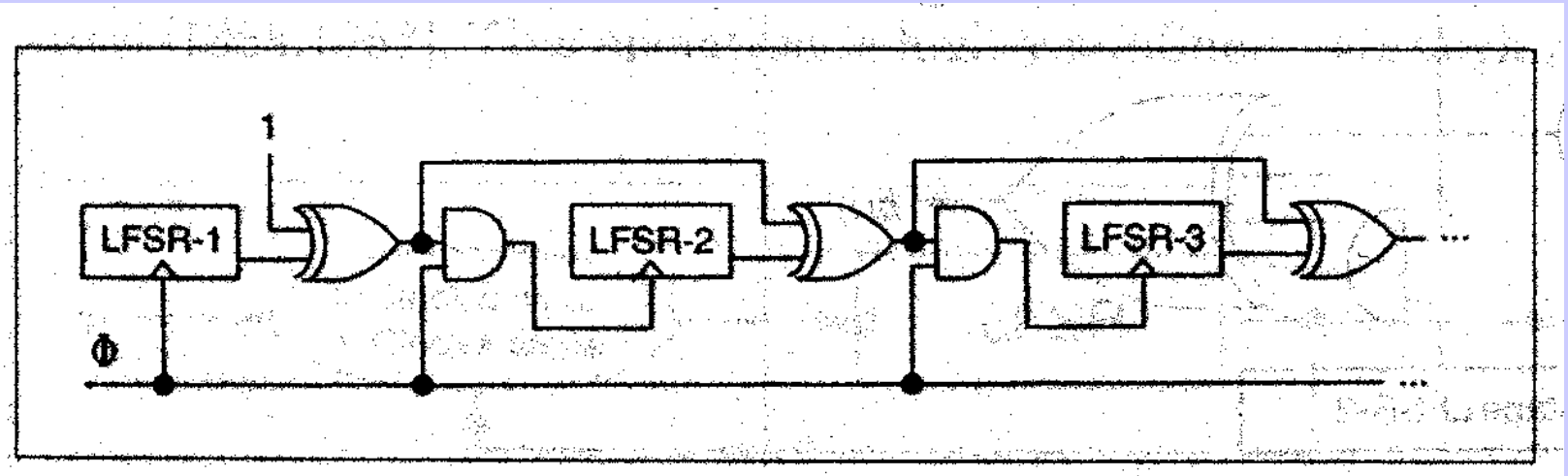


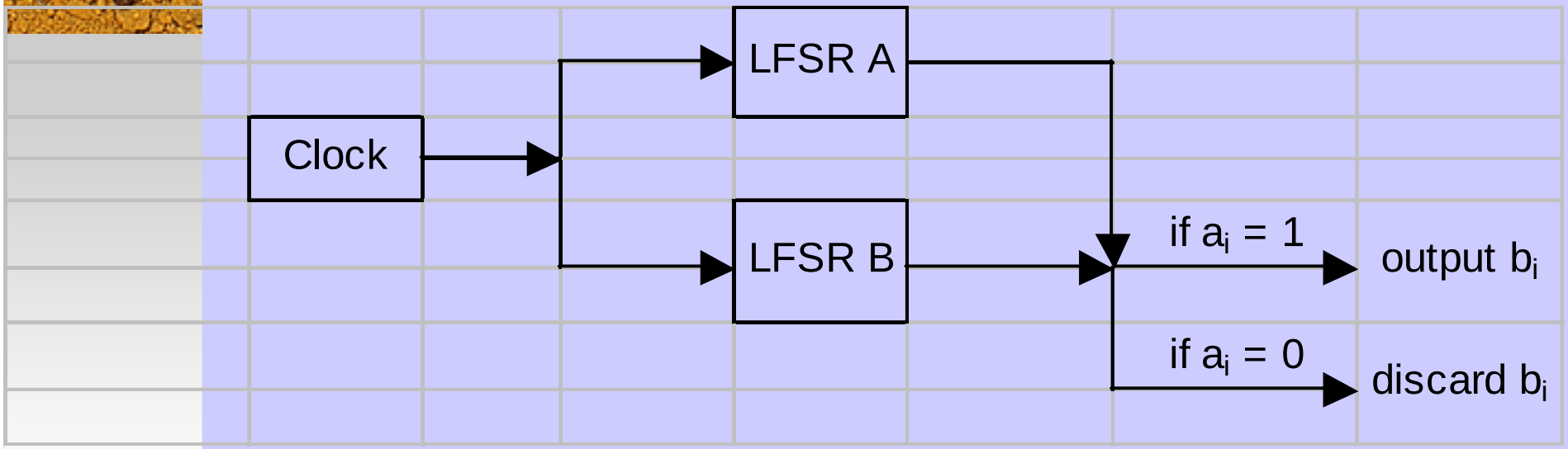
Figure 16.16 Gollmann cascade.

$$CL \geq L(2^L - 1)^{m-1}$$

$$T = (2^L - 1)^m$$

Combinadores no Lineales: Generador Stop-and-Go

- el reloj del subgenerador es controlado por otro subgenerador



Combinadores no Lineales: Generador Umbral

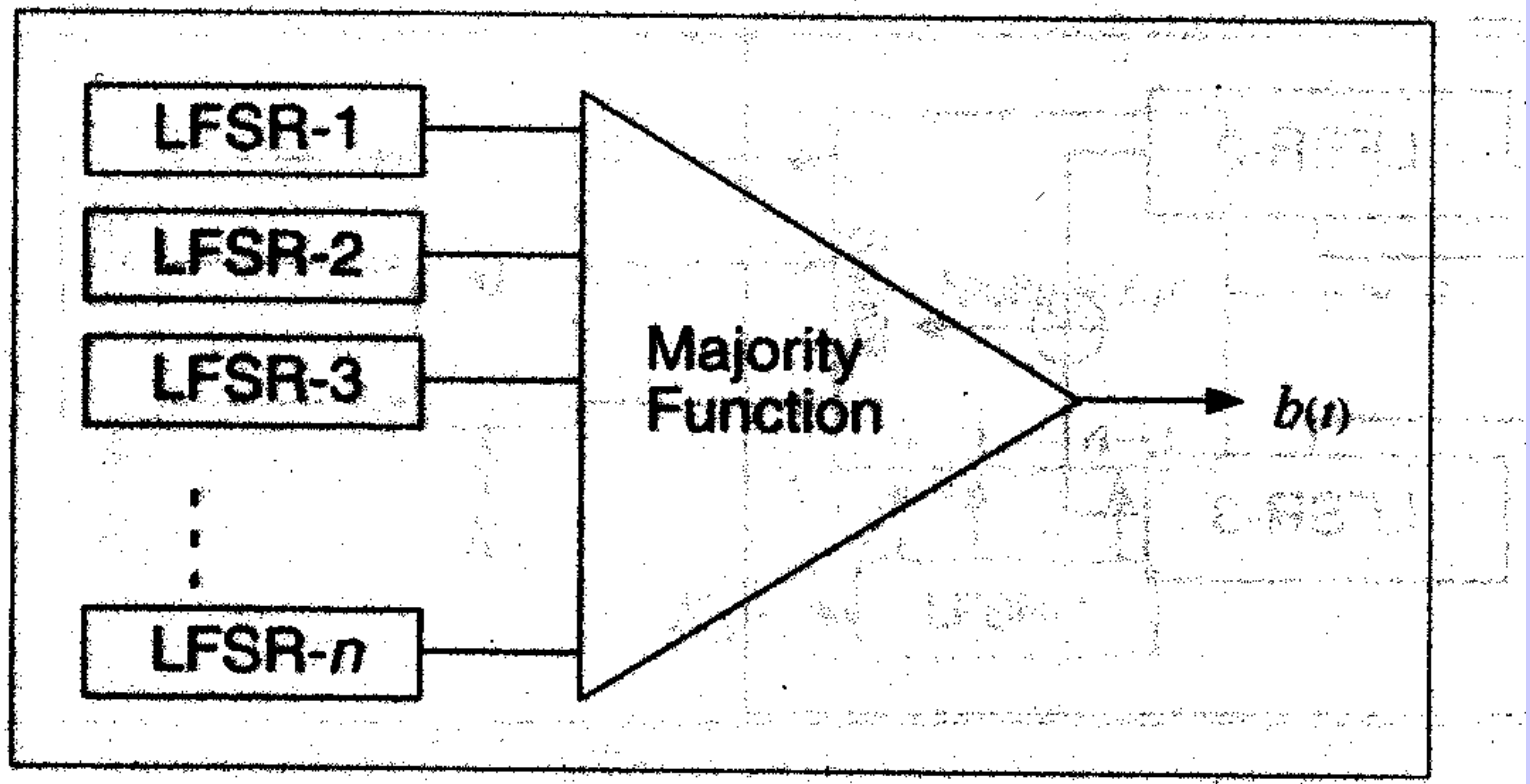
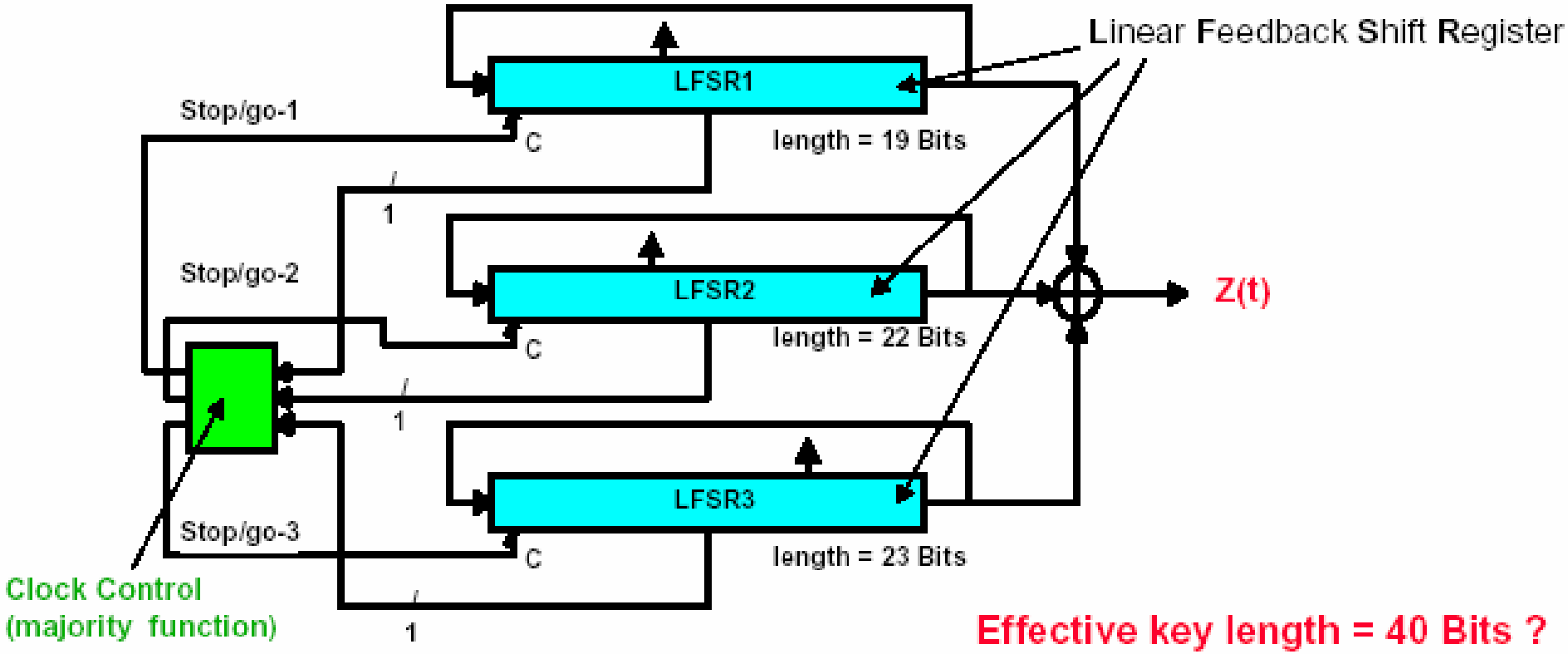


Figure 16.12 Threshold generator.

Combinadores no Lineales: Generador A5 para GSM

Clave de 64 bits: Semillas de los LFSR

LSFRs conocidos de periodos $2^{19}-1$, $2^{22}-1$, $2^{23}-1$

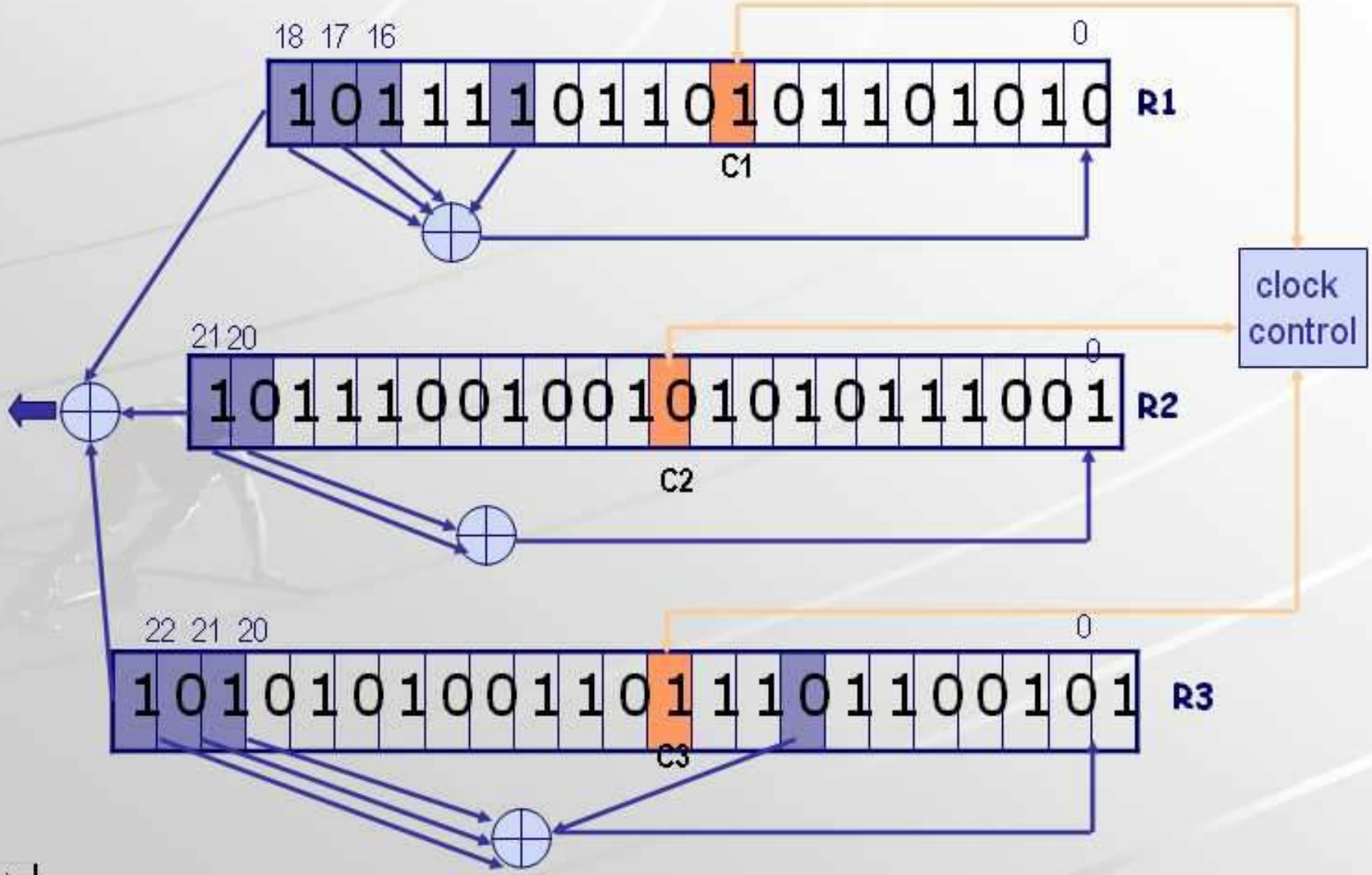


Combinadores no Lineales: Generador A5 para GSM

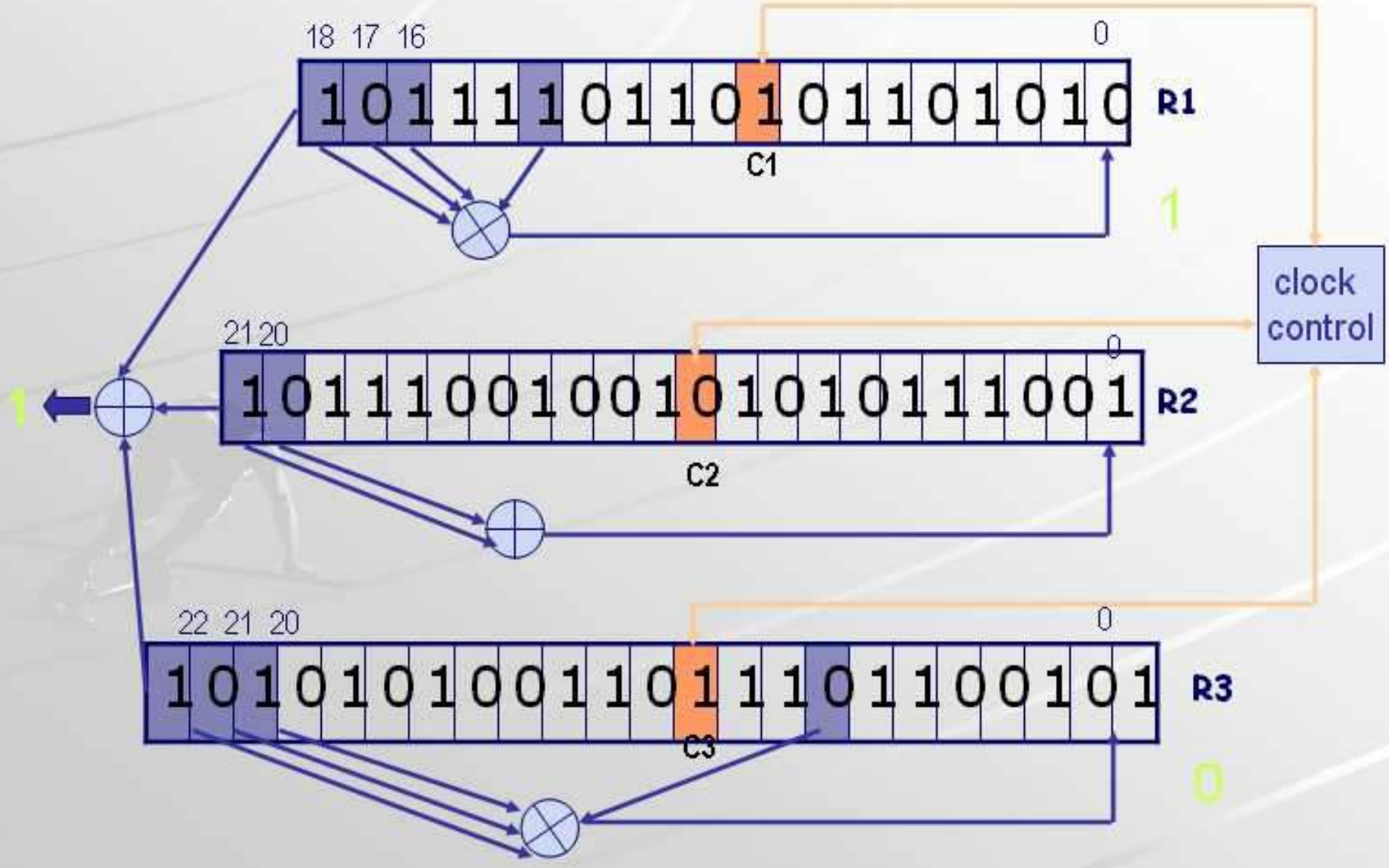
Función Mayoría f:

$f(a(t+11),b(t+12),c(t+13)) = (y_1,y_2,y_3)$	$a(t+11)$	$b(t+12)$	$c(t+13)$
(1,1,1)	0	0	0
	1	1	1
(1,1,0)	0	0	1
	1	1	0
(0,1,1)	0	1	1
	1	0	0
(1,0,1)	1	0	1
	0	1	0

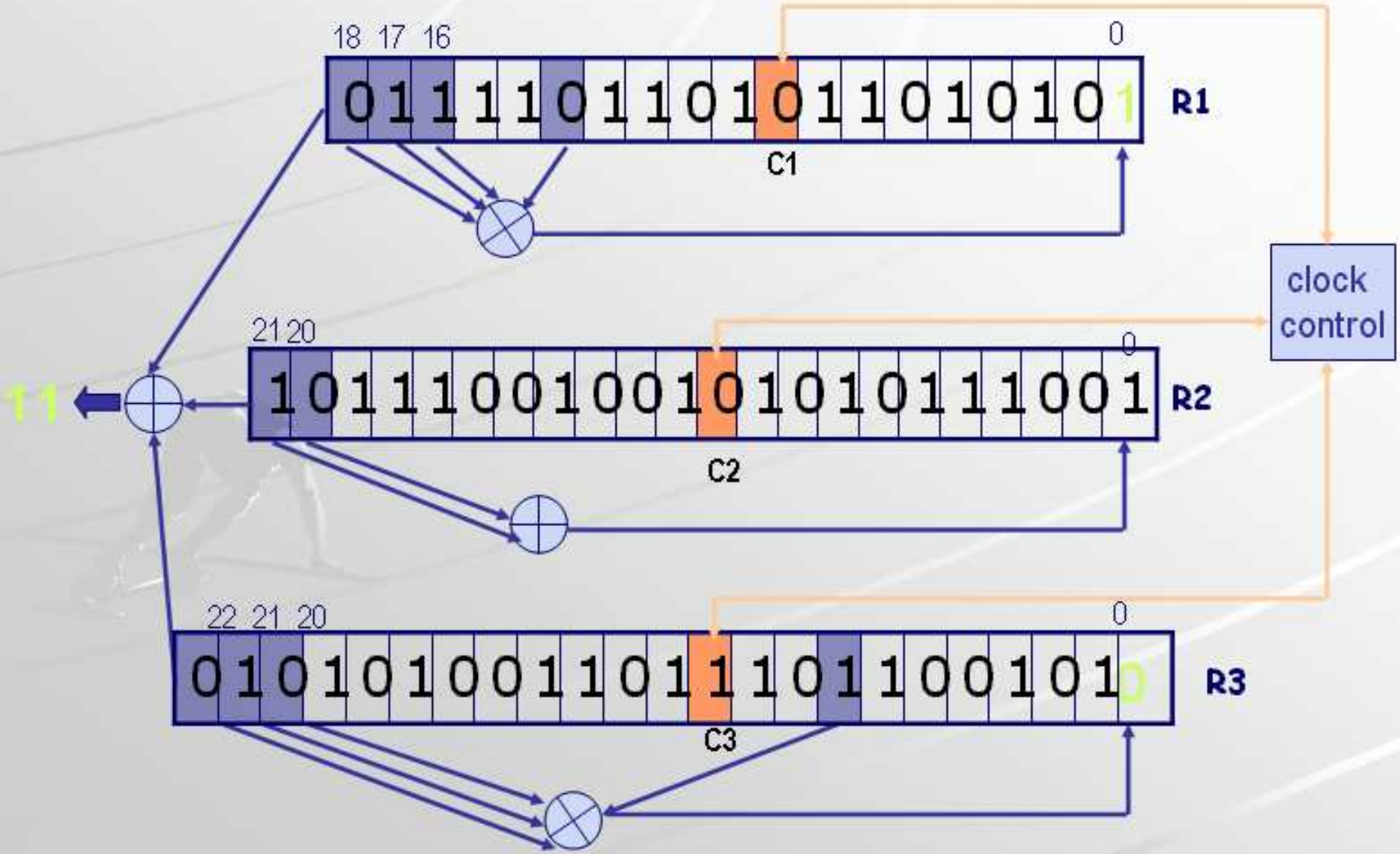
Cifrado A5 para GSM



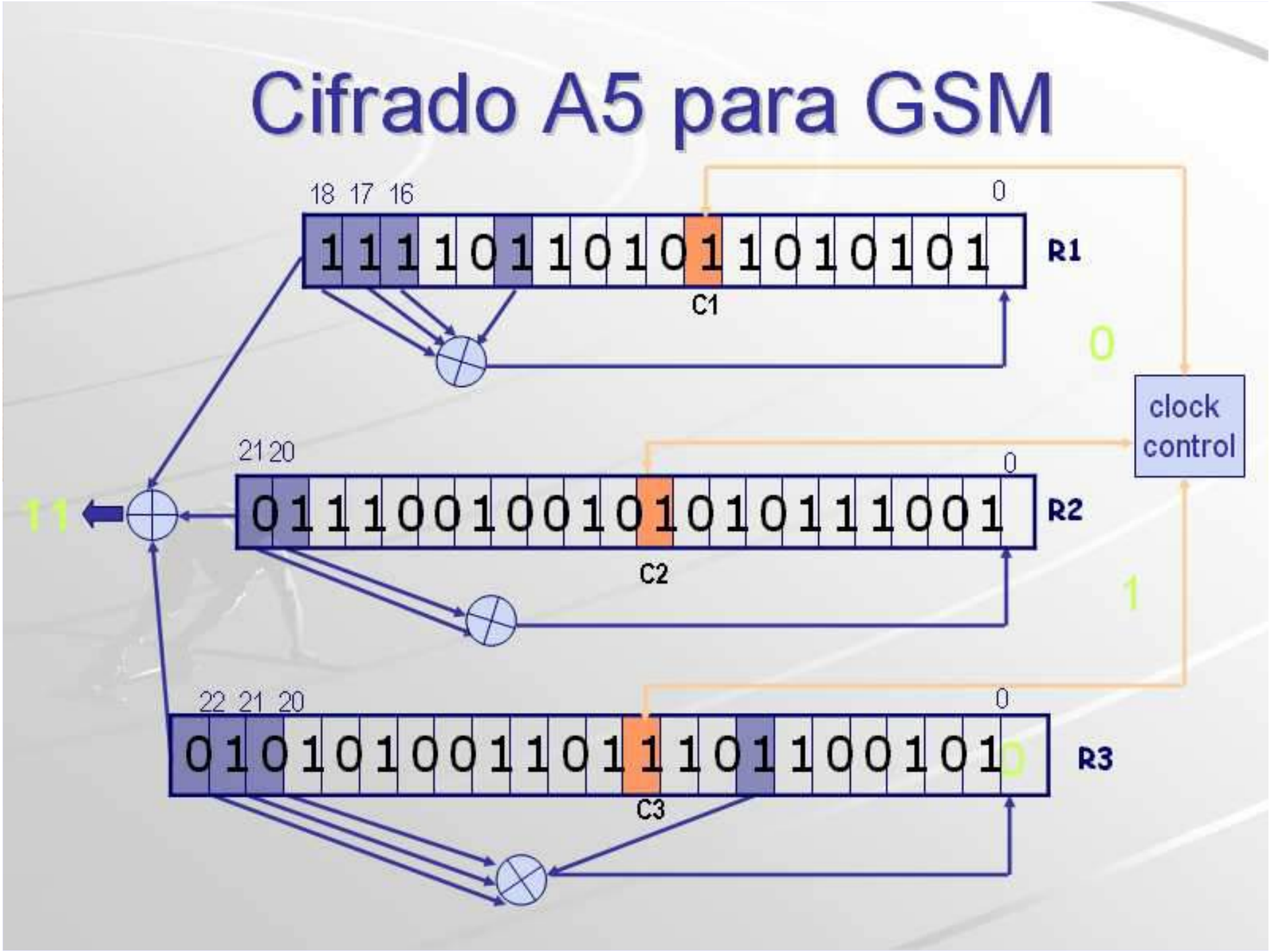
Cifrado A5 para GSM



Cifrado A5 para GSM



Cifrado A5 para GSM



Combinadores no Lineales: Generador A5 para GSM

- Para cifrar transmisiones aéreas entre el MS y el BTS
- Una conversación GSM se envía como una secuencia de 228 bits cada 4,6 miliseg.
- El periodo es de aprox. $(4/3)(2^{23} - 1)$.
- Existe el problema llamado de la colisión, debido a que diferentes semillas de los registros pueden producir la misma clave (el 70% de semillas distintas producen distintas claves)
- Una debilidad del protocolo de seguridad de GSM es que la identidad de los móviles no siempre se cifra en la transmisión aérea
- Puede romperse con un PC en pocas horas (Con un Pentium III comprobar 2^{54} claves requiere 250 horas)
- Se pueden hacer ataques por denegación de servicio

Combinador no Lineal

- Principios de diseño:

1. Usar registros con polinomios primitivos para lograr periodo máximo
2. Usar longitudes de registros primas entre sí para garantizar complejidad lineal grande
3. Incluir en F términos de cada orden para garantizar buena distribución de 0 y 1
4. …

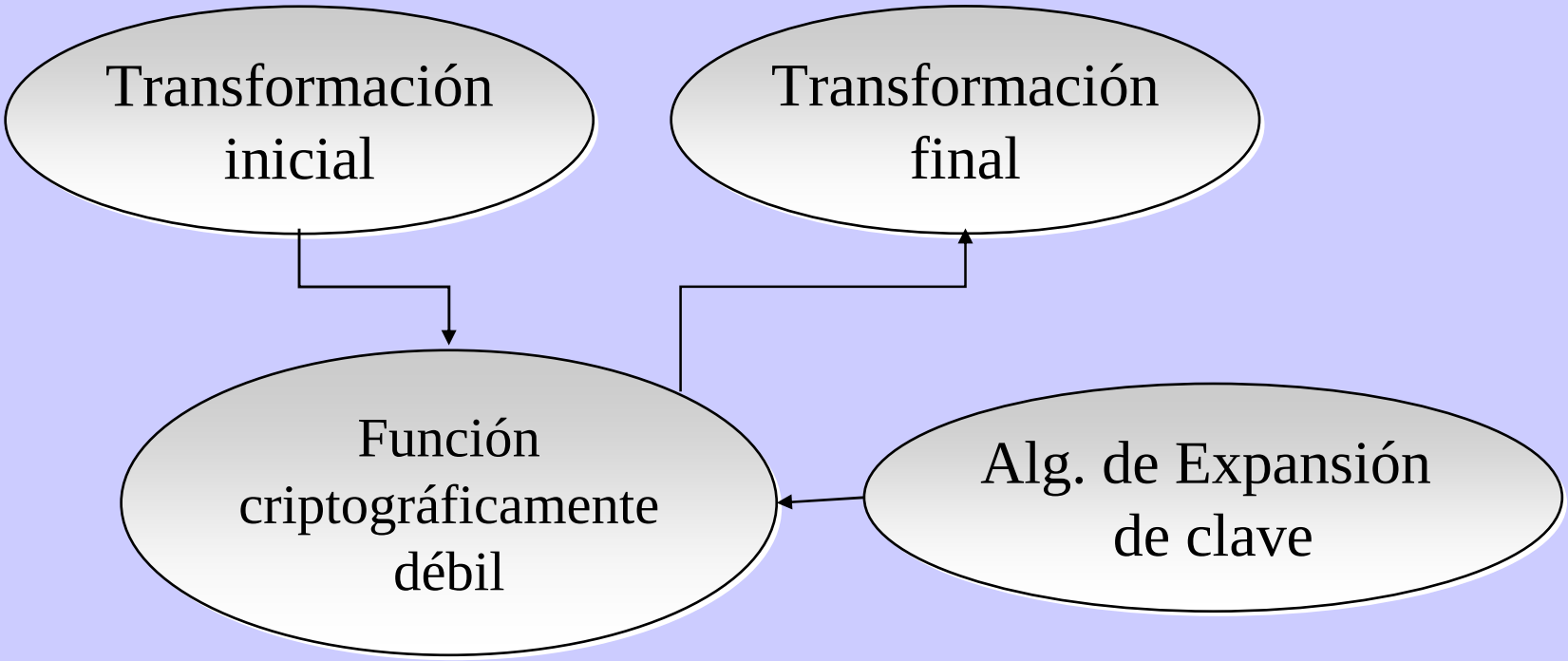
Cifrado en Bloque

El mensaje en claro se divide en bloques y se aplica el algoritmo sobre cada uno de forma independiente con la misma clave de manera que:

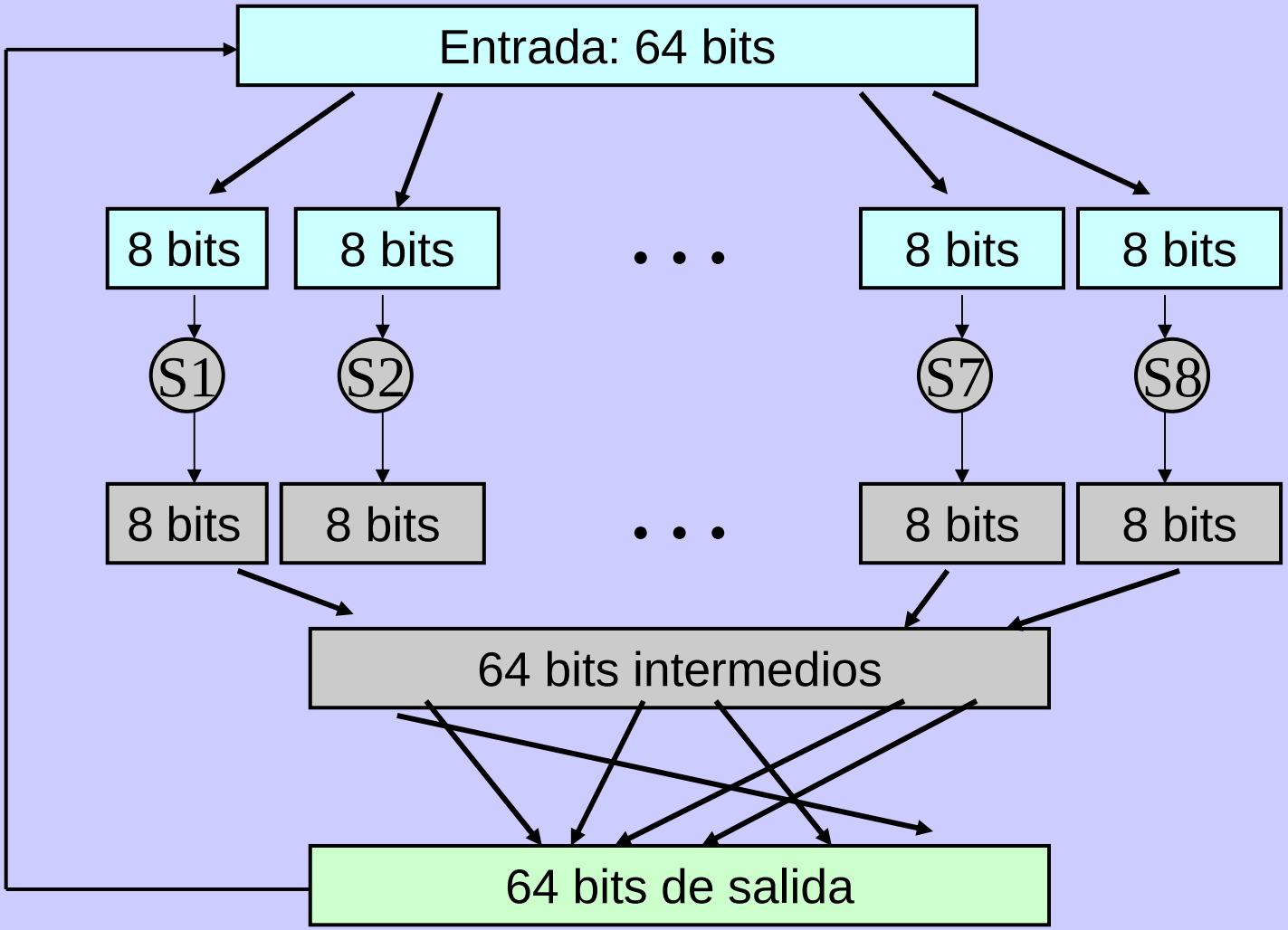
- –el cifrado de cada símbolo depende de los demás símbolos del mismo bloque,
- –para descifrar una parte no es preciso descifrar el mensaje completo.

Cifrado en Bloque

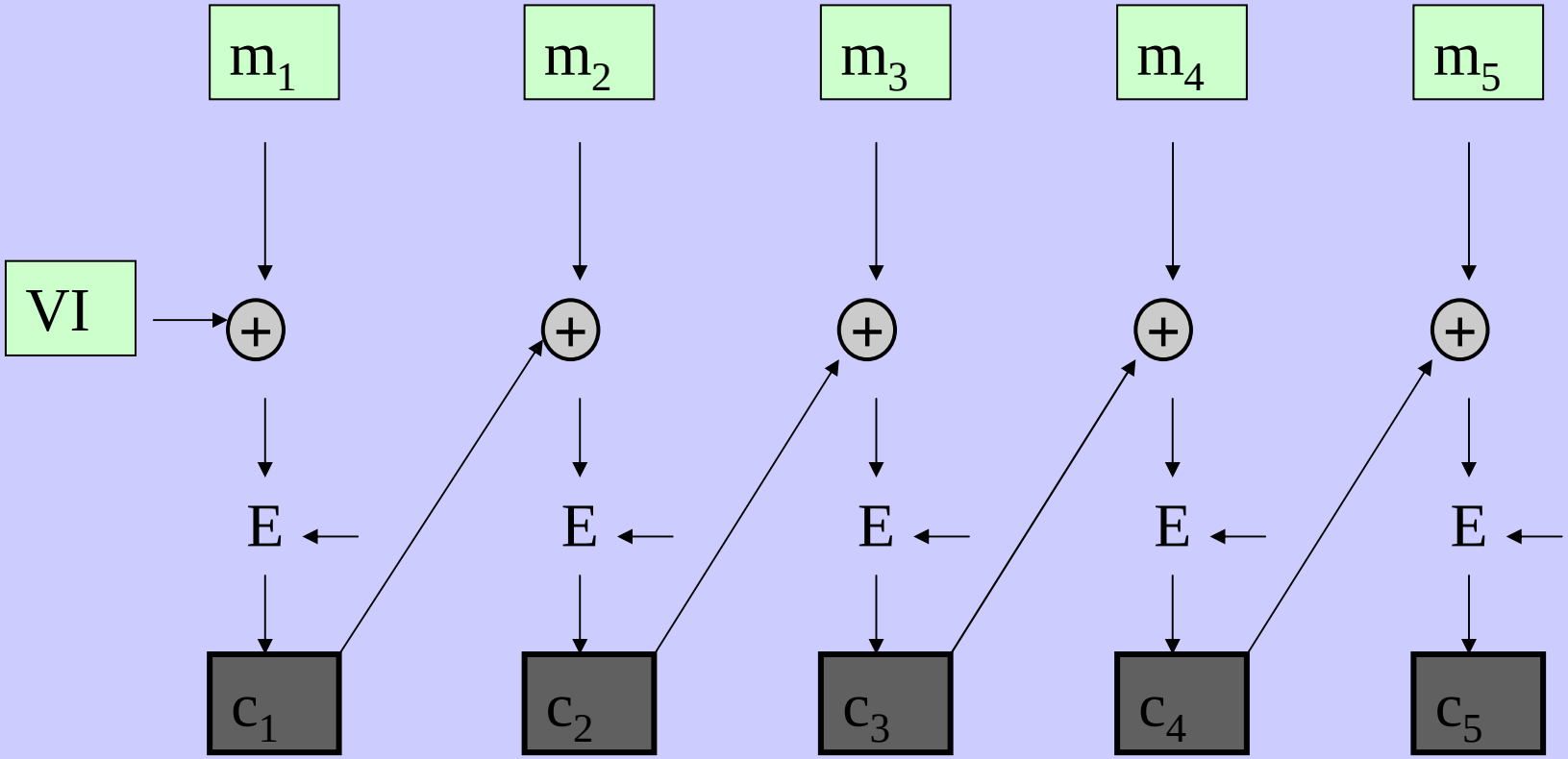
- Elementos básicos:



Cifrado en Bloque



Cifrado en Bloque: CBC

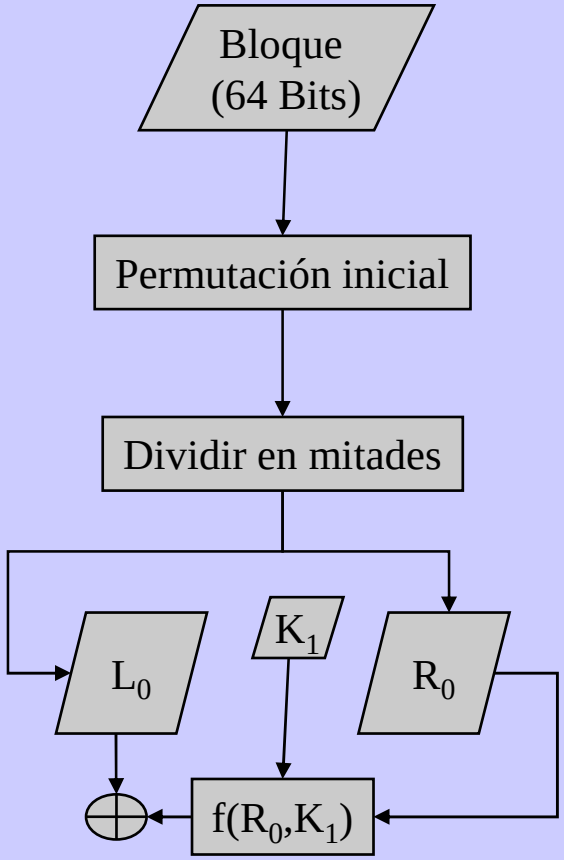


VI⇒vector de inicialización

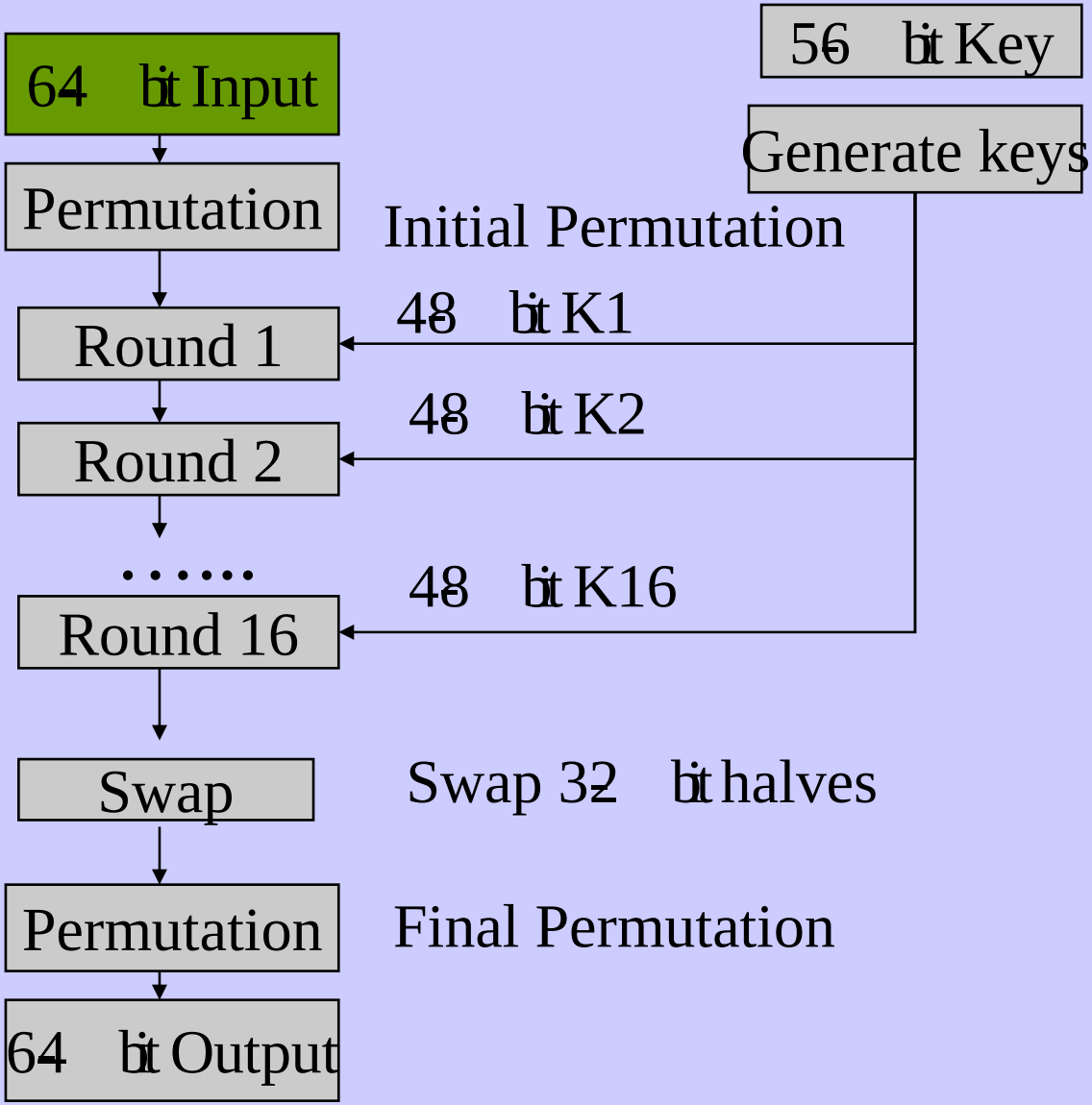
E ⇒función de cifrado

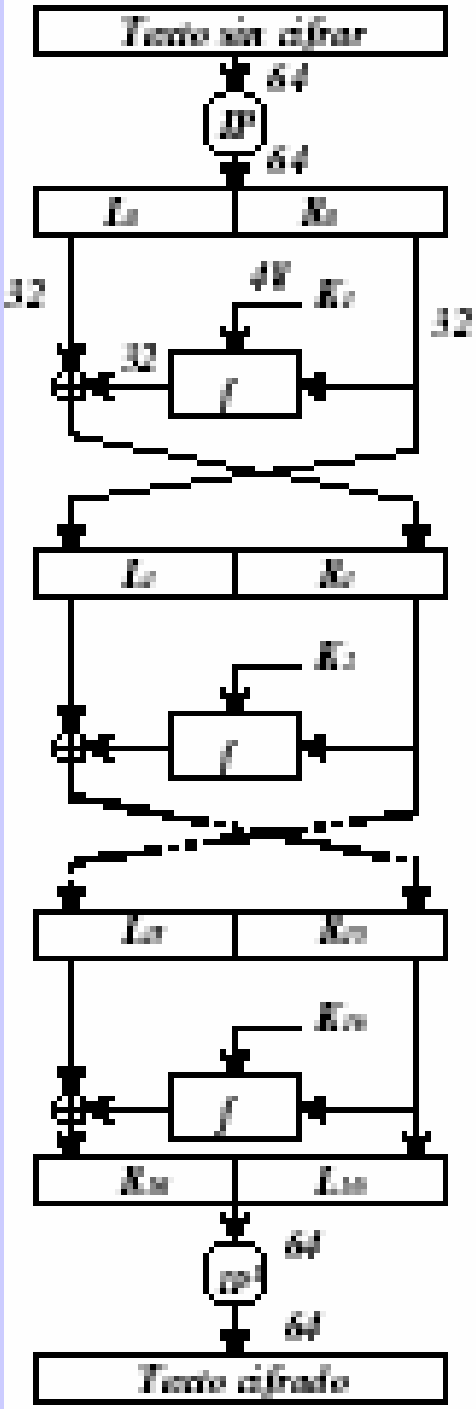
Cifrado en Bloque: DES

- Diseñado por IBM y NSA.
- Estándar en EEUU de 1973 a 2000
- Características:
  - Longitud de bloque: 64 bits
  - Longitud de clave : 56 bits
  - 16 iteraciones
  - Cifrado tipo Feistel



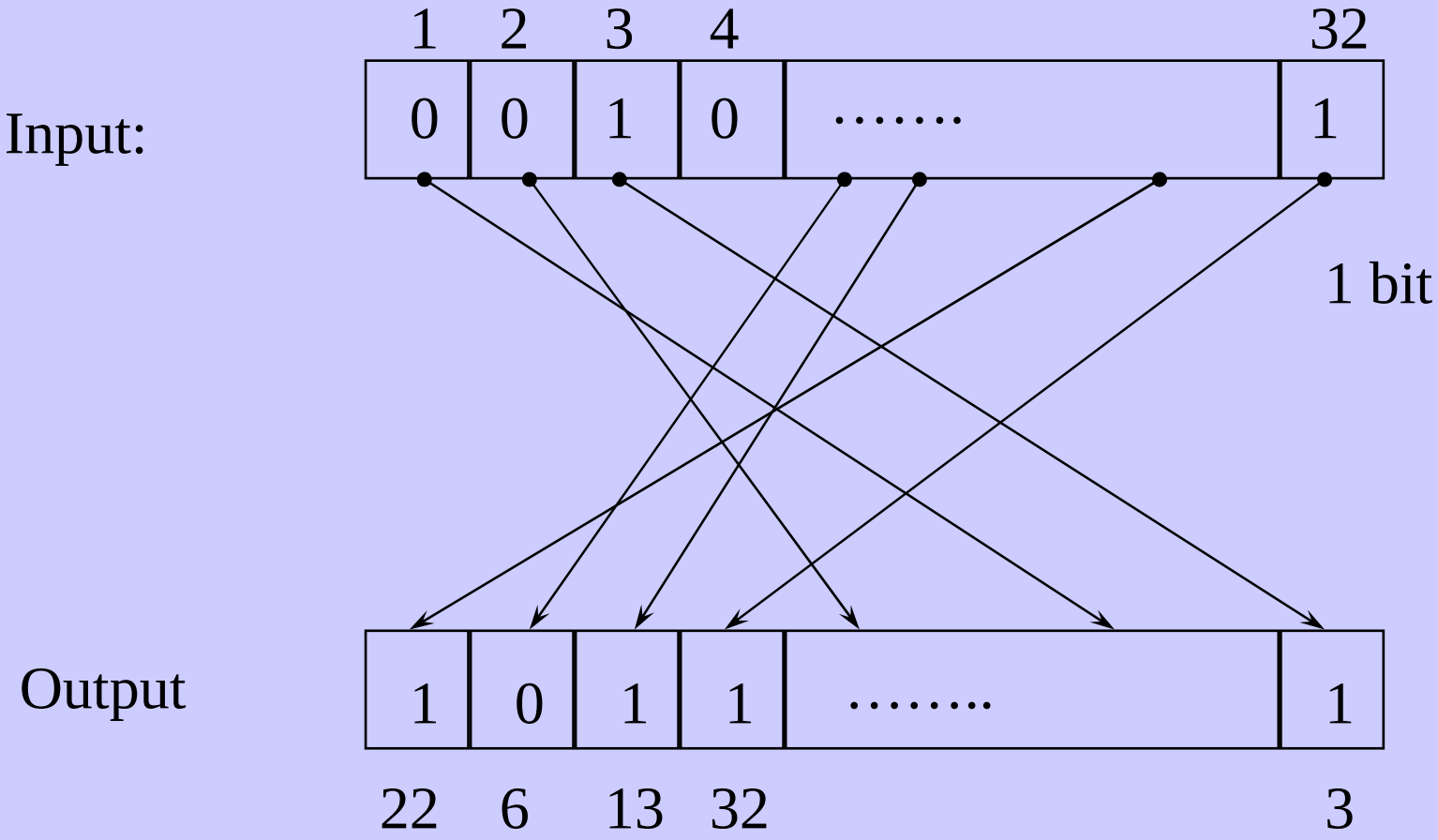
Cifrado en Bloque: DES





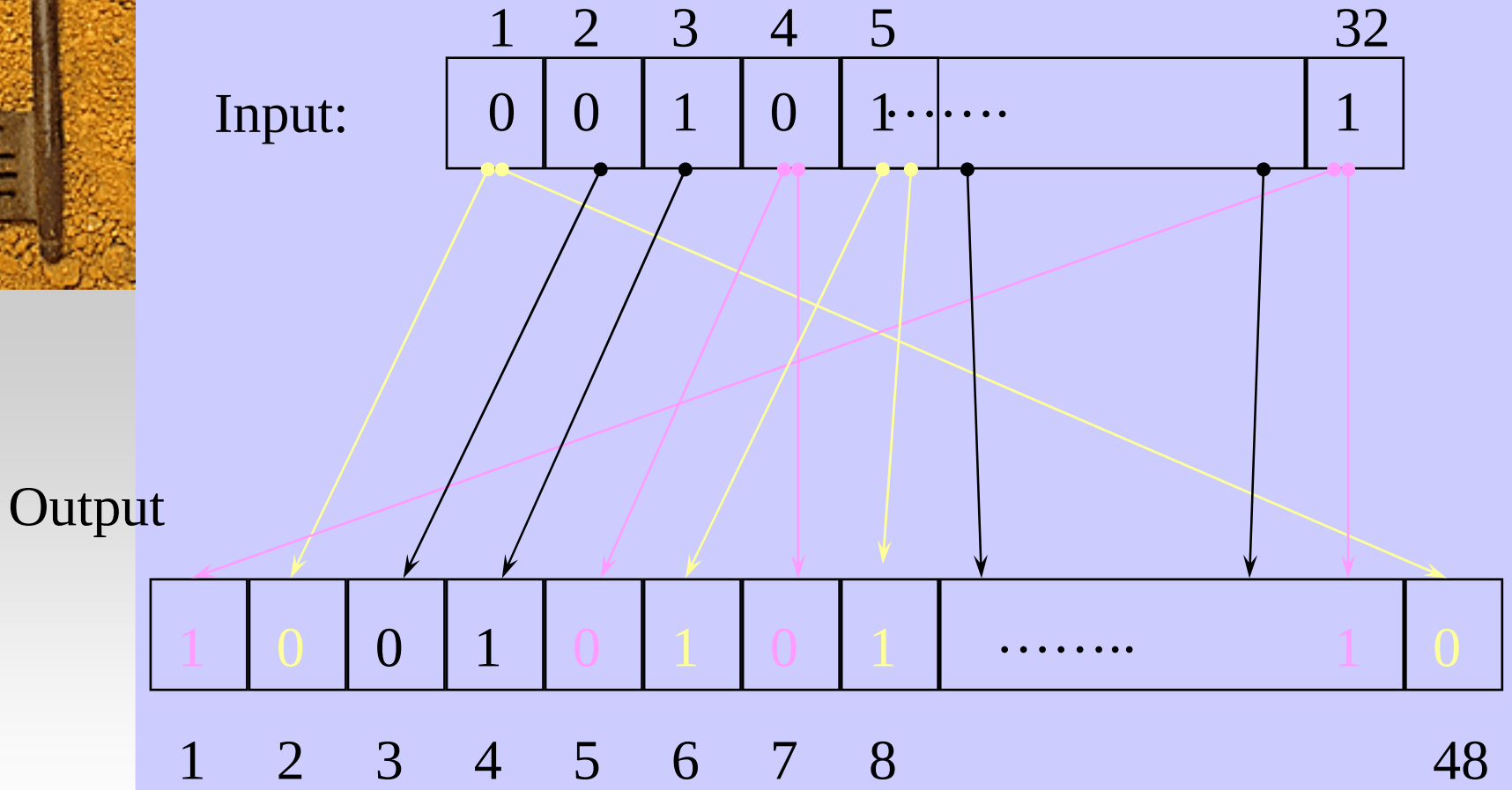
Cifrado en Bloque: DES

Permutación de Bits (1-to-1)



Cifrado en Bloque: DES

Expansión de Bits (1-to-m)



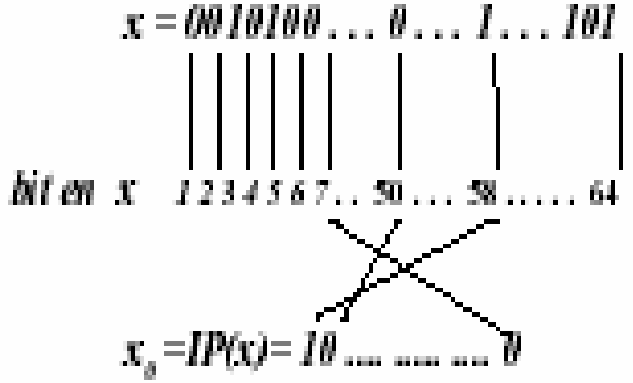
Cifrado en Bloque: DES

Permutaciones Inicial y Final

- Considerar la entrada como una matriz M de 8-byte por 8-bit
- Transformar M en M_1 en dos pasos
  - Transponer fila *x* en columna (9-*x*), 0<*x*<9
  - Aplicar permutación sobre las filas:
    - Cada columna par *y*, se convierte en fila *y*/2
    - Cada columa impar *y*, se convierte en fila (5+*y*/2)
- Permutación final $FP = IP^{-1}$

Permutación inicial IP

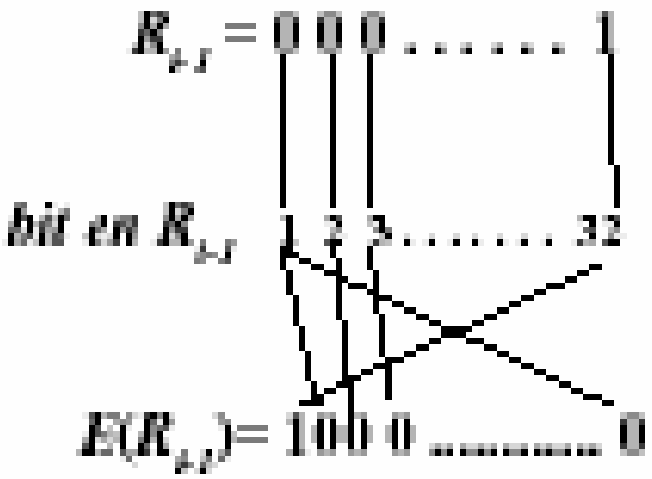
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7



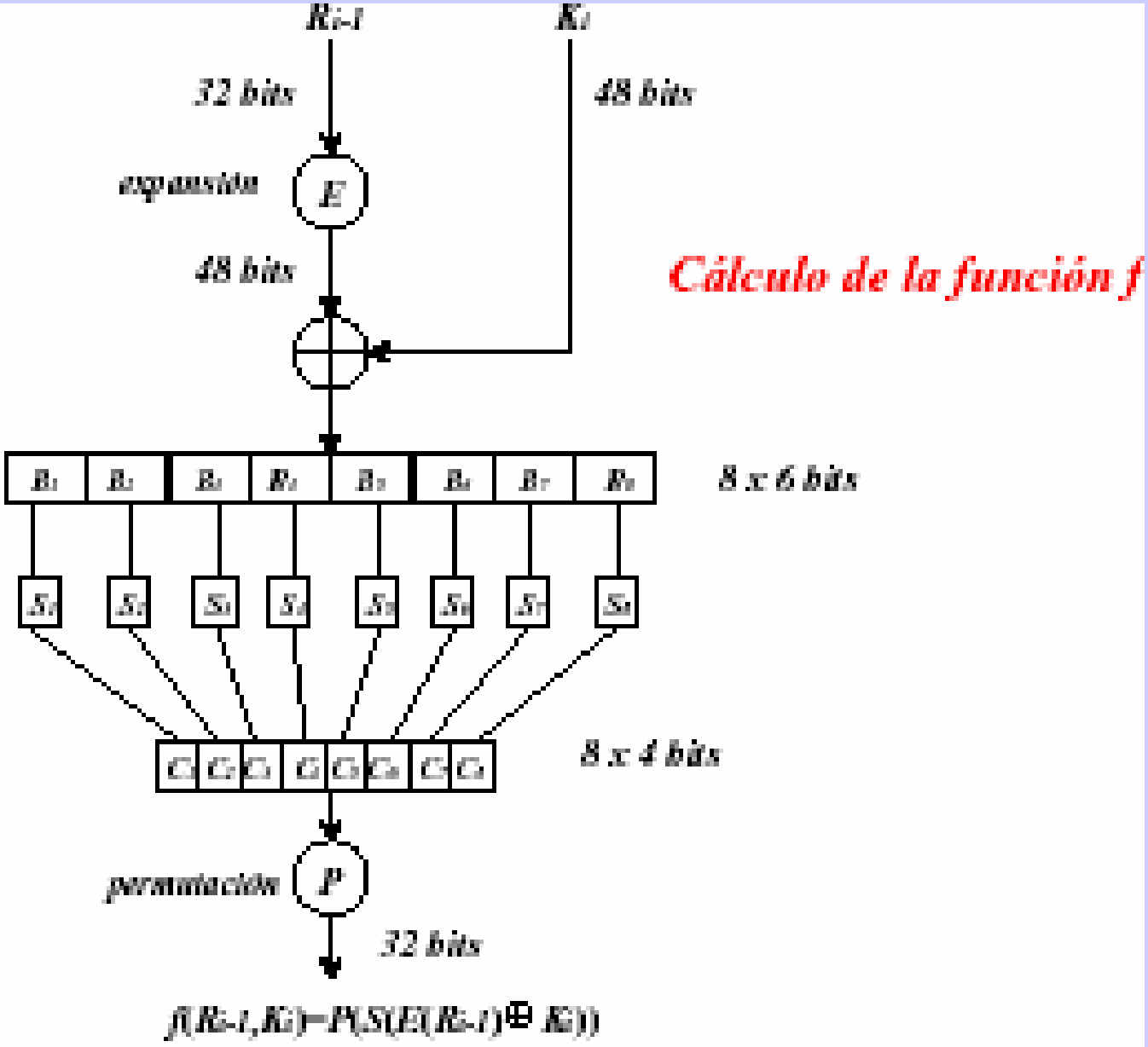
Cifrado en Bloque: DES

Función de expansión E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1



Cifrado en Bloque: DES



Cifrado en Bloque: DES

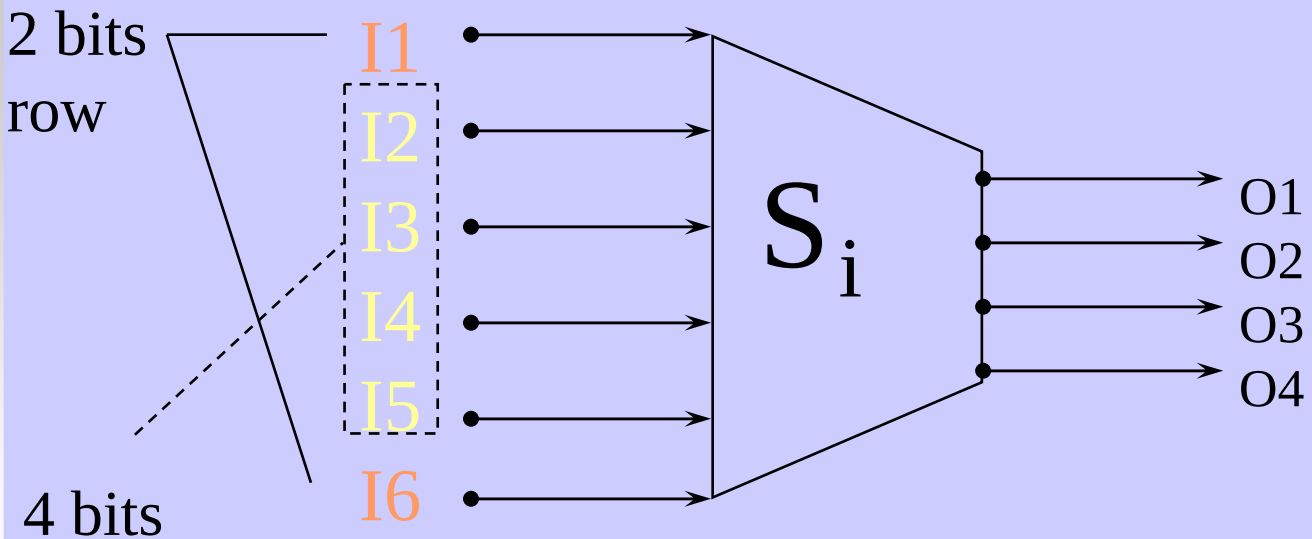
Descifrado

- Aplicar las mismas operaciones con las mismas claves K_i en cada iteración:
  - Entrada: $R_{n+1}|L_{n+1}$
    - Debido a la operación “swap”
  - Salida: $R_n|L_n$
    - La operación swap devuelve al final el resultado correcto: L|R

Cifrado en Bloque: DES

8 Cajas S

- 48 bits ==> 32 bits. (8*6 ==> 8 *4)
- 2 bits se usan para seleccionar las permutaciones para el resto de los 4 bits



i = 1,…8.

Cifrado en Bloque: DES

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

5	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

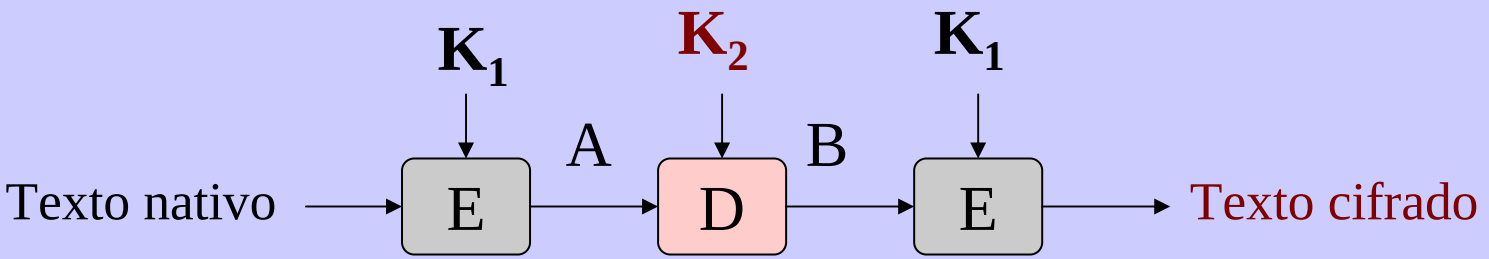
Cifrado en Bloque: DES

- No se puede deducir la clave a partir del mensaje cifrado.
- Existe un criptoanálisis teórico (diferencial) basado en examinar pares de textos cifrados cuyos correspondientes originales tienen diferencias particulares
- Con el tamaño de clave actual NO se considera fiable
  - Por fuerza bruta se puede romper desde 1999
  - En el último reto propuesto por Internet se encontró la clave por fuerza bruta en 21 horas
- Sin embargo sigue siendo muy utilizado, aunque
  - Dadas las mejoras en la tecnología (40% anual) la clave debería crecer 1 bit cada 2 años
  - 128 bits serían suficientes hasta el 2121
- La mejor opción para usar DES se conoce como Triple DES (TDES) que consiste en aplicar DES tres veces.

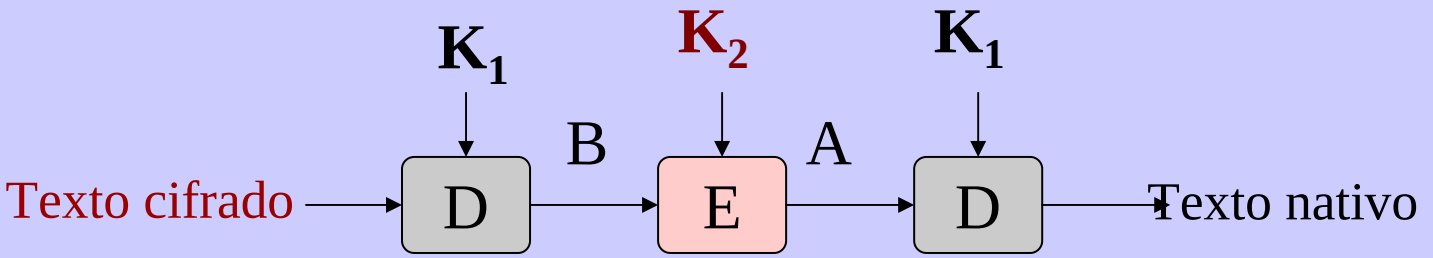
Cifrado en Bloque: Triple DES

- Utiliza dos claves y tres ejecuciones del algoritmo
- Longitud efectiva de la clave 112 bits

CIFRADO



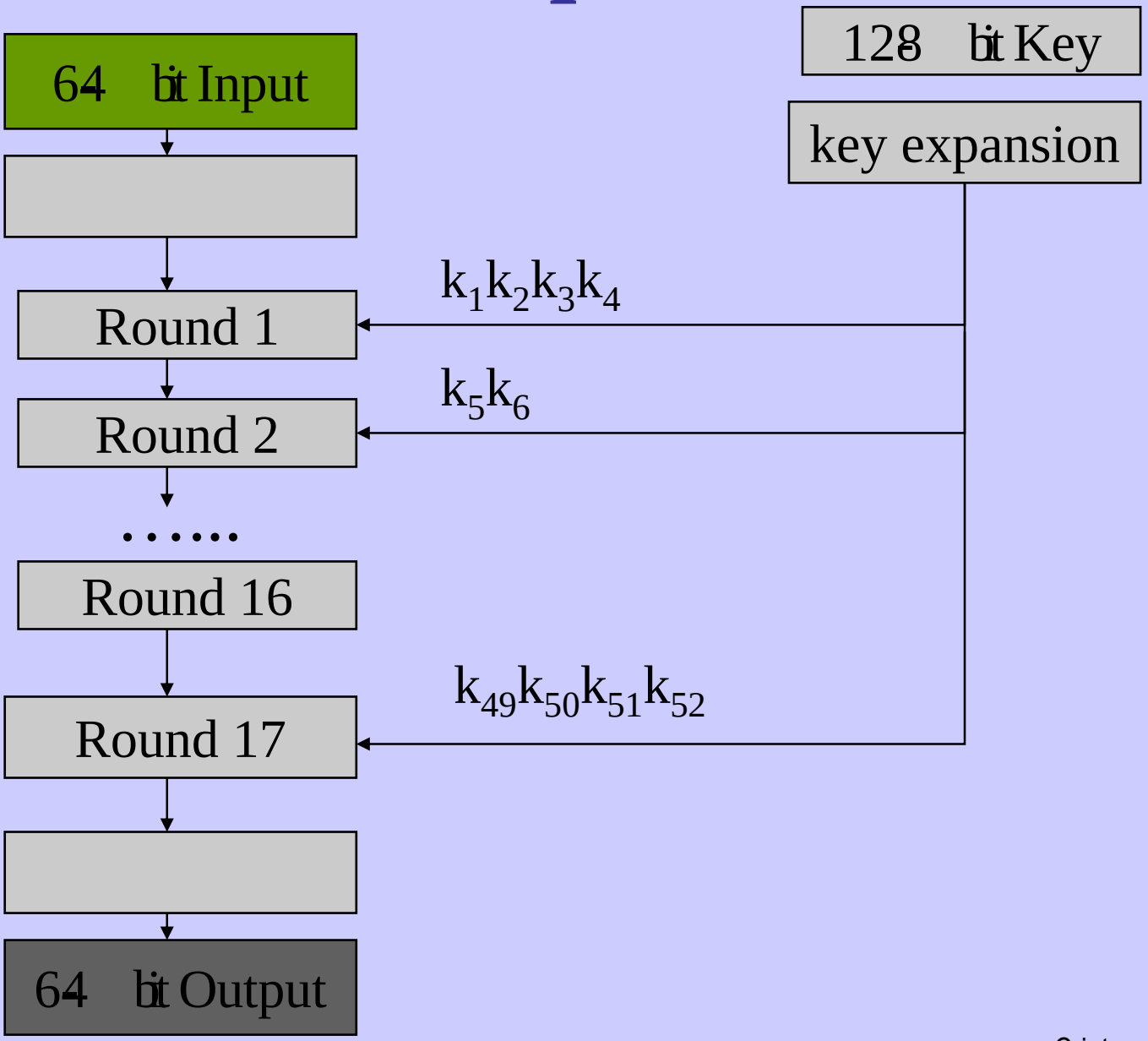
DESCIFRADO



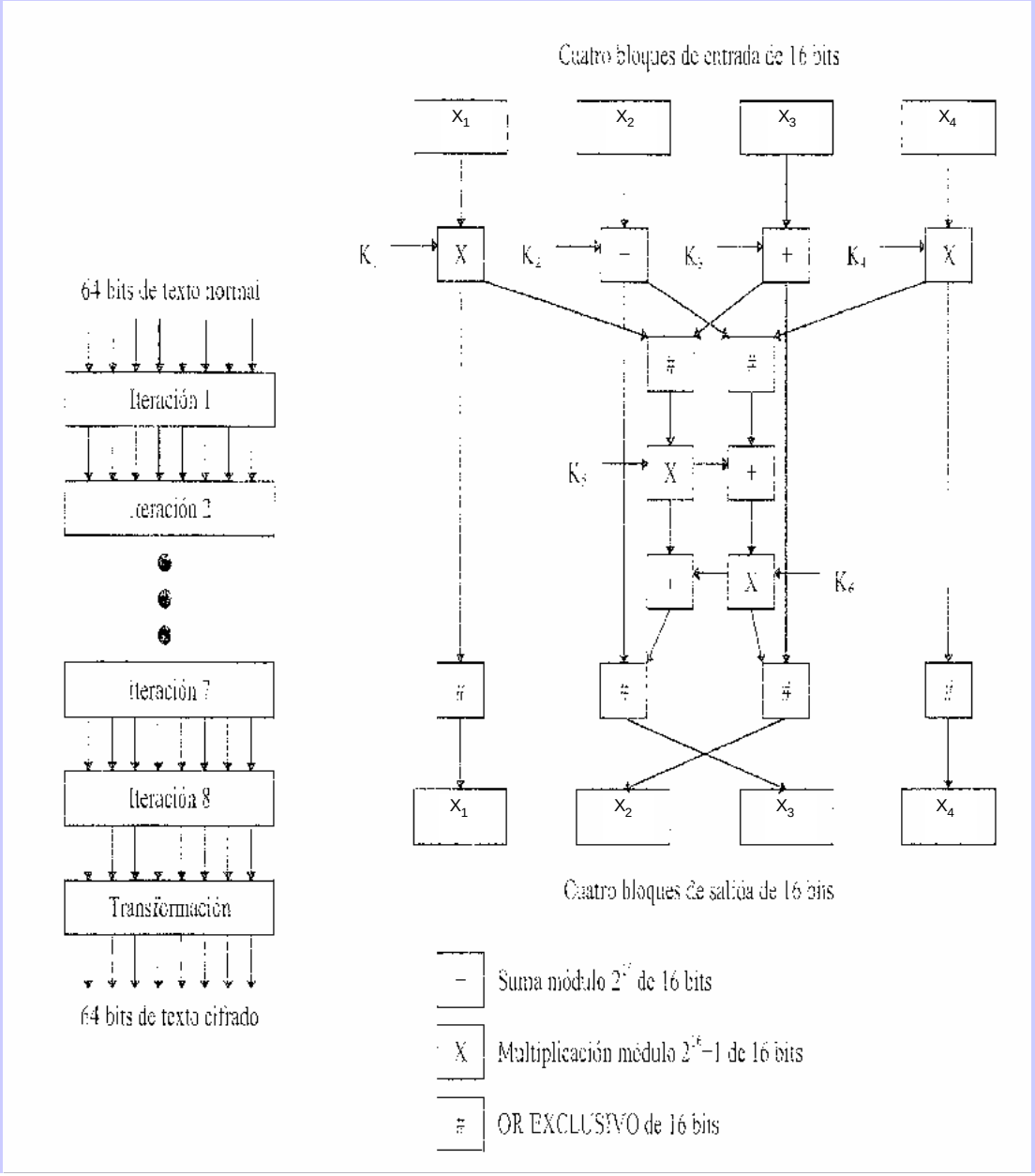
Cifrado en Bloque:IDEA

- International Data Encryption Algorithm (Zurich, 1991)
- Similar a DES: bloques de 64 bits
- Claves de 128 bits: búsqueda exhaustiva: 10^{38} intentos
- Operaciones:
  - 2 16-bit to 1 16-bit
  - $\oplus$
  - + mod 2^{16}
  - $\otimes$ mod $(2^{16} + 1)$
  - Reversible
- Clave inicial de 128 bits para generar 52 claves de 16 bits $K_1,\ldots,K_{52}$
- Espansión de claves en cifrado y descifrado diferentes

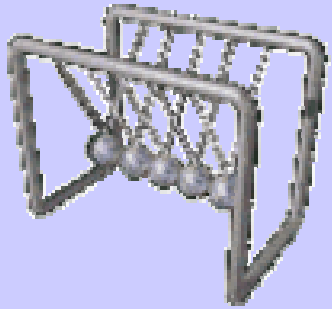
Cifrado en Bloque:IDEA



Cifrado en Bloque:IDEA



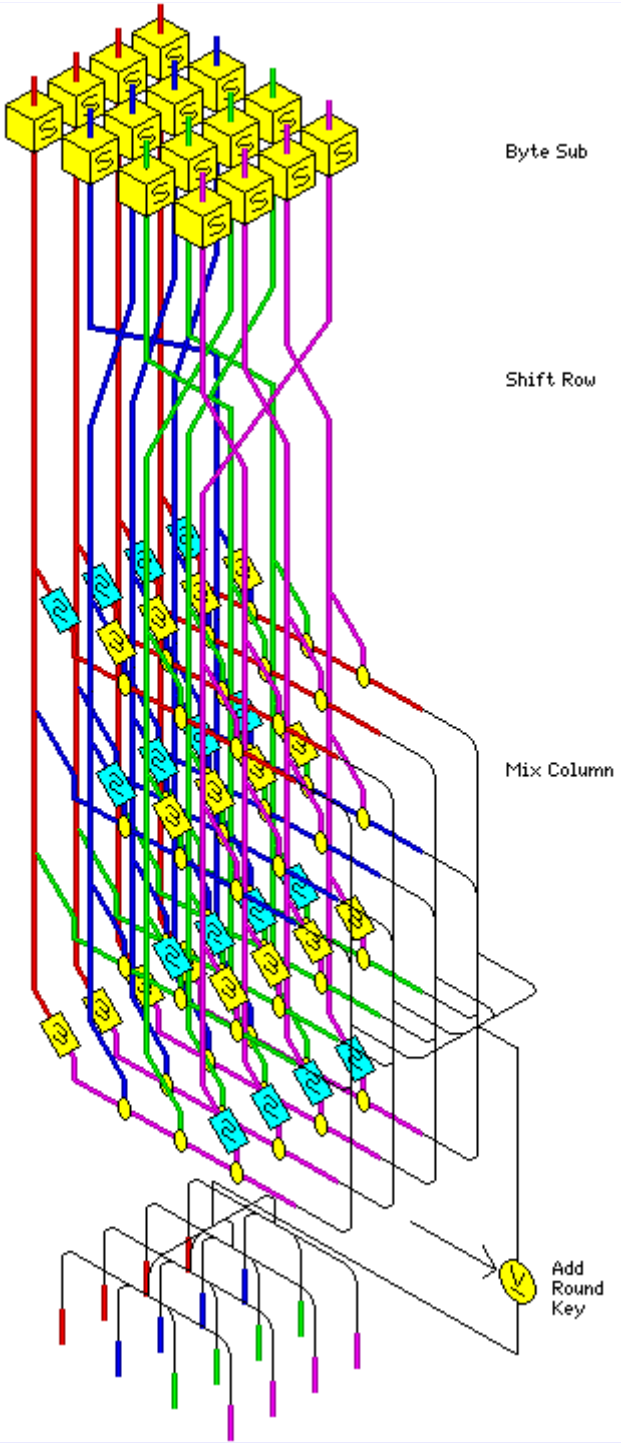
Cifrado en Bloque: Rijndael



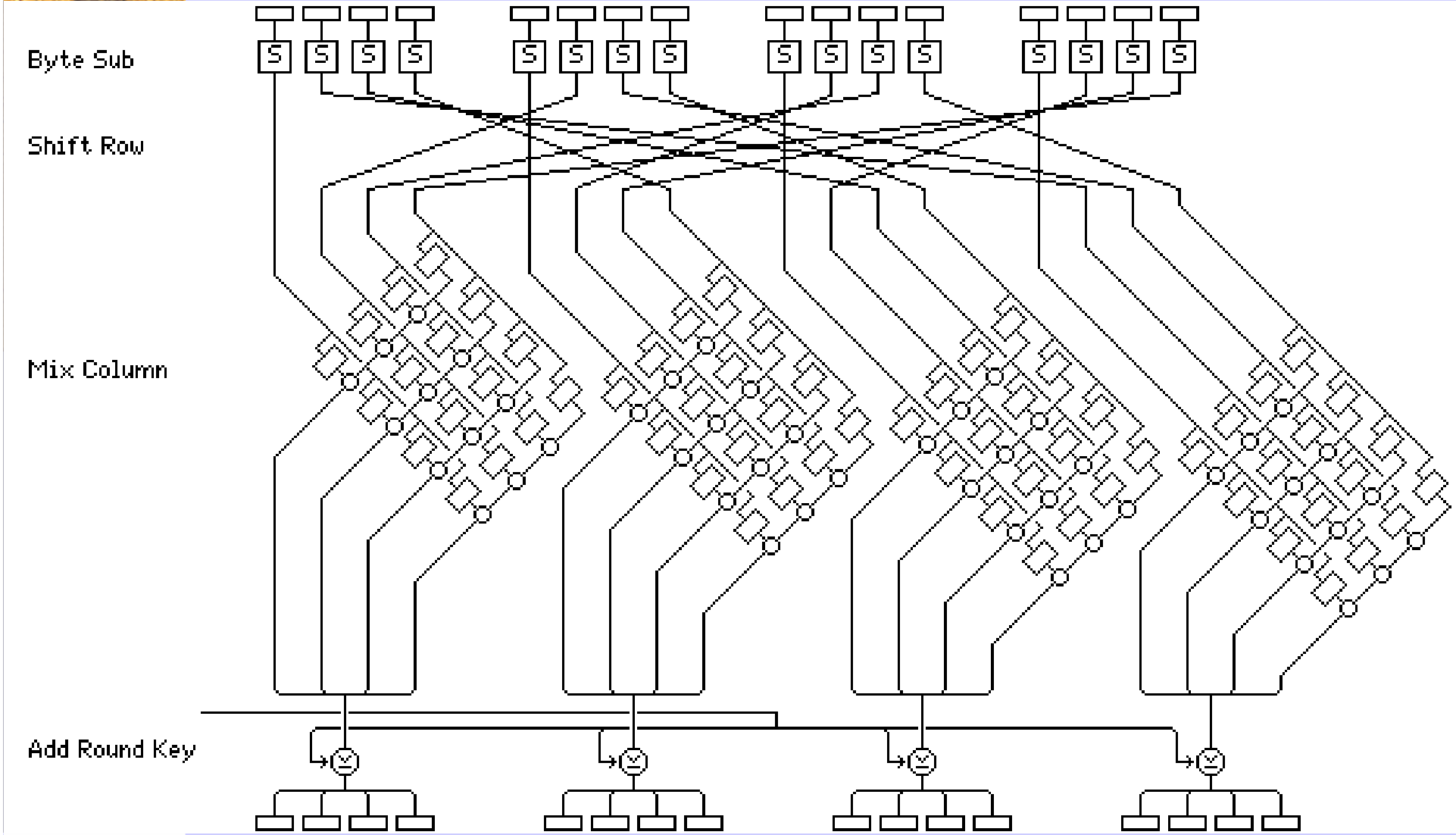
- Diseñado en Bélgica (Rijmen y Daemen).
- Estándar (AES) en EEUU desde 2000.
- Características:
  - Longitud variable de bloque y de clave: 128,192, 256 bits.
  - Número de iteraciones flexible: 10, 12, 14.
  - Operaciones a nivel de byte, con palabras de 4 bytes
  - No es de tipo Feistel
  - Implementación eficiente y con pocos requerimientos
  - Fácilmente paralelizable
  - Posible implementación en tarjeta inteligente
- http://csrc.nist.gov/encryption/aes/

Cifrado en Bloque: Rijndael

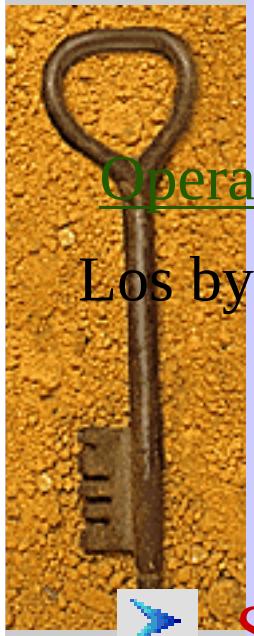
- Cálculos en GF(2^8).
- En cada iteración se realizan cuatro pasos:
  - Sustitución de cada byte por su recíproco
  - Desplazamiento de bytes
  - Producto de cada columna por una matriz
  - XOR de la subclave y la información del estado intermedio actual
- Descifrado:Operaciones inversas en orden inverso



Cifrado en Bloque: Rijndael



Algoritmo Rijndael



Operaciones en $GF(2^8)$.

Los bytes se representan en forma de polinomios. Por ejemplo

A = 5716 = 0101 01112 = x6 + x4 + x2 + x + 1

B = 8316 = 1000 00112 = x7 + x + 1

Suma en $GF(2^8)$

La suma en GF(2^8) se puede realizar de forma sencilla utilizando XOR (es equivalente a la suma de polinomios modulo 2)

A+B = 0101 0111 ⊕ 1000 0011
= 1101 0100 =$D4_{16}$

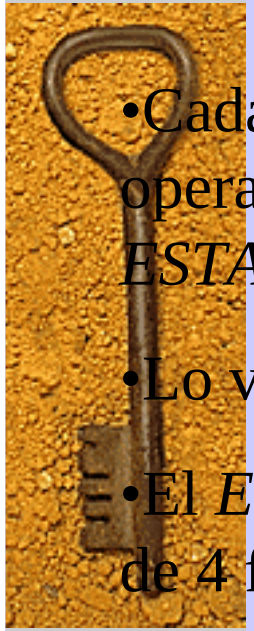
{57} ⊕ {83} = {D4}

Multiplicación en $GF(2^8)$

Consiste en multiplicar los polinomios (en módulo 2), y el resultado ir reduciéndolo módulo m(x). Donde m(x) es un polinomio irreducible.

{57} . {83} = {c1}

Algoritmo Rijndael



- Cada operación del algoritmo tiene como entrada la salida de la operación anterior. Este resultado intermedio se denomina *ESTADO*.
- Lo vemos para el caso de longitud de bloque 128.
- El *ESTADO* se representa como una matriz rectangular de bytes, de 4 filas y 4 columnas

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

- La clave también se representa con una matriz rectangular de bytes, de 4 filas y 4 columnas

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

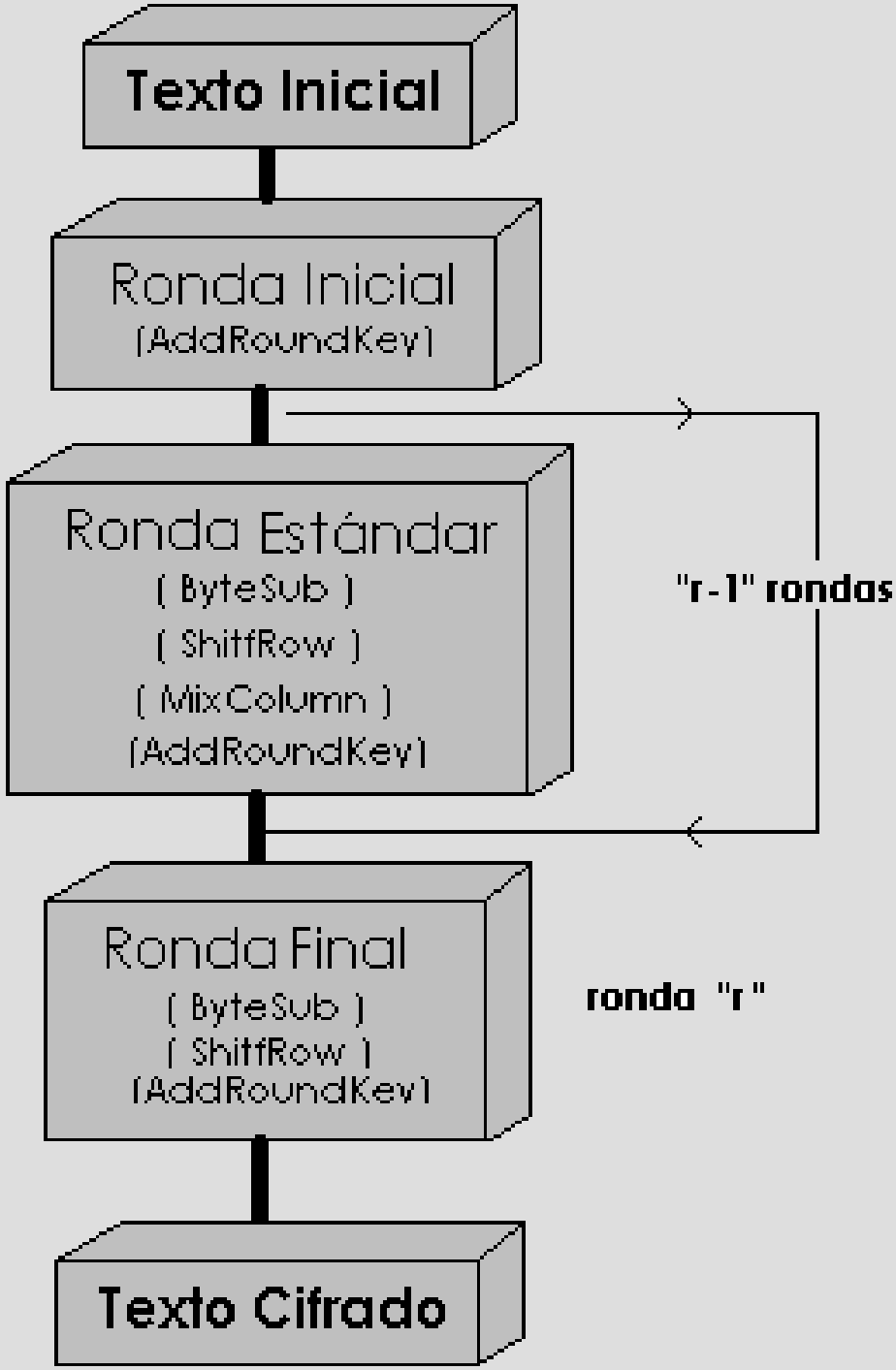
Algoritmo Rijndael

El número de iteraciones que se utilizan depende de la versión del algoritmo que utilicemos, es decir:

	$N_b = 4$ (128 bits)	$N_b = 6$ (192 bits)	$N_b = 8$ (256 bits)
$N_k = 4$ (128 bits)	10	12	14
$N_k = 6$ (192 bits)	12	12	14
$N_k = 8$ (256 bits)	14	14	14

Número de rondas para AES en función de los tamaños de clave y bloque.

En el caso que estudiamos, (longitud de bloque 128), el nº de iteraciones es 10.



Iteración 0

ARK: AddRoundKey:

Consiste en una OR EXCLUSIVA entre el bloque de entrada y la clave original de cifrado. El resultado es el ESTADO INTERMEDIO 1

Ejemplo:

Dado el bloque de entrada y la clave de 128 bits siguientes:

Bloque de entrada = **32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34**

Clave = **2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c**

Se suman los dos elementos:

32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

Bloque de entrada

$\oplus$

2b	28	ab	09
7e	ae	f7	c7
15	D2	15	4f
16	A6	88	3c

Clave

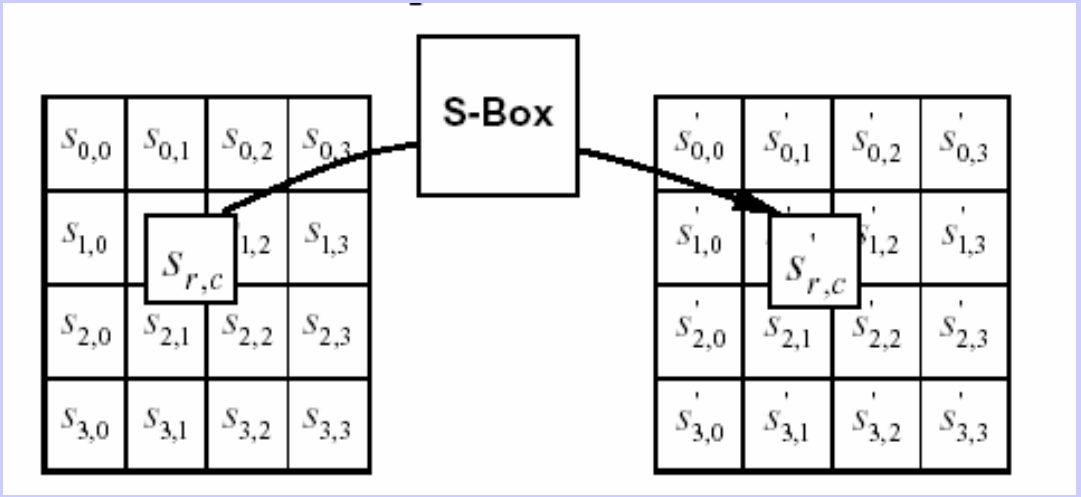
=

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

Estado intermedio 1

Transformación 1. ByteSub

Sustitución no lineal de los bytes de la matriz de estado basada en una S-Caja que para cada byte genera un nuevo byte.



Ejemplo:

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

Estado intermedio 1

⇒

d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

Estado intermedio 2

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Transformación 2. ShiftRow

- Consiste en desplazar a la izquierda los bytes de las filas que conforman la matriz del estado actual.
- Cada fila i=0,1,2,3 se desplaza un numero i de posiciones diferentes

Continuando el ejemplo

1. La fila 0 no es rotada
2. La fila 1 es rotada 1 byte
3. La fila 2 es rotada 2 bytes
4. La fila 3 es rotada 3 bytes

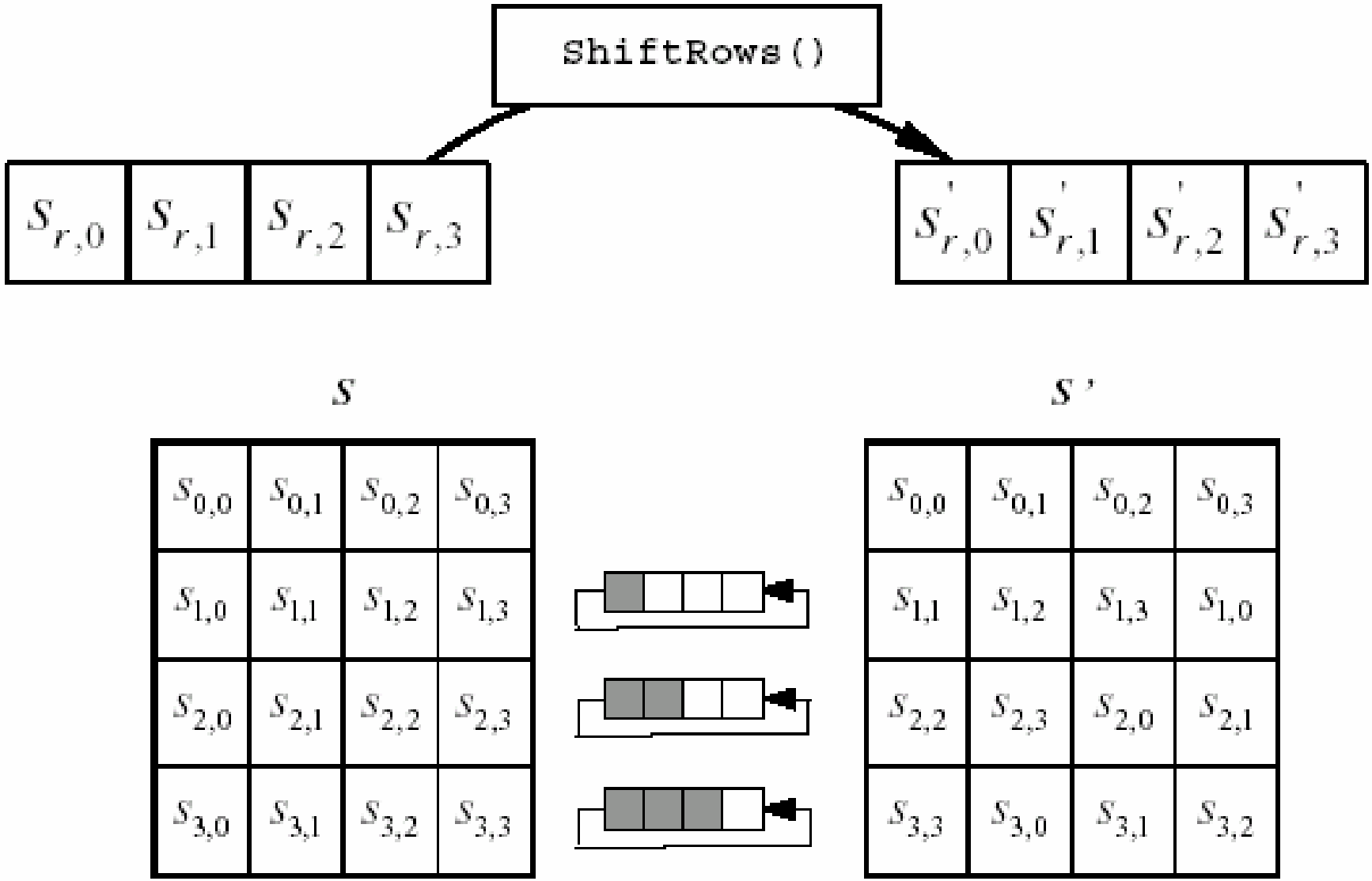
d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

Estado intermedio 2

⇒

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

Estado intermedio 3


ShiftRows()
$S_{r,0}$ $S_{r,1}$ $S_{r,2}$ $S_{r,3}$
$S'_{r,0}$ $S'_{r,1}$ $S'_{r,2}$ $S'_{r,3}$
S
S'
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,0}$
$S_{2,2}$ $S_{2,3}$ $S_{2,0}$ $S_{2,1}$
$S_{3,3}$ $S_{3,0}$ $S_{3,1}$ $S_{3,2}$

Transformación 3. MixColumn

Visto de forma matricial, donde "c" representa el índice de la columna que se procesa:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

Estado intermedio 3

⇒

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

Estado intermedio 4

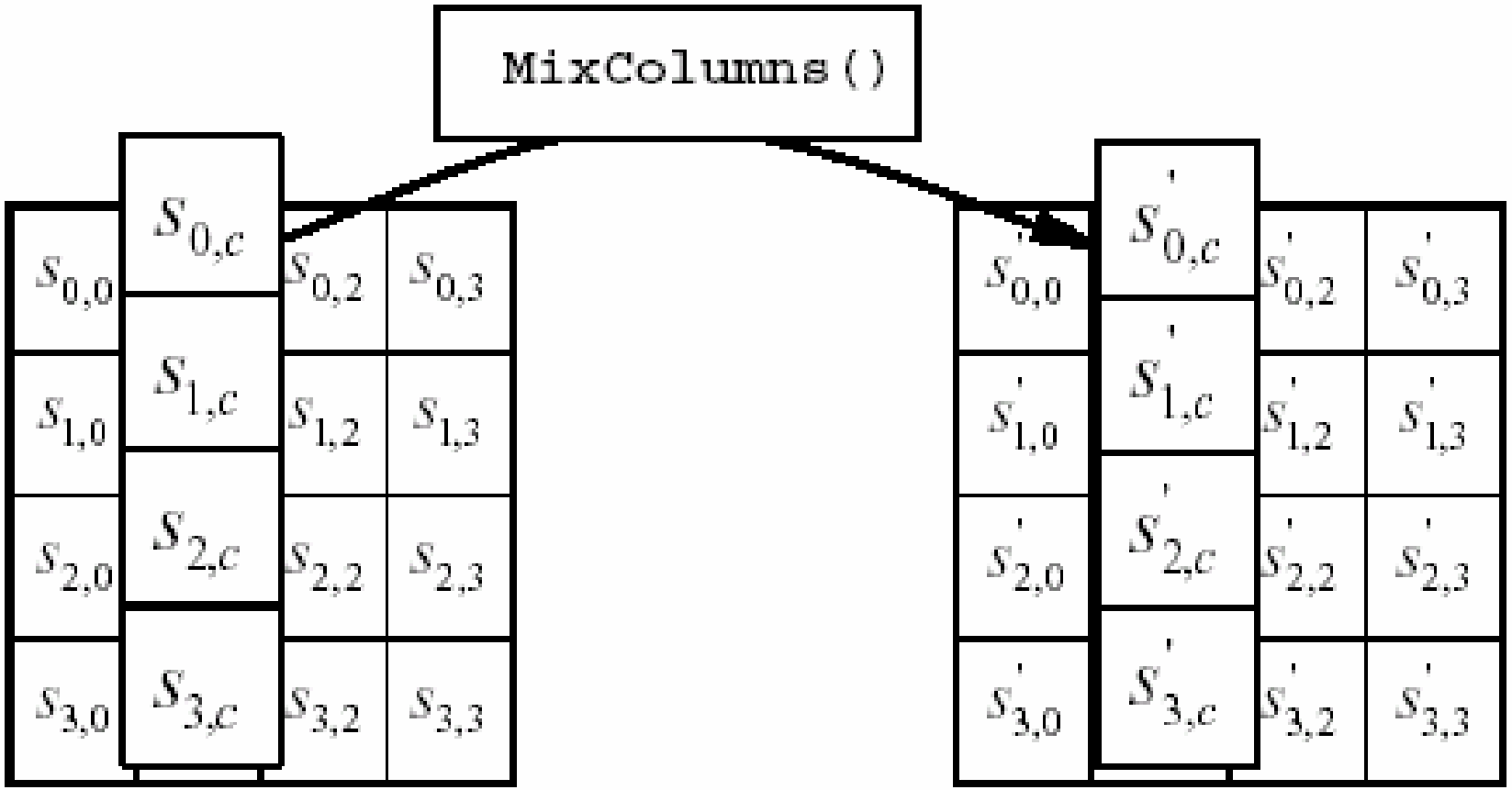
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \mathbf{Nb}.$$

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$



Transformación 4. AddRoundKey

Esta transformación consiste en una OR-EXCLUSIVO entre el Estado Intermedio 4 y una subclave que se genera a partir de la clave original de cifrado.

Ejemplo:

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

Estado intermedio 4

$\oplus$

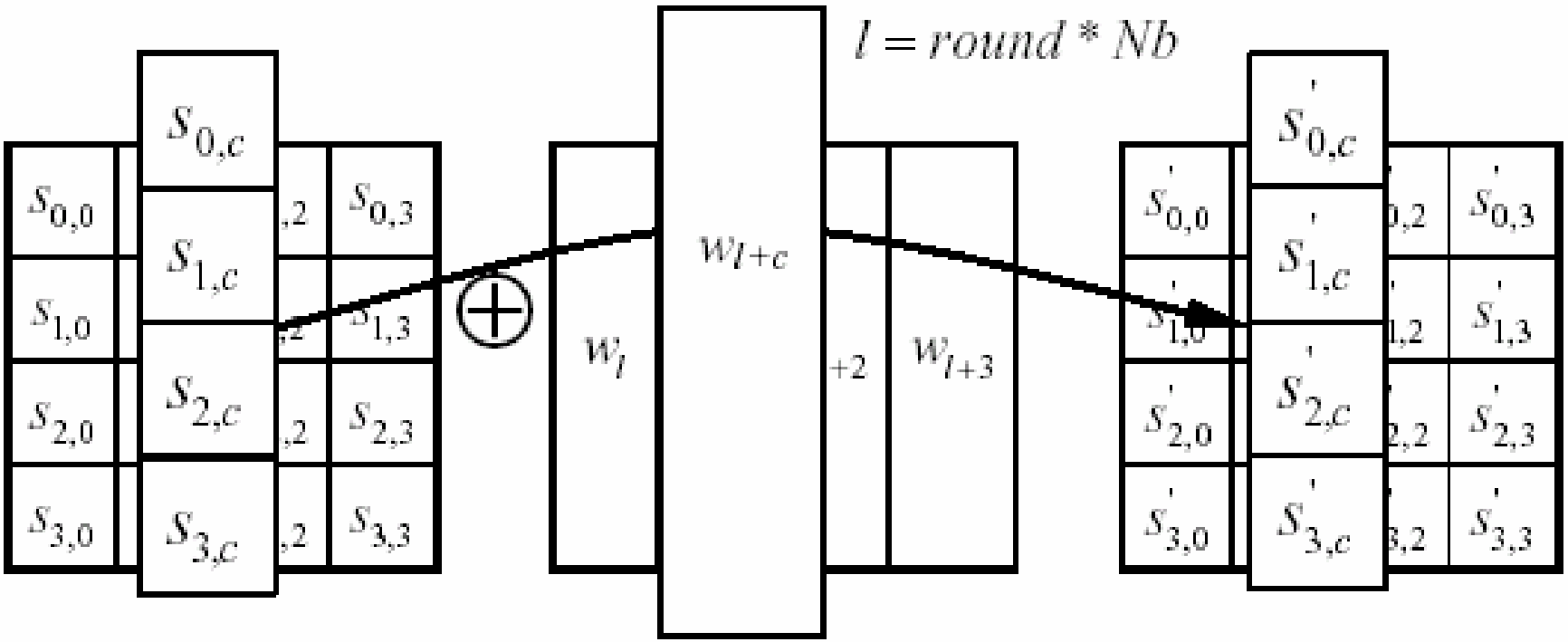
a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

Round Key

=

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

Estado intermedio 1, de la siguiente ronda, ó bloque de salida.


$l = round * Nb$
$s_{0,c}$
$s_{1,c}$
$s_{2,c}$
$s_{3,c}$
$s_{0,0}$
$s_{1,0}$
$s_{2,0}$
$s_{3,0}$
$s_{0,3}$
$s_{1,3}$
$s_{2,3}$
$s_{3,3}$
w_l
w_{l+c}
w_{l+3}
+2
$s'_{0,c}$
$s'_{1,c}$
$s'_{2,c}$
$s'_{3,c}$
$s'_{0,0}$
$s'_{1,0}$
$s'_{2,0}$
$s'_{3,0}$
$s'_{0,3}$
$s'_{1,3}$
$s'_{2,3}$
$s'_{3,3}$

Descifrado:

Es similar al de cifrado, pero algunas operaciones son las transformaciones inversas de las operaciones realizadas en el cifrado, y las subclaves se usan en orden inverso.

Transformación 1. InverseByteSub (S-Caja):

Hex		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	A	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	B	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	C	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	D	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	E	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

– Tabla de Substitución inversa – InvSbox [xy] (en hexadecimal)

Transformación 2. InverseShiftRow

Se realiza el mismo proceso que en el cifrado pero desplazando a la derecha los bytes de las filas que conforman la matriz del estado actual.

Transformación 3. InverseMixColumn

$$s_{0,c} = (\{0e\} \bullet s'_{0,c}) \oplus (\{0b\} \bullet s'_{1,c}) \oplus (\{0d\} \bullet s'_{2,c}) \oplus (\{09\} \bullet s'_{3,c})$$

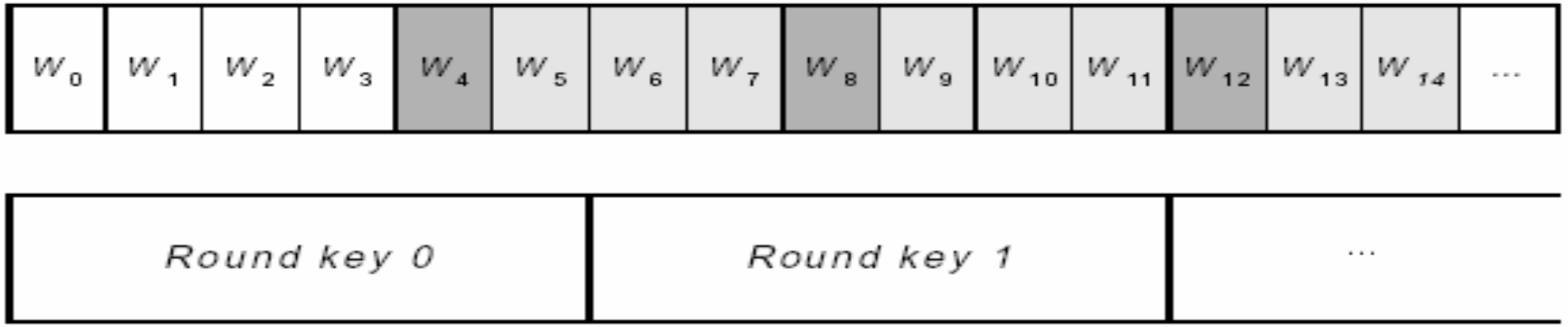
$$s_{1,c} = (\{09\} \bullet s'_{0,c}) \oplus (\{0e\} \bullet s'_{1,c}) \oplus (\{0b\} \bullet s'_{2,c}) \oplus (\{0d\} \bullet s'_{3,c})$$

$$s_{2,c} = (\{0d\} \bullet s'_{0,c}) \oplus (\{09\} \bullet s'_{1,c}) \oplus (\{0e\} \bullet s'_{2,c}) \oplus (\{0b\} \bullet s'_{3,c})$$

$$s_{3,c} = (\{0b\} \bullet s'_{0,c}) \oplus (\{0d\} \bullet s'_{1,c}) \oplus (\{09\} \bullet s'_{2,c}) \oplus (\{0e\} \bullet s'_{3,c})$$

Generación de Subclaves

El proceso de selección de la llave para la iteración i, será tomada de la subclave desde w[Nb*i] hasta w [Nb *(i+1)]



Seguridad de Rijndael

- Según los estudios publicados, no existe ningún mecanismo para invertir el Algoritmo que sea más eficiente que un ataque por fuerza bruta.
- El diseño de los parámetros del Algoritmo como: Número de iteraciones, número de rotaciones de bytes, los valores de las S-Cajas, etc, fueron elegidos para hacer robusto al Algoritmo frente a ataques conocidos como:
  - Criptoanálisis Diferencial
  - Criptoanálisis Lineal
- El primer ataque conocido para este algoritmo fue publicado en 1998, logra romper la seguridad teóricamente solo para 6 iteraciones, pero recordemos que Rijndael usa 10, 12 o 14 ciclos.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Criptoanálisis

Cifrado en Bloque/Flujo

CIFRADO EN BLOQUE

El algoritmo de cifrado se aplica sobre cada bloque de mensaje usando la misma clave.

Ventajas:

* Alta difusión
* Imposible introducir bloques falsos sin detectarlo.

Desventajas:

* Baja velocidad de cifrado
* Propenso a errores de cifrado: Un error se propaga a todo el bloque.

CIFRADO EN FLUJO

El algoritmo de cifrado se aplica sobre cada bit de mensaje usando los distintos bits de la secuencia de clave.

Ventajas:

* Alta velocidad de cifrado
* No propaga errores.

Desventajas:

* Baja difusión
* Vulnerable a alteraciones

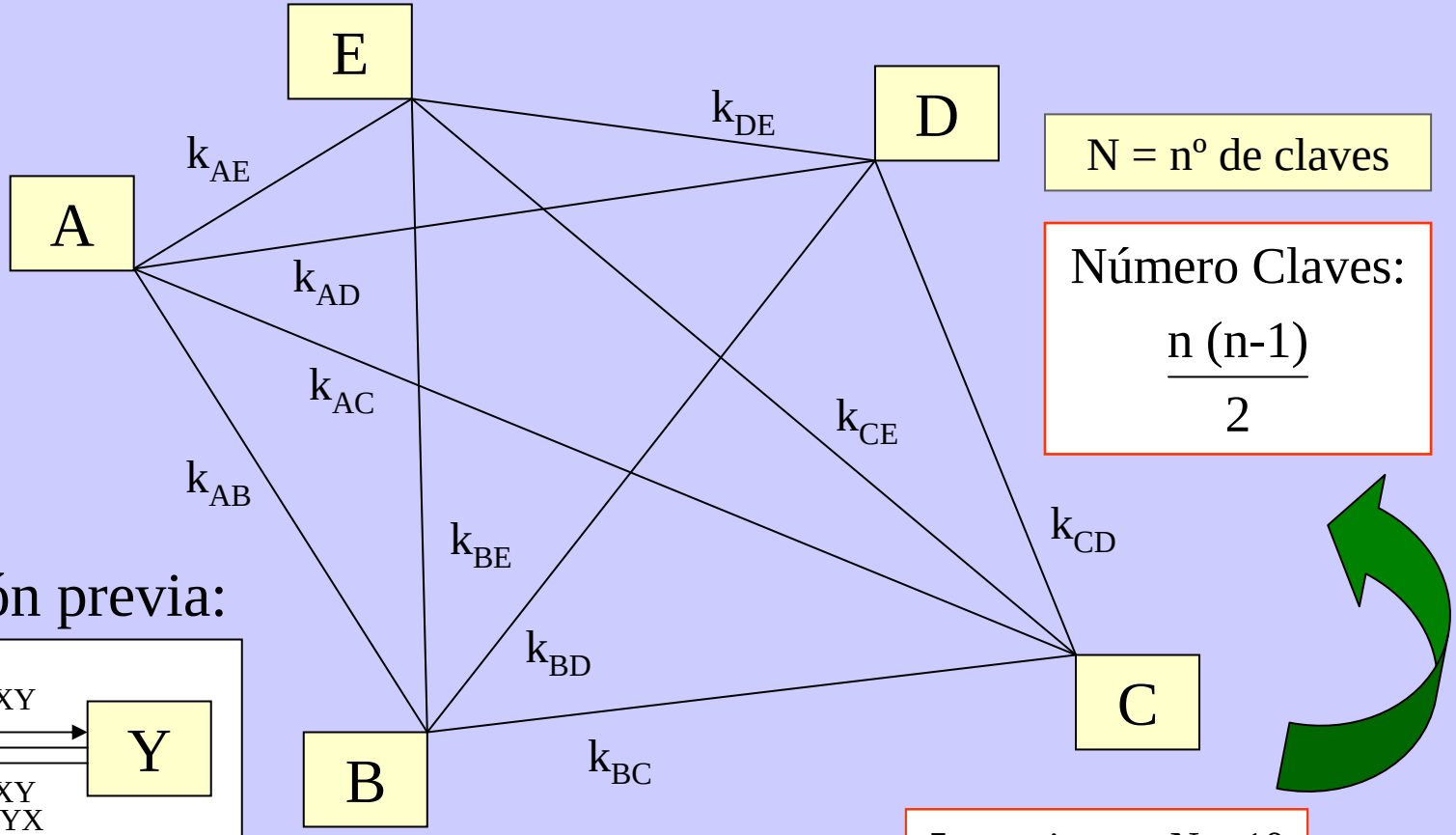
Gestión de Claves

- Generación, distribución, almacenamiento, tiempo de vida, destrucción y aplicación de las claves de acuerdo con una política de seguridad.
- Longitud de la clave
- Elección correcta de la clave para evitar ataque de diccionario. Generación seudoaleatoria.
- Tiempo de vida de las claves: Fecha de caducidad porque cuanto más tiempo se use una clave:
  - aumenta la probabilidad de que se comprometa
  - mayor será el daño si la clave se compromete, ya que toda la información protegida con esa clave queda al descubierto.
  - mayor será la tentación de alguien para intentar desbaratarla.
  - en general es más fácil realizar criptoanálisis con mucho texto cifrado con la misma clave.

Gestión de Claves

- Jerarquía del espacio de claves:
  - de sesión, empleadas para cifrar los datos de usuario y que son actualizadas frecuentemente,
  - de cifrado de claves, para proteger las claves de sesión, y
  - maestras, que son distribuidas manualmente y se utilizan para proteger las de cifrado de claves cuando son intercambiadas.

Distribución de Claves Secretas

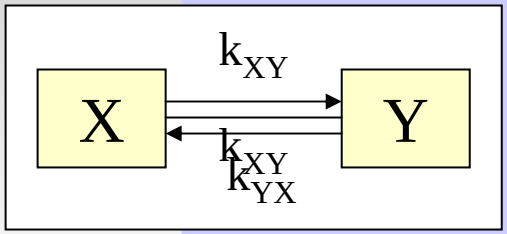


N = nº de claves

Número Claves:

$$\frac{n\,(n-1)}{2}$$

Definición previa:



5 usuarios: N = 10

La gestión de las claves es una tarea muy difícil

Distribución de Claves Secretas: Método de Diffie-Hellmann

escoge secreto x_A

calcula $y_A=\alpha^{xA} \pmod p$

genera $k= y_B^{xA} \pmod p$

escoge secreto x_B

calcula

$y_B=\alpha^{xB} \pmod p$

genera

$k= y_A^{xB} \pmod p$