3. - Cifrado en flujo con clave secreta

3.1 Características generales
3.2 Generadores pseudoaleatorios
3.3 Registros de desplazamiento realimentados
3.4 Algoritmo RC4
3.5 Algoritmo A5

Criptografía simétrica

- La criptografía de clave secreta o simétrica se refiere al conjunto de métodos que permiten tener comunicación segura entre las partes siempre y cuando anteriormente se hayan intercambiado la clave correspondiente.

- Ha sido la **más usada en toda la historia** y ha sido implementada en diferentes dispositivos, manuales, mecánicos, eléctricos, hasta los algoritmos actuales que son programables en cualquier ordenador.

- Todos los sistemas criptográficos **clásicos** se basan en criptografía **simétrica**.

Criptografía simétrica

- Generalmente el **algoritmo** de cifrado es **conocido** por lo que la **fortaleza** del mismo dependerá de su **complejidad** interna y sobre todo de la **longitud de la clave** empleada.

- Para que un algoritmo de este tipo sea considerado **fiable** debe cumplir varios requisitos básicos:
  - **conocido el criptograma** (texto cifrado) **no** se pueden obtener de él ni el **texto** en claro ni la **clave**.
  - **conocidos el texto en claro y el texto cifrado** debe resultar **más caro en tiempo o dinero descifrar la clave** que el **valor** posible de la **información** obtenida por terceros.

Criptografía simétrica

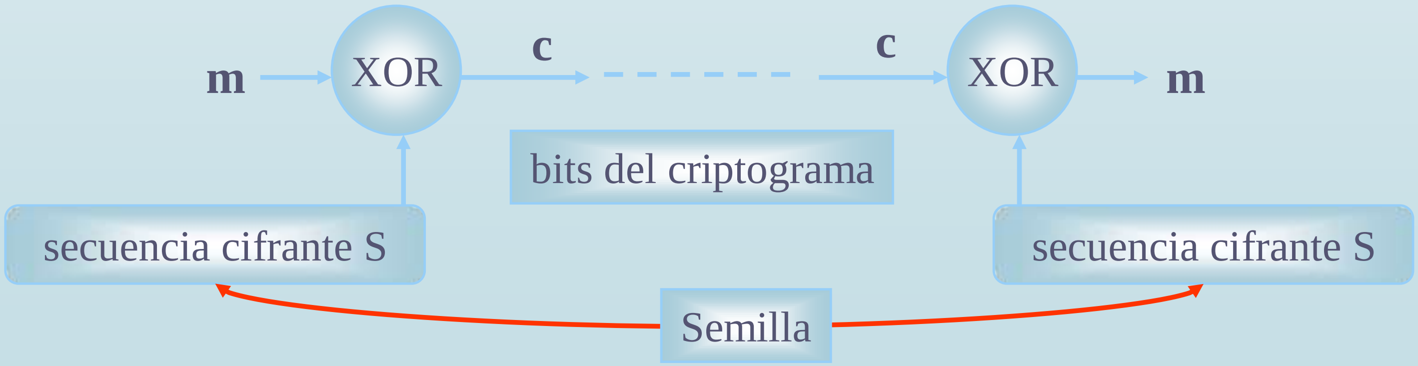
- El principal problema para este sistema de cifrado consiste en que para cada par de usuarios que quieran establecer comunicación se requiere una clave diferente, es decir, que un usuario de una red debe almacenar tantas claves como personas con las que quiera mantener una comunicación segura.

- Al principio (cuando las redes contaban con pocos usuarios) este hecho no constituía ningún problema, pero actualmente, con la cantidad de usuarios que existen en las redes se convierte en impracticable.

- Otro problema que presentan es el hecho de la distribución de claves y el peligro de que muchas personas deban conocer una misma clave.

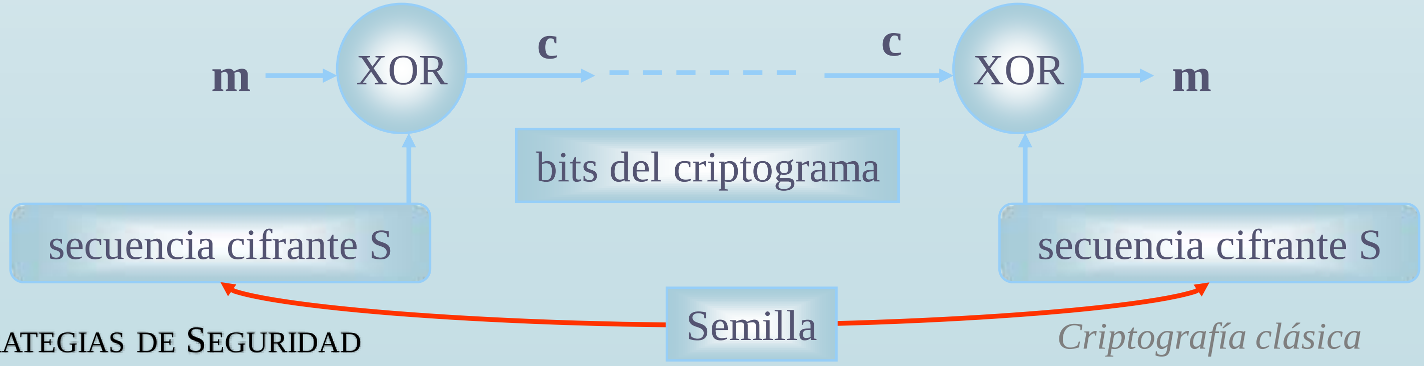
3.1 Características generales

- Como ya se ha visto, el cifrado Vernam verifica las condiciones de secreto perfecto definidas por Shannon.
- Es, en estos momentos, el **único procedimiento de cifrado incondicionalmente seguro**.
- Sin embargo presenta el inconveniente evidente de que requiere un bit de clave por cada bit de texto claro.
  - Puesto que el hacer llegar tal cantidad de clave a emisor y receptor por un canal seguro desbordaría la propia capacidad del canal, **en la práctica** se utiliza el método de cifrado en flujo, cuyo esquema fundamental es:



3.1 Características generales

- El emisor A, con **una clave corta (secreta)** y un **algoritmo determinista (público)**, genera una secuencia binaria S cuyos elementos se suman módulo 2 con los correspondientes bits de texto claro m, dando lugar a los bits de texto cifrado c.
    - Esta secuencia c es la que se envía a través de un canal público.
    - En recepción, B, con **la misma clave y el mismo algoritmo determinístico**, genera la misma secuencia cifrante S, que se suma módulo 2 con la secuencia cifrada c, dando lugar a los bits de texto claro m.
- Obsérvese que el cifrado en flujo es asimismo una **involución**, pues el procedimiento de cifrado y descifrado es idéntico.



3.1 Características generales

- Como la **secuencia cifrante** se ha obtenido a partir de un **algoritmo determinístico**, el cifrado en flujo ya no considera secuencias perfectamente aleatorias, **sino** solamente **pseudoaleatorias**.

- Sin embargo, lo que se **pierde** en cuanto a **seguridad**, por no verificarse en rigor las condiciones de Shannon, se **gana** en **viabilidad práctica** a la hora de utilizar este procedimiento de cifrado ya que **la única información que han de compartir emisor y receptor es la clave secreta**, cuya longitud oscila entre **128-256 bits**.

- En la actualidad, los bits de clave se suelen hacer llegar a ambos destinatarios mediante un procedimiento de clave pública; una vez que ambos disponen ya de la clave, se procede a aplicar el esquema tradicional del cifrado en flujo.

3.1 Características generales

Requerimientos de una secuencia cifrante

- Es difícil evaluar cuándo una secuencia binaria es suficientemente segura para su utilización en criptografía, ya que no existe un criterio general y unificado que lo certifique.

- Sin embargo sí se pueden señalar una serie de **requerimientos generales que toda secuencia cifrante ha de satisfacer** para su correcta aplicación al cifrado en flujo, entre ellos podemos señalar

3.1 Características generales

REQUERIMIENTOS DE UNA SECUENCIA CIFRANTE

Período

- El período de la secuencia cifrante ha de ser al menos tan largo como la longitud del texto a cifrar.
- En la práctica, se generan **secuencias con período del orden de 10^{38} (2^{128})** o superiores que cumplen sobradamente este requisito criptográfico.

Distribución de ceros y unos

- En una secuencia aleatoria, diferentes muestras de una determinada longitud han de estar **uniformemente distribuidas a lo 1argo de ella**.

3.1 Características generales

REQUERIMIENTOS DE UNA SECUENCIA CIFRANTE

Imprevisibilidad

- La secuencia cifrante ha de ser **imprevisible**; es decir,
  - **dada una porción de secuencia** de cualquier longitud, un criptoanalista no podría **predecir el siguiente dígito** con una **probabilidad de acierto superior a 1/2.**

Facilidad de generación

- La secuencia tiene que ser **fácil de generar con medios electrónicos** para su aplicabilidad en el proceso real de cifrado/descifrado.
- En este epígrafe se incluyen una serie de aspectos técnicos:
  - velocidad de generación, coste, tamaño, número de circuitos utilizados, consumo, etcétera, que han de tenerse en cuenta a la hora de implementar el generador de secuencia cifrante.

3.2 Generadores pseudoaleatorios

- La seguridad de muchos sistemas criptográficos está basada en el uso de números aleatorios como
  - claves de sesión
  - números primos
  - valores de desafío
  - secuencias cifrantes

Estos valores han de ser lo **suficientemente impredecibles** *para que un atacante no sea capaz de averiguarlos mediante el uso de técnicas probabilísticas*

Se obtienen de secuencias aleatorias que pueden ser
- *realmente aleatorias o,*
- *simplemente, comportarse como tal.*

3.2 Generadores pseudoaleatorios

- Los generadores realmente aleatorios presentan ciertos inconvenientes para su uso criptográfico.

  - Un **atacante podría observar o manipular la fuente de aleatoriedad** de forma que le permitiera predecir la secuencia con cierta probabilidad.

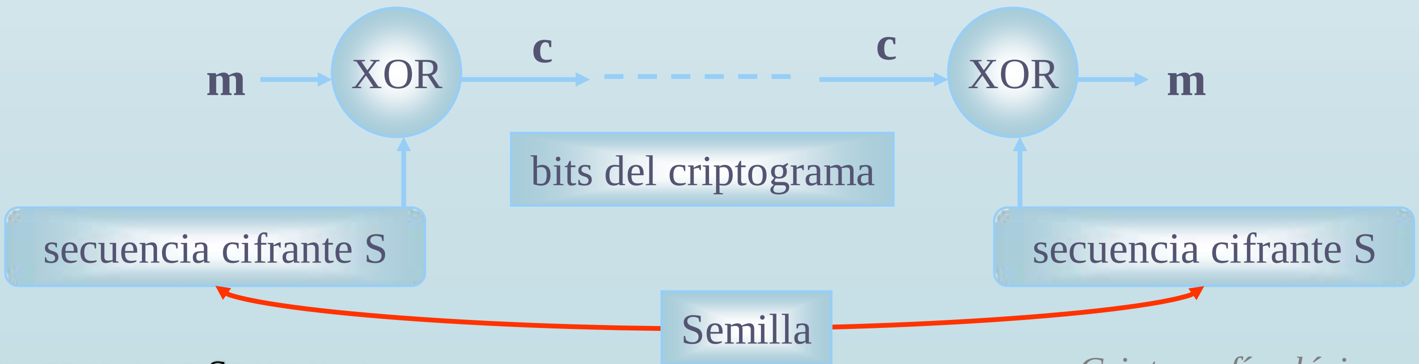
  - Se ha de **monitorizar** continuamente el **funcionamiento** de la fuente para evitar que **deje de ser suficientemente aleatoria**.

3.2 Generadores pseudoaleatorios

- Para determinadas aplicaciones criptográficas, resulta mucho más conveniente utilizar **generadores pseudoaleatorios**
  - basados en **algoritmos deterministas**
  - las **secuencias** generadas
    - son perfectamente **reproducibles** en función de la **entrada o semilla**
    - no son genuinamente aleatorias pero **se comportan como tal**

3.2 Generadores pseudoaleatorios

- En las aplicaciones de los generadores pseudoaleatorios a la seguridad, necesitamos producir secuencias
  - de grandes períodos
  - complejidades lineales altas y
  - una buena distribución estadística que las haga impredecibles.
- La secuencias generadas se pueden utilizar para cifrar la información haciendo uso del método Vernam
  - XOR bit a bit entre el texto en claro y la secuencia cifrante.



3.2 Generadores pseudoaleatorios

Veamos algunos de los métodos más simples y conocidos para la generación de secuencias pseudoaleatorias.

- Uno de los métodos más simples para la **generación de secuencias pseudoaleatorias es el basado en congruencias lineales** que no es criptográficamente seguro.
- Otro método muy simple es un **Registro de Desplazamiento Realimentado Linealmente** (*Linear Feedback Shift Register*, **LFSR**).
  - Por sí solo es inseguro pero combinado apropiadamente puede ser muy seguro
    - A5 – inseguro
    - SNOW - seguro

¿ A5, SNOW, LFSR?

3.2 Generadores pseudoaleatorios

ALGORITMOS DE CIFRADO EN FLUJO

- Entre los sistemas de cifrado en flujo más conocidos y utilizados destacamos:
  - **RC4**: Algoritmo de RSA, Rivest Cipher #4, desarrollado en el año 1987.
  - **SPRITZ**: variante mejorada de RC4 desarrollada en 2014.

    http://people.csail.mit.edu/rivest/pubs/RS14.pdf
    http://crypto.2014.rump.cr.yp.to/3de41b60e32a494c8f0fc9c21c67063a.pdf
  - **Salsa20**: uno de los algoritmos ganadores de eSTREAM PROJECT

    https://en.wikipedia.org/wiki/Salsa20
  - **A5**: Algoritmo no publicado propuesto en 1994. Versiones A5/1 fuerte (Europa) y A5/2 débil (exportación). Utilizado para cifrar el enlace en la telefonía móvil GSM (Global Systems for Mobile communications) o telefonía 2G.
  - **SNOW 3G**, núcleo de la integridad y la confidencialidad de las comunicaciones 4G.

    https://rua.ua.es/dspace/bitstream/10045/40395/1/RECSI-2014_12.pdf

3.2 Generadores pseudoaleatorios

GENERADORES BASADOS EN CONGRUENCIAS LINEALES

- Este procedimiento de generación de números pseudoaleatorios está basado en relaciones de recurrencia del tipo

$$x_{i+1} = ax_i + b \mod n$$

donde ***(a,b,n)*** son los parámetros que caracterizan al generador y pueden utilizarse como **clave secreta**.

- Se toma un valor $\mathbf{x_0}$ como **semilla** para inicializar del proceso..

3.2 Generadores pseudoaleatorios

GENERADORES BASADOS EN CONGRUENCIAS LINEALES

- Si los **parámetros** han sido **elegidos** de forma **conveniente**, **los números resultantes** x_i **no** se **repetirán** hasta haber **cubierto íntegramente el intervalo** [0, *n*- 1].

- **Boyar** demostró que las secuencias obtenidas a partir de congruencias lineales **no son criptográficamente seguras**, pues dada una **porción suficientemente larga** de las mismas, se podían **deducir los parámetros n, a, b**.

3.2 Generadores pseudoaleatorios

Ejemplo de generador basado en congruencias lineales

Sean:
$a = 5$ $b = 1$
$n = 16$ $x_0 = 10$

$x_{i+1} = ax_i + b \bmod n = 5x_i + 1 \bmod 16$

FlujoLab

http://www.criptored.upm.es/software/sw_m001m.htm

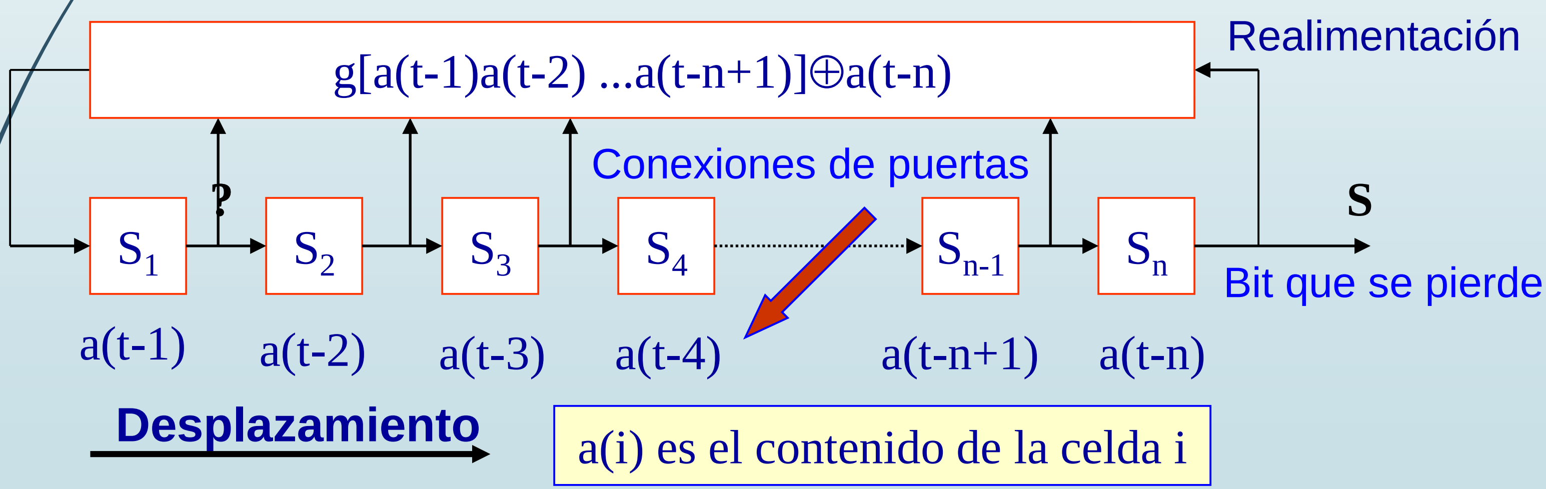
S = 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10….

$x_1 = 5 \cdot 10 + 1 \bmod 16 = 3$	$x_2 = 5 \cdot 3 + 1 \bmod 16 = 0$
$x_3 = 5 \cdot 0 + 1 \bmod 16 = 1$	$x_4 = 5 \cdot 1 + 1 \bmod 16 = 6$
$x_5 = 5 \cdot 6 + 1 \bmod 16 = 15$	$x_6 = 5 \cdot 15 + 1 \bmod 16 = 12$
$x_7 = 5 \cdot 12 + 1 \bmod 16 = 13$	$x_8 = 5 \cdot 13 + 1 \bmod 16 = 2$
$x_9 = 5 \cdot 2 + 1 \bmod 16 = 11$	$x_{10} = 5 \cdot 11 + 1 \bmod 16 = 8$
$x_{11} = 5 \cdot 8 + 1 \bmod 16 = 9$	$x_{12} = 5 \cdot 9 + 1 \bmod 16 = 14$
$x_{13} = 5 \cdot 14 + 1 \bmod 16 = 7$	$x_{14} = 5 \cdot 7 + 1 \bmod 16 = 4$
$x_{15} = 5 \cdot 4 + 1 \bmod 16 = 5$	$x_{16} = 5 \cdot 5 + 1 \bmod 16 = 10$

3.3 Registros de desplazamiento realimentados

- Un registro de desplazamiento realimentado es un circuito digital secuencial constituido por n celdas (etapas o *flip-flops*) que almacenan un 1 o un 0 y una función de realimentación g que permite expresar cada nuevo elemento de la secuencia a(t), con t>n, en función de los *n* elementos anteriores
a(t - n), a(t - n - 1), ... a(t -1).

- El contenido de las celdas se desplaza un lugar en el mismo sentido a cada impulso de reloj, de manera que el nuevo elemento de la secuencia *a(t)* se realimenta directamente a la primera celda.



Genera una secuencia con un período máximo 2^n

3.3 Registros de desplazamiento realimentados

- Se denomina estado del registro al contenido de las celdas entre dos impulsos; el estado inicial del registro corresponde al contenido de las celdas en el momento de comenzar el proceso.

- E1 diagrama de estados de un registro de desplazamiento (y consecuentemente la secuencia generada) es cíclico siempre que la función de realimentación sea no singular; es decir, de la forma

$$a(t)=g[a(t-1),a(t-2),\ldots,a(t-n-1)]\oplus a(t-n)$$

($\oplus$ representa la operación lógica XOR)

pues en caso contrario el nuevo elemento *a(t)* no tendría constancia de *a(t - n),* que es el dígito que se pierde en el siguiente desplazamiento.

3.3 Registros de desplazamiento realimentados

- El período de la secuencia producida dependerá del número de celdas del registro y de las características de la función *g*.
  - Lógicamente, el máximo período que puede alcanzar una secuencia de este tipo corresponde al máximo número de estados distintos, siendo éste 2^n para el caso de un registro de *n* celdas.

- La clave en este tipo de generadores está constituida por el contenido inicial del registro y/o el conocimiento de la función de realimentación.

3.3 Registros de desplazamiento realimentados

- Dependiendo de si la función g es o no lineal, así será, respectivamente, el registro de desplazamiento realimentado.

- **R**egistros de **D**esplazamiento **R**ealimentados **N**o **L**inealmente → **NLFSR**
  Non Linear **F**eedback **S**hift **R**egister
- **R**egistros de **D**esplazamiento **R**ealimentados **L**inealmente → **LFSR**
  Linear **F**eedback **S**hift **R**egister

3.3.1 Generadores no lineales NLFSR

REGISTROS DE DESPLAZAMIENTO REALIMENTADOS NO LINEALMENTE (NLFSR)

Non Linear Feedback Shift Register

- E1 problema que presenta este tipo de generador es que no hay un método sistemático para su análisis y manipulación, las secuencias generadas por estos registros pueden tener ciclos pequeños que se repiten indefinidamente a lo largo de todas ellas, lo que criptográficamente hablando es peligroso.

3.3.2 Generadores lineales LFSR

REGISTROS DE DESPLAZAMIENTO REALIMENTADOS LINEALMENTE (LFSR)

Linear Feedback Shift Register

- Se cuentan entre los dispositivos más importantes para la generación de secuencias pseudoaleatorias.

- Su función de realimentación g es lineal de la forma

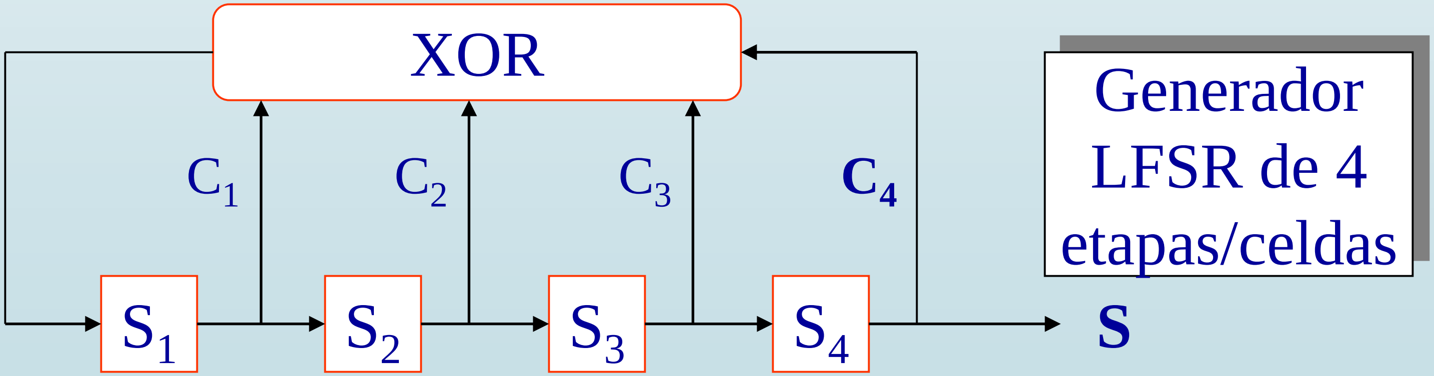
$$a(t) = c_1\ a(t\text{-}l) \oplus c_2\ a(t\text{-}2) \oplus c_n\ a(t\text{–}n)$$

con $c_i \in \{1,0\}$ y $c_n=1$.

3.3.2 Generadores lineales LFSR

- Su modelización es sencilla, al igual que su implementación electrónica.

- Obviamente, el estado inicial tiene que ser distinto del estado todo ceros, para evitar la secuencia idénticamente nula; el mayor número de estados diferentes será, pues, 2^n-1.



3.3.2 Generadores lineales LFSR

- Todo registro de desplazamiento realimentado linealmente tiene asociado un polinomio de realimentación de grado *n*

$$f(x) = 1+c_1\,x+c_2\,x^2+\ldots+c_{n-1}\,x^{n-1}+x^n$$

con coeficientes binarios 1 ó 0, respectivamente, según que la correspondiente etapa esté o no conectada a la puerta XOR

- Estudiando las características de este **polinomio**, se pueden determinar las características de la secuencia generada por el registro.
- De este modo se distinguen varios tipos de generadores

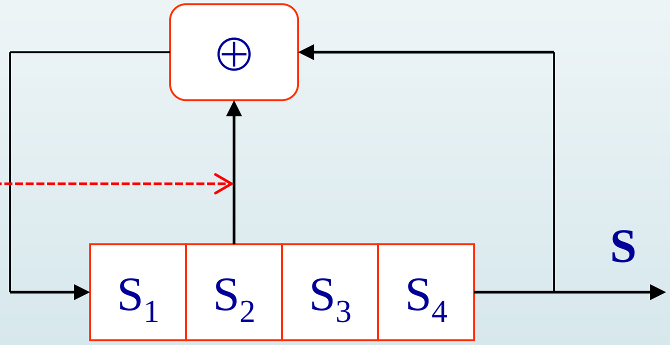
LFSR con polinomios factorizables
LFSR con polinomios irreducibles
LFSR con polinomios primitivos

3.3.2 Generadores lineales LFSR

LFSR con polinomio factorizable

- Dan lugar a secuencias caracterizadas por:
  - La longitud de la secuencia depende del estado inicial.
  - El máximo período T verifica $n<T<2^n-1$, pudiendo aparecer períodos secundarios que son divisores de T.

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



Es factorizable porque:

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

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

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

Reduciendo mod 2:

$$f(x_1)\bullet f(x_2) = x^4+x^2+1$$

Problema

T depende de la semilla

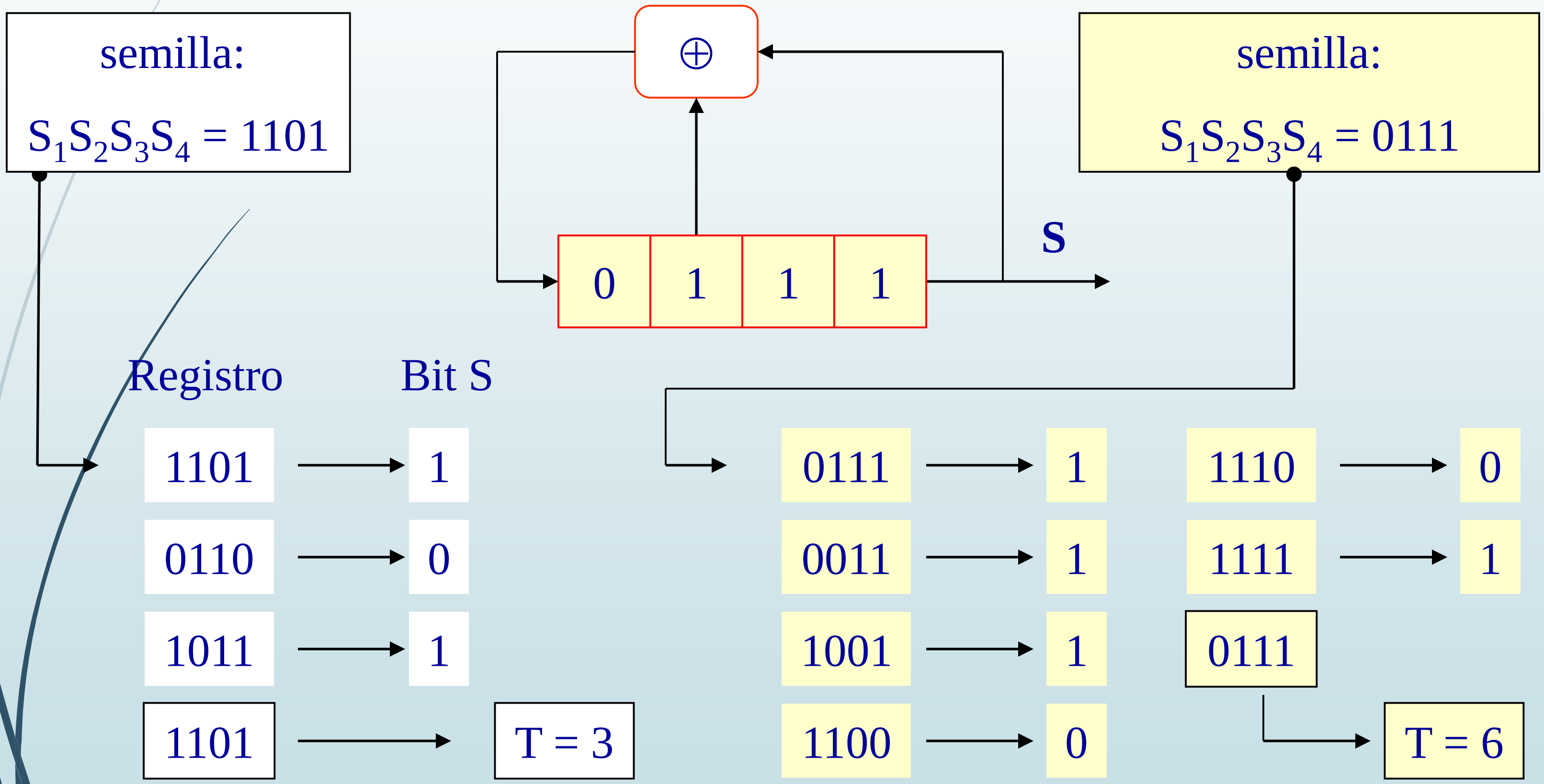
$n \le T \le 2^n - 1$

Períodos secundarios divisores

3.3.2 Generadores lineales LFSR

LFSR con polinomio factorizable

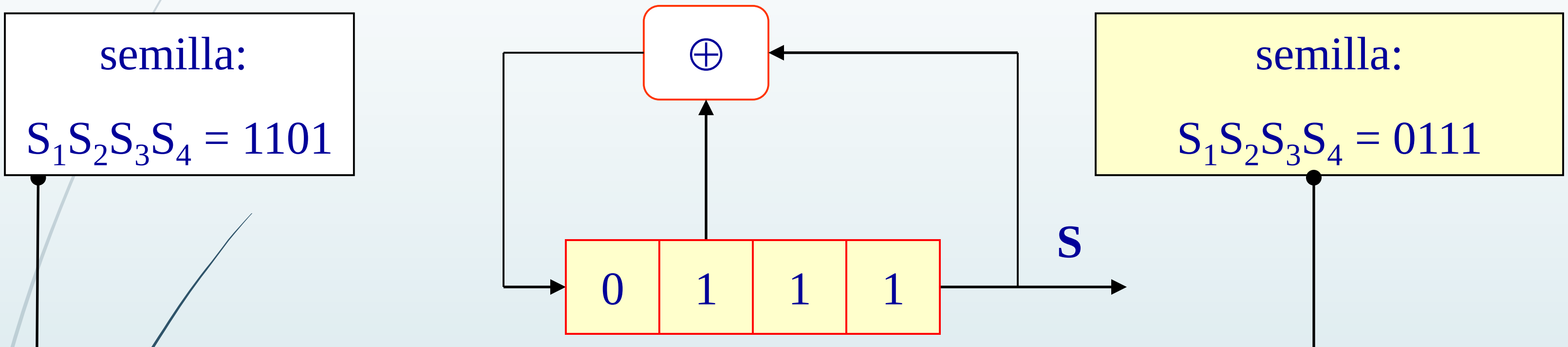
- Generador factorizable de cuatro celdas $f(x) = 1+x^2+x^4$



3.3.2 Generadores lineales LFSR

LFSR con polinomio factorizable

- Generador factorizable de cuatro celdas $f(x) = 1+x^2+x^4$



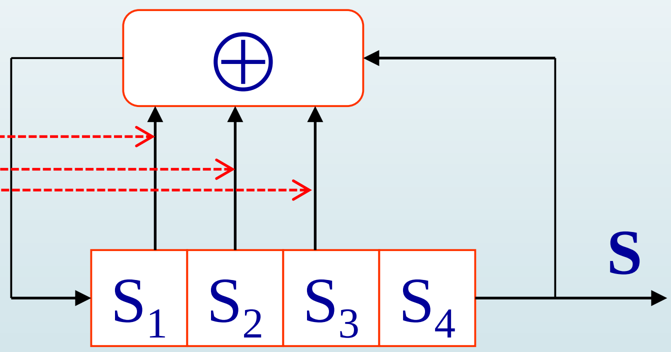
Dependiendo de la semilla, el período T de la secuencia toma valores distintos. En este caso, el período principal es $T_1 = 6$ y el período secundario $T_2 = 3$

3.3.2 Generadores lineales LFSR

LFSR con polinomio irreducible

- Dan lugar a secuencias caracterizadas por:
  - La longitud de la secuencia no depende del estado inicial.
  - El período T es un divisor de 2^n-1.

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



Es imposible factorizar en módulo 2 el polinomio f(x) como producto de dos polinomios $f(x_1)$ y $f(x_2)$ de grado menor

Problema

Ahora T ya no depende de la semilla pero será un divisor de $T_{máx} = 2^n - 1$

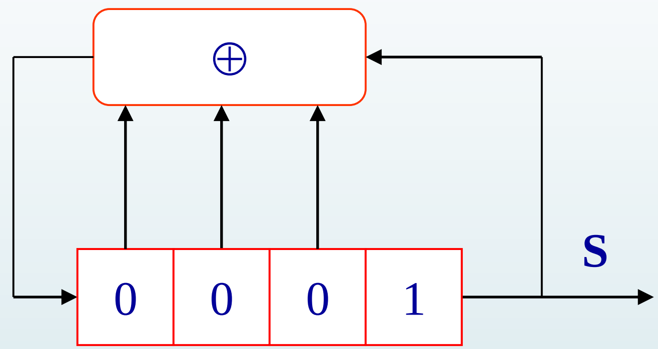
3.3.2 Generadores lineales LFSR

LFSR con polinomio irreducible

- Generador irreducible de cuatro celdas $f(x) = 1 + x + x^2 + x^3 + x^4$

semilla:

$S_1S_2S_3S_4 = 0001$



Registro	Bit S
0001	1
1000	0
1100	0
0110	0
0011	1
0001	

$T = 5$

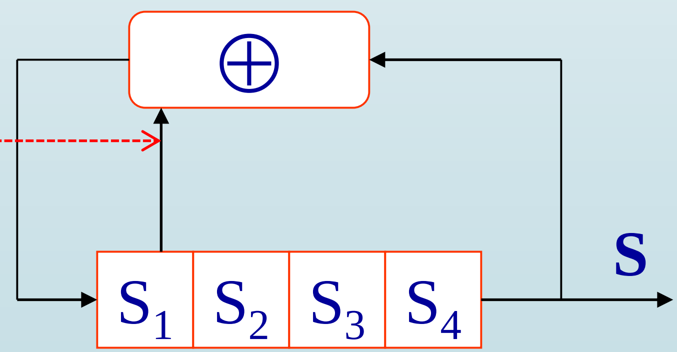
El período $T = 5$
es un divisor de
$2^n - 1 = 2^4 - 1 = 15$

3.3.2 Generadores lineales LFSR

LFSR CON POLINOMIO PRIMITIVO

- Dan lugar a secuencias caracterizadas por:
  - La longitud de la secuencia no depende del estado inicial.
  - El período T es 2^n-1
- Estos generadores son los que ofrecen una secuencia de período máximo 2^n-1; luego son los recomendados para su aplicación criptográfica.
- Existen algoritmos para la determinación de polinomios primitivos con coeficientes binarios:
  - GOLOMB, S. W., *Shift Register Sequences*

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



f(x) no es factorizable como $f(x_1) \cdot f(x_2)$ en módulo dos. Es además un generador del grupo.

3.3.2 Generadores lineales LFSR

LFSR CON POLINOMIO PRIMITIVO

- El número de polinomios primitivos de grado n obedece a la expresión

$$\frac{\Phi(2^n - 1)}{n}$$

donde $\Phi(x)$ es la función de Euler que denota el número de enteros positivos menores que x y primos con él.

- Este número crece exponencialmente con n, por ejemplo:
  - para n=11 existen 176 polinomios primitivos,
  - mientras que para n = 24 existen 276480.

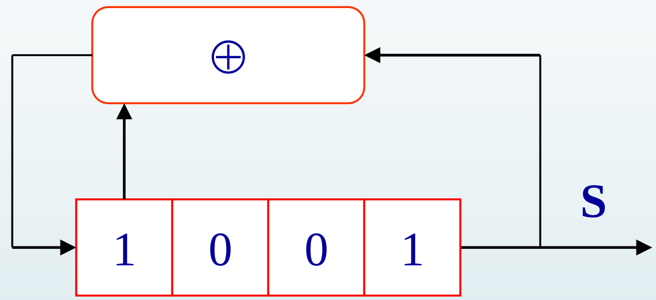
- Una secuencia generada por un registro de desplazamiento de polinomio primitivo se denomina secuencia de máxima longitud o, abreviadamente, ***m-secuencia***.

3.3.2 Generadores lineales LFSR

LFSR con polinomio primitivo

- Generador primitivo de cuatro celdas $f(x) = 1 + x + x^4$

$S_1S_2S_3S_4 = 1001$



Registro	Bit S	Registro	Bit S	Registro	Bit S
1001	1				
0100	0	1110	0	1010	0
0010	0	1111	1	1101	1
0001	1	0111	1	0110	0
1000	0	1011	1	0011	1
1100	0	0101	1	1001	T = 15

$T = 2^n - 1$
$T = 2^4 - 1$
$T = 15$

FlujoLab

3.4 Algoritmo RC4

- El Algoritmo RC4 fue diseñado por Ron Rivest en 1987 para la compañía RSA Data Security.

- Su nombre completo es Rivest Cipher 4, teniendo el acrónimo RC un significado alternativo al de Ron's Code utilizado para los algoritmos de cifrado RC2, RC5 y RC6

http://es.wikipedia.org/wiki/RC4

3.4 Algoritmo RC4

- Su implementación es extremadamente sencilla y rápida, y esta orientado a generar secuencias en unidades de un byte, además de permitir claves de diferentes longitudes.

- Se trata de un algoritmo patentado, lo cual implica que no puede ser incluido en aplicaciones de tipo comercial sin pagar los royalties correspondientes.

http://www.genbeta.com/actualidad/los-trolls-tambien-ganan-newegg-pierde-el-caso-por-una-patente-sobre-el-cifrado-ssl

3.4 Algoritmo RC4

- El código del algoritmo no se había publicado nunca oficialmente, pero en 1994 alguien difundió en los grupos de noticias de Internet (Cypherpunks, sci.crypt) una descripción que, como posteriormente se ha comprobado, genera las mismas secuencias.
- Reconocido por el propio Rivest en Spritz

  http://people.csail.mit.edu/rivest/pubs/RS14.pdf

- Actualmente la implementación no oficial de RC4 es legal, no puede ser utilizada con el nombre de RC4.
- Por este motivo, y con el fin de evitar problemas legales a raíz de la marca registrada, a menudo podemos verlo nombrado como
  - ARCFOUR,
  - ARC4 o
  - Alleged-RC4.

3.4 Algoritmo RC4

- Como se ha indicado, RC4 es un **algoritmo muy simple** que **genera una secuencia pseudoaleatoria** de bits que puede ser utilizada para el cifrado de la información mediante el **método de Vernam**.

- Consta de 2 subalgoritmos que utilizan una S-Caja almacenadora de una permutación del conjunto {0,1,…,255} ({0,1,…,2^8-1}),

  - el algoritmo de programación de clave (*Key Scheduling Algorithm-**KSA***), que se encarga de realizar la primera mezcla en la S-Caja a partir de la clave o semilla, y

  - el algoritmo de generación pseudoaleatoria (*Pseudo-Random Generation Algorithm-**PRGA***) que emite un byte de secuencia cifrante por cada iteración.

- Estos algoritmos se pueden describir mediante el siguiente pseudocódigo.

3.4 Algoritmo RC4

- ***Key Scheduling Algorithm (KSA)***

Para calcular los valores iniciales de la S-Caja, se hace lo siguiente:

1. $S(i) = i \quad \forall i \in \{0,1,\ldots,255\}$
2. Rellenar el array K(0) a K(255) repitiendo la clave tantas veces como sea necesario.
3. $j = 0$
4. Para i = 0 hasta 255 hacer:

   $j = [\, j + S(i) + K(i)\,] \bmod 256$

   Intercambiar S(i) y S(j) .

- ***Pseudo-Random Generation Algorithm (PRGA)***

Dos contadores i y j se ponen a cero.

En la iteración r, cada byte, O_r, de la secuencia cifrante se calcula como sigue:

1. $i = (i + 1) \bmod 256$
2. $j = [j + S(i)] \bmod 256$
3. Intercambiar los valores de S(i) y S(j)
4. $t = [S(i) + S(j)] \bmod 256$
5. $O_r = S(t)$ (2)

3. Mientras se necesite secuencia cifrante volver a 1

3.4 Algoritmo RC4

- El algoritmo RC4 genera secuencias en las que los ciclos son bastante grandes y es inmune a los criptoanálisis diferencial y lineal, si bien algunos estudios indican que puede poseer claves débiles, y que es sensible a estudios analíticos del contenido de la S-Caja.

- En julio de **2001** S. Fluhrer, I. Mantin y A. Shamir **publicaron un artículo** describiendo una **vulnerabilidad** en el algoritmo de cifrado RC4.

- Según ella, se puede recuperar la clave empleada si la inicialización del algoritmo cumple determinadas premisas, muy comunes, y se interceptan el suficiente número de mensajes.

- R. Rivest afirma que basta con aplicar el descarte de los 256 primeros bytes para evitar los ataques.

https://www.jcea.es/artic/rc4.htm

3.4 Algoritmo RC4

- A pesar de las dudas que existen en la actualidad sobre su seguridad, es un algoritmo ampliamente utilizado en muchas aplicaciones de tipo comercial como, por ejemplo, el protocolo WEP (Wired Equivalent Privacy) de WLAN (estándar IEEE 802.11b-g).

RC4 didáctico

El algoritmo se puede generalizar para obtener una secuencia cifrante de ***b*** bits por cada iteración del siguiente modo

3.4 Algoritmo RC4 - b bits por cada iteración

Key Scheduling Algorithm (KSA)

Para calcular los valores iniciales de la S-Caja, se hace lo siguiente:

1. $S(i) = i \quad \forall i \in \{0,1,\ldots,2^b-1\}$
2. Rellenar el array $K(0)$ a $K(2^b-1)$ repitiendo la clave tantas veces como sea necesario.
3. $j = 0$
4. Para $i = 0$ hasta 2^b-1 hacer:

   $j = [j + S(i) + K(j)] \bmod 2^b$

   Intercambiar $S(i)$ y $S(j)$.

Pseudo-Random Generation Algorithm (PRGA)

Dos contadores i y j se ponen a cero.

En la iteración r, cada bloque de b bits, O_r, de la secuencia cifrante se calcula como sigue:

1. $i = (i + 1) \bmod 2^b$
2. $j = [j + S(i)] \bmod 2^b$
3. Intercambiar los valores de $S(i)$ y $S(j)$
4. $t = [S(i) + S(j)] \bmod 2^b$
5. $O_r = S(t)$ (2)
3. Mientras se necesite secuencia cifrante volver a 1

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

RC4 didáctico

- En este caso trabajaremos en $Z_4= \{0,1,2,3\}$. Supongamos que la clave k=[2,1]
- ***Key Scheduling Algorithm (KSA)***

1. S = [S(0), S(1), S(2), S(3)] = [0,1,2,3]

2. K= [K(0), K(1), K(2), K(3)] = [2,1,2,1]

3. j=0

4. ***i=0*** (j=0, S = [0,1,2,3])

j = [j+S(i)+K(i)] mod 4 = [0+S(0)+K(0)] mod 4 = (0+0+2) mod 4 = 2

Intercambiar S(i) con S(j); S(0)↔S(2); S=[**2**,1,**0**,3]

4. ***i=1*** (j=2, S = [2,1,0,3])

j = [j+S(i)+K(i)] mod 4 = [2+S(1)+K(1)] mod 4 = (2+1+1) mod 4 = 0

Intercambiar S(i) con S(j); S(1)↔S(0); S=[**1**,**2**,0,3]

4. ***i=2*** (j=0, S = [1,2,0,3])

j = [j+S(i)+K(i)] mod 4 = [0+S(2)+K(2)] mod 4 = (0+0+2) mod 4 = 2

Intercambiar S(i) con S(j); S(2)↔S(2); S=[1,2,**0**,3]

4. ***i=3*** (j=2, S = [1,2,0,3])

j = [j+S(i)+K(i)] mod 4 = [0+S(3)+K(3)] mod 4 = (2+3+1) mod 4 = 2

Intercambiar S(i) con S(j); S(3)↔S(2); S=[1,2,**3**,**0**]

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

- ***Pseudo-Random Generation Algorithm (PRGA)***

S = [S(0), S(1), S(2), S(3)] = [1,2,3,0]

i=0, j=0

Iteración 1 (i=0, j=0, S=[1,2,3,0])

1. i = (i+1) mod 4 = (0+1) mod 4 = 1

2. j = [j+S(i)] mod 4 = [0+S(1)] mod 4 = (0+2) mod 4 = 2

3. Intercambiar S(i) con S(j); S(1)↔S(2); S=[1,**3**,**2**,0]

4. t = [S(i)+S(j)] mod 4 = [S(1)+S(2)] mod 4 = (3+2) mod 4 = 1

5. $O_1 = S(t)_{(2)} = S(1)_{(2)} = 3_{(2)} = \mathbf{11}$

Iteración 2 (i=1, j=2, S=[1,3,2,0])

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

2. j = [j+S(i)] mod 4 = [2+S(1)] mod 4 = (2+2) mod 4 = 0

3. Intercambiar S(i) con S(j); S(2)↔S(0); S=[**2**,3,**1**,0]

4. t = [S(i)+S(j)] mod 4 = [S(2)+S(0)] mod 4 = (2+1) mod 4 = 3

5. $O_2 = S(t)_{(2)} = S(3)_{(2)} = 0_{(2)} = \mathbf{00}$

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

- ***Pseudo-Random Generation Algorithm (PRGA)***

Iteración 3 (i=2, j=0, S=[2,3,1,0])

1. i = (i+1) mod 4 = (2+1) mod 4 = 3

2. j = [j+S(i)] mod 4 = [0+S(3)] mod 4 = (0+0) mod 4 = 0

3. Intercambiar S(i) con S(j); S(3)↔S(0); S=[**0**,3,1,**2**]

4. t = [S(i)+S(j)] mod 4 = [S(3)+S(0)] mod 4 = (2+0) mod 4 = 2

5. O_3= $S(t)_{(2)}$= $S(2)_{(2)}$= $1_{(2)}$= **01**

Iteración 4 (i=3, j=0, S=[0,3,1,2])

1. i = (i+1) mod 4 = (3+1) mod 4 = 0

2. j = [j+S(i)] mod 4 = [0+S(0)] mod 4 = (0+0) mod 4 = 0

3. Intercambiar S(i) con S(j); S(0)↔S(0); S=[**0**,3,1,2]

4. t = [S(i)+S(j)] mod 4 = [S(0)+S(0)] mod 4 = (0+0) mod 4 = 0

5. O_4 = $S(t)_{(2)}$= $S(0)_{(2)}$= $0_{(2)}$= **00**

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

- Si el texto en claro fuese m = C y consideramos su codificación en ASCII, tendríamos:

Texto en claro	$m = C = 67_{(2)}$	$= 0100\ 0011$
Secuencia cifrante	$k = O_1O_2O_3O_4$	$= 1100\ 0100$
Texto cifrado	$c = m \oplus k$	$= 1000\ 0111$

RC4 didáctico

Clave: 1,2,1,0

Texto en claro: 1,0,0,3 (01 00 00 11)

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

Key Scheduling Algorithm (KSA)

1. S=[S(0),S(1),S(2),S(3)]= [0,1,2,3]

   Semilla =[1,2,1,0]

2. K=[K(0),K(1),K(2),K(3)]= [1,2,1,0]

3. j= 0

4. i=0 (j=0, S=[0,1,2,3])

   j=[j+S(i)+K(i)] mod 4=[0+S(0)+K(0)] mod 4=(0+0+1) mod 4=1

   Intercambiar S(0) con S(1)

   S=[1,0,2,3]

4. i=1 (j=1, S=[1,0,2,3])

   j=[j+S(i)+K(i)] mod 4=[1+S(1)+K(1)] mod 4=(1+0+2) mod 4=3

   Intercambiar S(1) con S(3)

   S=[1,3,2,0]

4. i=2 (j=3, S=[1,3,2,0])

   j=[j+S(i)+K(i)] mod 4=[3+S(2)+K(2)] mod 4=(3+2+1) mod 4=2

   Intercambiar S(2) con S(2)

   S=[1,3,2,0]

4. i=3 (j=2, S=[1,3,2,0])

   j=[j+S(i)+K(i)] mod 4=[2+S(3)+K(3)] mod 4=(2+0+0) mod 4=2

   Intercambiar S(3) con S(2)

   S=[1,3,0,2]

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

Pseudo-Random Generation Algorithm (PRGA)

S=[S(0),S(1),S(2),S(3)]= [1,3,0,2]

i=0,j=0

Iteración 1 (i=0, j=0, S=[1,3,0,2])

1. i=(i+1) mod 4=(0+1) mod 4=1
2. j=[j+S(i)] mod 4=[0+S(1)] mod 4=(0+3) mod 4=3
3. Intercambiar S(1) con S(3) -> S=[1,2,0,3]
4. t=[S(i)+S(j)] mod 4=[S(1)]+S(3)] mod 4=(2+3) mod 4=1
5. O_1=S(t)=S(1)=2=$10_{(2)}$

Iteración 2 (i=1, j=3, S=[1,2,0,3])

1. i=(i+1) mod 4=(1+1) mod 4=2
2. j=[j+S(i)] mod 4=[3+S(2)] mod 4=(3+0) mod 4=3
3. Intercambiar S(2) con S(3) -> S=[1,2,3,0]
4. t=[S(i)+S(j)] mod 4=[S(2)]+S(3)] mod 4=(3+0) mod 4=3
5. O_2=S(t)=S(3)=0=$00_{(2)}$

3.4 Algoritmo RC4

Ejemplo b=2 bits de salida por iteración

Iteración 3 (i=2, j=3, S=[1,2,3,0])

1. i=(i+1) mod 4=(2+1) mod 4=3
2. j=[j+S(i)] mod 4=[3+S(3)] mod 4=(3+0) mod 4=3
3. Intercambiar S(3) con S(3) -> S=[1,2,3,0]
4. t=[S(i)+S(j)] mod 4=[S(3)]+S(3)] mod 4=(0+0) mod 4=0
5. O_3=S(t)=S(0)=1=$01_{(2)}$

Iteración 4 (i=3, j=3, S=[1,2,3,0])

1. i=(i+1) mod 4=(3+1) mod 4=0
2. j=[j+S(i)] mod 4=[3+S(0)] mod 4=(3+1) mod 4=0
3. Intercambiar S(0) con S(0) -> S=[1,2,3,0]
4. t=[S(i)+S(j)] mod 4=[S(0)]+S(0)] mod 4=(1+1) mod 4=2
5. O_4=S(t)=S(2)=3=$11_{(2)}$

Secuencia de salida 2,0,1,3 (binaria) ===> 10,00,01,11

3.4 Algoritmo RC4

Ejemplo b=3 bits de salida por iteración

Key Scheduling Algorithm (KSA)

1. S=[S(0),S(1),S(2),S(3),S(4),S(5),S(6),S(7)]= [0,1,2,3,4,5,6,7]

   Semilla =[1,2,1,0]

2. K=[K(0),K(1),K(2),K(3),K(4),K(5),K(6),K(7)]= [1,2,1,0,1,2,1,0]

3. j= 0

4. i=0 (j=0, S=[0,1,2,3,4,5,6,7])

   j=[j+S(i)+K(i)] mod 8=[0+S(0)+K(0)] mod 8=(0+0+1) mod 8=1

   Intercambiar S(0) con S(1)

   S=[1,0,2,3,4,5,6,7]

4. i=1 (j=1, S=[1,0,2,3,4,5,6,7])

   j=[j+S(i)+K(i)] mod 8=[1+S(1)+K(1)] mod 8=(1+0+2) mod 8=3

   Intercambiar S(1) con S(3)

   S=[1,3,2,0,4,5,6,7]

4. i=2 (j=3, S=[1,3,2,0,4,5,6,7])

   j=[j+S(i)+K(i)] mod 8=[3+S(2)+K(2)] mod 8=(3+2+1) mod 8=6

   Intercambiar S(2) con S(6)

   S=[1,3,6,0,4,5,2,7]

3.4 Algoritmo RC4

Ejemplo b=3 bits de salida por iteración

4. i=3 (j=6, S=[1,3,6,0,4,5,2,7])

   j=[j+S(i)+K(i)] mod 8=[6+S(3)+K(3)] mod 8=(6+0+0) mod 8=6

   Intercambiar S(3) con S(6)

   S=[1,3,6,2,4,5,0,7]

4. i=4 (j=6, S=[1,3,6,2,4,5,0,7])

   j=[j+S(i)+K(i)] mod 8=[6+S(4)+K(4)] mod 8=(6+4+1) mod 8=3

   Intercambiar S(4) con S(3)

   S=[1,3,6,4,2,5,0,7]

4. i=5 (j=3, S=[1,3,6,4,2,5,0,7])

   j=[j+S(i)+K(i)] mod 8=[3+S(5)+K(5)] mod 8=(3+5+2) mod 8=2

   Intercambiar S(5) con S(2)

   S=[1,3,5,4,2,6,0,7]

4. i=6 (j=2, S=[1,3,5,4,2,6,0,7])

   j=[j+S(i)+K(i)] mod 8=[2+S(6)+K(6)] mod 8=(2+0+1) mod 8=3

   Intercambiar S(6) con S(3)

   S=[1,3,5,0,2,6,4,7]

4. i=7 (j=3, S=[1,3,5,0,2,6,4,7])

   j=[j+S(i)+K(i)] mod 8=[3+S(7)+K(7)] mod 8=(3+7+0) mod 8=2

   Intercambiar S(7) con S(2)

   S=[1,3,7,0,2,6,4,5]

3.4 Algoritmo RC4

Ejemplo b=3 bits de salida por iteración

Pseudo-Random Generation Algorithm (PRGA)

S=[S(0),S(1),S(2),S(3),S(4),S(5),S(6),S(7)]= [1,3,7,0,2,6,4,5]

i=0,j=0

Iteración 1 (i=0, j=0, S=[1,3,7,0,2,6,4,5])

1. i=(i+1) mod 8=(0+1) mod 8=1
2. j=[j+S(i)] mod 8=[0+S(1)] mod 8=(0+3) mod 8=3
3. Intercambiar S(1) con S(3) -> S=[1,0,7,3,2,6,4,5]
4. t=[S(i)+S(j)] mod 8=[S(1)]+S(3)] mod 8=(0+3) mod 8=3
5. O_1=S(t)=S(3)=3=$011_{(2)}$

Iteración 2 (i=1, j=3, S=[1,0,7,3,2,6,4,5])

1. i=(i+1) mod 8=(1+1) mod 8=2
2. j=[j+S(i)] mod 8=[3+S(2)] mod 8=(3+7) mod 8=2
3. Intercambiar S(2) con S(2) -> S=[1,0,7,3,2,6,4,5]
4. t=[S(i)+S(j)] mod 8=[S(2)]+S(2)] mod 8=(7+7) mod 8=6
5. O_2=S(t)=S(6)=4=$100_{(2)}$

Iteración 3 (i=2, j=2, S=[1,0,7,3,2,6,4,5])

1. i=(i+1) mod 8=(2+1) mod 8=3
2. j=[j+S(i)] mod 8=[2+S(3)] mod 8=(2+3) mod 8=5
3. Intercambiar S(3) con S(5) -> S=[1,0,7,6,2,3,4,5]
4. t=[S(i)+S(j)] mod 8=[S(3)]+S(5)] mod 8=(6+3) mod 8=1
5. O_3=S(t)=S(1)=0=$000_{(2)}$

3.4 Algoritmo RC4

Ejemplo b=3 bits de salida por iteración

Iteración 4 (i=3, j=5, S=[1,0,7,6,2,3,4,5])

1. i=(i+1) mod 8=(3+1) mod 8=4
2. j=[j+S(i)] mod 8=[5+S(4)] mod 8=(5+2) mod 8=7
3. Intercambiar S(4) con S(7) -> S=[1,0,7,6,5,3,4,2]
4. t=[S(i)+S(j)] mod 8=[S(4)]+S(7)] mod 8=(5+2) mod 8=7
5. O_4=S(t)=S(7)=2=$010_{(2)}$

Iteración 5 (i=4, j=7, S=[1,0,7,6,5,3,4,2])

1. i=(i+1) mod 8=(4+1) mod 8=5
2. j=[j+S(i)] mod 8=[7+S(5)] mod 8=(7+3) mod 8=2
3. Intercambiar S(5) con S(2) -> S=[1,0,3,6,5,7,4,2]
4. t=[S(i)+S(j)] mod 8=[S(5)]+S(2)] mod 8=(7+3) mod 8=2
5. O_5=S(t)=S(2)=3=$011_{(2)}$

Iteración 6 (i=5, j=2, S=[1,0,3,6,5,7,4,2])

1. i=(i+1) mod 8=(5+1) mod 8=6
2. j=[j+S(i)] mod 8=[2+S(6)] mod 8=(2+4) mod 8=6
3. Intercambiar S(6) con S(6) -> S=[1,0,3,6,5,7,4,2]
4. t=[S(i)+S(j)] mod 8=[S(6)]+S(6)] mod 8=(4+4) mod 8=0
5. O_6=S(t)=S(0)=1=$001_{(2)}$

3.4 Algoritmo RC4

Ejemplo b=3 bits de salida por iteración

Iteración 7 (i=6, j=6, S=[1,0,3,6,5,7,4,2])

1. i=(i+1) mod 8=(6+1) mod 8=7
2. j=[j+S(i)] mod 8=[6+S(7)] mod 8=(6+2) mod 8=0
3. Intercambiar S(7) con S(0) -> S=[2,0,3,6,5,7,4,1]
4. t=[S(i)+S(j)] mod 8=[S(7)]+S(0)] mod 8=(1+2) mod 8=3
5. O_7=S(t)=S(3)=6=$110_{(2)}$

Iteración 8 (i=7, j=0, S=[2,0,3,6,5,7,4,1])

1. i=(i+1) mod 8=(7+1) mod 8=0
2. j=[j+S(i)] mod 8=[0+S(0)] mod 8=(0+2) mod 8=2
3. Intercambiar S(0) con S(2) -> S=[3,0,2,6,5,7,4,1]
4. t=[S(i)+S(j)] mod 8=[S(0)]+S(2)] mod 8=(3+2) mod 8=5
5. O_8=S(t)=S(5)=7=$111_{(2)}$

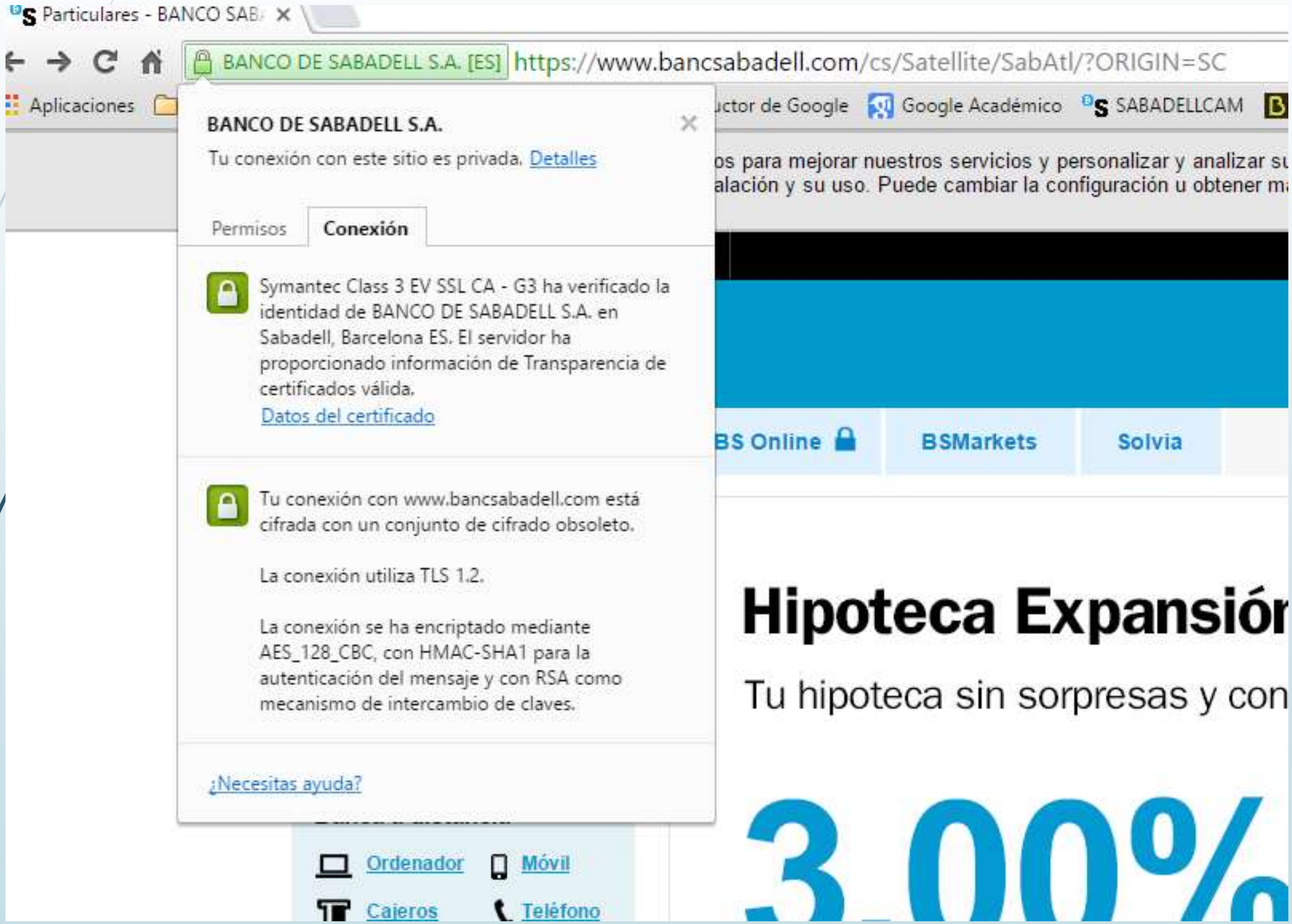
Secuencia de salida 3,4,0,2,3,1,6,7 (binario) ===> 011,100,000,010,011,001,110,111

3.4 Algoritmo RC4

Año 2015

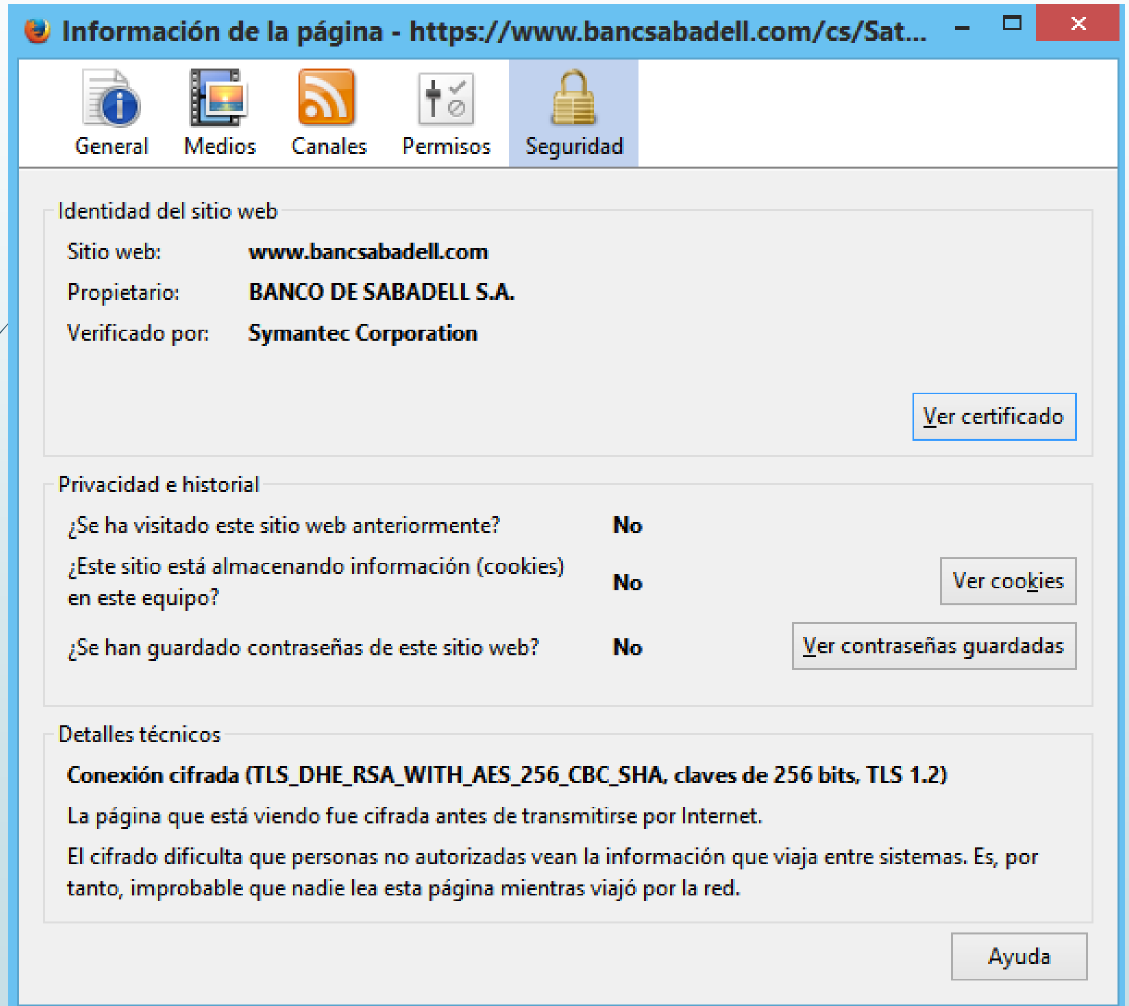
3.4 Algoritmo RC4

Año 2016



3.4 Algoritmo RC4

Años 2017, 2018



3.4 Algoritmo RC4

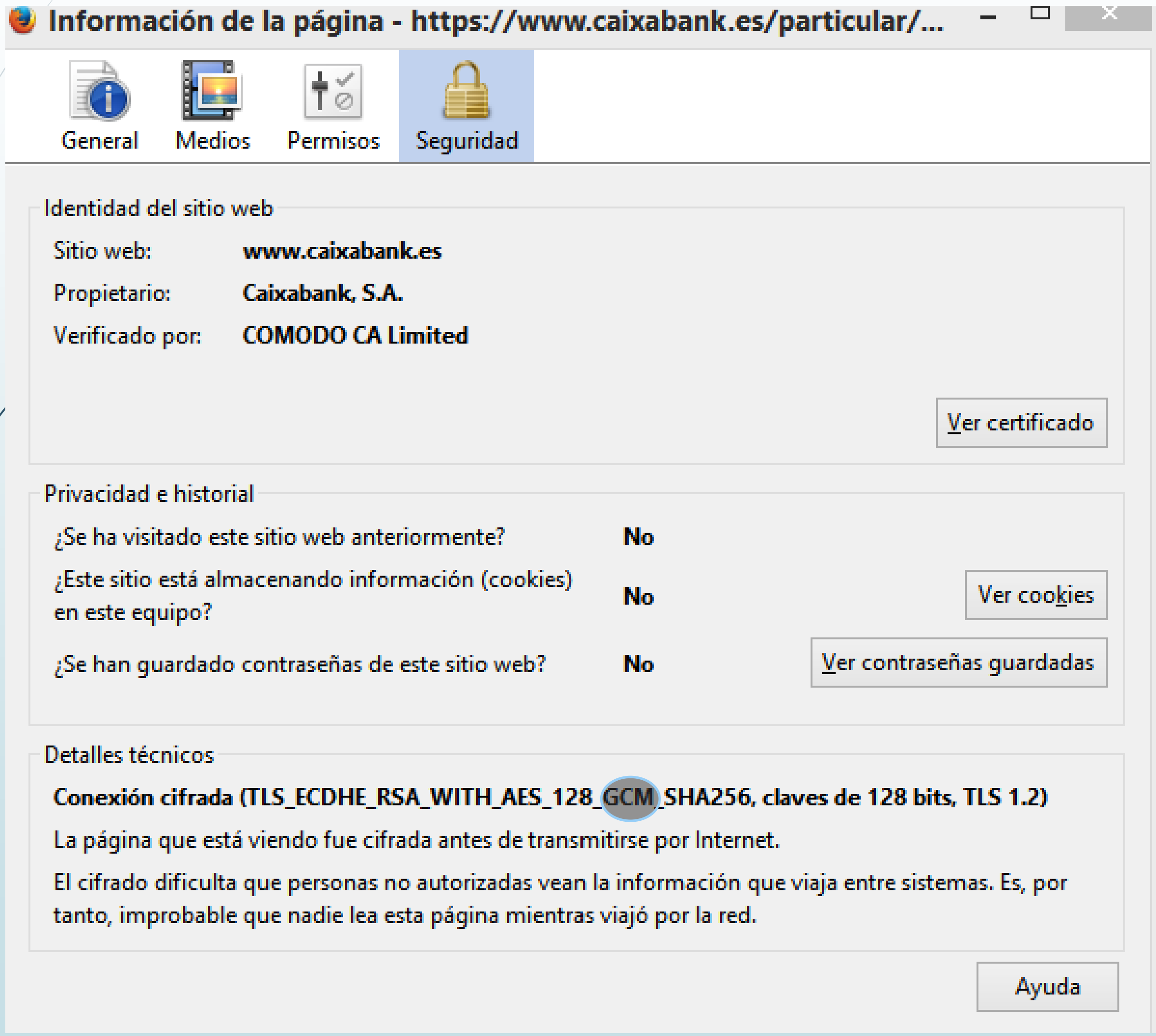
Año 2015

3.4 Algoritmo RC4

Año 2016

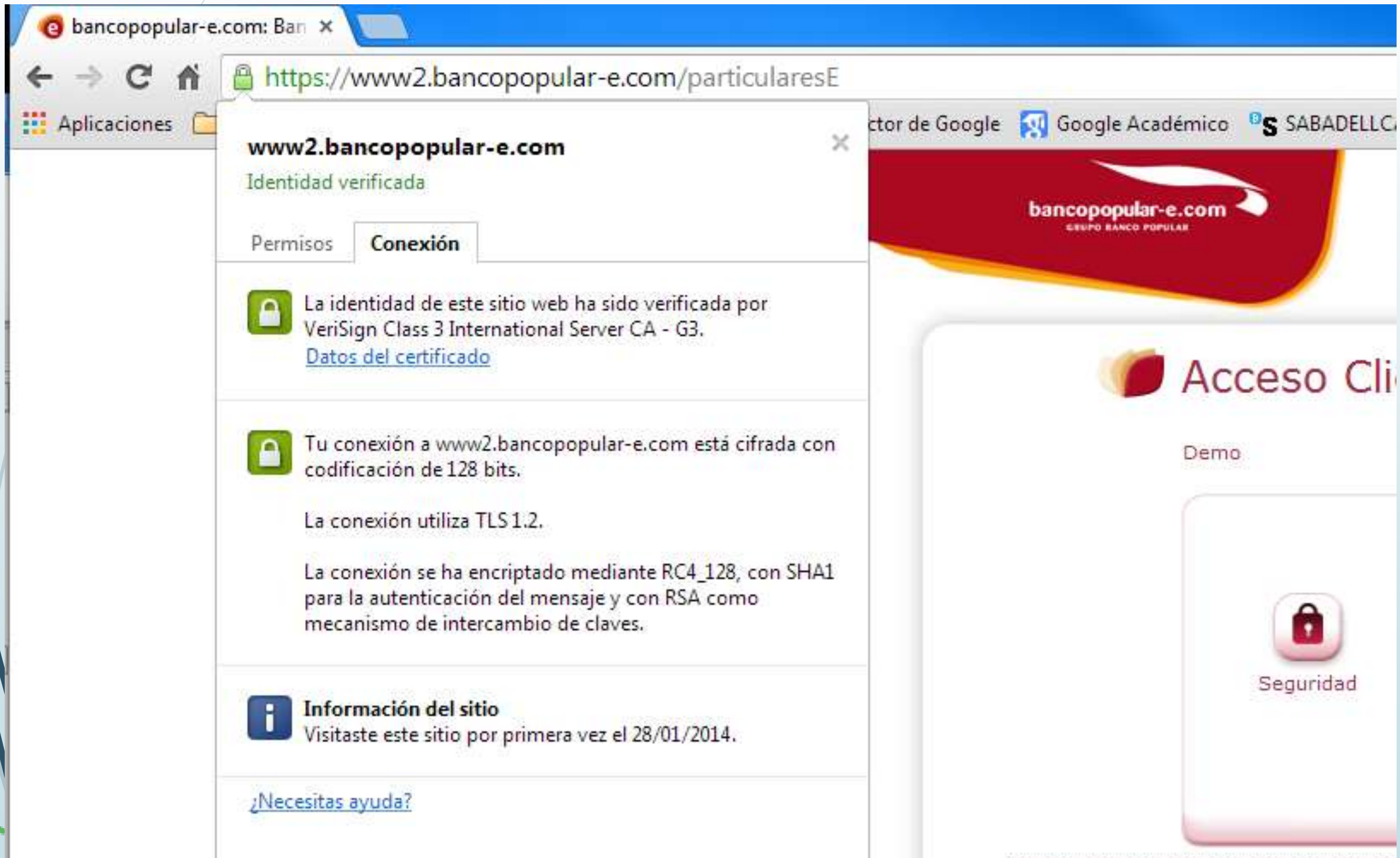
3.4 Algoritmo RC4

Años 2017, 2018



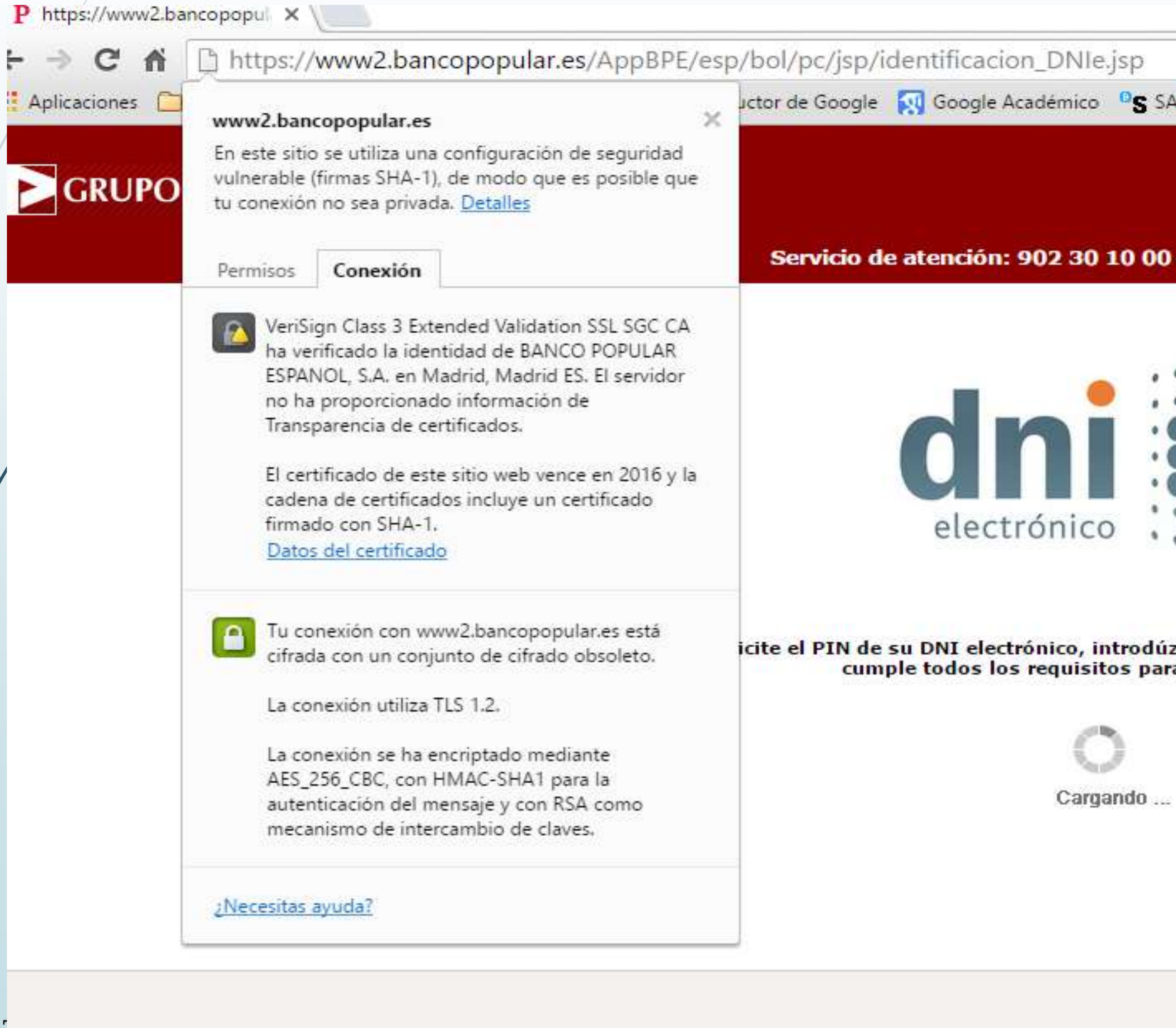
3.4 Algoritmo RC4

Año 2015



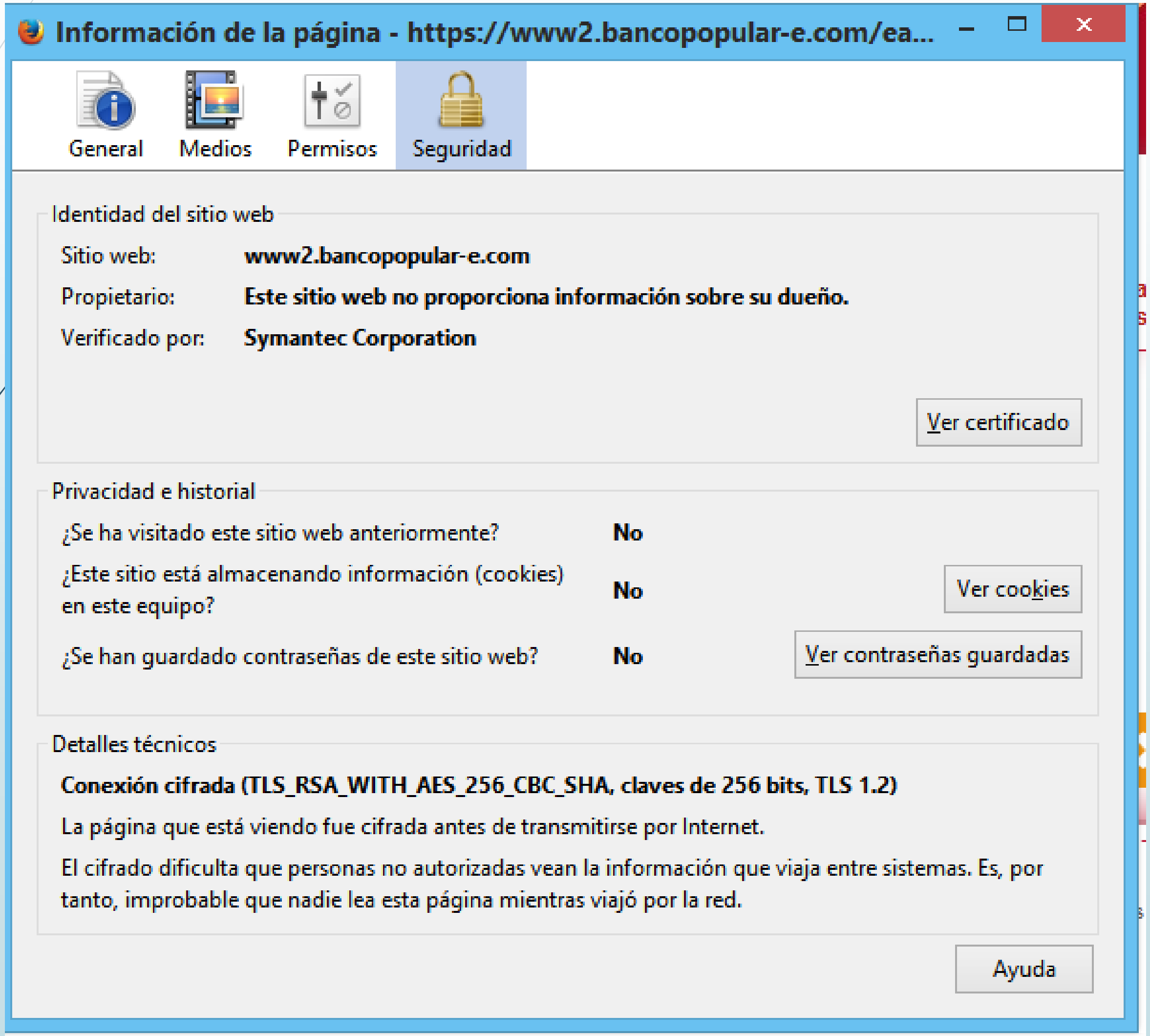
3.4 Algoritmo RC4

Año 2016



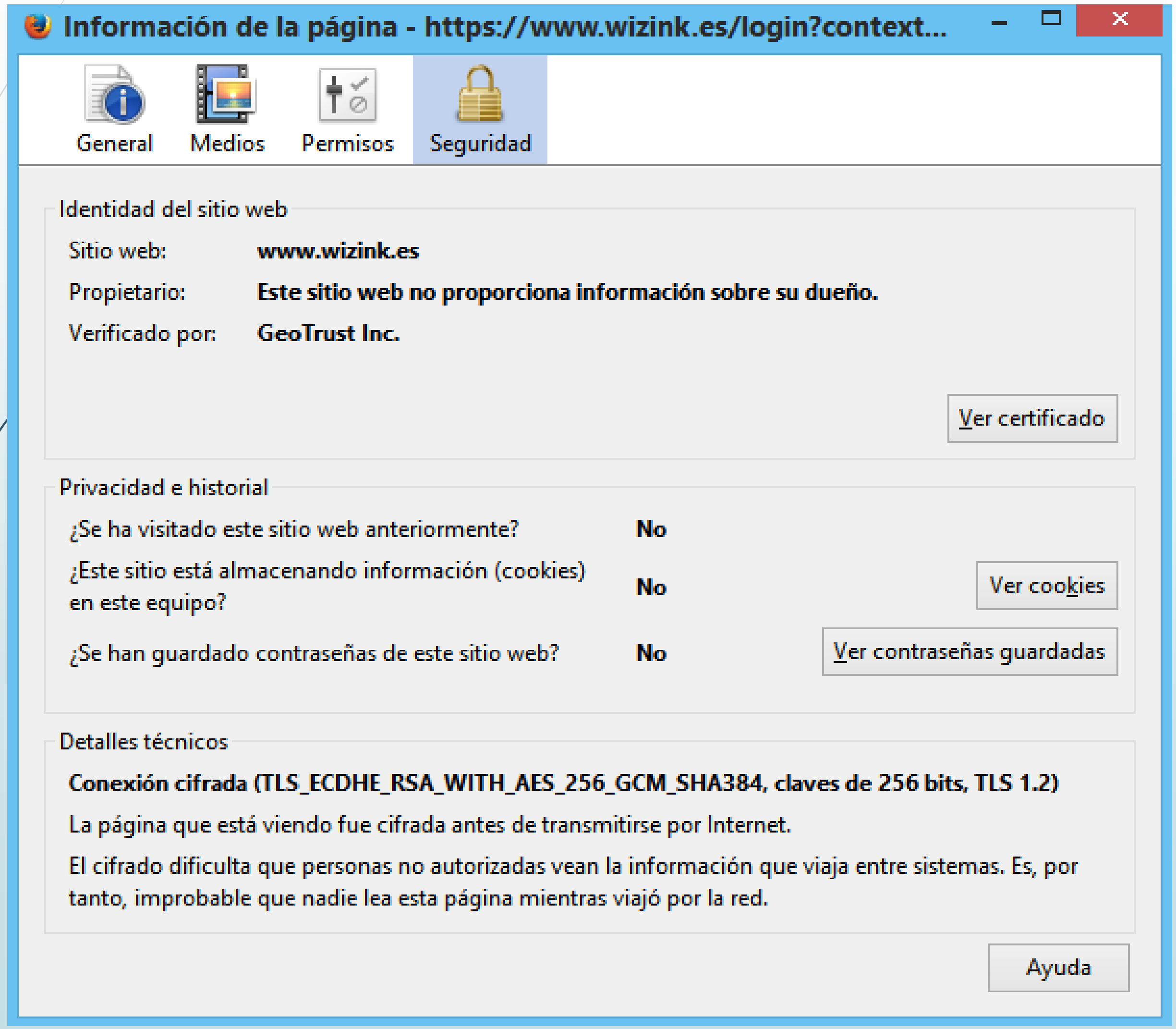
3.4 Algoritmo RC4

Año 2017



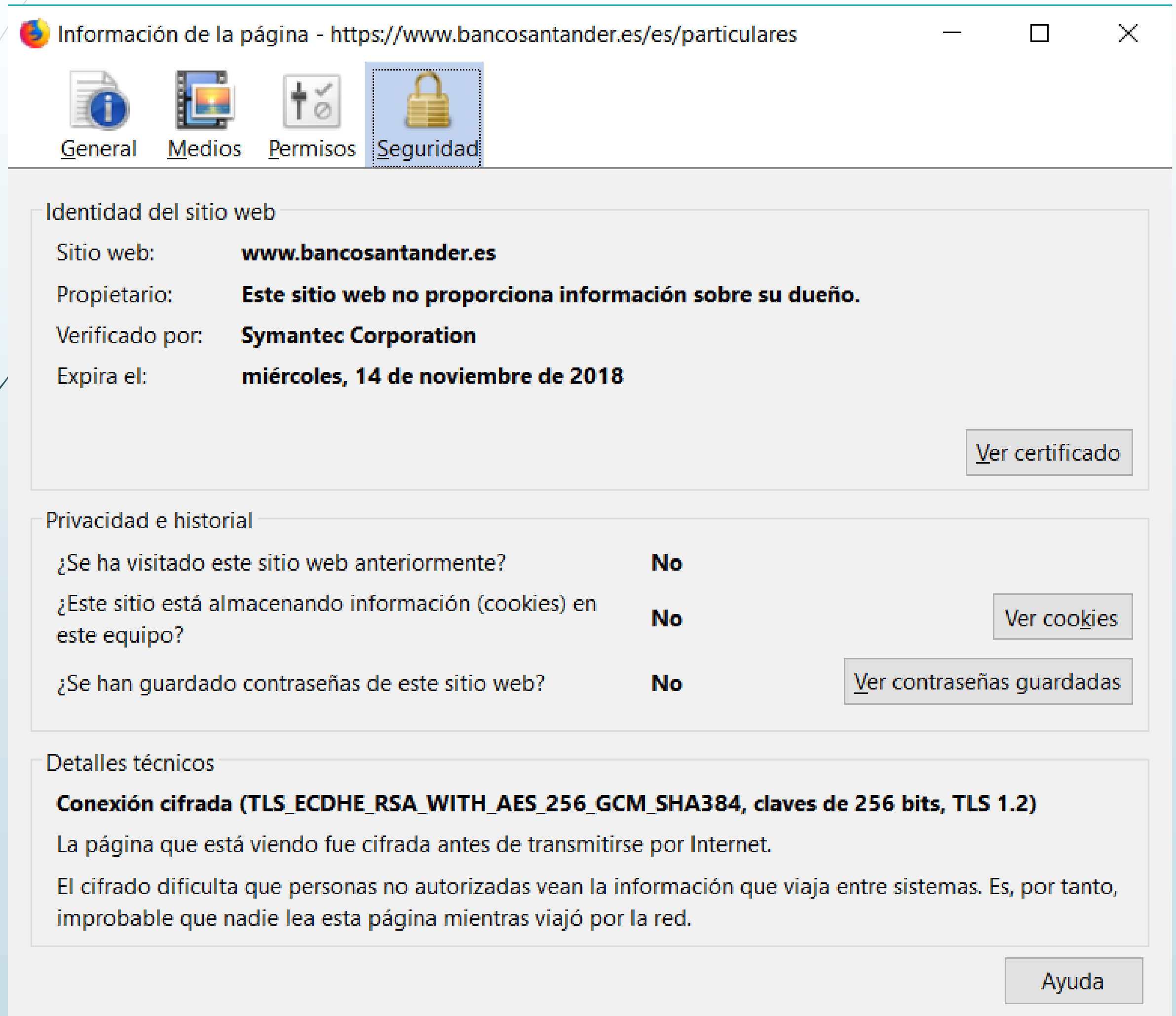
3.4 Algoritmo RC4

Años 2017, 2018



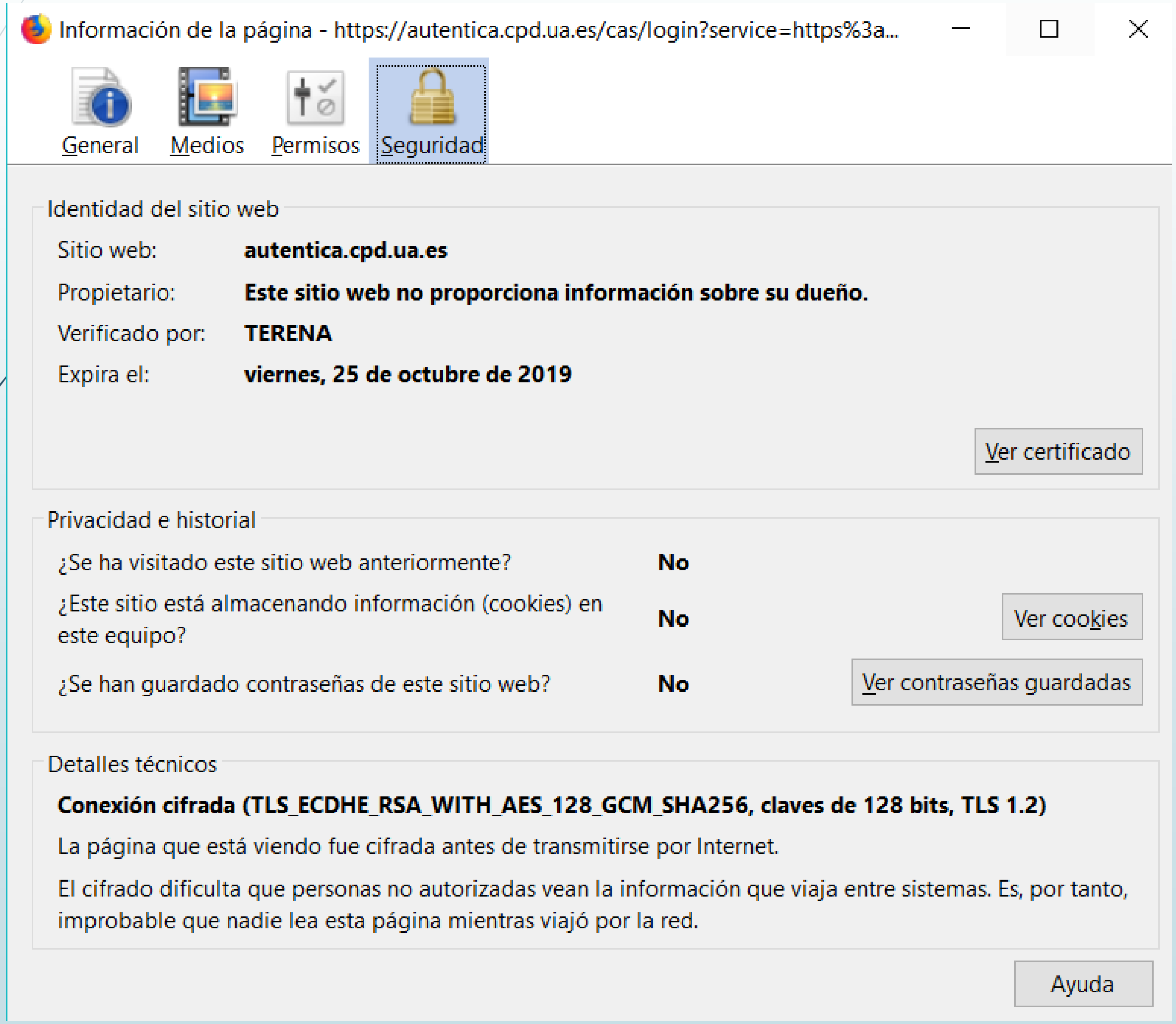
3.4 Algoritmo RC4

Año 2018



3.4 Algoritmo RC4

Año 2018



3.5 Algoritmo A5

- El algoritmo de cifrado en flujo A5 es un generador binario de secuencia cifrante utilizado para cifrar el enlace entre el teléfono móvil y la estación base en el sistema de telefonía móvil GSM (*Global Systems for Mobile communications*) o telefonía 2G.
- Utiliza 3 LFSR con 19, 22 y 23 celdas, que generan un periodo muy pequeño.
- La longitud de clave es, por tanto, de 64 bits.

3.5 Algoritmo A5

- Una conversación GSM puede visualizarse como una sucesión de tramas donde cada una de ellas contiene
  - 114 bits que representan la comunicación digitalizada entre móvil/estación base y otros
  - 114 bits que representan la comunicación digitalizada en sentido contrario.
- Una vez inicializado, el generador de secuencia cifrante produce 228 bits que se suman módulo 2 con los 228 bits de conversación en claro para producir los 228 bits de conversación cifrada.
- El procedimiento se repite para cada trama.

3.5 Algoritmo A5

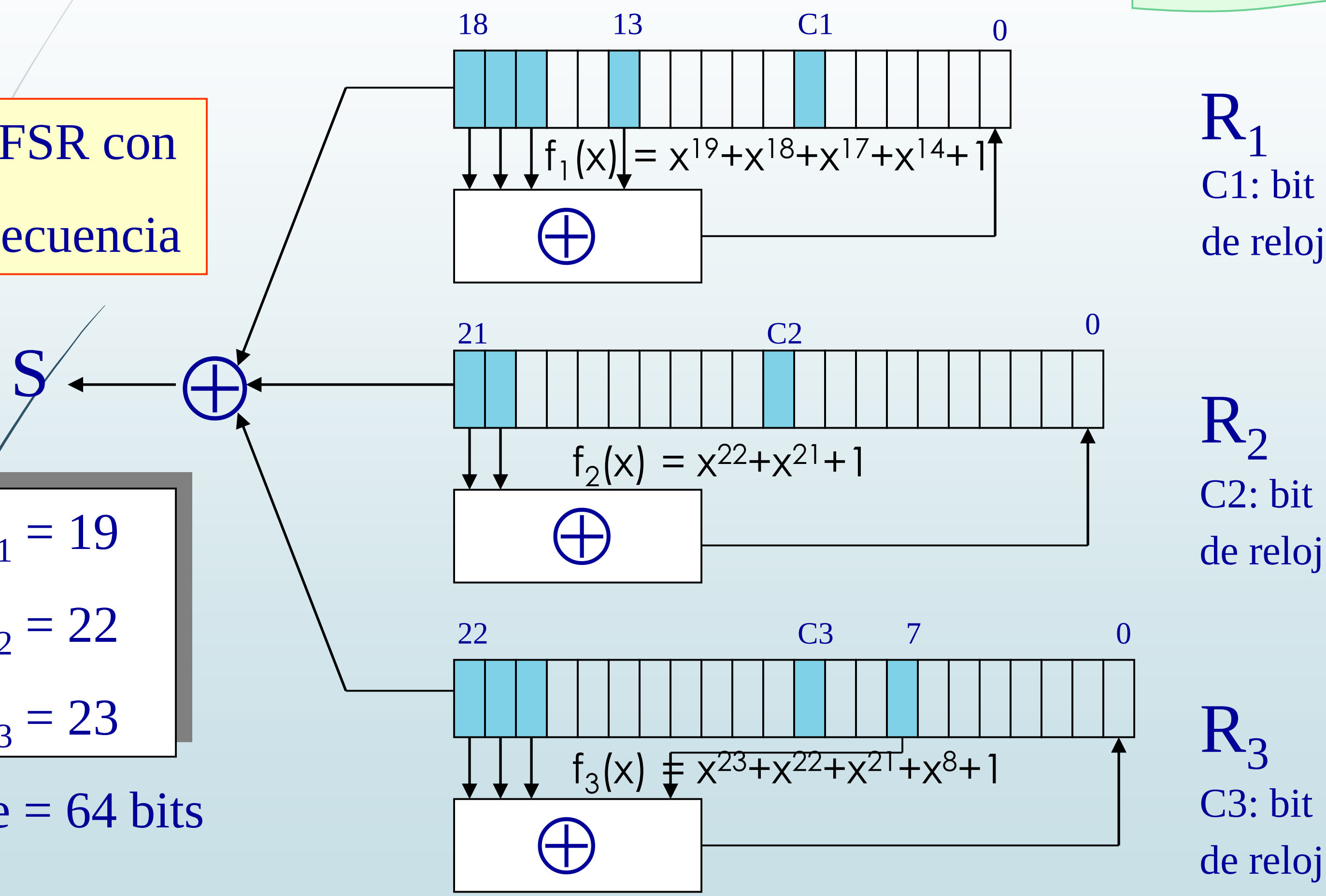
- Existen dos versiones del generador:
  - La versión A5/1 o versión fuerte, utilizada mayoritariamente en sistemas de telefonía móvil europea y estadounidense, y
  - La versión A5/2 o versión débil, utilizada fuera de Europa y EE.UU.
- En ambas versiones se detectaron serias debilidades, siendo reemplazado por el sistema de cifrado en bloque Kasumi en la tecnología 3G o UMTS.
- En 2010, el cifrado Kasumi fue atacado y roto con recursos computacionales muy modestos y, en consecuencia,
  - el sistema de cifrado tuvo que ser modificado de nuevo para la tecnología del nuevo estándar 4G o LTE, de manera que fue
  - el cifrado en flujo SNOW 3G el que se propuso para la protección de la confidencialidad e integridad de las comunicaciones.

3.5 Algoritmo A5

Esquema del algoritmo A5/1

FlujoLab

3 LFSR con m-secuencia



$n_1 = 19$

$n_2 = 22$

$n_3 = 23$

Clave = 64 bits

3.5 Algoritmo A5

CONSIDERACIONES SOBRE EL PERÍODO DE A5/1

- El período T viene dado por el mínimo común múltiplo de los tres períodos individuales:

$$T = mcm\ (2^{n_1} - 1,\ 2^{n_2} - 1,\ 2^{n_3} - 1)$$

- Como n_1, n_2 y n_3 son primos entre sí, también lo son los valores $2^{n_1} - 1$, $2^{n_2} - 1$ y $2^{n_3} - 1$. Entonces el período T es el producto de estos tres períodos:

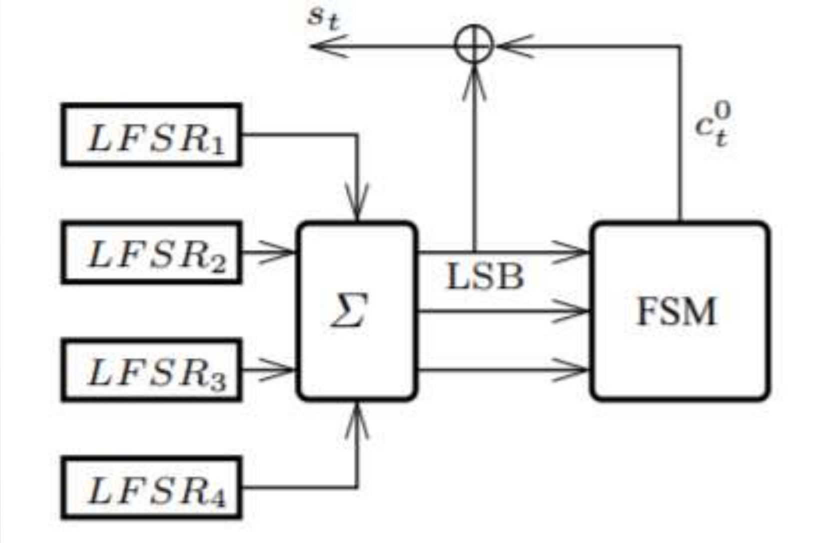
$$T = T_1 T_2 T_3$$

Criptoanálisis de A5/2

- En todo caso: $T < 2^{64}$, es un valor muy bajo.

Algoritmo E0 - Bluetooth

- El cifrado de la información transmitida mediante tecnología Bluetooth se lleva a cabo mediante un procedimiento de cifrado en flujo cuyo generador de secuencia cifrante es el algoritmo E0.



- Consta de 4 registros de desplazamiento LFSR1, LFSR2, LFSR3 y LFSR4 de longitudes respectivamente 25, 31, 33 y 39 celdas.

- La longitud de clave es, por tanto, de 128 bits.

Algoritmo E0 - Bluetooth

- Al igual que la tecnología GSM, Bluetooth también funciona **a nivel de un sistema de tramas** que se van cifrando sucesivamente.
- La longitud de trama es ahora de 2745 bits. La diferencia con GSM es que **la tecnología Bluetooth cifra cada trama con una clave distinta**.
- Esto implica que el criptoanalista dispone solamente de 2746 bits para desarrollar su criptoanálisis lo cual, en términos criptográficos, es poco.
  - De ahí que ninguno de los ataques criptoanalíticos desarrollados por vía algebraica o por correlación haya resultado fructífero.
- La **inseguridad** achacable al Bluetooth es más debida a un **mal método de inicialización y cambio de clave** que a una debilidad detectada en el diseño del generador de secuencia cifrante.

Security Mechanism	Legacy	Secure Simple Pairing	Secure Connections
Encryption	E0	E0	AES-CCM
Authentication	SAFER+	SAFER+	HMAC-SHA256
Key Generation	SAFER+	P-192 ECDH HMAC-SHA-256	P-256 ECDH HMAC-SHA-256

Table 1.1: Security algorithms

The eSTREAM Project

- http://www.ecrypt.eu.org/stream/
- El portafolios de eSTREAM contiene los siguientes cifradores en flujo:

Profile 1 (SW)	Profile 2 (HW)
HC-128	Grain v1
Rabbit	MICKEY 2.0
Salsa20/12	Trivium
SOSEMANUK	

Cifrado en flujo con clave secreta

ALGORITMOS MÁS UTILIZADOS

- RC4
- Salsa20
- SNOW
- HC256
- Rabbit

OTROS ALGORITMOS

https://en.wikipedia.org/wiki/Stream_cipher

Conclusiones

- El procedimiento criptográfico del cifrado en flujo demuestra ser rápido y eficaz en un mundo en el que cada vez hay más necesidad de proteger información.

- **No existe un criterio unificado** que dictamine si la secuencia cifrante utilizada está **suficientemente** próxima a una secuencia **aleatoria** que sería la única en garantizar la perfecta seguridad del método.

Conclusiones

- **No puede asegurarse** que un **generador** de secuencia cifrante sea intrínsecamente **bueno**; a veces, la fortaleza de un generador se fundamenta simplemente en que no se ha sabido aplicar el criptoanálisis adecuado o bien en que nadie se ha parado a criptoanalizarlo.

- Conviene resaltar que los procedimientos matemáticos que engloba la criptografía de clave pública son tan costosos **computacionalmente** que los **métodos de cifrado bit a bit son una buena opción.**

- Una operación lógica entre dos bits siempre será más sencilla y fácil de implementar que una operación matemática sobre un número de cientos de dígitos decimales.

Conclusiones

- La **rapidez** de ejecución del cifrado en flujo es y será siempre la **mejor garantía** de su vigencia.

- La gran velocidad de cifrado de estos sistemas hace que su aplicación esté sobre todo orientada al cifrado de grandes bloques de información como puede ser el intercambio de documentos hipermedia a través de redes públicas o streaming de vídeo.