



CRIPTOGRAFÍA PARA INGENIER@S

Class4crypt

© Jorgeramió 2022

Aula virtual de
criptografía
aplicada

Diapositivas
utilizadas en las
clases grabadas
de Class4crypt

Módulo 9 Criptografía simétrica en flujo

Dr. Jorge Ramió Aguirre © 2022



Attribution-NonCommercial-
NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)



*El ingenio es intrínseco al ser humano,
solo hay que darle una oportunidad
para que se manifieste.*

<https://www.criptored.es/cvJorge/index.html>

Tu aula virtual de criptografía aplicada

Módulo 10. Criptografía asimétrica

Class4crypt

Módulo 9. Criptografía simétrica en flujo

9.1 Fundamentos de la cifra simétrica en flujo

9.2 Registros de desplazamiento realimentados lineales LFSR y no lineales NLFSR

9.3 Aleatoriedad en registros LFSR con polinomio primitivo

9.4 Complejidad en LFSR, A5, RC4 y Chacha20

Lista de reproducción del módulo 9 en el canal Class4crypt

https://www.youtube.com/playlist?list=PLq6etZPDh0kv5eX_3RUMrq8tsePRmWadm

Class4crypt c4c9.1

Módulo 9. Criptografía simétrica en flujo

Lección 9.1. Fundamentos de la cifra simétrica en flujo

9.1.1. El código Baudot

9.1.2. El cifrado de Vernam con código Baudot

9.1.3. Vernam, la libreta de un solo uso OTP y el secreto perfecto

9.1.4. Introducción y estructura de un cifrado en flujo

9.1.5. Características que debe presentar una buena secuencia cifrante binaria

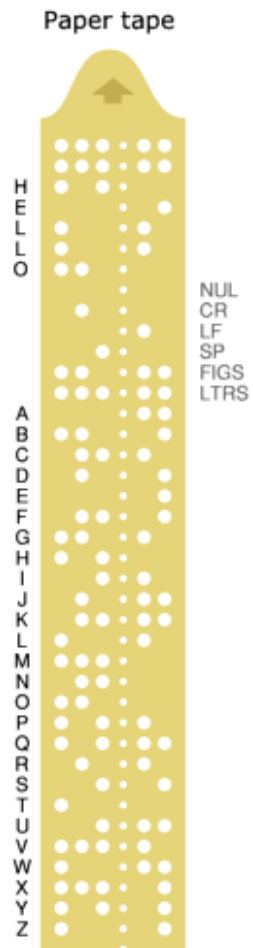
9.1.5.1. Tamaño de la secuencia

9.1.5.2. Imprevisibilidad de la secuencia

9.1.5.3. Aleatoriedad de la secuencia

Class4crypt c4c9.1 Fundamentos de la cifra simétrica en flujo
<https://www.youtube.com/watch?v=VpQj-SkOa10>

Código de Baudot (1874) y sus versiones



LETTERS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	CARRIAGE RETURN	LINE FEED	LETTERS	FIGURES	SPACE	ALL SPACE	
FIGURES	—	?	:	WHO ARE YOU	3	%	@	£	8	BELL	()	.	,	9	0	1	4	'	5	7	=	2	/	6	+							
CODE ELEMENTS	1	●	●		●	●				●	●					●	●		●		●	●	●	●	●	●			●	●	●	○	
2	●			●			●		●	○	●						●	●				●	●		●				●	●			
3	○	○	○	○	○	○	○	○	○		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
4		●	●	●		●	●			●	●	●		●	●		●	●				●	●	●	●	●	●	●	●	●	●	●	
5		●				●	●	●			●	●	●	●	●	●	●	●	●	●			●	●	●	●	●	●	●	●	●	●	

● INDICATES A MARK ELEMENT (A HOLE PUNCHED IN THE TAPE)

○ INDICATES POSITION OF A SPROCKET HOLE IN THE TAPE

The International Telegraph Alphabet



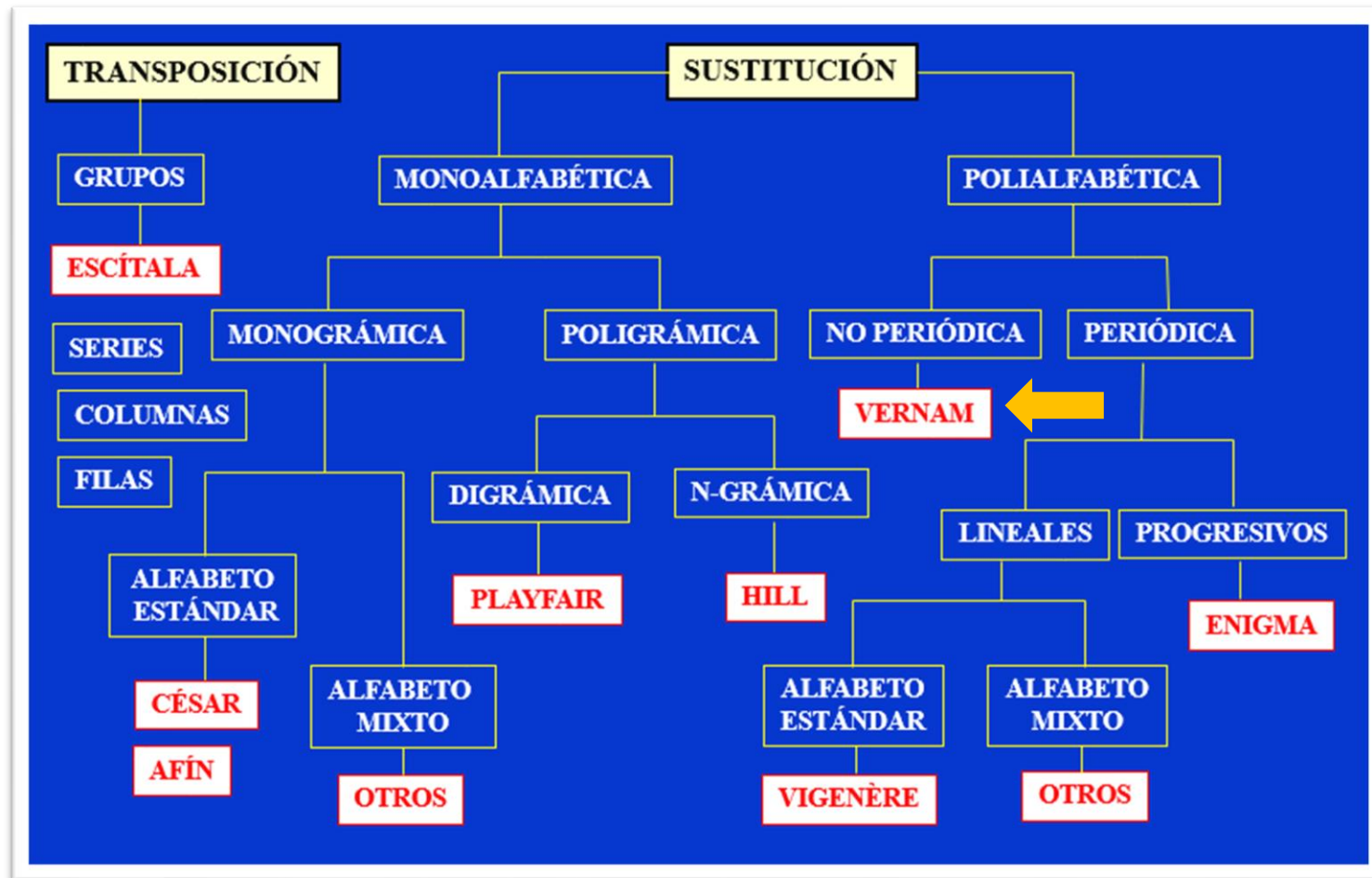
Jean Maurice Émile Baudot
(1845 - 1903)



Código Binario	Carácter	Código Binario	Carácter
00000	Blanco	10000	T
00001	E	10001	Z
00010	=	10010	L
00011	A	10011	W
00100	Espacio	10100	H
00101	S	10101	Y
00110	I	10110	P
00111	U	10111	Q
01000	<	11000	O
01001	D	11001	B
01010	R	11010	G
01011	J	11011	↑
01100	N	11100	M
01101	F	11101	X
01110	C	11110	V
01111	K	11111	↓

Baudot (Murray) binario CCITT-2

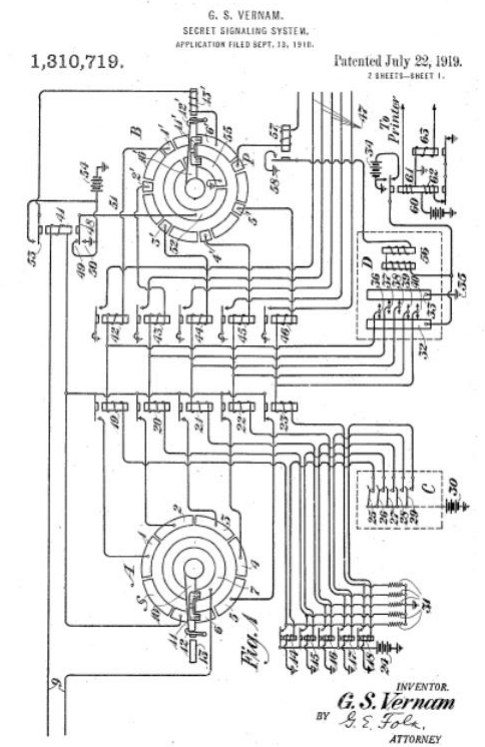
Vernam y la cifra clásica: los inicios



La figura de Gilbert Sandford Vernam

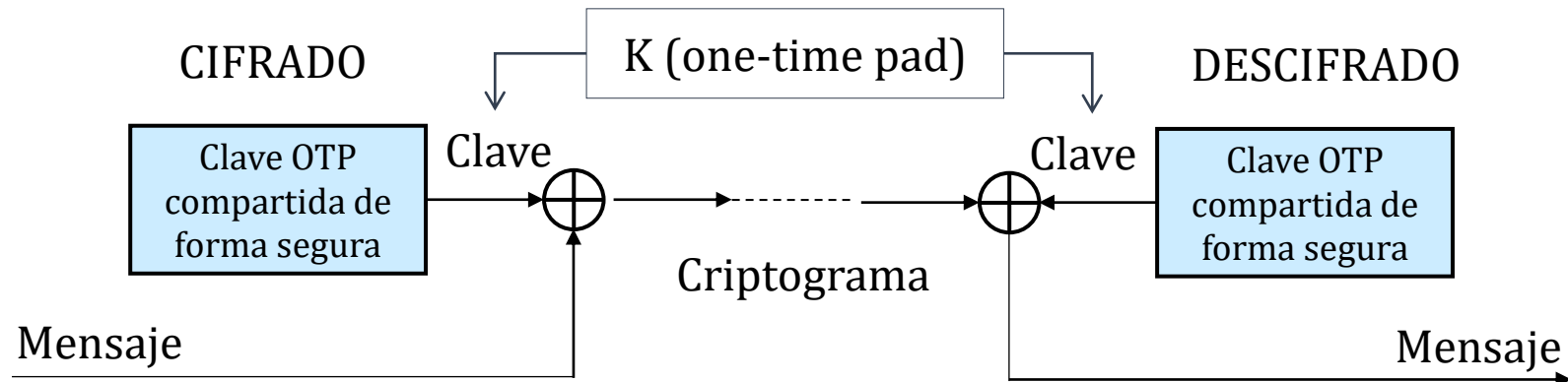


- Gilbert S. Vernam (1890 - 1960), ingeniero en AT&T Bell Labs, inventa en 1917 un cifrador de flujo aditivo y polialfabético
- Posteriormente, co-inventa con Joseph Mauborgne, capitán del Cuerpo de Señales del Ejército de los Estados Unidos, el sistema de cifra denominado one-time pad (OTP) o libreta de un solo uso
- Patenta en 1919 una función de combinación para implementar el xor en la lógica de relés, codificando caracteres del código Baudot
- Ejemplo de Vernam en su patente:
 - A con código Baudot = ++--- [+ hoyo en cinta (1), - sin hoyo en cinta (0)]
 - B con código Baudot = +---++
 - G con código Baudot = -+---++ [es la cifra de A con B]
- Haciendo la cifra G con B se recupera A
- Una de las patentes más importantes en la historia de la criptografía (NSA)



El cifrado de Vernam

- Se usaba el código Baudot de 5 caracteres o bits
- La operación de cifra era la función XOR implementada con relés
- La clave K es una libreta de un solo uso, one-time pad, compartida con anterioridad y de forma segura entre emisor y receptor
- La clave debía ser aleatoria y tan larga o más que el mensaje
- El sistema de descifrado era igual que el de cifrado por la involución de la función XOR



Ejemplo de cifrado de Vernam

Con el código Baudot de 5 bits, cifrar el mensaje $M = \text{BYTES}$ con clave $K = \text{VERNAM}$

- CIFRADO

- $B \oplus V = 11001 \oplus 11110 = 00111 = U$
- $Y \oplus E = 10101 \oplus 00001 = 10100 = H$
- $T \oplus R = 10000 \oplus 01010 = 11010 = G$
- $E \oplus N = 00001 \oplus 01100 = 01101 = F$
- $S \oplus A = 00101 \oplus 00011 = 00110 = I$

- DES CIFRADO

- $U \oplus V = 00111 \oplus 11110 = 11001 = B$
- $H \oplus E = 10100 \oplus 00001 = 10101 = Y$
- $G \oplus R = 11010 \oplus 01010 = 10000 = T$
- $F \oplus N = 01101 \oplus 01100 = 00001 = E$
- $I \oplus A = 00110 \oplus 00011 = 00101 = S$

- El esquema de Vernam es el único cifrador matemáticamente seguro
- Cuenta con secreto perfecto, demostrado por Claude Shannon
- Es imposible de criptoanalizar pues la clave K se usa una sola vez, es aleatoria y de longitud igual o mayor que el propio mensaje

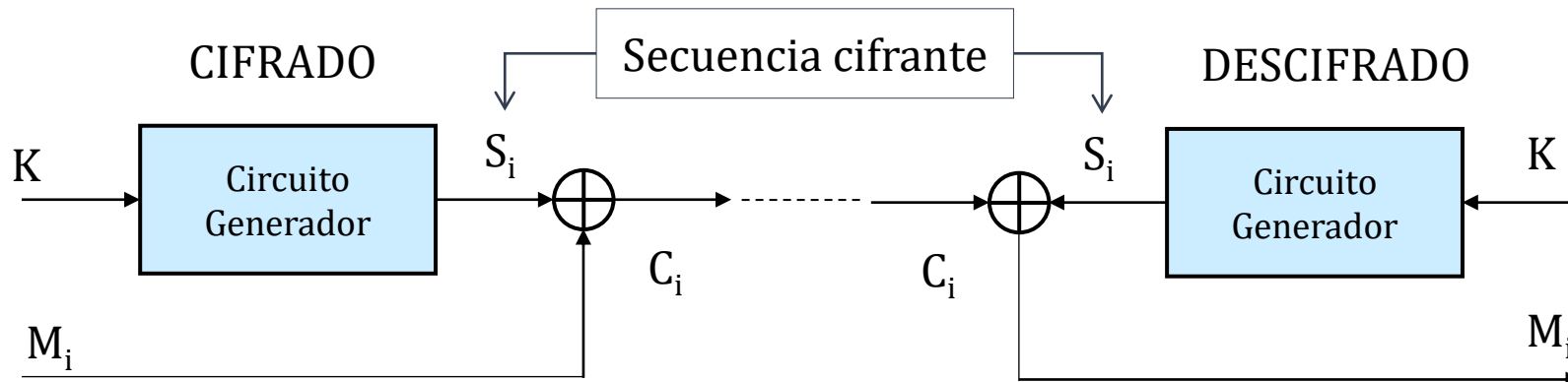
Vernam, cifra en flujo y secreto perfecto

- Para que un sistema de cifra en flujo (en criptografía clásica cifra polialfabética monográfica) tenga un secreto perfecto, deberá cumplir las condiciones del cifrador de Vernam
 - Tener una clave aleatoria
 - Que la longitud de la clave sea de igual tamaño o mayor que la del mensaje
 - Que dicha clave (libreta) sea de un único uso (one-time pad)
- Para mensajes relativamente pequeños, se puede aceptar que la clave (cinta) sea intercambiada entre emisor y receptor previamente. Para mensajes muy grandes, esto ya no es práctico
- Por tanto, habrá que inventar un sistema (circuito) que permita generar la clave en ambos extremos, emisor y receptor, y no necesariamente de forma simultánea
- Esto hará que la clave ya no sea aleatoria sino pseudoaleatoria y, por lo tanto, se pierde el secreto perfecto, al menos en teoría

Introducción a la cifra en flujo

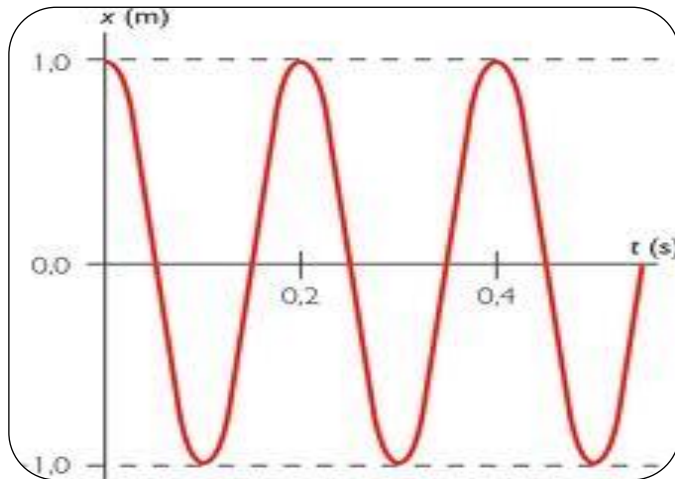


- El mensaje en claro se cifra bit a bit o byte a byte, mediante una operación XOR entre el texto en claro y una **secuencia cifrante** S_i
- Emisor y receptor comparten de forma segura una semilla secreta K y un circuito generador de la secuencia de bits S_i que permita generar secuencias binarias pseudoaleatoria y seguras
- Para descifrar, se vuelve a realizar el XOR entre criptograma y la misma secuencia S_i en destino, pues la función XOR es involutiva



Características secuencias cifrantes (1)

El periodo de S_i



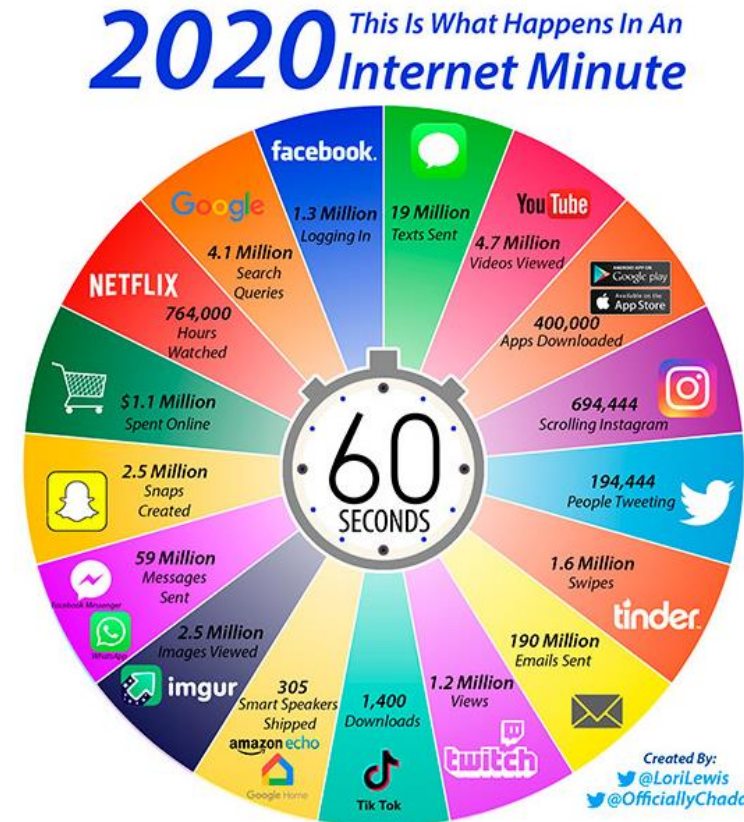
- La secuencia cifrante S_i debe tener un período muy elevado, tanto o más que el propio mensaje, para que la clave no se repita durante la operación de cifrado
- En la práctica, se usará una semilla K de mínimo 120 bits para generar períodos de secuencia cifrante mínimos de 2^{120} bits, es decir 10^{37} bits
- Pregunta recurrente: ¿es suficientemente grande este valor de bits de S_i ?
 1. “El ingenioso hidalgo Don Quijote de La Mancha” tiene algo más de dos millones de caracteres, espacio en blanco incluido, menos de $2 \cdot 10^7$ bits
 2. Cisco: tráfico anual de móviles en Internet 2022: 1 zettabyte (2^{70} bytes)

Magnitudes de bytes y tráfico en Internet

Múltiplos de bytes			
Sistema Internacional (decimal)		ISO/IEC 80000-13 (binario)	
Múltiplo (símbolo)	SI	Múltiplo (símbolo)	ISO/IEC
kilobyte (kB)	10 ³	kibibyte (KiB)	2 ¹⁰
megabyte (MB)	10 ⁶	mebibyte (MiB)	2 ²⁰
gigabyte (GB)	10 ⁹	gibibyte (GiB)	2 ³⁰
terabyte (TB)	10 ¹²	tebibyte (TiB)	2 ⁴⁰
petabyte (PB)	10 ¹⁵	pebibyte (PiB)	2 ⁵⁰
exabyte (EB)	10 ¹⁸	exbibyte (EiB)	2 ⁶⁰
zettabyte (ZB)	10 ²¹	zebibyte (ZiB)	2 ⁷⁰
yottabyte (YB)	10 ²⁴	yobibyte (YiB)	2 ⁸⁰
Véase también: nibble • byte • sistema octal			

El peso aproximado de todo Internet en el año 2018 era de 10 zettabytes, en 2020 este valor se podría duplicar

En 2020 estamos ya en estos órdenes de magnitud en cuanto a datos en Internet



<https://www.allaccess.com/merge/archive/31294/infographic-what-happens-in-an-internet-minute>

Características secuencias cifrantes (2)

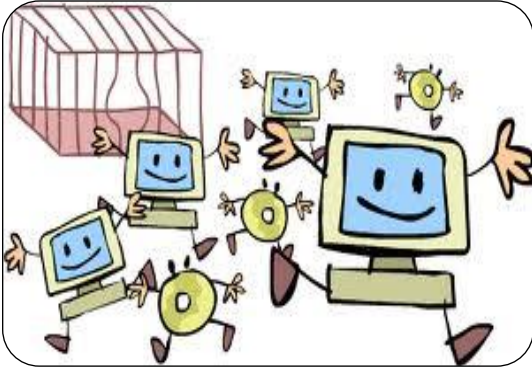
Tamaño de la semilla K



- Para generar la misma secuencia cifrante S_i en fase, ambos extremos usarán una semilla K que no puede ser un valor demasiado pequeño
- Para protegerse de ataques por fuerza bruta la semilla debe tener un tamaño suficientemente grande, por ejemplo sobre los 100 bits, para generar secuencias de al menos 2^{100} bits
- Pregunta recurrente: ¿la semilla K es lo mismo que la secuencia de cifra S_i ?
- No. La semilla K es el secreto que deben guardar emisor y receptor y S_i es la secuencia de bits que generan para cifrar. La seguridad del sistema de cifra recae en el tamaño de la semilla y en la idoneidad del circuito generador

Características secuencias cifrantes (3)

Implementación



- Debe ser fácil su implementación y que permita obtener algoritmo rápidos
- Se puede lograr con hardware y con software mediante diferentes sistemas o circuitos

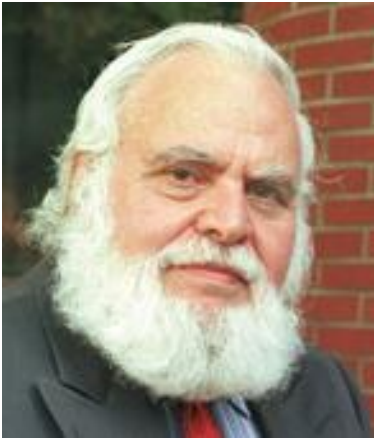
- Aunque se conozca una parte de la secuencia cifrante S_i , la probabilidad de predecir el próximo bit no debería ser superior al 50%
- Una función aleatoria es, por definición, imprevisible

Imprevisibilidad



Características secuencias cifrantes (4)

Aleatoriedad de S_i



Solomon Golomb
(1932 – 2016)

- S_i debe aproximarse a una secuencia aleatoria
- Cada generador de S_i debe cumplir con los 3 Postulados de Golomb: “Shift Register Sequences”, Holden-Day, Inc., San Francisco (1967), 2a ed. Aegean Park Press (1982)
- Postulado G1: mitad de 0s y de 1s (probabilidad 50%)
- Postulado G2: probabilidad 50%, independiente de bits
- Postulado G3: la información de S_i derivada de zonas con diferentes cadenas de bits conocidos, debe ser nula
- Pregunta recurrente: ¿existen otros test de aleatoriedad de cadenas binarias?
- Sí. Los postulados de Golomb son solamente los más básicos, aunque los más simples de entender. Otros test más avanzados: Diehard, Dieharder y NIST

Prácticas con Criptoclásicos y FlujoLab

- Criptoclásicos v2.1
 - https://www.criptored.es/software/sw_m001c.htm
- Con Criptoclásicos cifrar con Vernam el texto en claro “Cifrado de Vernam” con la clave “Tiene secreto perfecto”
- FlujoLab: software para la generación de claves y el seguimiento de cifradores de flujo
 - https://www.criptored.es/software/sw_m001m.htm
- Con FlujoLab, cifrar el texto en claro “Mamá dice que la vida es como una caja de bombones, nunca sabes el que te va a tocar.” con una secuencia cifrante de 1.023 bits generada con un registro de desplazamiento con realimentación lineal LFSR, polinomio primitivo asociado $x^{10} + x^3 + 1$ y semilla 1101011111

Conclusiones de la lección 9.1

- La cifra en flujo tiene como precedente el cifrado de Vernam (1917), que cifra los caracteres representados con el código Baudot (1874), usando para ello una cinta perforada: con hoyo significa un 1 y sin hoyo significa un 0
- Vernam patenta en 1919 un sistema de relés que permite ejecutar la función XOR, un invento vital según la NSA . Posteriormente inventa, junto a Joseph Mauborgne, el sistema de libreta de un solo uso, one-time pad OTP
- El cifrado de Vernam es el único sistema con secreto perfecto, al ser la clave aleatoria, de un solo uso y mayor o de igual longitud que el mensaje
- En cifra de flujo moderna la secuencia de clave deberá generarse en emisión y en recepción, partiendo de una semilla segura previamente intercambiada
- Las secuencias cifrantes deberán cumplir con principios de aleatoriedad

Lectura recomendada (1/2)

- Cryptology and Data Secrecy: The Vernam Cipher
 - http://www.pro-technix.com/information/crypto/pages/vernam_base.html
- Gilbert Vernam (Wikipedia)
 - https://en.wikipedia.org/wiki/Gilbert_Vernam
- Securing Record Communications: The TSEC/KW-26, Melville Klein (NSA)
 - https://web.archive.org/web/20121010011445/https://www.nsa.gov/about/_files/cryptologic_heritage/publications/misc/tsec_kw26.pdf
- Baudot code (Wikipedia)
 - https://en.wikipedia.org/wiki/Baudot_code
- Crypto Museum
 - <https://www.cryptomuseum.com/crypto/ baudot.htm>

Lectura recomendada (2/2)

- Cisco: Mobile internet traffic will approach a zettabyte by 2022, Kyle Wiggers, 2019
 - <https://venturebeat.com/2019/02/19/cisco-mobile-internet-traffic-will-approach-a-zettabyte-by-2022/>
- Zettabyte Era (Wikipedia - Cisco)
 - https://en.wikipedia.org/wiki/Zettabyte_Era
- Comprobando la aleatoriedad (INCIBE Cert)
 - <https://www.incibe-cert.es/blog/comprobando-aleatoriedad>

Class4crypt c4c9.2

Módulo 9. Criptografía simétrica en flujo

Lección 9.2. Registros de desplazamiento realimentados LFSR y NLFSR

9.2.1. Generadores de secuencias cifrantes

9.2.2. Registro de desplazamiento realimentado

9.2.3. Registros de desplazamiento realimentados no lineales NLFSR

9.2.4. Registros de desplazamiento realimentados lineales LFSR

9.2.4.1. Registros con polinomio asociado factorizable

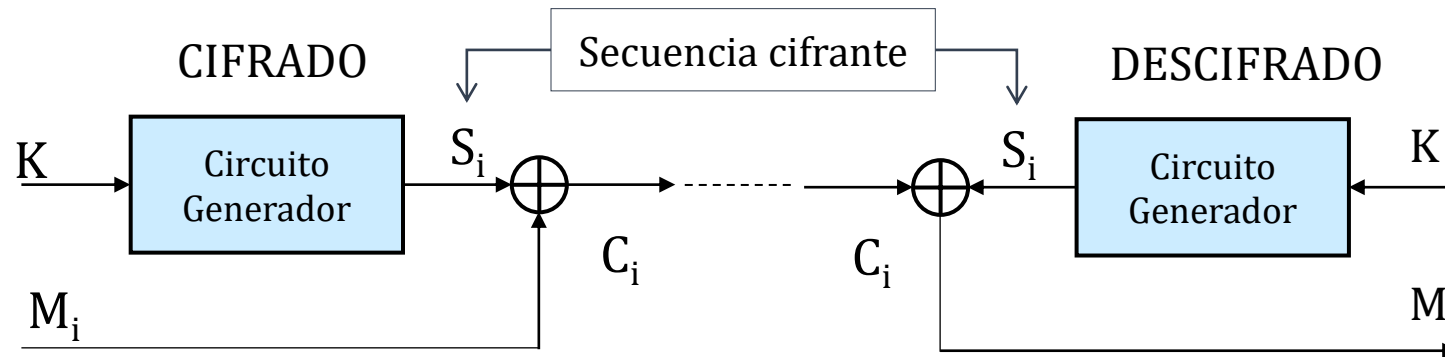
9.2.4.2. Registros con polinomio asociado irreducible

9.2.4.3. Registros con polinomio asociado primitivo

Class4crypt c4c9.2 Registros de desplazamiento realimentados lineales LFSR y no lineales NLFSR

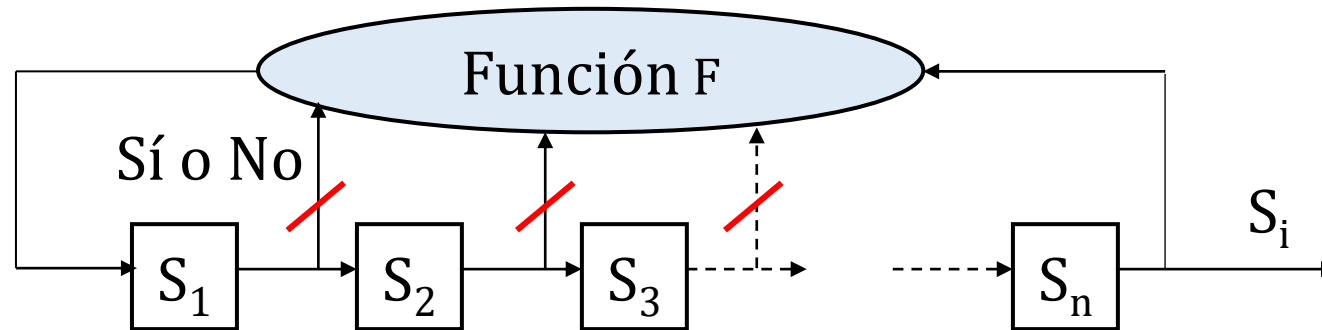
<https://www.youtube.com/watch?v=j-Bx95hLUcE>

Generadores de secuencias cifrantes



- Los generadores de la secuencia cifrante pueden ser de los siguientes tipos:
- ➡
- Registros de desplazamiento realimentados. Por ejemplo, el algoritmo A5 usado en telefonía móvil GSM o 2G, entre los años 1990 y 2000
 - Registros numéricos. Por ejemplo, el algoritmo RC4, usado desde 1995 en WEP y WPA de los sistemas inalámbricos, y en SSL/TLS de Internet hasta 2014
 - Algoritmos de cifra simétrica en bloque. Por ejemplo, el algoritmo AES en modos de cifra contador CTR o contador con campos de Galois GCM

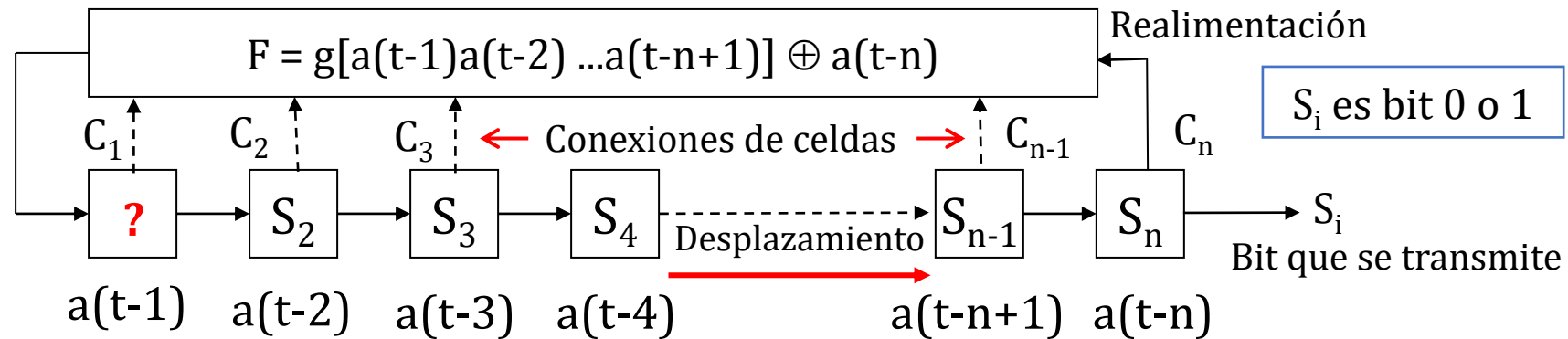
Registro de desplazamiento realimentado



- Los FSR (Feedback Shift Register) son la base de muchos de los generadores de secuencias cifrantes. Lo conforman un conjunto de n celdas $\{S_1 \dots S_n\}$ en las que se almacena un bit (0 o 1)
- Las celdas contendrán inicialmente un valor 0 o 1, que será parte de la semilla o clave inicial
- El circuito estará controlada por un reloj, produciéndose un desplazamiento hacia la derecha del registro en cada pulso de reloj
- S_n se convierte en el nuevo bit de S_i y S_1 se obtiene como resultado de aplicar la función F al contenido de las n celdas (0 o 1) conectadas, antes del desplazamiento hacia la derecha del registro. De forma similar se opera con los siguientes bits del registro

Generador con registro de desplazamiento

Generador de secuencia cifrante con registro de desplazamiento genérico



a(i) es el contenido de la celda i | C_i es 0 si no está conectada y 1 si está conectada

Genera una secuencia con un período máximo igual a 2^n

NLFSR Non Linear Feedback Shift Register
Registros de Desplazamiento Realimentados No Linealmente

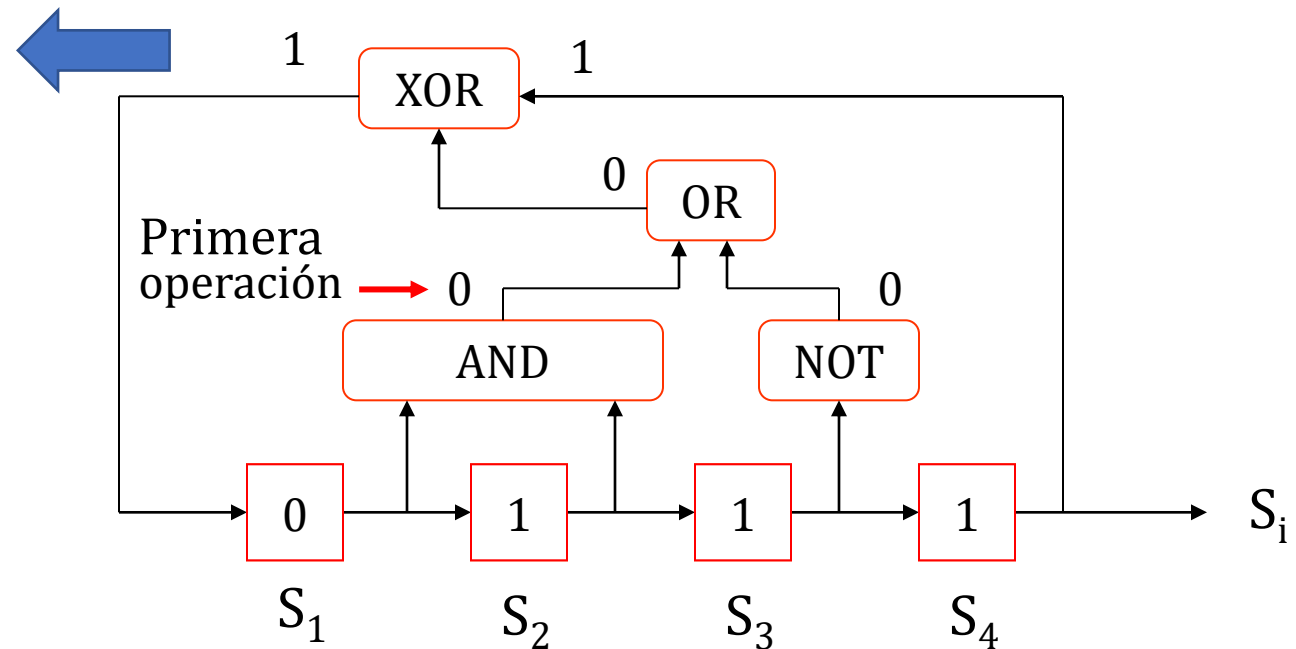
LFSR Linear Feedback Shift Register
Registros de Desplazamiento Realimentados Linealmente

Generadores con registros NLFSR

Ejemplo con generador NLFSR de cuatro celdas ($n = 4$)

Este es el estado de las celdas y operaciones previas en el registro, antes de realizar el desplazamiento de un bit hacia a la derecha

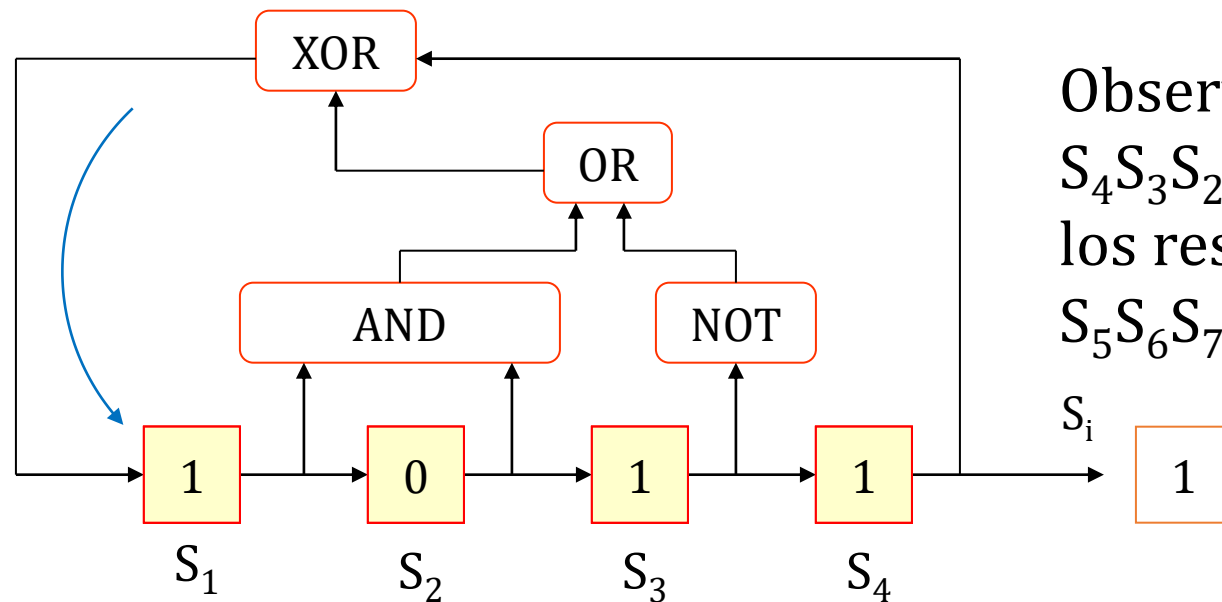
El bit S_4 (1) saldrá como S_i y los demás bits (0, 1, 1) se desplazan a la derecha y dejan a la celda S_1 vacía, sin valor



Sea la semilla: $S_1S_2S_3S_4 = 0111$

Generadores con registros NLFSR (cont.)

Semilla: $S_1S_2S_3S_4 = 0111$

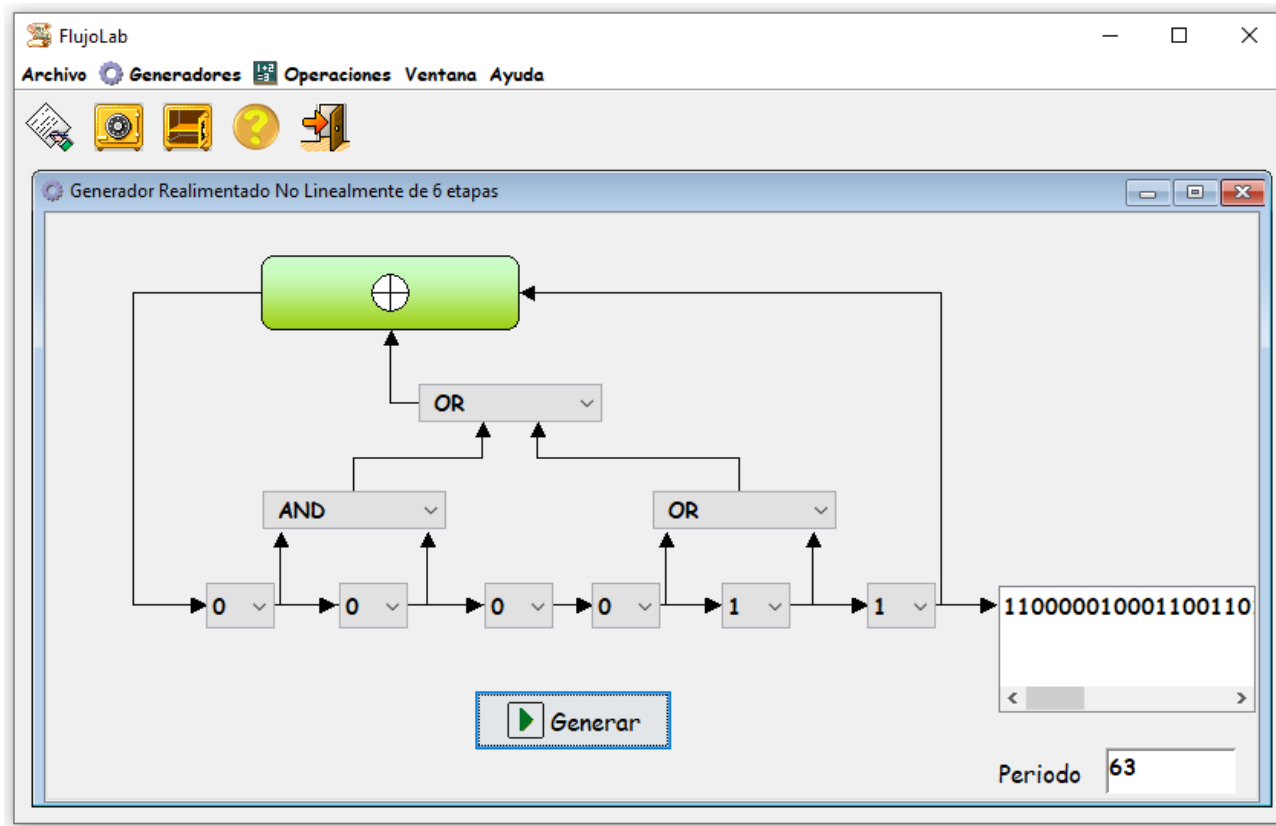


Observa que se transmite primero $S_4S_3S_2S_1$ (semilla al revés) y luego los restantes bits de la secuencia $S_5S_6S_7S_8S_9S_{10}S_{11}S_{12}S_{11}S_{14}S_{15}S_{16}$

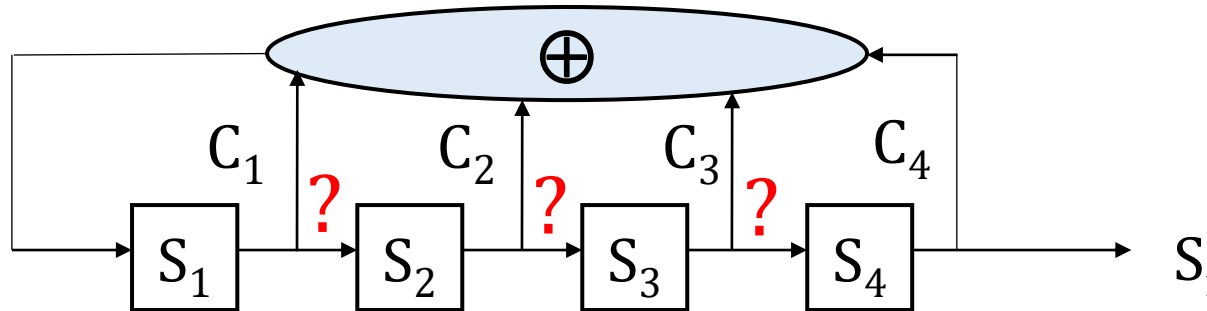
- $S_i = \underline{1110} 1100 1010 0001$. Su período es máximo: $T_{\text{máx}} = 2^n = 2^4 = 16$
- Se conoce como secuencia de Bruijn. El contenido de las celdas pasará por todos los estados posibles del registro, desde 0000 hasta 1111, no en orden

Práctica de NLFSR con FlujoLab

FlujoLab: https://www.criptored.es/software/sw_m001m.htm



Generadores con registros LFSR



- La función F es el XOR de algunas celdas, ya que cada C_i puede valer 0 (si no está conectada) o 1 (si está conectada), excepto C_n que siempre valdrá 1. La secuencia será: $S_1(t) = C_n S_n(t-1) \oplus \dots \oplus C_2 S_2(t-1) \oplus C_1 S_1(t-1)$
- A esta función de conexión de celdas se le asociará un polinomio, que nos permitirá clasificar los generadores en muy malos, malos y buenos para generar claves. Esta función es $f(x) = C_n x^n + C_{n-1} x^{n-1} + \dots + C_2 x^2 + C_1 x + 1$
- Aquí el período máximo $T_{\text{máx}} = 2^n - 1$, ya que se podrá pasar por todos los estados del registro, excepto n ceros. Con n ceros el registro no evoluciona al tener sólo un XOR

Tipos de polinomios usados en LFSR

- **Polinomios factorizables** (cuando pueden expresarse como producto de dos o más polinomios de menor grado). Se obtienen secuencias con un periodo inferior a $T_{\text{máx}}$ y estos valores además dependen de la semilla utilizada
- **Polinomios irreducibles** (cuando son no factorizables). En este caso se obtienen secuencias con un periodo fijo, que no depende de la semilla, pero son inferior a $T_{\text{máx}}$ al ser un múltiplo de ésta
- **Polinomios primitivos** (cuando, además de no factorizables, generan todos los restos). Son los más adecuados y los que deben usarse, ya que se obtienen secuencias máximas (m-secuencias) con periodo fijo $T_{\text{máx}} = 2^n - 1$
- En bibliografía se entrega una página para la factorización de polinomios y otras páginas con listas de polinomios primitivos

Generador LFSR con $f(x)$ factorizable

Generador $f(x)$ factorizable de cuatro celdas ($n = 4$)

Sea $f(x) = x^4 + x^2 + 1$

$f(x)$ es factorizable porque:

Sea $f(x_1) = f(x_2) = (x^2 + x + 1)$

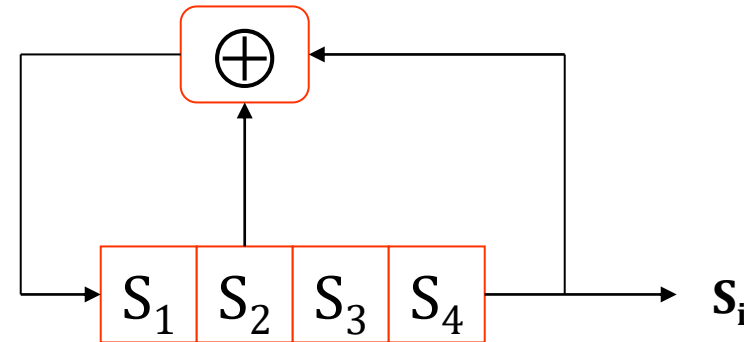
$f(x) = f(x_1) \cdot f(x_2)$

$f(x) = (x^2 + x + 1) \cdot (x^2 + x + 1)$

$f(x) = x^4 + \cancel{2x^3} + \cancel{3x^2} + \cancel{2x} + 1$

Tras la reducción módulo 2

Luego $f(x) = x^4 + x^2 + 1$

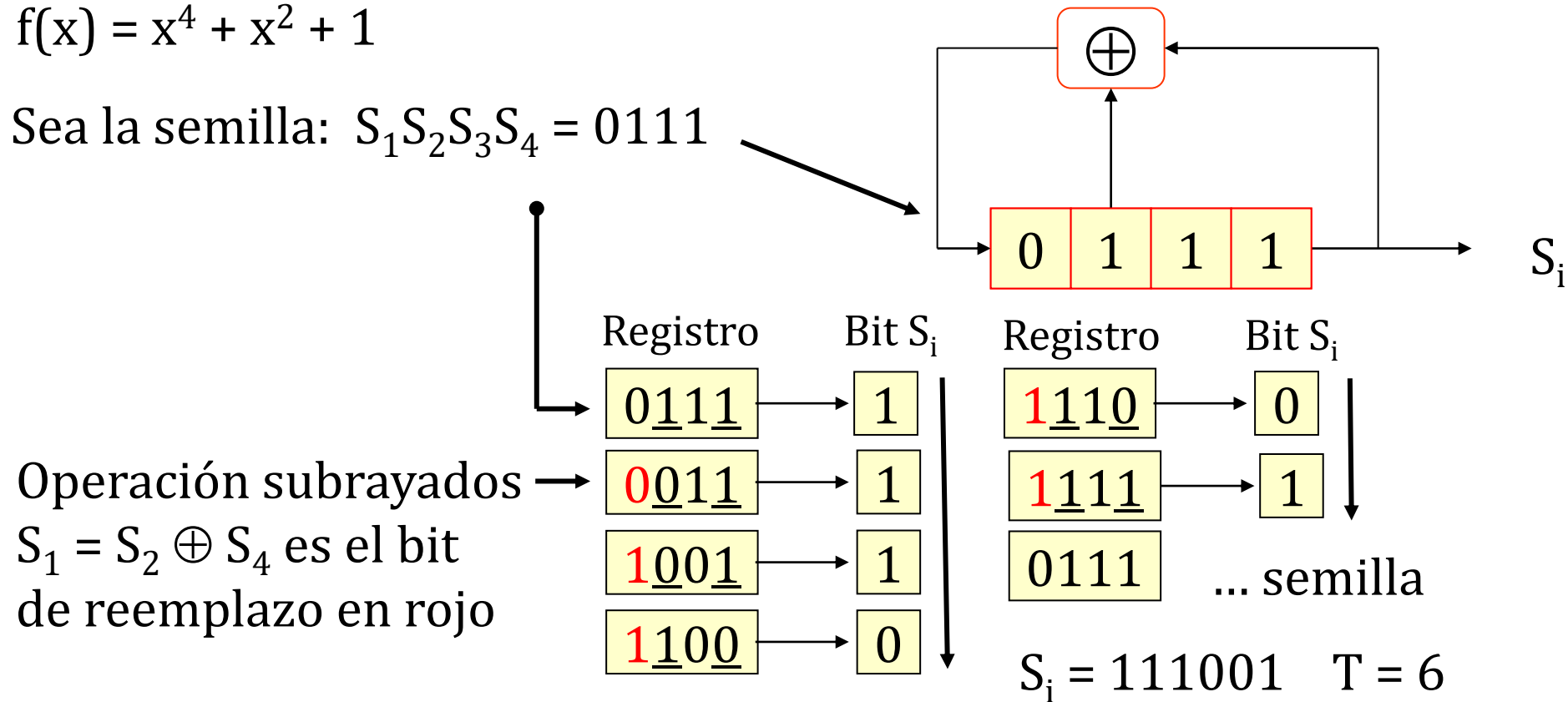


- Este polinomio es muy mala elección
- $T \leq 2^n - 1$
- Y además T dependerá de la semilla, habiendo períodos secundarios

Ejemplo LFSR con $f(x)$ factorizable (1/2)

$$f(x) = x^4 + x^2 + 1$$

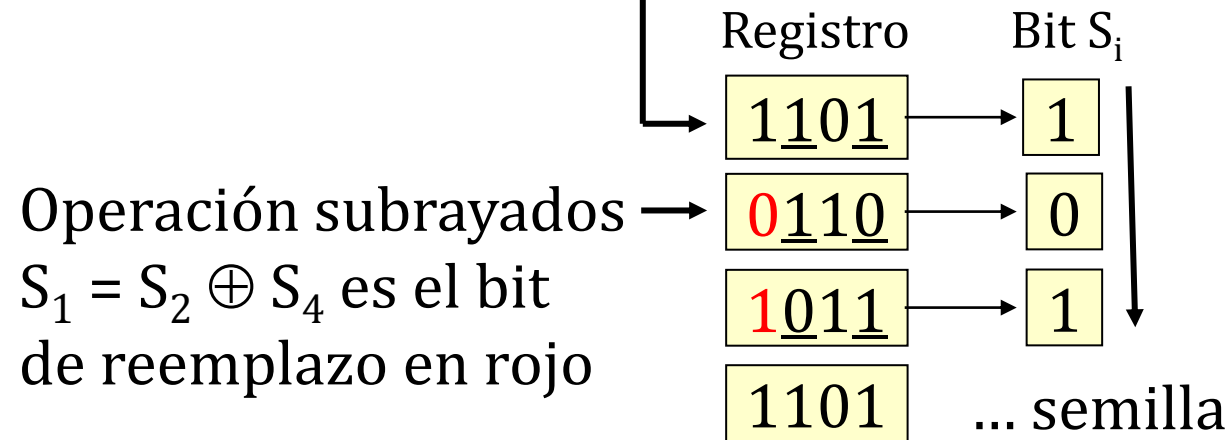
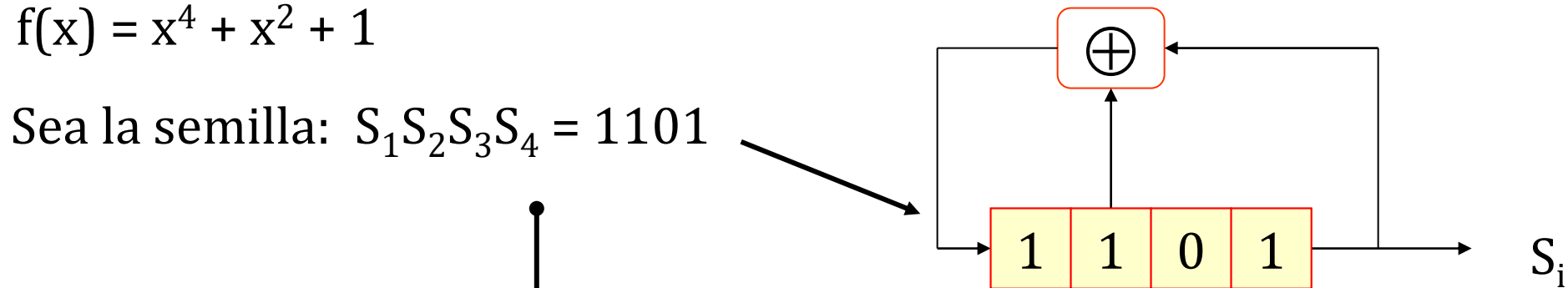
Sea la semilla: $S_1S_2S_3S_4 = 0111$



Ejemplo LFSR con $f(x)$ factorizable (2/2)

$$f(x) = x^4 + x^2 + 1$$

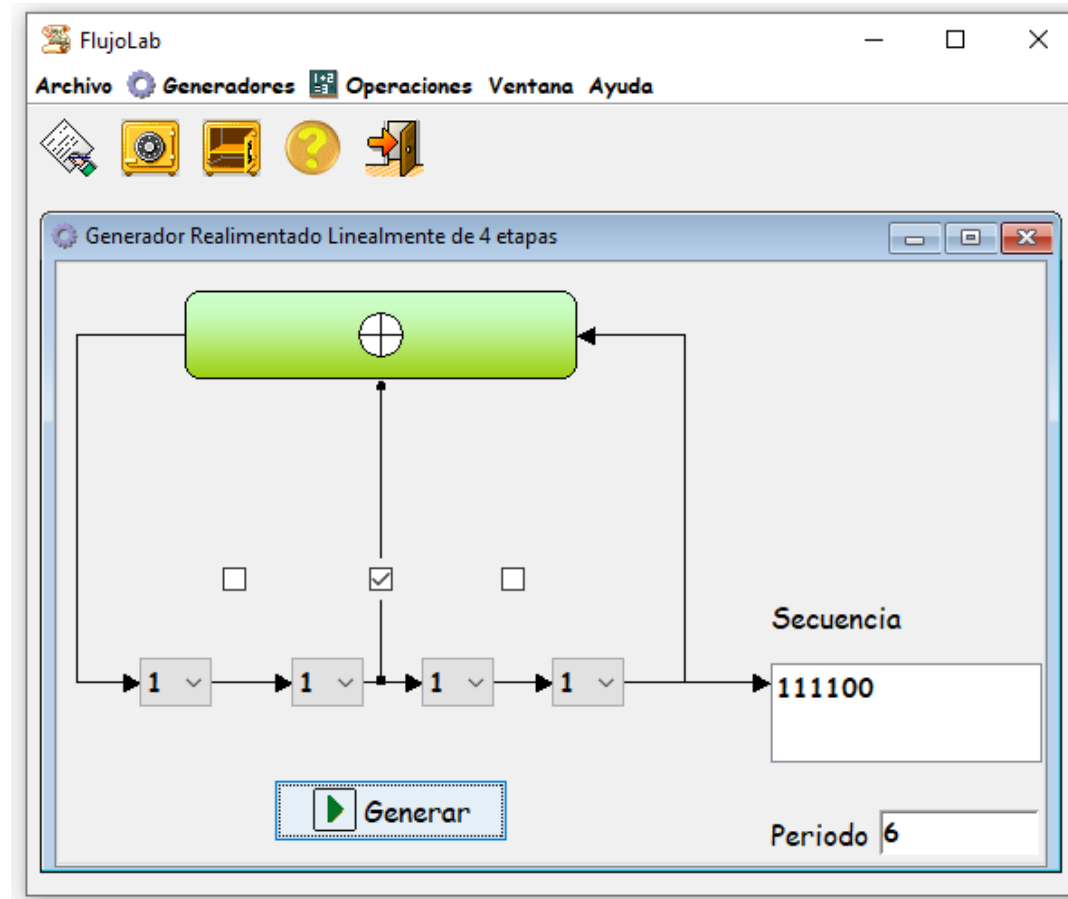
Sea la semilla: $S_1S_2S_3S_4 = 1101$



$S_1 = S_2 \oplus S_4$ es el bit de reemplazo en rojo

- $S_i = 101$
- $T = 3$
- T es un periodo secundario, aquí es incluso menor que la propia semilla

Prácticas de LFSR con $f(x)$ factorizable

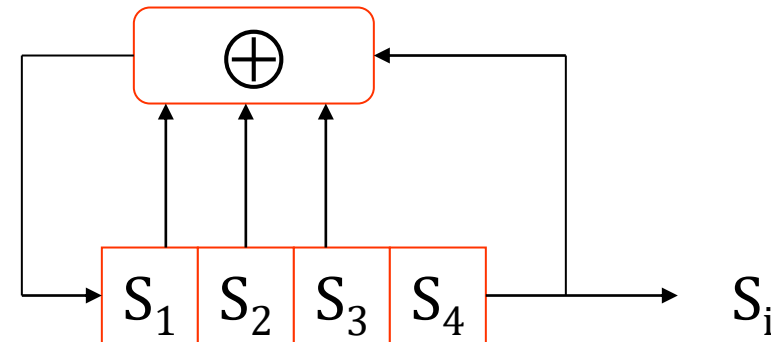


Generador LFSR con $f(x)$ irreducible

Generador $f(x)$ irreducible de cuatro celdas ($n = 4$)

Sea $f(x) = x^4 + x^3 + x^2 + x + 1$

- Es imposible factorizar el polinomio $f(x)$ en dos o más polinomios $f(x_i)$ de grado menor
- Ahora T ya no depende de la semilla, es un valor constante
- Este polinomio irreducible es mejor que el anterior factorizable
- Pero el periodo será un factor de $T_{\text{máx}} = 2^n - 1$ y no obtendremos un período máximo

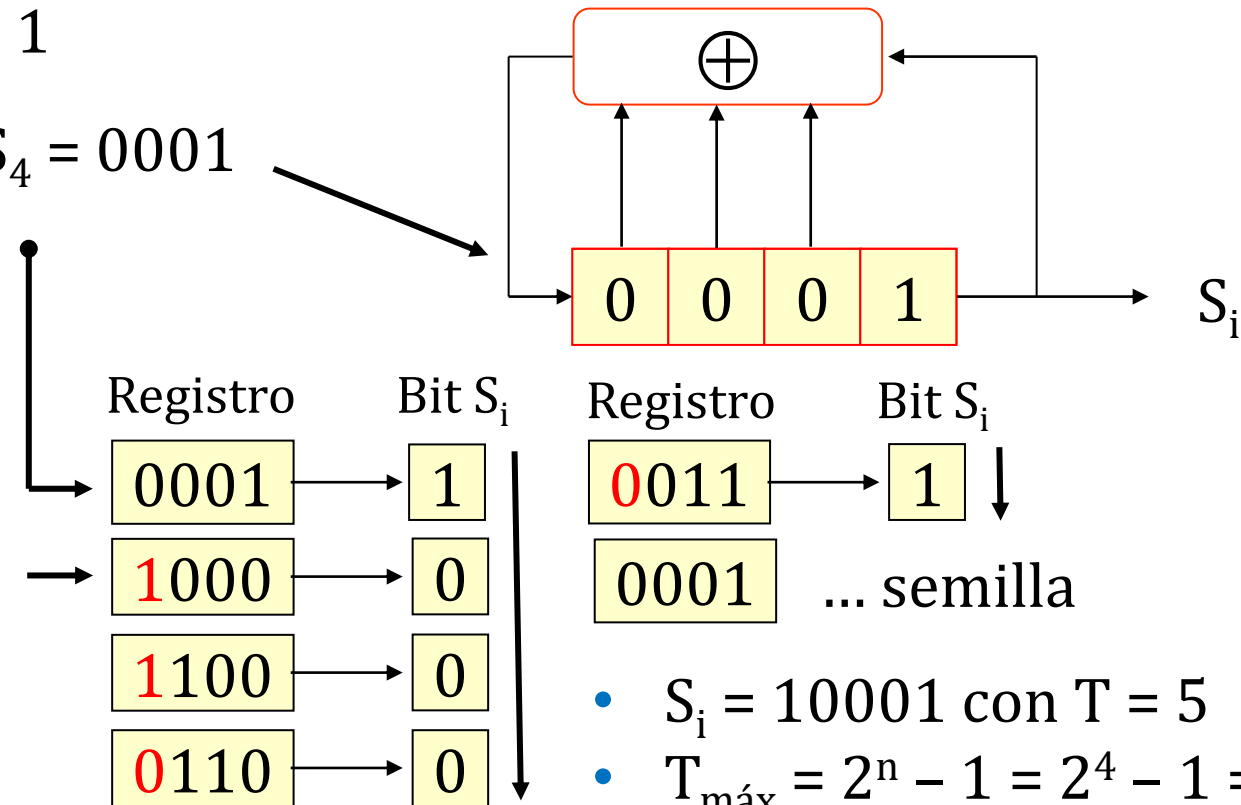


Ejemplo LFSR con $f(x)$ irreducible

$$f(x) = x^4 + x^3 + x^2 + x + 1$$

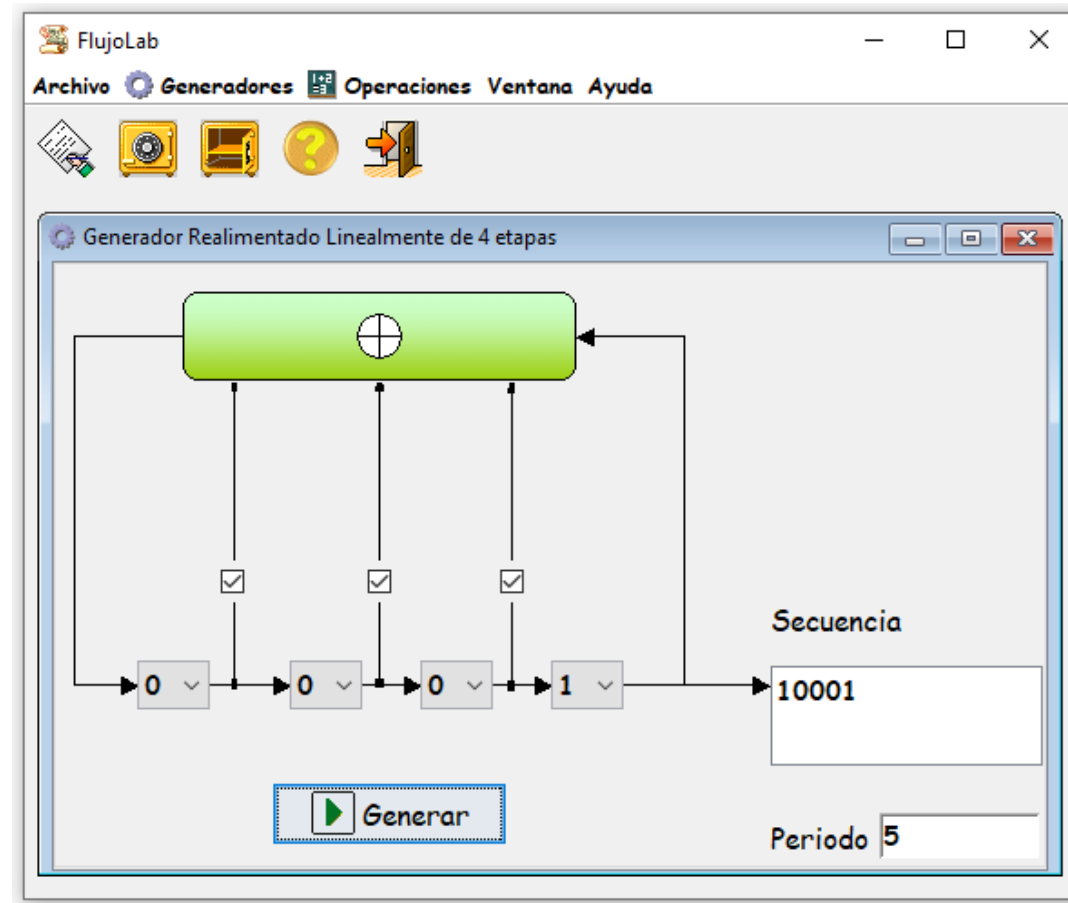
Sea la semilla: $S_1 S_2 S_3 S_4 = 0001$

Operación con todos
 $S_1 = S_1 \oplus S_2 \oplus S_3 \oplus S_4$
 marcando el bit de
 reemplazo en rojo



- $S_i = 10001$ con $T = 5$
- $T_{\text{máx}} = 2^n - 1 = 2^4 - 1 = 15$

Prácticas de LFSR con $f(x)$ irreducible

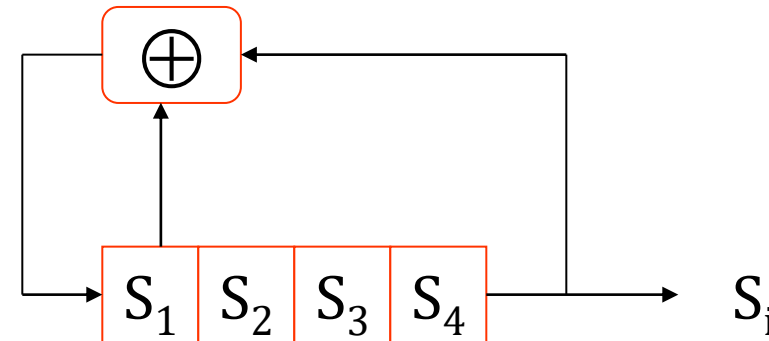


Generador LFSR con $f(x)$ primitivo

Generador $f(x)$ primitivo de cuatro celdas ($n = 4$)

Sea $f(x) = x^4 + x + 1$

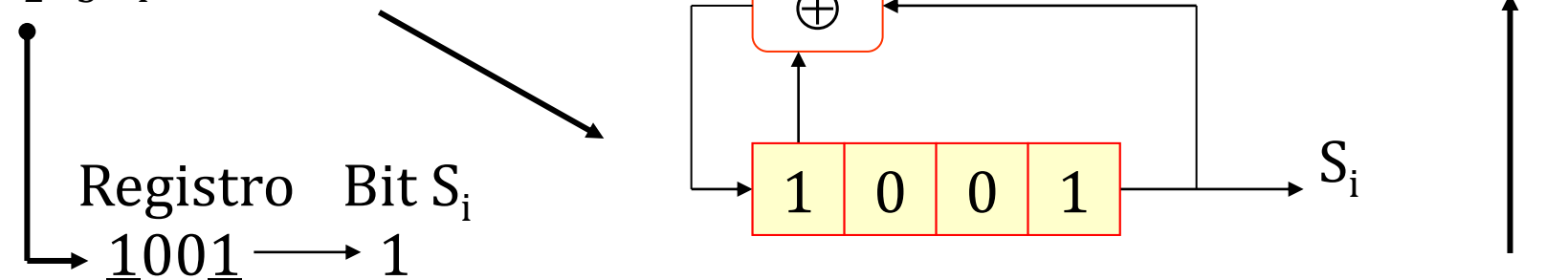
- El polinomio $f(x)$ no es factorizable mediante dos o más polinomios $f(x_i)$ de grado menor
- T será un valor constante y el máximo posible para registros lineales
- $T_{\text{máx}} = 2^n - 1$, característico de las denominadas m-secuencias
- Interesante: habrá $\phi(2^n - 1)/n$ polinomios primitivos para un generador LSFR primitivo de grado n



Ejemplo LFSR con $f(x)$ primitivo

$$f(x) = x^4 + x + 1$$

Sea la semilla: $S_1 S_2 S_3 S_4 = 1001$



Operación con bits
subrayados, rojo
bit de reemplazo

$$S_1 = S_1 \oplus S_4$$

$$\underline{0}10\underline{0} \longrightarrow 0$$

$$\underline{0}01\underline{0} \longrightarrow 0$$

$$\underline{0}00\underline{1} \longrightarrow 1$$

$$\underline{1}00\underline{0} \longrightarrow 0$$

$$\underline{1}10\underline{0} \longrightarrow 0$$

$$\underline{1}11\underline{0} \longrightarrow 0$$

$$\underline{1}11\underline{1} \longrightarrow 1$$

$$\underline{0}11\underline{1} \longrightarrow 1$$

$$\underline{1}01\underline{1} \longrightarrow 1$$

$$\underline{0}10\underline{1} \longrightarrow 1$$

$$\underline{1}01\underline{0} \longrightarrow 0$$

$$\underline{1}10\underline{1} \longrightarrow 1$$

$$\underline{0}11\underline{0} \longrightarrow 0$$

$$\underline{0}01\underline{1} \longrightarrow 1$$

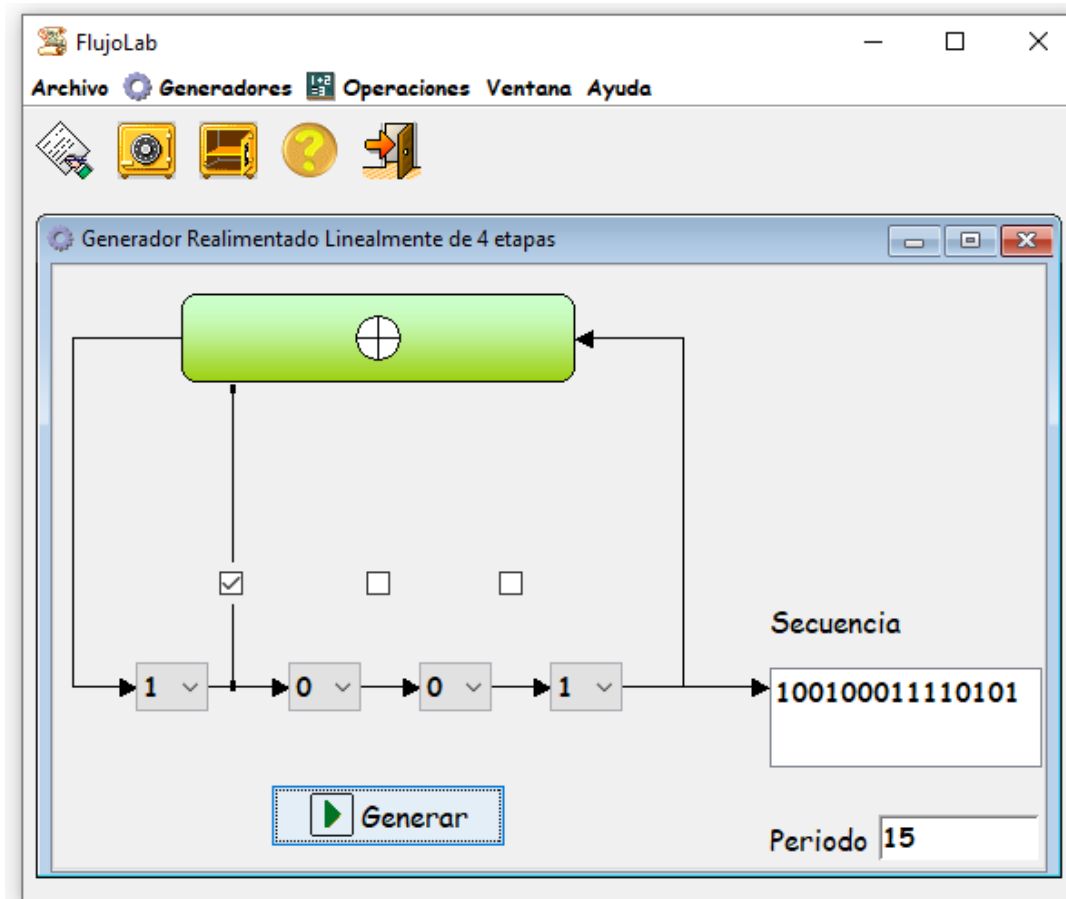
1001 ... semilla

$$T = 2^n - 1$$

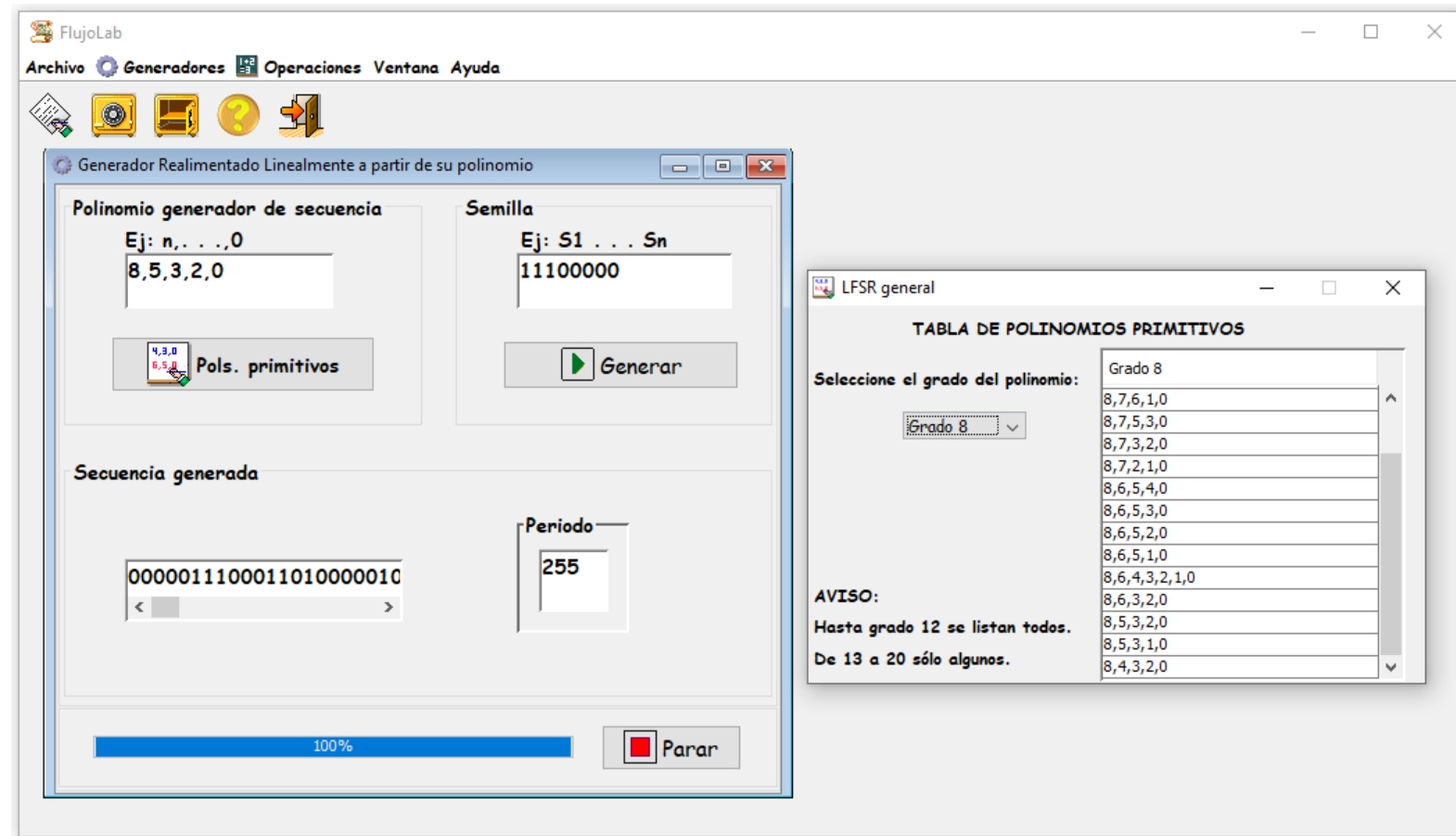
$$T = 2^4 - 1$$

$$T = 15$$

Prácticas de LFSR con $f(x)$ primitivo



Prácticas de LFSR con $f(x)$ general



Más información en píldoras Thoth



<https://www.youtube.com/watch?v=vfq3onw-uiM>

Conclusiones de la lección 9.2

- Los registros de desplazamiento realimentados son las máquinas o circuitos utilizados en los comienzos de los cifradores de flujo, en la década de los 90
- Los registros no lineales, conocidos como NLFSR contienen, además de la puerta xor que conecta el bit de la última celda, un conjunto de puertas lógicas que conectan las otras celdas al xor. No interesantes su uso porque en algunos casos se obtiene como máximo un periodo igual a 2^n bits, siendo n el número de celdas, pero a costa de un alto coste computacional
- Los registros lineales, conocidos como LFSR, contienen solamente una puerta xor como realimentación y están representados por un polinomio asociado. Si éste es factorizable, el periodo no es máximo y depende de la semilla. Si es irreducible, no depende de la semilla pero es un factor del máximo. Y si es primitivo, se obtiene un periodo máximo $2^n - 1$ bits, casi igual que con NLFSR

Lectura recomendada

- Guion píldora formativa Thoth nº 33, ¿Cómo se usan los registros de desplazamiento en la cifra?, Jorge Ramió, 2015
 - <https://www.criptored.es/thoth/material/texto/pildora033.pdf>
- Wolfram|Alpha, Factoring Polynomials Calculator
 - <https://www.wolframalpha.com/widgets/gallery/view.jsp?id=15b128aa42d812ef9f3a9640bf1fb3fa>
- Complete list of binary irreducible polynomials up to degree 11, Joerg Arndt, 2003
 - <https://www.jjj.de/mathdata/all-irredpoly.txt>
- Primitive Polynomial List, Arash Partow
 - <http://www.partow.net/programming/polynomials/index.html>
- Primitive Polynomials for the Field GF(2), Degree 2 Through Degree 16, Peter M. Maurer
 - <https://baylor-ir.tdl.org/baylor-ir/bitstream/handle/2104/8792/GF2%20Polynomials.pdf?sequence=1>

Class4crypt c4c9.3

Módulo 9. Criptografía simétrica en flujo

Lección 9.3. Aleatoriedad en registros LFSR con polinomio primitivo

9.3.1. Postulado de Golomb R1

9.3.2. Rachas de bits

9.3.3. Postulado de Golomb R2

9.3.4. Autocorrelación fuera de fase

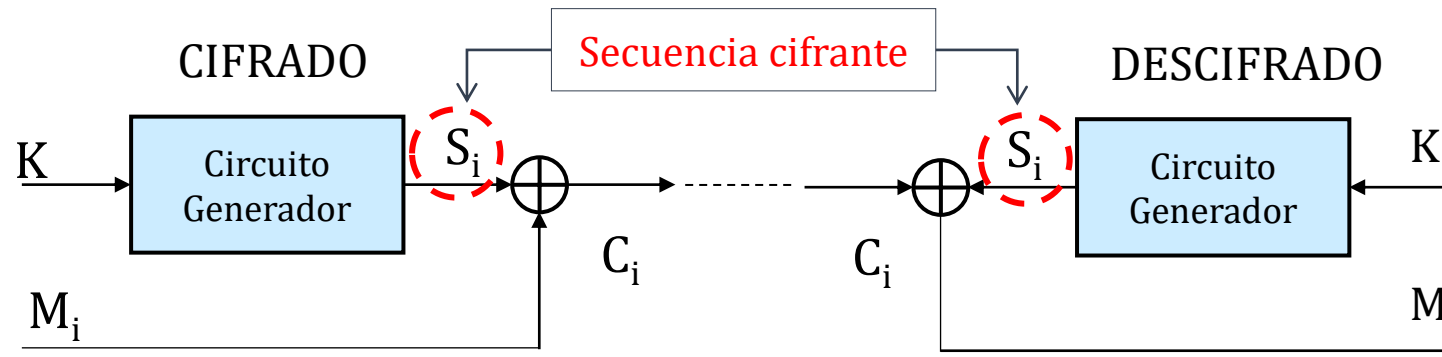
9.3.5. Postulado de Golomb R3

9.3.6. Prácticas con el software FlujoLab

Class4crypt c4c9.3 Aleatoriedad en registros LFSR con polinomio primitivo

https://www.youtube.com/watch?v=CoH_gI1rJgc

La seguridad en la cifra en flujo



- La seguridad de los sistemas de cifra en flujo depende exclusivamente de la secuencia cifrante S_i que debe tener las siguientes características:
 - La secuencia S_i debe tener una longitud en bits mayor que la del mensaje M
 - Aunque la secuencia S_i sea pseudoaleatoria, debe tener una apariencia aleatoria
 - La semilla K con la cual se inicia el circuito generador de secuencia S_i deberá usarse una sola vez (one-time pad)



Aleatoriedad de una secuencia de bits LFSR

- Existen varios test para poder demostrar la aleatoriedad de una secuencia. Por ejemplo, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* del NIST, Diehard y Dieharder (ver bibliografía)
- No obstante, en esta primera etapa vamos a usar los test básicos propuestos por Solomon Golomb en *Golomb's Randomness Postulates*
 - R1: Postulado de Golomb 1
 - R2: Postulado de Golomb 2
 - R3: Postulado de Golomb 3



Solomon Golomb
(1932 – 2016)

Postulado R1 de Golomb

- Equiprobabilidad de bits tomados independientemente
- Deberá existir igual número de ceros que de unos. Se acepta como máximo una diferencia igual a la unidad (en favor de los unos)

S_1 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 $T = 15$

- La secuencia S_1 cumple con R1 pues hay 8 unos y 7 ceros

S_2 1 1 1 0 1 0 1 1 0 1 0 0 1 0 0 1 $T = 16$

- La secuencia S_2 no cumple con R1 pues hay 9 unos y 7 ceros

Significado del postulado R1 de Golomb

S_i 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0

- Si una secuencia S_i como la indicada cumple con R1, quiere decir que la probabilidad de recibir un bit 1 es igual a la de recibir un bit 0, es decir un 50% o equiprobabilidad
- Por lo tanto, a lo largo de una secuencia S_i , independientemente de cuál bit se analiza, en media será igualmente probable recibir un 1 que un 0, pues en la secuencia hay mitad de unos y mitad de ceros

Ejemplo: S_i ? 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0

 ↑ ↑ ↑ ↑

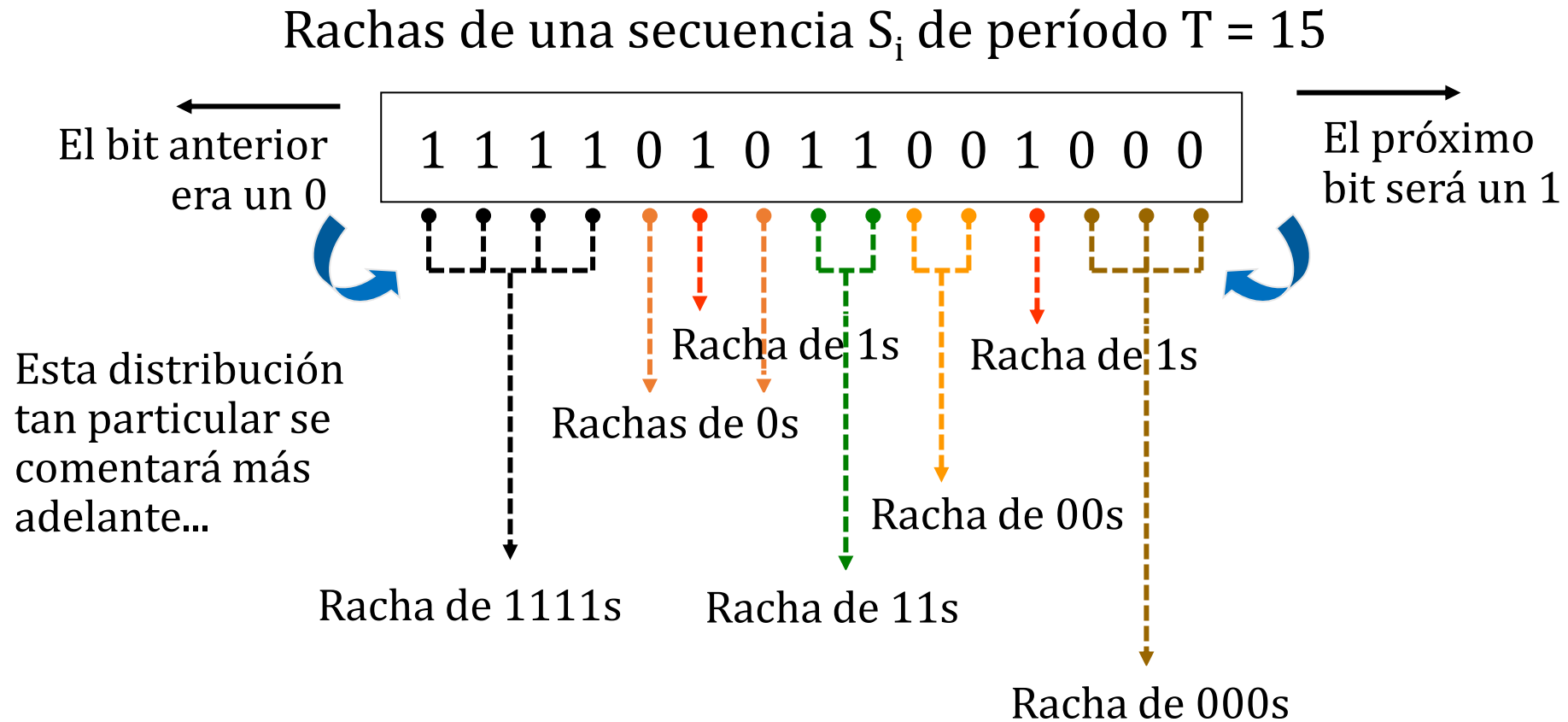
Postulado R2 de Golomb y rachas de bits

- Equiprobabilidad de próximo bit con secuencia anterior conocida
- En un período T la mitad de las rachas de S_i serán de longitud 1, la cuarta parte de longitud 2, la octava parte de longitud 3, etc., lo que se conoce como una distribución geométrica

S_i 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 $T = 15$

- Pero, ¿qué son las rachas de bits en una secuencia binaria?
- Una racha de bits es un conjunto de L bits ($L = 1, 2, 3, \dots$) iguales, separados por bits diferentes. Así, 010 es una racha de longitud de uno de 1s, 1001 es una racha de longitud dos de 0s y 01110 es una racha de longitud tres de 1s

Rachas de bits de una secuencia binaria



Distribución de las rachas de bits en S_i

- Las rachas deberán seguir una distribución estadística, de forma que la secuencia cifrante pseudoaleatoria S_i tenga un comportamiento lo más parecido a una secuencia aleatoria
- Para que esto se cumpla, es obvio que será necesario que haya mayor número de rachas de longitud corta ($L = 1, 2, 3, \dots$) que número de rachas de longitud larga ($L = \dots 7, 8, 9, \dots$), como se observaba en el ejemplo de la diapositiva anterior
- Este tipo de distribución seguirá una progresión geométrica
- Es decir, una secuencia S_i podría tener 256 rachas de longitud uno, 128 rachas de longitud dos, 64 rachas de longitud tres, 32 rachas de longitud cuatro, etc., siempre igual cantidad de rachas de 0s que de 1s

Significado del postulado R2 de Golomb

S_i 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 $T = 15$

- Si una secuencia S_i como la indicada cumple con R2, quiere decir que la probabilidad de recibir un bit 1 o un bit 0, después de haber recibido un conjunto de bits 0s o 1s sigue siendo la misma, es decir un 50%
- En este caso de $T = 15$, conocido un 1 el par 10 será igual de probable que el par 11. Recibido un 0, las cadenas 00 y 01 son equiprobables. Lo mismo un conocido 00, 01, 10, 11, 001, 010, 011, 100, 101, 110 o 111
- Es decir, existirá una equiprobabilidad a pesar de los bits ya conocidos


S_i ? 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0

↑ ↑ ↑ ↑

Importancia del postulado R2 de Golomb

- Cuando una secuencia S_i cumple con R2, significa que el registro LFSR pasará por todos los estados o restos posibles del mismo, desde 0000...001 hasta 1111...111, no orden lógicamente
- Por lo tanto, se obtendrá la mayor secuencia posible que puede aportar ese registro, con un periodo $T_{\text{máx}} = 2^n - 1$
- Esto es importante porque, además de la equiprobabilidad del próximo bit a pesar de conocer los anteriores bits, que es vital de cara para la apariencia de aleatoriedad de la secuencia, obtenemos una secuencia máxima, que también lo estábamos buscando
- Pero esto también entregará pistas al atacante...

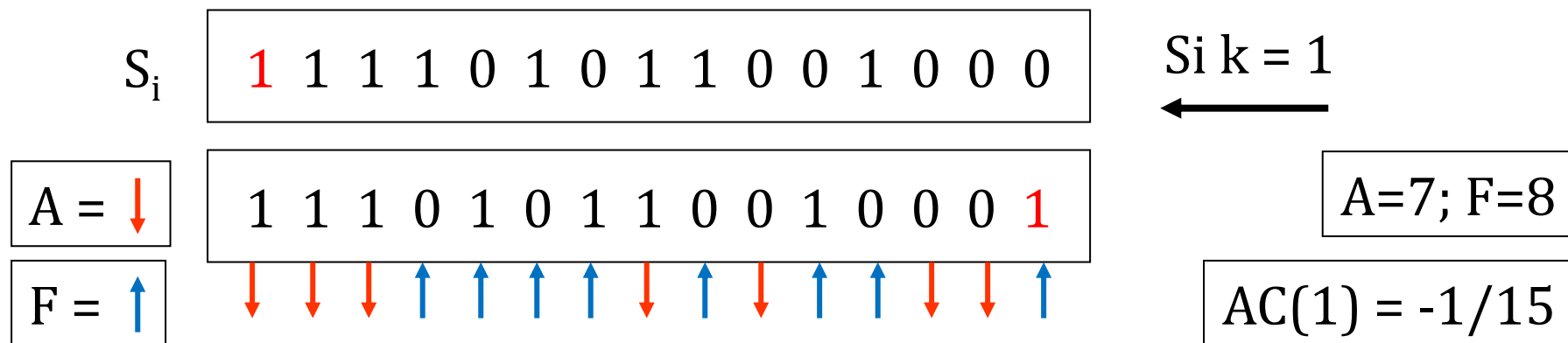
Polinomio primitivo, rachas y m-secuencias

- Cumplir con R2 nos llevará a un periodo $T_{\text{máx}} = 2^n - 1$, lo que se conoce como m-secuencia
 - Las m-secuencias solamente se logran con un LFSR que tenga polinomio asociado primitivo
 - Habrá un total de 2^{n-1} rachas en una m-secuencia (50% serán de 1s y 50% serán de 0s)
 - Sea $f(x) = x^{11} + x^2 + 1$ con periodo $T_{\text{máx}} = 2^{11} - 1 = 2.047$
- 

Longitud	Rachas totales	Rachas de ceros	Rachas de unos
1	512	256	256
2	256	128	128
3	128	64	64
4	64	32	32
5	32	16	16
6	16	8	8
7	8	4	4
8	4	2	2
9	2	1	1
10	1	1	0 siempre
11	1	0	1 siempre
Totales	1024	512	512

Postulado R3 de Golomb y AC(k)

- La autocorrelación fuera de fase AC(k) de una cadena o secuencia de bits, desplazada k bits, debe permanecer constante para todos los desplazamientos, es decir, desde $k = 1$ hasta $k = n - 1$
- $AC(k) = (A - F)/T$, donde A son los aciertos (bits iguales) y F son los fallos (bits diferentes) al comparar la cadena con su desplazada



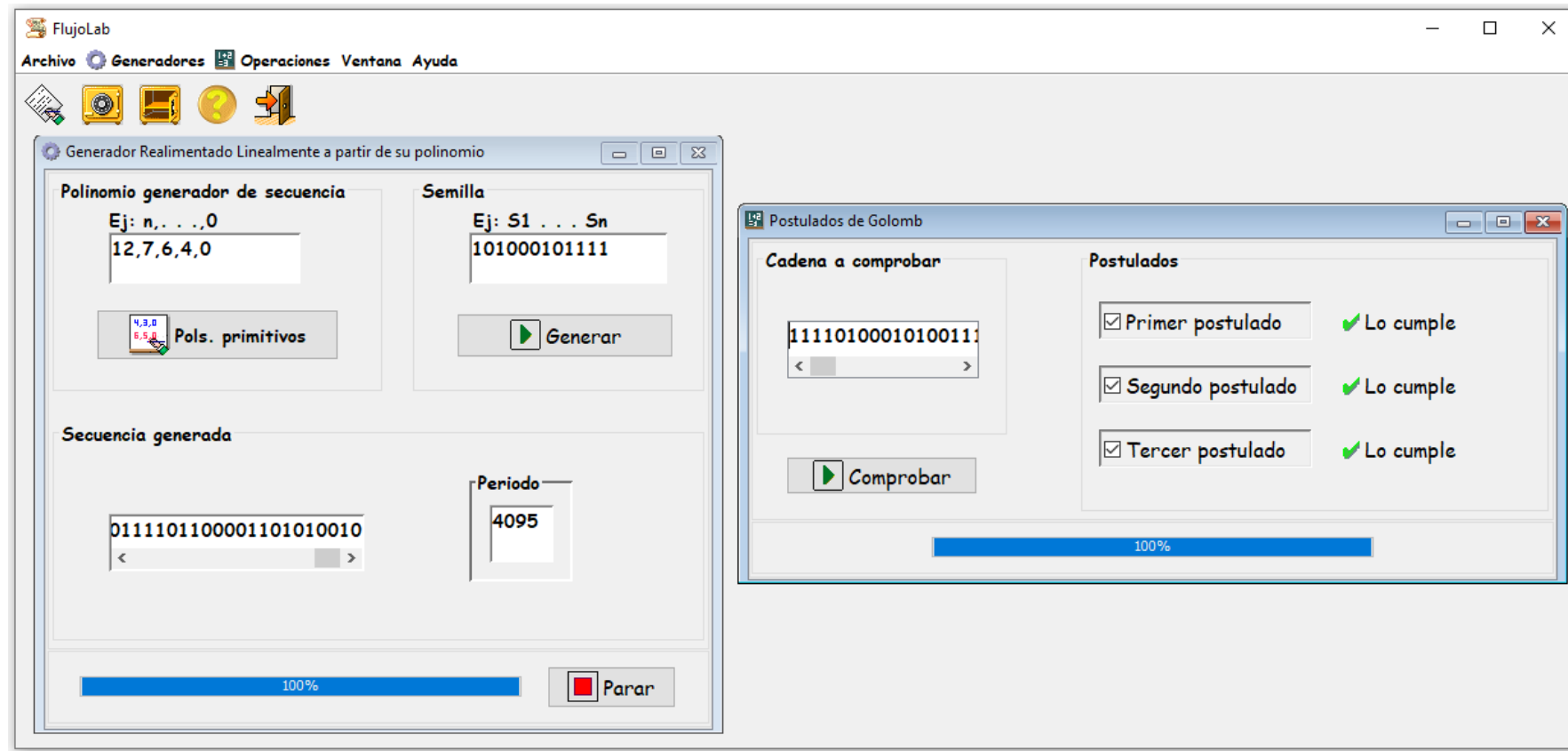
Significado del postulado R3 de Golomb

S_i	1 0 0 1 1 1 0	Aciertos: color Fallos: color
$k = 1$	0 0 1 1 1 0 1	$AC(1) = (3-4)/7 = -1/7$
$k = 2$	0 1 1 1 0 1 0	$AC(2) = (3-4)/7 = -1/7$
$k = 3$	1 1 1 0 1 0 0	$AC(3) = (3-4)/7 = -1/7$
$k = 4$	1 1 0 1 0 0 1	$AC(2) = (3-4)/7 = -1/7$
$k = 5$	1 0 1 0 0 1 1	$AC(3) = (3-4)/7 = -1/7$
$k = 6$	0 1 0 0 1 1 1	$AC(3) = (3-4)/7 = -1/7$

- Si una secuencia S_i cumple con el postulado R3, quiere decir que independientemente del trozo de secuencia elegido o conocido por el atacante, no tendrá una mayor cantidad de información que en otro trozo cualquiera de la misma
- Será imposible entonces deducir cómo continúa la secuencia desde ese trozo u obtener información de otro tipo... “analogía” con el ataque de Kasiski a Vigenère...



Prácticas de postulados de Golomb



FlujoLab: https://www.criptored.es/software/sw_m001m.htm

Más información en píldoras Thoth



<https://www.youtube.com/watch?v=fLN-jYUlsv8>

Conclusiones de la lección 9.3

- Aunque existen muchos test mejores y más eficientes, una buena primera aproximación para poder apreciar las características de aleatoriedad de una secuencia S_i cifrante en un LFSR, es estudiar los postulados de Golomb
- El postulado de Golomb R1 habla de la equiprobabilidad de encontrar un bit 1 o un bit 0 en cualquier lugar de la secuencia
- El postulado de Golomb R2 habla de la misma equiprobabilidad de encontrar un bit 1 o un bit 0 en la secuencia, pero conociendo de antemano un conjunto de bits anteriores. Está relacionado con las rachas de bits
- El postulado de Golomb R3 habla de la nula información que debería entregar un trozo de la secuencia S_i sobre los bits precedentes o los bits posteriores. Está relacionado con la autocorrelación fuera de fase $AC(k)$

Lectura recomendada (1/2)

- A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST, Revised 2010
 - <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf>
- Diehard tests
 - https://en.wikipedia.org/wiki/Diehard_tests
- Dieharder, Robert G. Brown's General Tools Page
 - <https://webhome.phy.duke.edu/~rgb/General/dieharder.php>

Lectura recomendada (2/2)

- Golomb's Randomness Postulates
 - https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_351
 - <http://www.iitg.ac.in/pinaki/Golomb.pdf>
- Shift Register Sequences, Golomb Solomon, Aegean Park Press, Laguna Hills, CA, 1982
 - <https://www.amazon.com/exec/obidos/ASIN/0894120484/acmorg-20>
- Guion píldora formativa Thoth nº 32, ¿Qué son los postulados de Golomb?, Jorge Ramió, 2015
 - <https://www.criptored.es/thoth/material/texto/pildora032.pdf>

Class4crypt c4c9.4

Módulo 9. Criptografía simétrica en flujo

Lección 9.4. Complejidad en LFSR, A5, RC4 y ChaCha20

9.4.1. Complejidad algorítmica de los registros de desplazamiento FSR

9.4.2. Ataque de Berlekamp-Massey a una m-secuencia

9.4.3. Uso de dos o más registros para generar secuencias cifrantes

9.4.4. Algoritmo A5

9.4.5. Algoritmo RC4

9.4.6. Algoritmo ChaCha20

Class4crypt c4c9.4 Complejidad en LFSR, A5, RC4 y ChaCha20
<https://www.youtube.com/watch?v=trb3eh7hh3c>

Complejidad algorítmica de los FSR

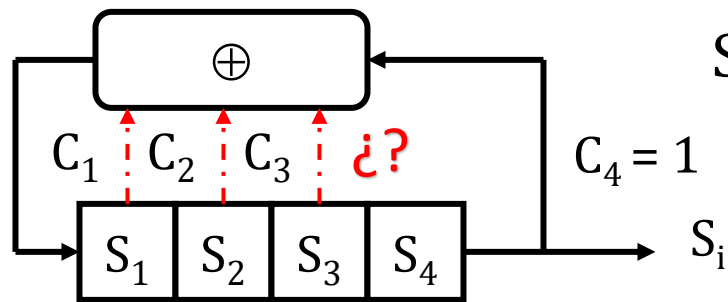
- Se entiende por complejidad de un registro a la predictibilidad de la secuencia cifrante que genera y de su sencillez matemática
- Los registros NLFSR tienen una complejidad mayor que los LFSR, ya que los bits de la secuencia cifrante dependen de operaciones donde intervienen funciones lógicas y en LFSR es un simple xor
- Por lo tanto, los NLFSR tendrán una complejidad exponencial y los LFSR una complejidad lineal (LC)
- Así, los NLSFR serán más robustos pero no será fácil determinar el periodo; en cambio, en los LFSR ese periodo será siempre el de una m-secuencia $2^n - 1$ bits para un polinomio asociado primitivo

Características de las m-secuencias

- Cumplen con los tres postulados de Golomb G1, G2 y G3
- Su debilidad reside en que tienen una baja complejidad lineal LC, entendida ésta como la longitud del registro de desplazamiento más corto que es capaz de generar dicha secuencia cifrante S_i . En concreto, tendremos que para un único LFSR la $LC = n$
- El ataque de Berlekamp-Massey será capaz de reconstruir toda la secuencia S_i de $2^n - 1$ bits a partir solo de $2 \cdot n$ bits consecutivos
- Así, con 80 bits consecutivos de S_i , se puede romper el registro que genera la secuencia cifrante S_i de $2^{40} - 1 = 1.099.511.627.775$ bits
- Para solucionar este problema, vamos a aumentar la complejidad lineal LC, uniendo dos o más registros LFSR en un generador

Ataque Berlekamp-Massey a m-secuencias

Si conocemos $2 \cdot n = 8$ bits consecutivos $b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8$ de un LFSR de 4 celdas, con conexiones C_1, C_2, C_3 y C_4 , podemos plantear este sistema de ecuaciones



$$S_5 = C_1 \cdot S_1 \oplus C_2 \cdot S_2 \oplus C_3 \cdot S_3 \oplus C_4 \cdot S_4$$

$$S_6 = C_1 \cdot S_5 \oplus C_2 \cdot S_1 \oplus C_3 \cdot S_2 \oplus C_4 \cdot S_3$$

$$S_7 = C_1 \cdot S_6 \oplus C_2 \cdot S_5 \oplus C_3 \cdot S_1 \oplus C_4 \cdot S_2$$

$$S_8 = C_1 \cdot S_7 \oplus C_2 \cdot S_6 \oplus C_3 \cdot S_5 \oplus C_4 \cdot S_1$$

Si conocemos esos $2 \cdot n = 8$ bits $S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8$, vamos a poder resolver este sistema

Es importante recordar que se transmite primero la semilla $S_4 S_3 S_2 S_1$ y, a continuación, los siguientes bits de la secuencia $S_5 S_6 S_7 S_8$

Ataque de B-M con la cadena 11001000

- Si conocemos estos 8 bits consecutivos $b_1b_2b_3b_4b_5b_6b_7b_8 = 11001000$, se puede decir entonces que $S_4S_3S_2S_1S_5S_6S_7S_8 = 11001000$, porque primero se transmite o sale del registro la semilla $S_1S_2S_3S_4$ al revés, es decir, $S_4S_3S_2S_1$

$$S_5 = C_1 \bullet S_1 \oplus C_2 \bullet S_2 \oplus C_3 \bullet S_3 \oplus C_4 \bullet S_4 \quad \text{Ecuación 1} \quad 1 = C_1 \bullet 0 \oplus C_2 \bullet 0 \oplus C_3 \bullet 1 \oplus C_4 \bullet 1$$

$$S_6 = C_1 \bullet S_5 \oplus C_2 \bullet S_1 \oplus C_3 \bullet S_2 \oplus C_4 \bullet S_3 \quad \text{Ecuación 2} \quad 0 = C_1 \bullet 1 \oplus C_2 \bullet 0 \oplus C_3 \bullet 0 \oplus C_4 \bullet 1$$

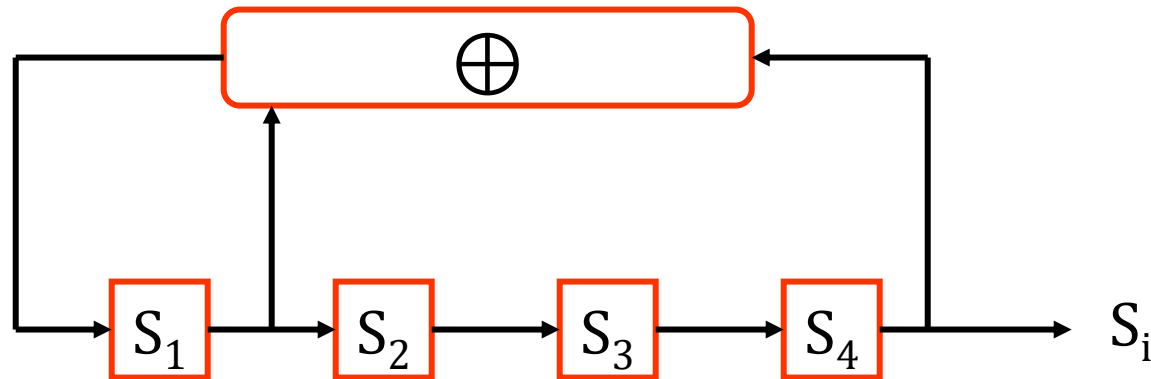
$$S_7 = C_1 \bullet S_6 \oplus C_2 \bullet S_5 \oplus C_3 \bullet S_1 \oplus C_4 \bullet S_2 \quad \text{Ecuación 3} \quad 0 = C_1 \bullet 0 \oplus C_2 \bullet 1 \oplus C_3 \bullet 0 \oplus C_4 \bullet 0$$

$$S_8 = C_1 \bullet S_7 \oplus C_2 \bullet S_6 \oplus C_3 \bullet S_5 \oplus C_4 \bullet S_1 \quad \text{Ecuación 4} \quad 0 = C_1 \bullet 0 \oplus C_2 \bullet 0 \oplus C_3 \bullet 1 \oplus C_4 \bullet 0$$

- Como C_n debe ser 1 (aquí $C_4 = 1$), puede resolverse usando simple lógica
 - Ecuación 1** Si $1 = C_3 \bullet 1 \oplus C_4 \bullet 1 = C_3 \bullet 1 \oplus 1 \bullet 1$, entonces C_3 debe ser 0
 - Ecuación 2** Si $0 = C_1 \bullet 1 \oplus C_4 \bullet 1 = C_1 \bullet 1 \oplus 1 \bullet 1$, entonces C_1 debe ser 1
 - Ecuación 3** Si $0 = C_2 \bullet 1$, entonces C_2 debe ser 0

Solución del ataque B-M anterior

- El registro LFSR tenía como valores de conexión $C_1 = 1$, $C_2 = 0$, $C_3 = 0$ y $C_4 = 1$



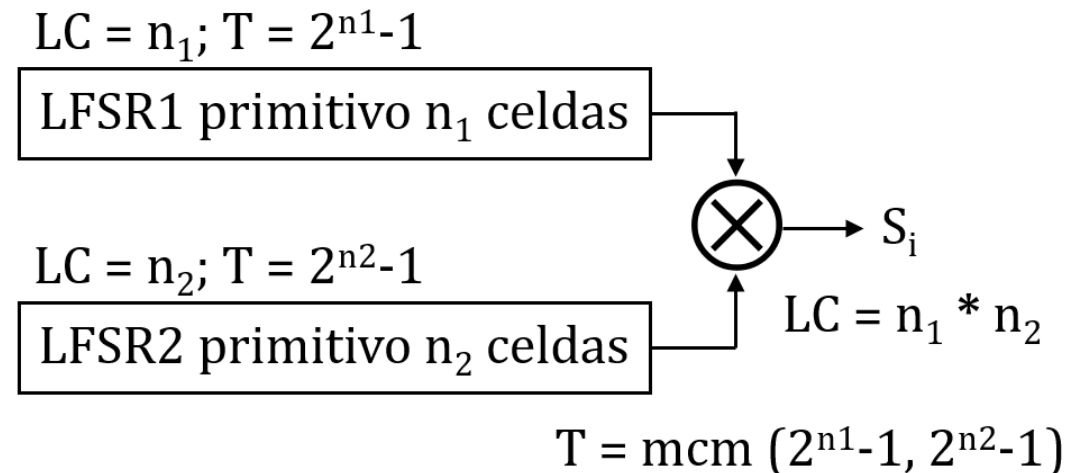
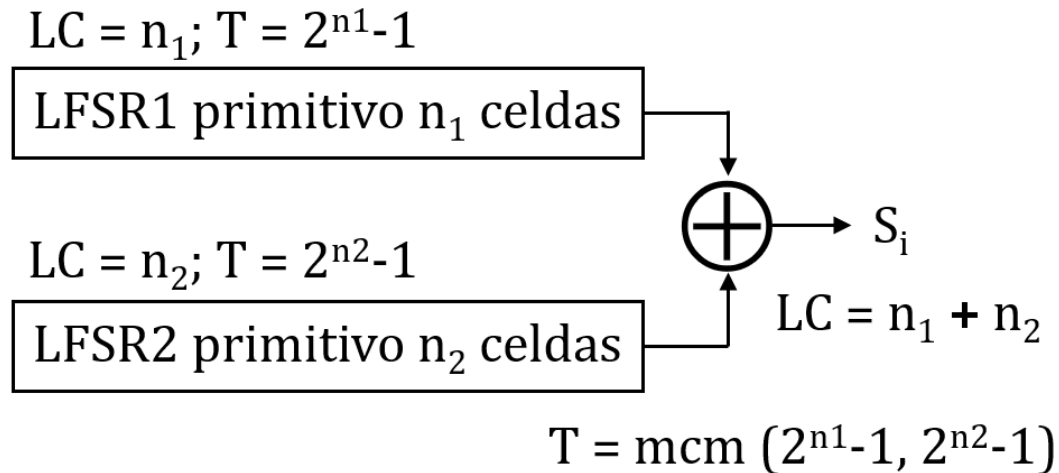
- $f(x) = x^4 + x + 1$

- Con cualquier semilla $S_1S_2S_3S_4$ que usemos, desde 0001 hasta 1111, se obtiene la misma secuencia de $2^4 - 1 = 15$ bits, **PERO** su inicio va a depender de la semilla que se elija. Por ello, con este ataque se rompe solamente el registro, si éste fuese secreto, pero **NO** la misma secuencia cifrante en fase usada en la cifra, salvo que se parta con la semilla original que obviamente era secreta

¿Cómo podemos romper ahora la cifra?

- Como la semilla tendrá n bits y el valor de n ceros está prohibido, conocido el polinomio del registro LFSR habrá que probar como máximo con las $2^n - 1$ semillas posibles mediante fuerza bruta
- Si el mensaje fuese texto o archivo txt, entonces no sería necesario generar una a una las $2^n - 1$ secuencias para descifrar, sino tomar una ventana de algunas decenas de bits de la secuencia (por ejemplo 80 bits) y hacer un xor con los bits del criptograma
- Si no se obtiene texto en claro, se desplaza esa ventana un bit a la derecha, se hace el xor y se sigue recorriendo el criptograma
- Cuando aparezca texto en claro, conoceremos la semilla del LFSR

Aumentando la complejidad lineal de LFSR



- ¿Se cumplen ahora los postulados de Golomb G1, G2 y G3?
- Existen muchos ejemplos de generadores con filtrado no lineal
 - Geffe, Beth-Piper, GollMann, Massey-Rueppel
 - Pero no los estudiaremos en este curso

Ejemplos con software FlujoLab

The screenshot displays the FlujoLab software interface, which is used for simulating Linear Feedback Shift Registers (LFSRs) and related cryptographic operations. The interface consists of four main windows:

- Generador Realimentado Linealmente a partir de su polinomio:** This window allows users to define a polynomial generator and a seed. The polynomial is given as coefficients (e.g., 10, 9, 4, 2, 0) and the seed as a binary string (e.g., 1111100000). It shows the generated sequence (010011111111011011011) and the period (1023).
- Aumento de la complejidad lineal con LFSR's. Suma:** This window demonstrates the combination of two LFSRs using addition. It shows two polynomials (5, 2, 0 and 7, 4, 0) and two seeds (00111 and 1000111). The resulting sequence is 00000101011010101011001100001010011000001101100111101000101010001001000001, with a period of 3937.
- Aumento de la complejidad lineal con LFSR's. Multiplicación:** This window demonstrates the combination of two LFSRs using multiplication. It shows two polynomials (5, 2, 0 and 7, 4, 0) and two seeds (00111 and 1000111). The resulting sequence is 11100010100100000010011011010000110010010000000000100010001000001010110010000010000101010001100, with a period of 3937.
- Ataque de Berlekamp-Massey:** This window shows the Berlekamp-Massey algorithm being applied to a sequence (0100001111010001111) and a polynomial generator (10, 9, 4, 2, 0). It includes a progress bar and an 'Ataque' button.

Algoritmo A5

- Desarrollado en 1987, se usa en el cifrado del enlace de las comunicaciones de voz entre el abonado y la red de telefonía móvil en redes GSM o 2G
- Existen dos versiones: A5/1 y A5/2, esta última mucho más débil
- Cada trama de una comunicación entre A y B tiene 228 bits (114 bits en cada sentido). El generador entregará los 228 bits pseudoaleatorios para la cifra de cada trama a partir de la clave de sesión K_c de 64 bits
- La clave de sesión K_c puede utilizarse durante varios días y se calcula en la tarjeta SIM del abonado a partir de la clave individual K_i y de una trama RAND de 128 bits que la red de telefonía envía al usuario
- Su código se mantuvo en secreto (grave error) hasta que fue atacado
- El sistema A5/1 sucumbió en diciembre de 1999 a un ataque realizado por Alex Biryukov, Adi Shamir y David Wagner con un ataque con texto en claro conocido
- En diciembre de 2009, el ingeniero alemán Karsten Nohl criptoanaliza el sistema A5/1 a través de un ataque activo, ahora sin texto en claro conocido

Esquema de A5/1

Usa 3 registros
LFSR primitivos

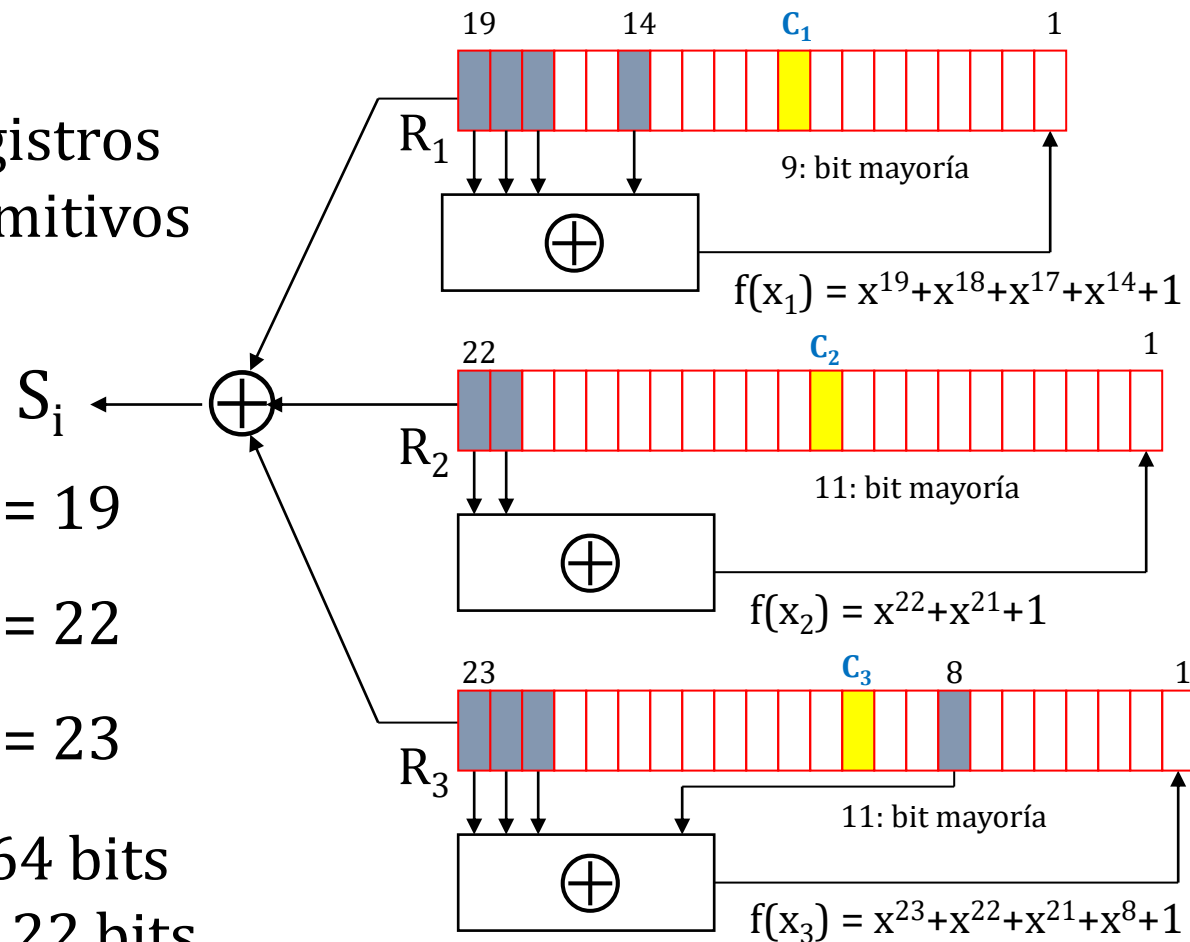
$$R_1 \Rightarrow n_1 = 19$$

$$R_2 \Rightarrow n_2 = 22$$

$$R_3 \Rightarrow n_3 = 23$$

Clave = 64 bits

Nonce = 22 bits



Una función mayoría
entre C_1 , C_2 y C_3 hace que
sólo en los registros en
los que el bit coincide
haya un desplazamiento

El nonce es un valor
público, que se modifica
para cada una de las
tramas intercambiadas y
que se emplea para la
sincronización de las
comunicaciones

Función mayoría y periodo en A5/1

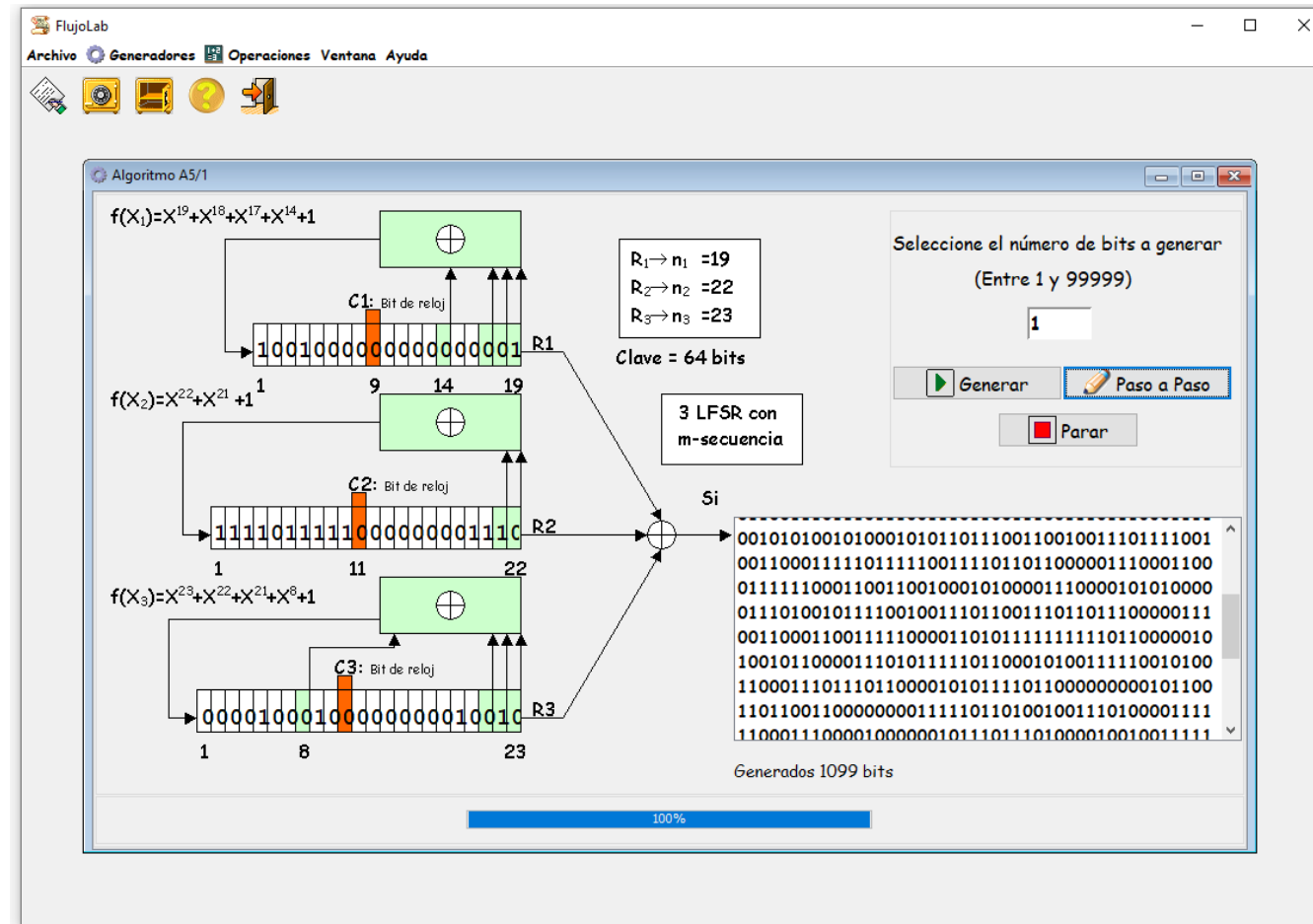
$$F(C_1, C_2, C_3) = C_1C_2 \oplus C_1C_3 \oplus C_2C_3$$

C_1	C_2	C_3	
0	0	0	$F = 0$: Se desplazan todos
0	0	1	$F = 0$: No desplaza R_3
0	1	0	$F = 0$: No desplaza R_2
0	1	1	$F = 1$: No desplaza R_1

C_1	C_2	C_3	
1	0	0	$F = 0$: No desplaza R_1
1	0	1	$F = 1$: No desplaza R_2
1	1	0	$F = 1$: No desplaza R_3
1	1	1	$F = 1$: Se desplazan todos

- Si el bit de la celda del registro coincide con el resultado de F , dicho registro estará en movimiento y se desplazará, en caso contrario no desplazará
- Esta función mayoría entre las celdas C_1 , C_2 y C_3 , permite que al menos dos de los tres registros se desplacen en cada paso
- El periodo T será igual al mcm $(T_1, T_2, T_3) = \text{mcm}(2^{n_1}-1, 2^{n_2}-1, 2^{n_3}-1)$
- Como $\text{mcd}(n_1, n_2, n_3) = 1$, entonces $T = (2^{n_1}-1) * (2^{n_2}-1) * (2^{n_3}-1)$
- $T = (2^{19}-1)(2^{22}-1)(2^{23}-1) = 524.287 * 4.194.303 * 8.388.607 \approx 10^{19} < 2^{64}$

Ejemplo de A5 con software FlujoLab



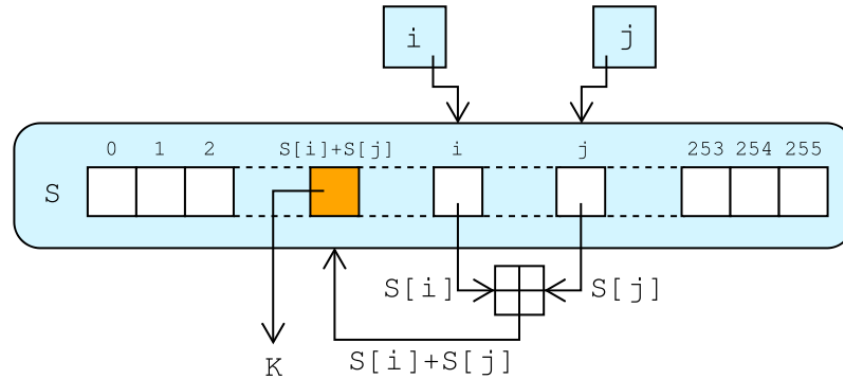
Algoritmo RC4

- Sistema de cifrado en flujo diseñado por Ron Rivest en 1987, cuyo algoritmo era secreto pero en 1994 aparece una descripción pública de su funcionamiento
- Al ser una marca registrada, las implementaciones no oficiales de RC4 se conocen con otros nombres como ARC4, ARCFOUR o Alleged-RC4
- Hasta 2015 era uno de los algoritmos en flujo más utilizados, por ejemplo, en las comunicaciones web basadas en el protocolo SSL/TLS
- Fue utilizado también ampliamente en el pasado en redes Wi-Fi inalámbricas basadas en WEP y WPA/TKIP para el cifrado de esas comunicaciones
- El uso extendido de RC4 en el pasado se debía a su alta velocidad y a su simplicidad
- RC4 es unas 10 veces más rápido que DES, el estándar de cifra simétrica en bloque hasta 1997 y unas 4 veces más rápido que AES, el nuevo estándar desde 2001
- En 2015 se prohíbe su uso en TLS al detectarse fallos relevantes (IETF – RFC 7465), al igual que en redes Wi-Fi WPA/TKIP (Wi-Fi Alliance)

Características de RC4

- Utiliza una clave de 1 a 256 bytes (de 8 a 2.048 bits)
- Es un registro o vector de estado S con 256 bytes $S_0, S_1, S_3, \dots S_{254}, S_{255}$
- En todo momento S contiene una permutación de todos los posibles valores de bytes entre 0 (00000000) y 255 (11111111), que cambia dinámicamente
- La implementación se basa en dos algoritmos
 1. El algoritmo KSA, Key Scheduling Algorithm, que convierte una semilla o clave en una permutación inicial del vector S
 2. El algoritmo PRGA, Pseudo-Random Generation Algorithm, que genera la secuencia cifrante pseudoaleatoria K_S
 3. El cifrado de cada byte del texto en claro con cada byte de la secuencia cifrante se hace mediante la función XOR, es decir, $C = M \oplus K_S$

Rutinas KSA y PRGA en RC4



Se recomienda ver la píldora formativa Thoth 35 ¿Cómo funciona el algoritmo RC4?

```
for i from 0 to 255
  S[i] := i
endfor
j := 0
for i from 0 to 255
  j := (j + S[i] + key[i mod keylength]) mod 256
  swap values of S[i] and S[j]
endfor
```

KSA

Key Scheduling Algorithm

Inicialización del array S

$1 < \text{keylength} < 256$ bytes

$5 \leq \text{Típico} \leq 16$ bytes (40 - 128 bits)

```
i := 0
j := 0
while GeneratingOutput:
  i := (i + 1) mod 256
  j := (j + S[i]) mod 256
  swap values of S[i] and S[j]
  K := S[(S[i] + S[j]) mod 256]
  output K
endwhile
```

PRGA

Pseudo-Random Generation Algorithm

Mientras sea necesario, modifica el estado y entrega el byte de clave K

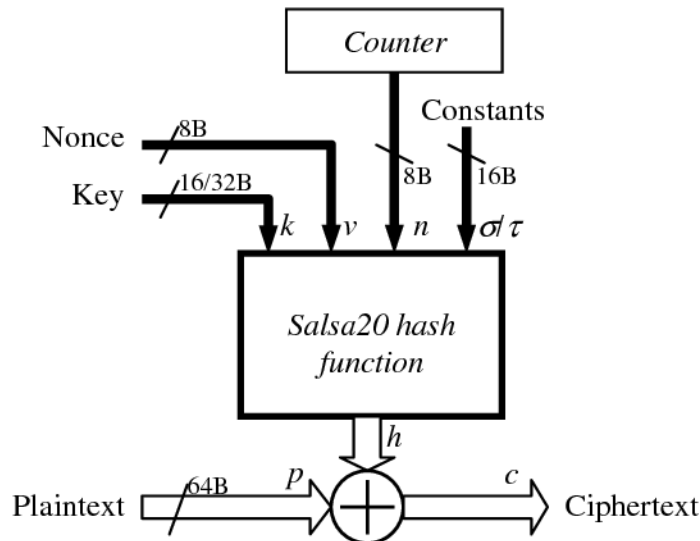
El byte de salida K se obtiene sumando $S[i] + S[j] \bmod 256$

Algoritmos Salsa20 y ChaCha20

- Salsa20 fue diseñado por Daniel J. Bernstein en 2005 y remitido a eSTREAM, ECRYPT Stream Cipher Project (Europa 2004-2008)
- Cada celda: palabra de 32 bits
- ChaCha20 es una variante de Salsa20, diseñado por el mismo autor en 2008
- De gran actualidad, al ser la competencia al estándar AES

Initial state of Salsa20

Cons	Key	Key	Key
Key	Cons	Nonce	Nonce
Pos	Pos	Cons	Key
Key	Key	Key	Cons

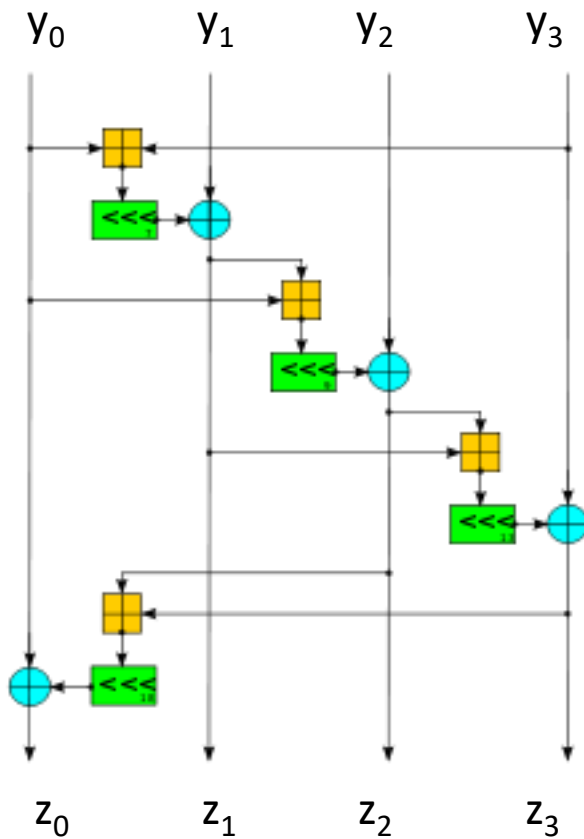


Initial state of ChaCha

Cons	Cons	Cons	Cons
Key	Key	Key	Key
Key	Key	Key	Key
Pos	Pos	Nonce	Nonce

Función Quarter-Round (QR)

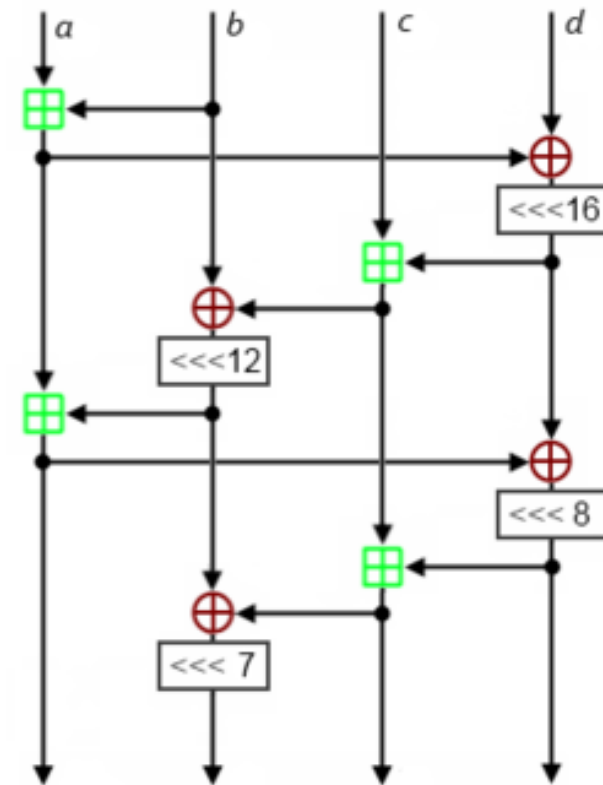
- Salsa20



- Operaciones ARX
- Add-Rotate-XOR
- Cuatro palabras de 32 bits cada una
- QR (y_0, y_1, y_2, y_3) = z_0, z_1, z_2, z_3

$$\begin{aligned}
 z_1 &= y_1 \oplus ((y_0 + y_3) \lll 7), \\
 z_2 &= y_2 \oplus ((z_1 + y_0) \lll 9), \\
 z_3 &= y_3 \oplus ((z_2 + z_1) \lll 13), \\
 z_0 &= y_0 \oplus ((z_3 + z_2) \lll 18).
 \end{aligned}$$

- ChaCha20



ChaCha20 versus AES

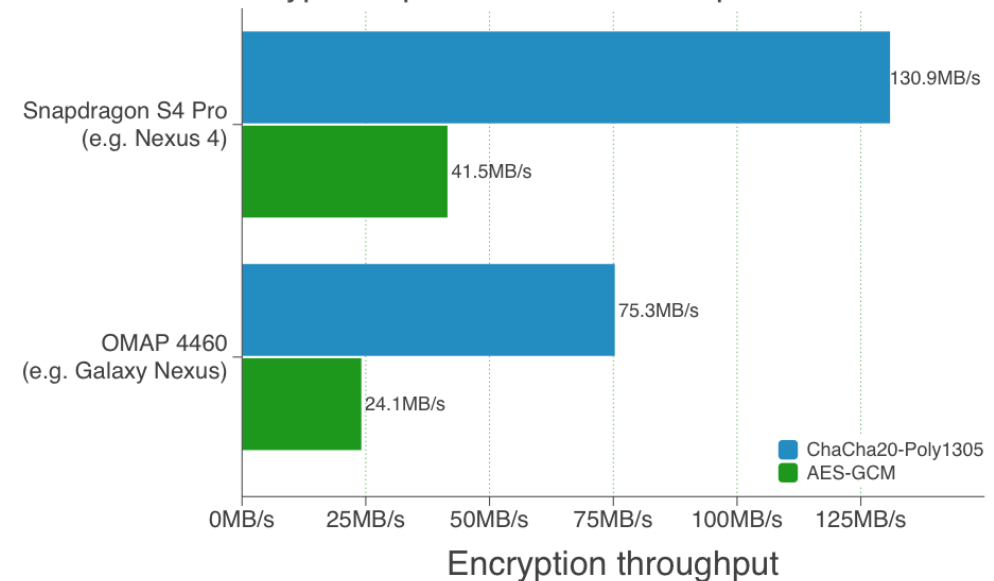
Given Cipher Suites

The cipher suites your client said it supports, in the order it sent them, are:

- TLS_AES_128_GCM_SHA256
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

<https://www.howssmyssl.com/>

Encryption speed for some widespread mobile CPUs

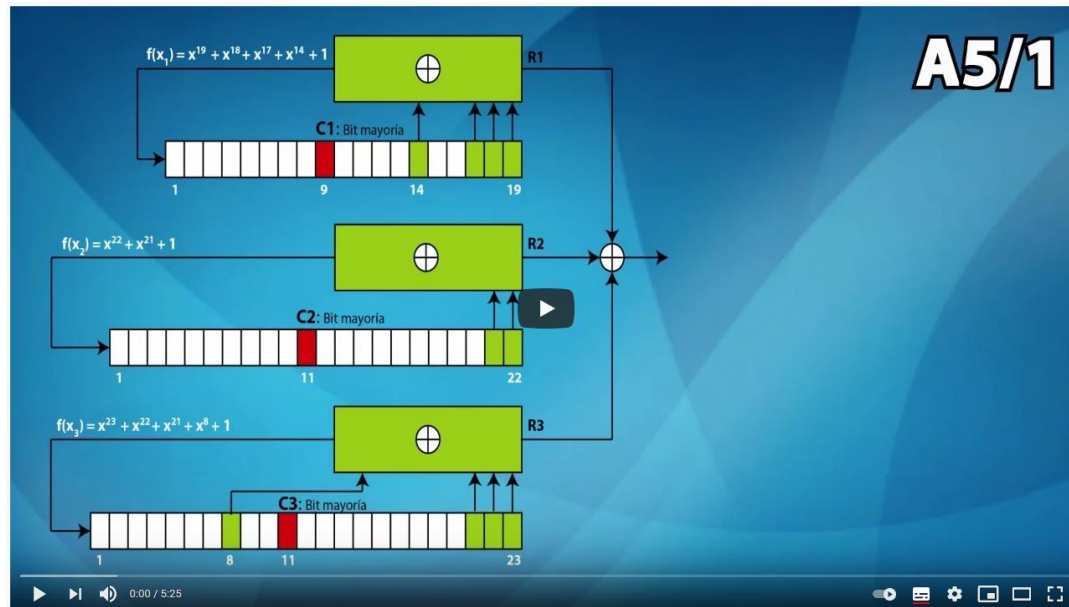


<https://blog.segu-info.com.ar/2014/04/protocolos-chacha20-y-poly1305-para.html>

Comparativa algoritmos de cifra en flujo

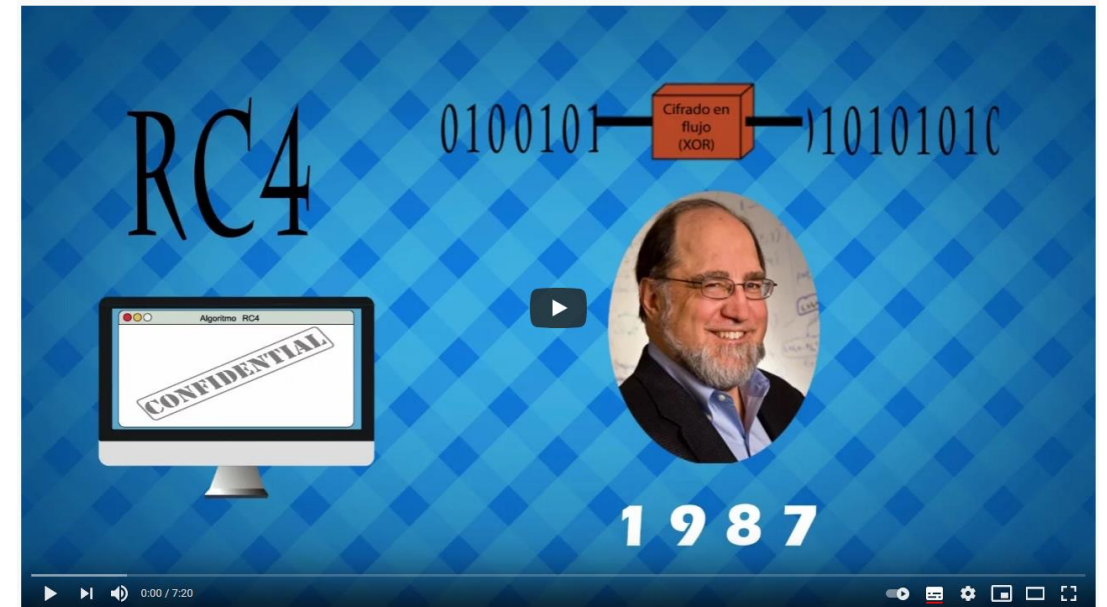
Cifrador	Bits estado	Periodo máximo	Tipo: HW/SW	Clave (bits)	Nonce	Notas
LFSR	n	$2^n - 1$	HW	-	-	Ataque B-M: $2 * n$
NLFSR	n	2^n	HW	-	-	-
A5/1	64 bits	$2^{64} - 1$	HW (LFSR)	64	22 bits	$64 = 19 + 22 + 23$ Diversos ataques
RC4	256 bytes	Elevado	SW	1-256 bytes	No	IETF recomienda no usarlo en 2015
Salsa20	512 bytes	2^{73}	SW	128/256 bytes	64 bits	eSTREAM del 2004 al 2008 (Europa)
ChaCha20	512 bytes	2^{73}	SW	128/256 bytes	64 bits	Actual competidor del AES

Más información en píldoras Thoth



Píldora formativa 34: ¿Cómo ciframos en flujo con A5, RC4 y en modo CTR?

<https://www.youtube.com/watch?v=WZfttmgqqeM>



Píldora formativa 35: ¿Cómo funciona el algoritmo RC4?

<https://www.youtube.com/watch?v=G3HajuqYH2U>

Conclusiones de la Lección 9.4

- Los registros lineales LFSR generan una m-secuencia y, por ello mismo, su complejidad es muy baja
- Mediante el ataque de Berlekamp-Massey, con solo 2^n bits consecutivos de una secuencia es posible recuperar las conexiones del LFSR que la ha generado, aunque la cifra no se ha roto ya que desconocemos la semilla
- A5/1 usado en la telefonía móvil GSM de la década de los 90, sucumbe en varios ataques, entre ellos Alex Biryukov, Adi Shamir y David Wagner en 2000
- RC4 de Ron Rivest fue un algoritmo muy popular durante más de una década en SSL/TLS y redes inalámbricas, hasta que en 2015 la IETF no lo recomienda
- El algoritmo ChaCha20 comienza a ser considerado como el más adecuado en una suite de cifrado, dada su velocidad mayor que AES y su buena seguridad

Lectura recomendada (1/2)

- Analysis of Geffe Generator LFSR properties on the application of algebraic attack, Fitri Handayani, N. P. R. Adiati, 2019
 - https://www.researchgate.net/publication/337024630_Analysis_of_Geffe_Generator_LFSR_properties_on_the_application_of_algebraic_attack
- Guion píldora formativa Thoth nº 34, ¿Cómo ciframos en flujo con A5, RC4 y en modo CTR?, Jorge Ramió, 2016
 - <https://www.criptored.es/thoth/material/texto/pildora034.pdf>
- A5/1 Security, Wikipedia
 - <https://en.wikipedia.org/wiki/A5/1#Security>
- Guion píldora formativa Thoth nº 35, ¿Cómo funciona el algoritmo RC4?, Jorge Ramió, 2016
 - <https://www.criptored.es/thoth/material/texto/pildora035.pdf>

Lectura recomendada (2/2)

- RC4 NOMORE: <https://www.rc4nomore.com/>
- eSTREAM: the ECRYPT Stream Cipher Project (Wikipedeia)
 - <https://en.wikipedia.org/wiki/ESTREAM>
- Salsa20, Wikipedia
 - <https://en.wikipedia.org/wiki/Salsa20>
- ChaCha20 and Poly1305 for IETF Protocols
 - <https://datatracker.ietf.org/doc/html/rfc8439>
- Protocolos ChaCha20 y Poly1305 para reforzar conexiones HTTPS
 - <https://blog.segu-info.com.ar/2014/04/protocolos-chacha20-y-poly1305-para.html>
- ChaCha, a variant of Salsa20, Daniel J. Bernstein, 2008
 - <http://cr.yp.to/chacha/chacha-20080128.pdf>