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

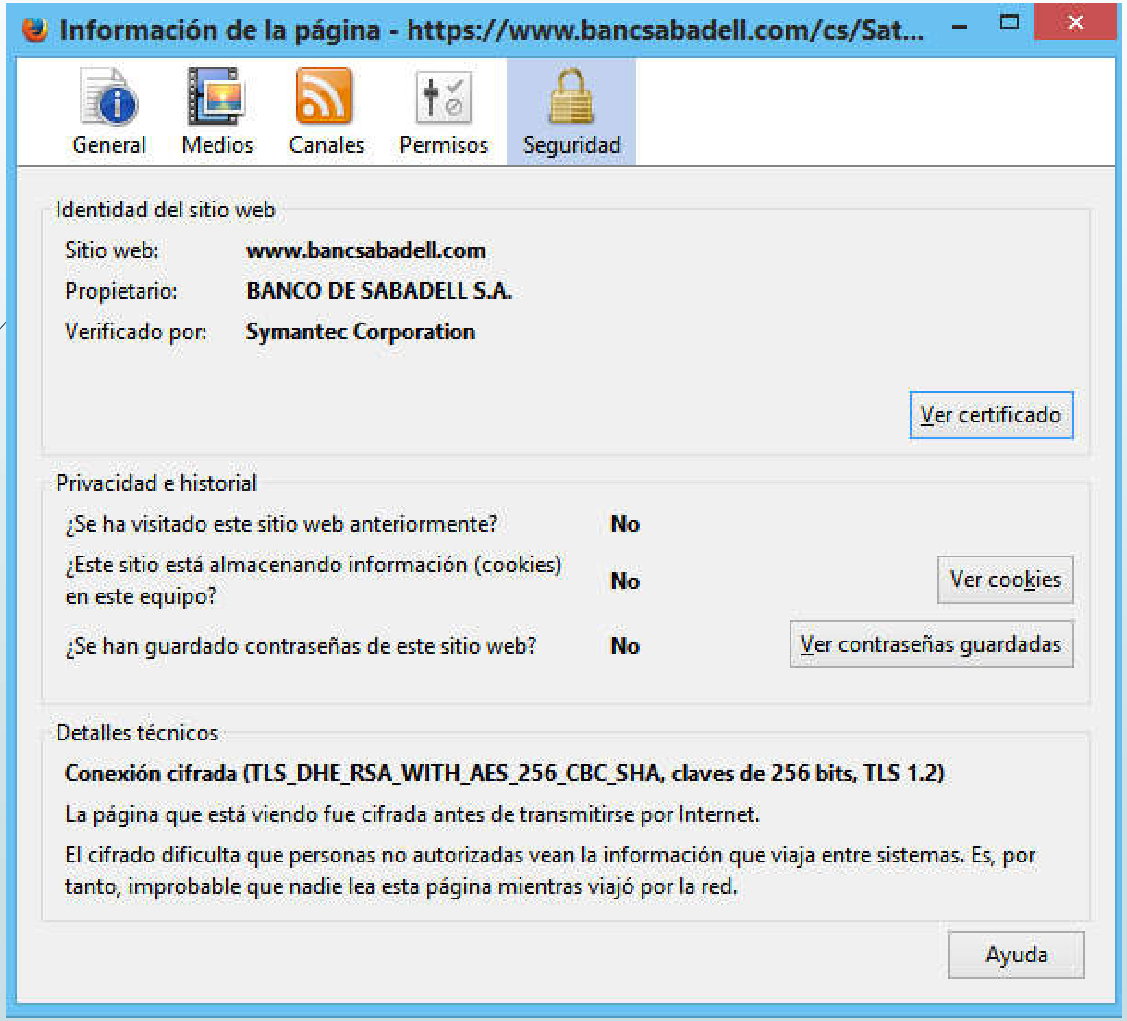
Año 2015

3.4 Algoritmo RC4

Año 2016

3.4 Algoritmo RC4

Años 2017, 2018



3.4 Algoritmo RC4

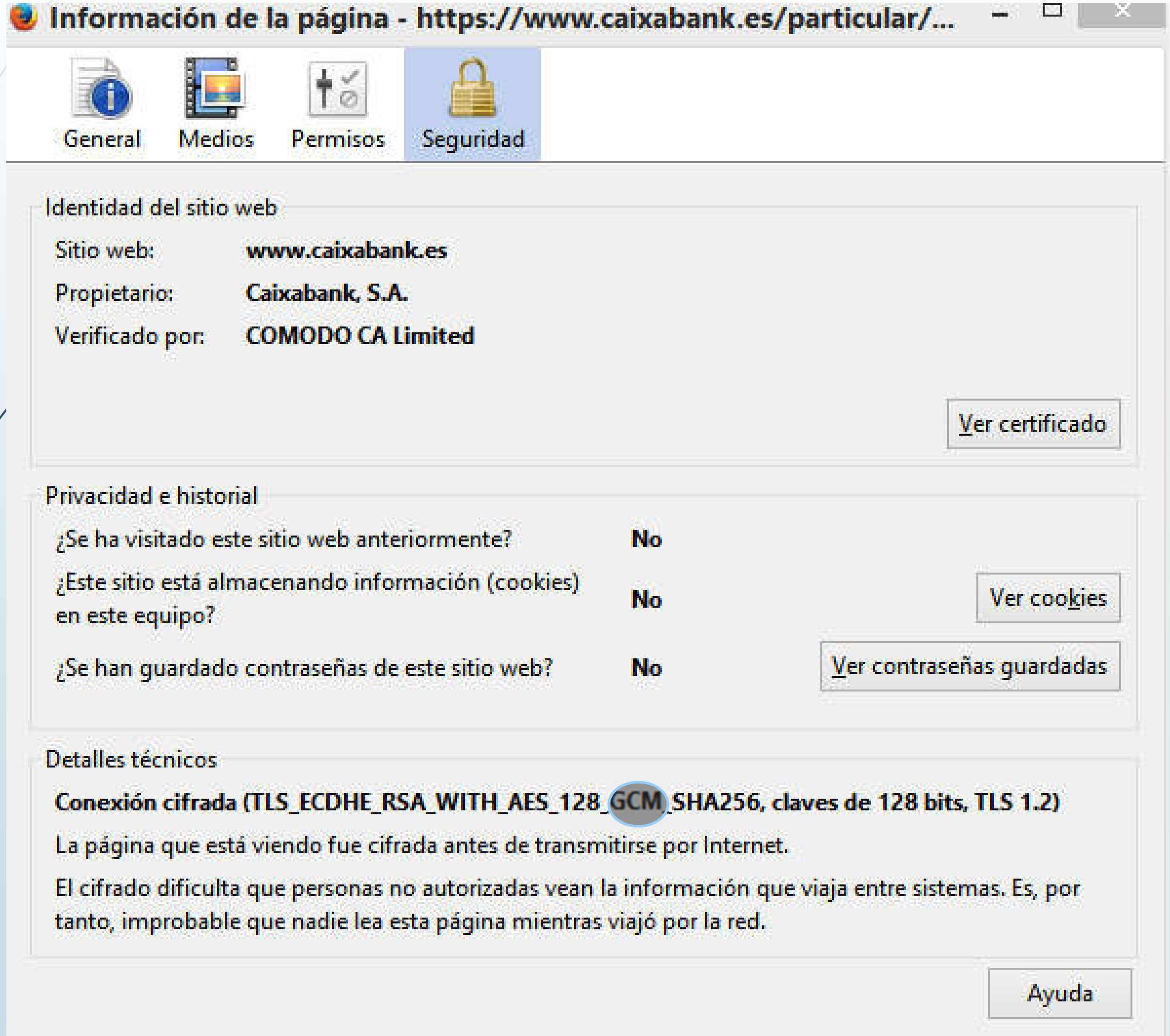
Año 2015

3.4 Algoritmo RC4

Año 2016

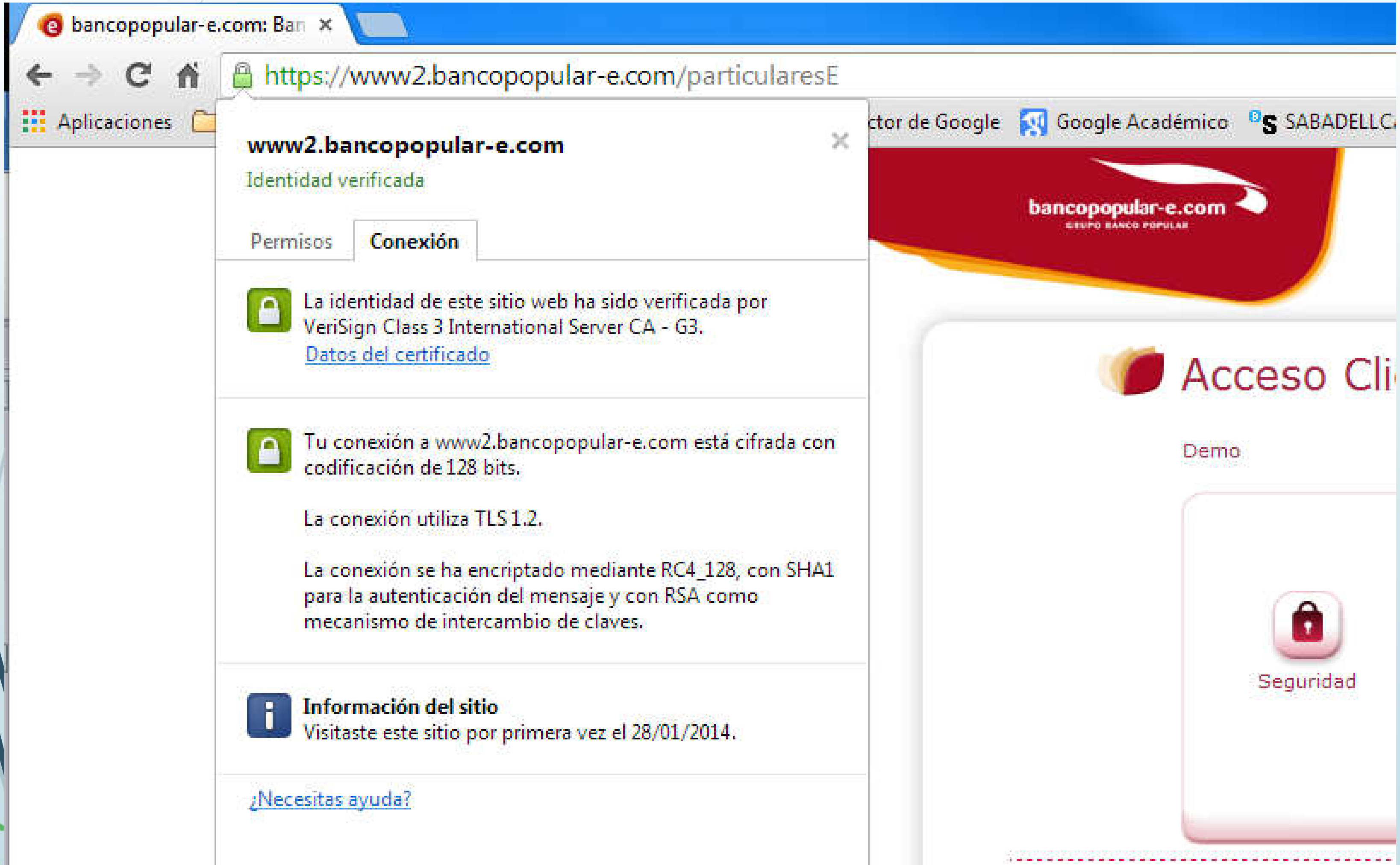
3.4 Algoritmo RC4

Años 2017, 2018



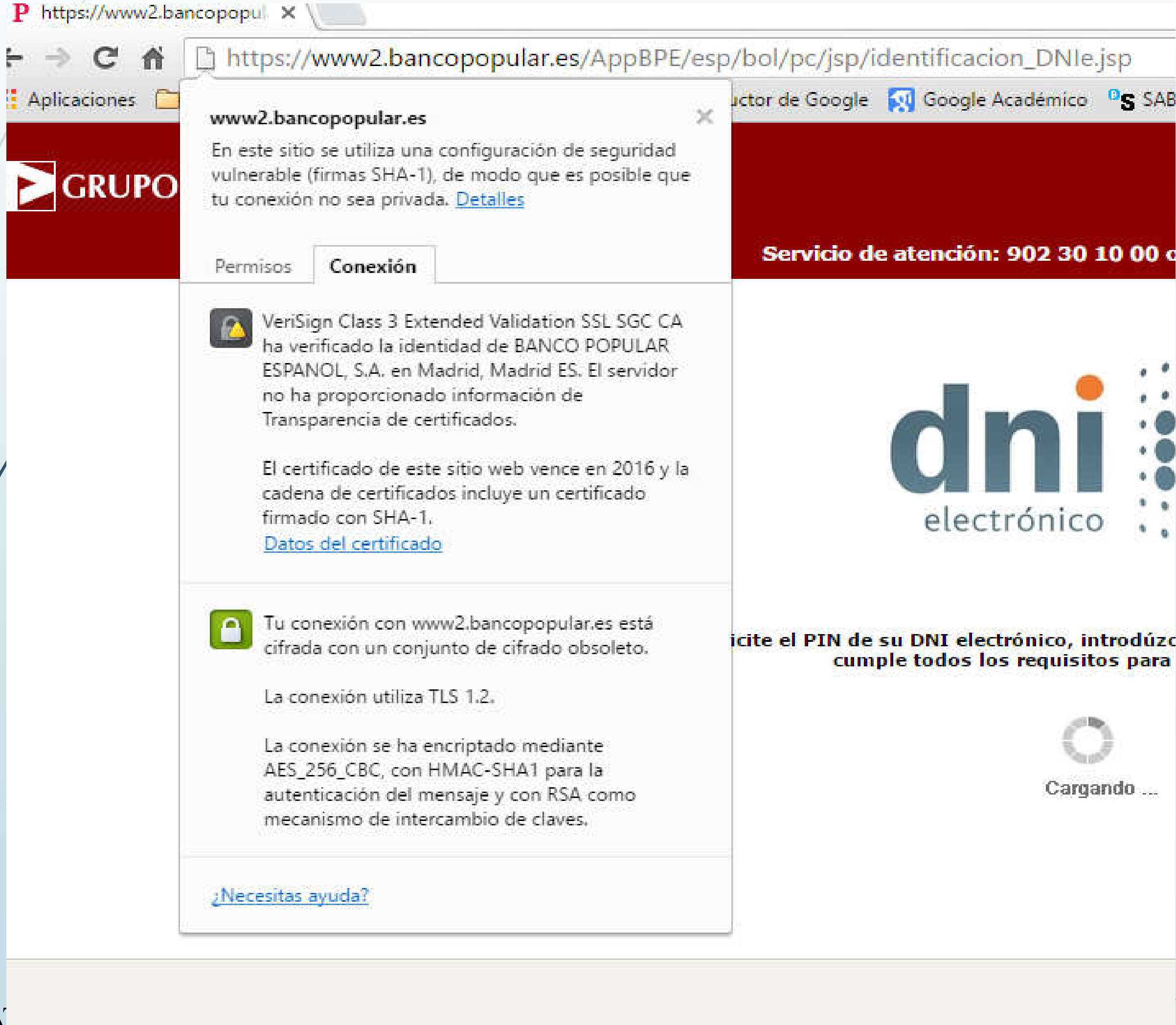
3.4 Algoritmo RC4

Año 2015



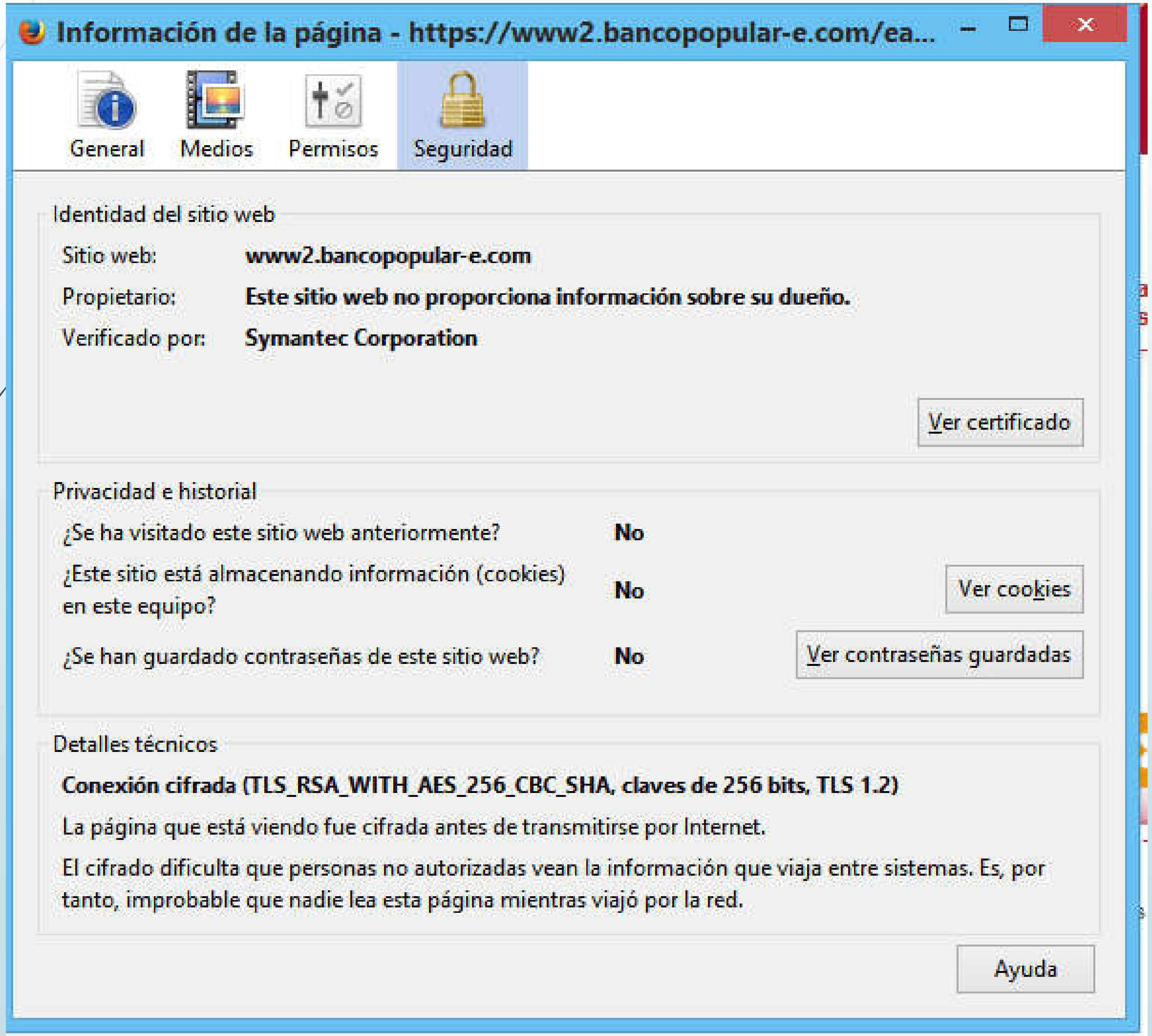
3.4 Algoritmo RC4

Año 2016



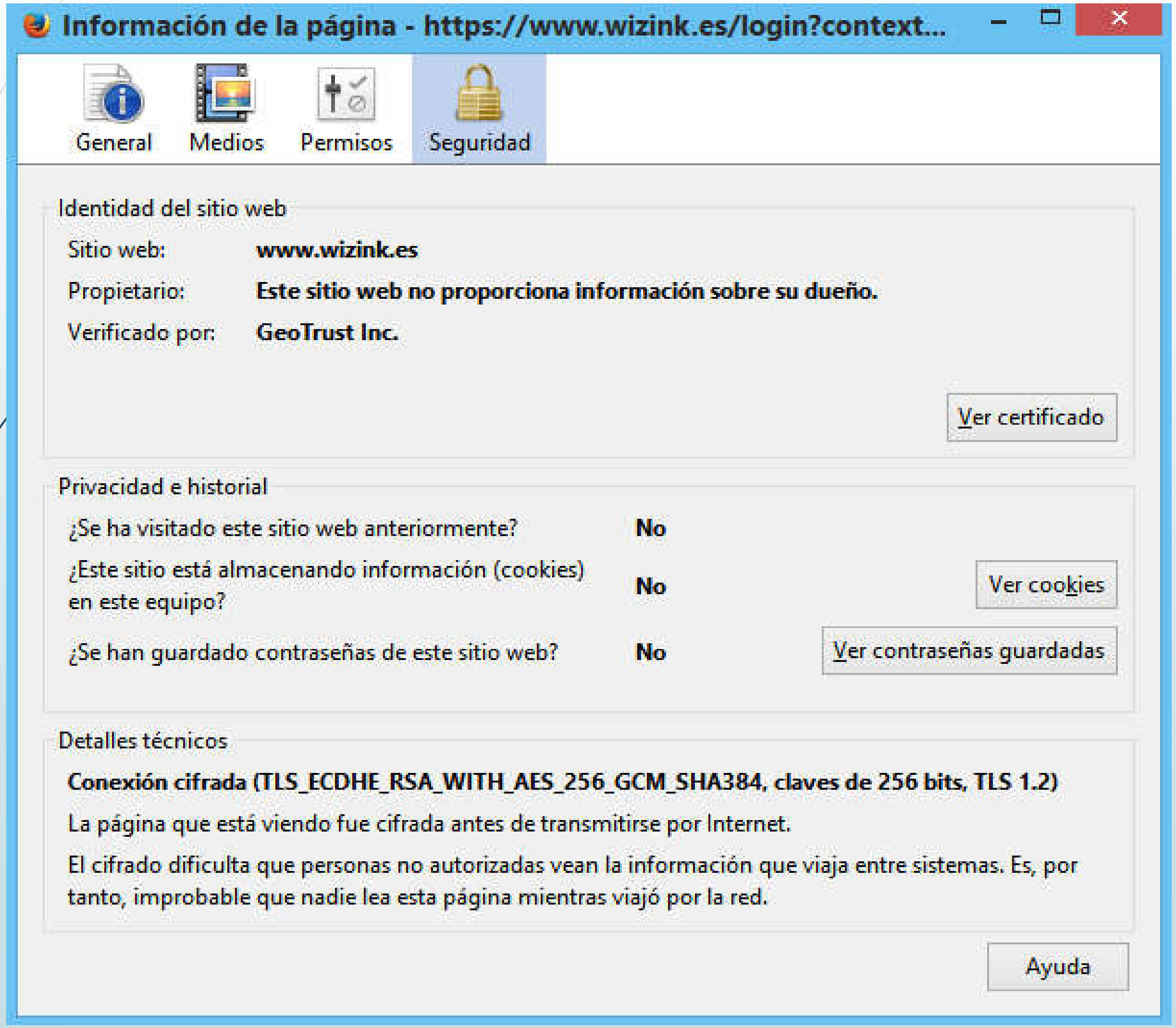
3.4 Algoritmo RC4

Año 2017



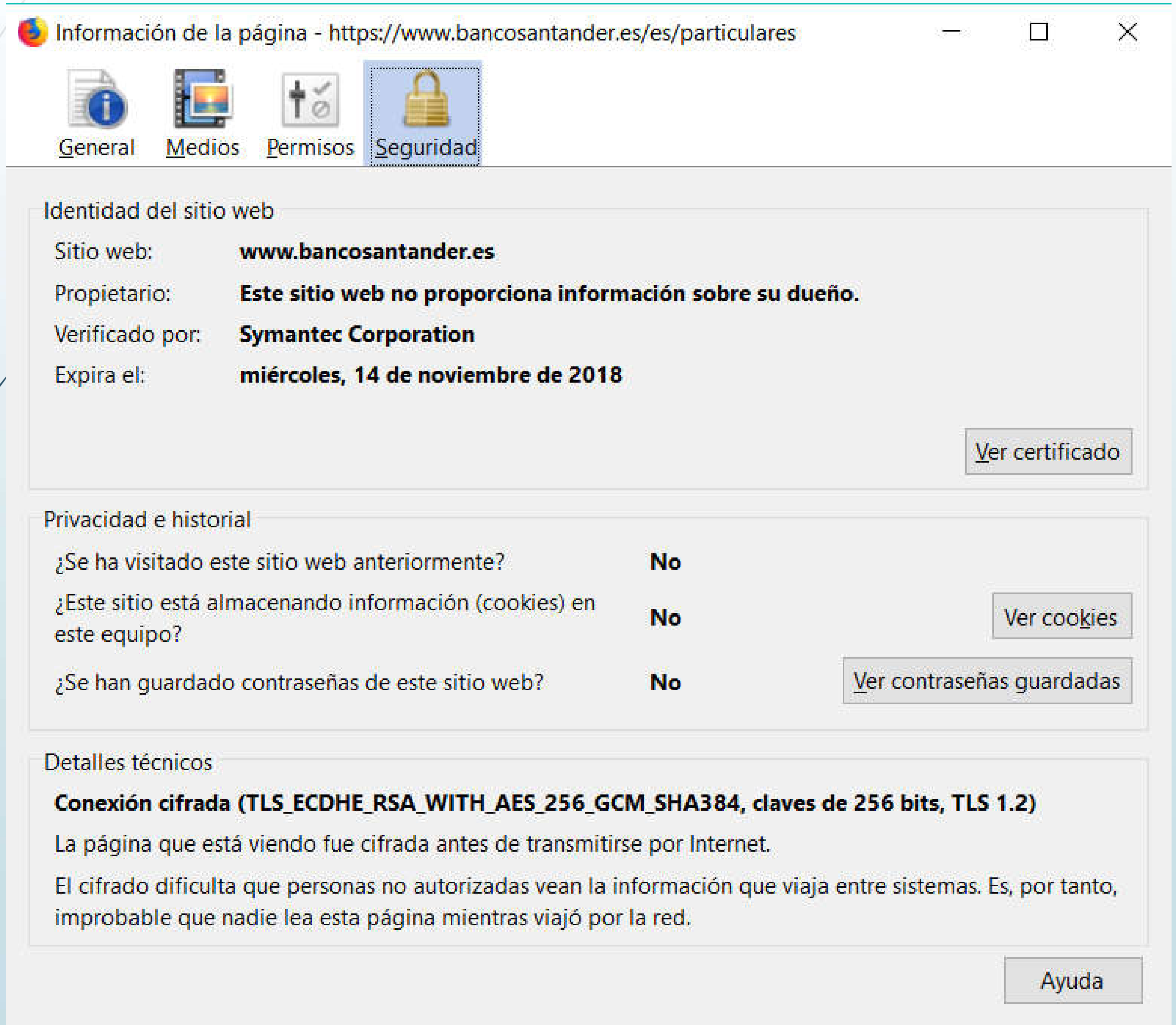
3.4 Algoritmo RC4

Años 2017, 2018



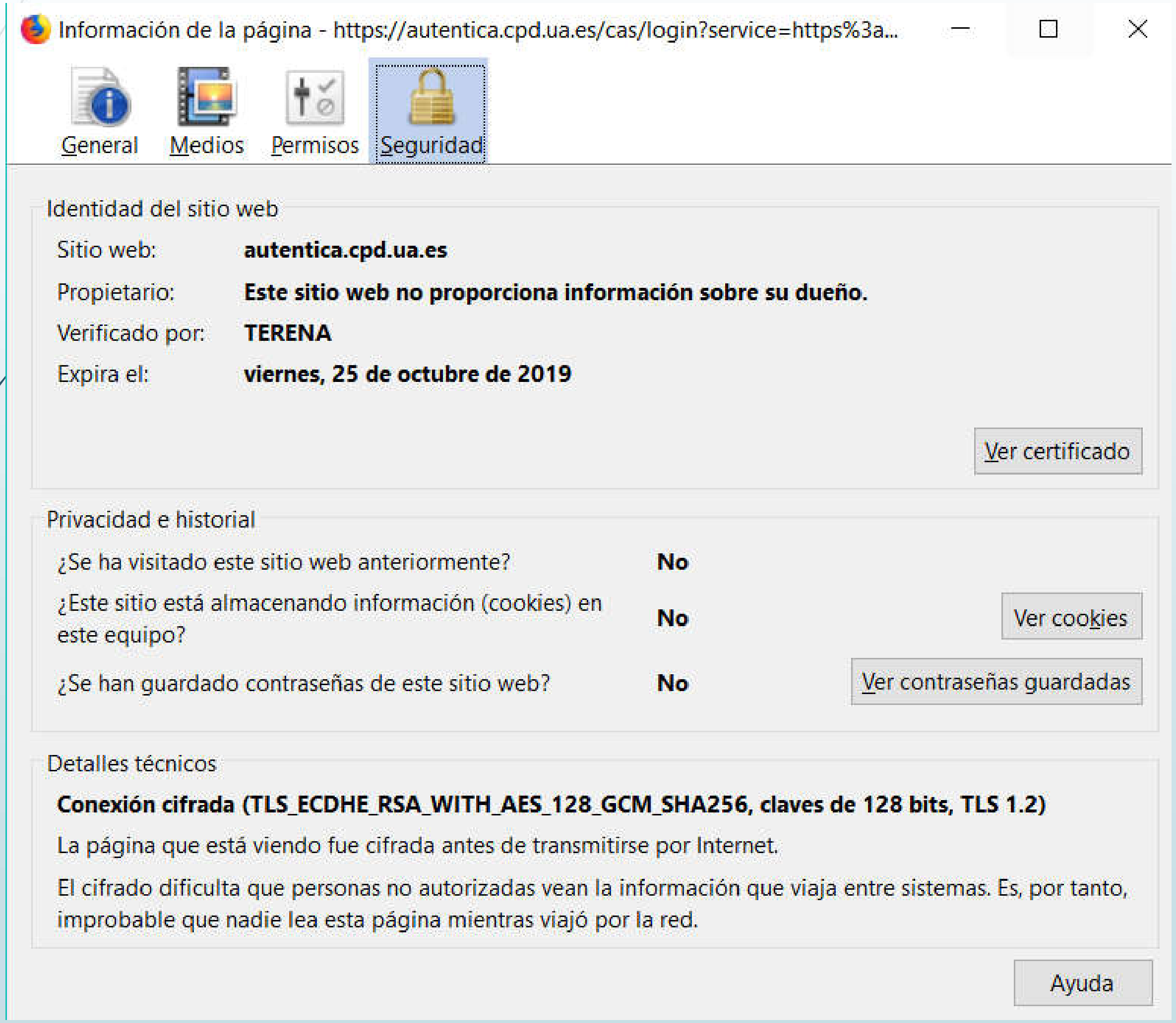
3.4 Algoritmo RC4

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.

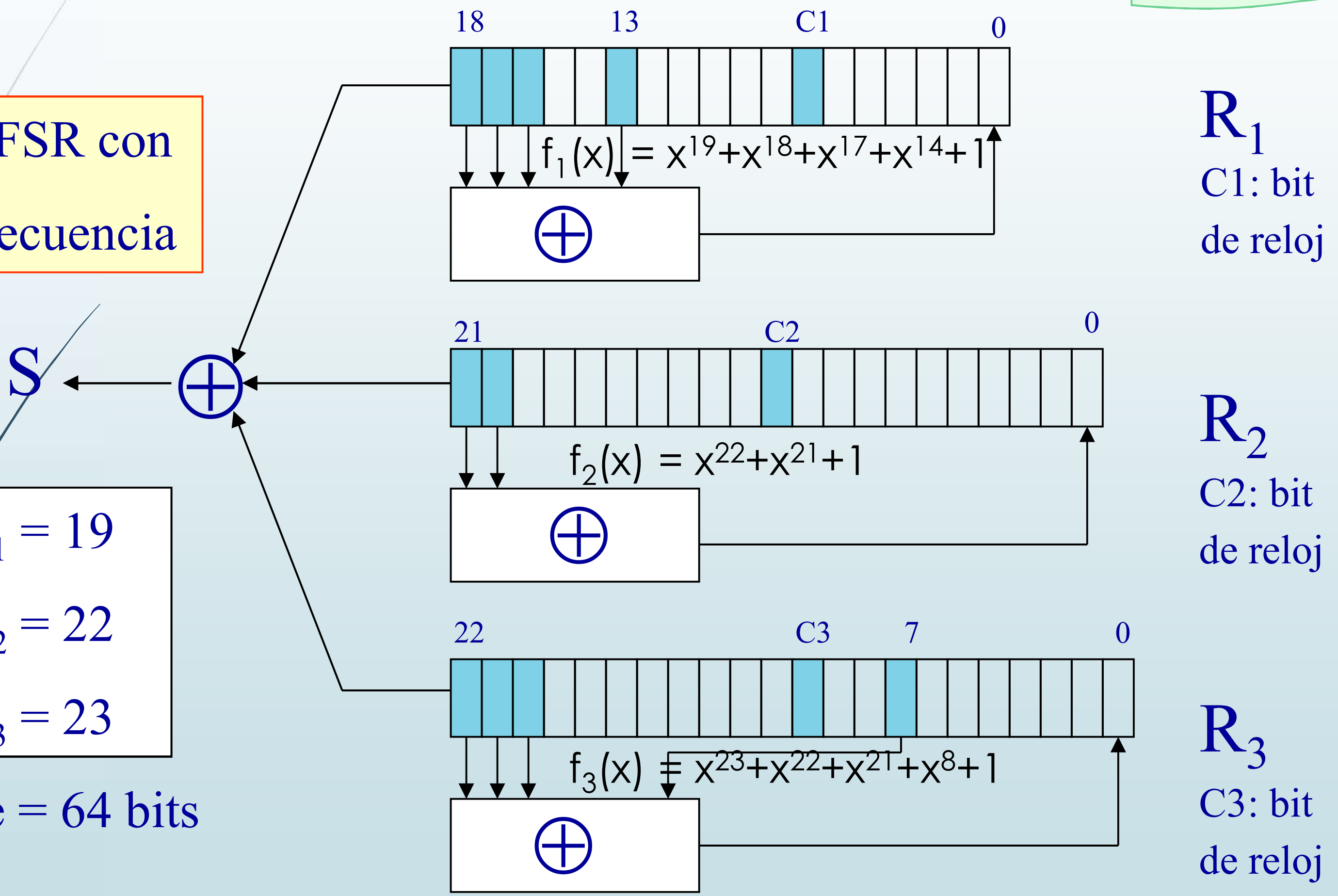
- En 2010, el cifrado Kasumi fue atacado y roto con recursos computacionales muy modestos y, en consecuencia,
  - el sistema de cifrado tuvo que ser modificado de nuevo para la tecnología del nuevo estándar 4G o LTE, de manera que fue
  - el cifrado en flujo SNOW 3G el que se propuso para la protección 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:

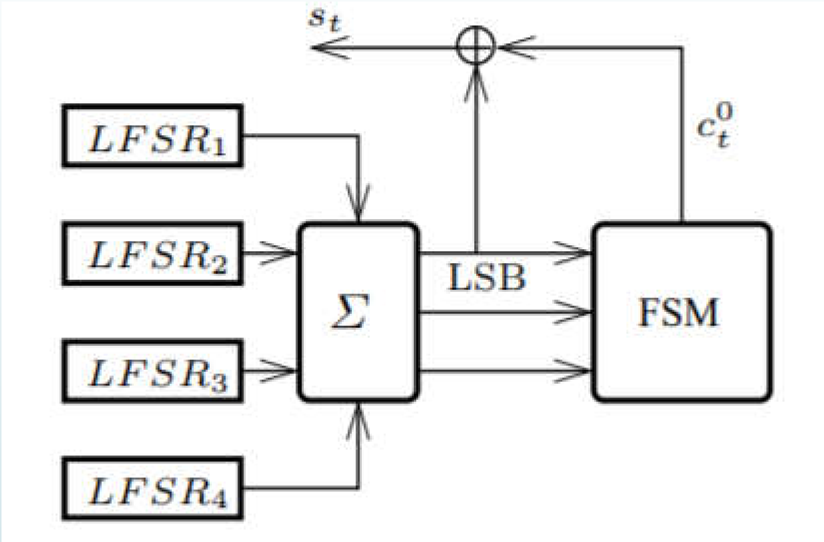
$$T = T_1 T_2 T_3$$

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.