Cifrado en Flujo

Cifradores en bloque y en flujo

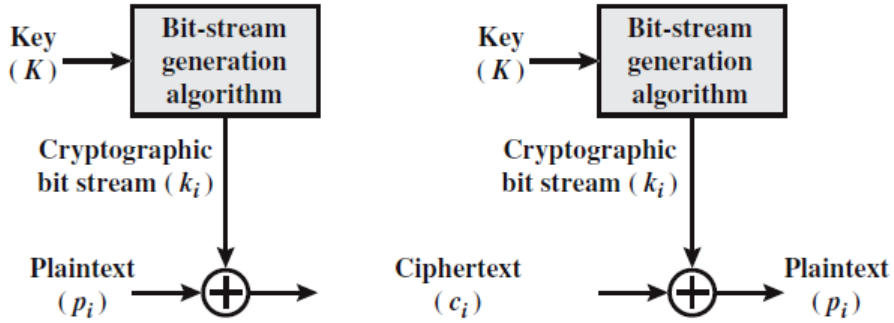
Cifrador en bloque

- Divide el texto en claro en bloques y cifra un bloque cada vez
- Aplica una **transformación fija** a cada bloque que sólo depende de la clave
- La mayoría se basa en una red Feistel

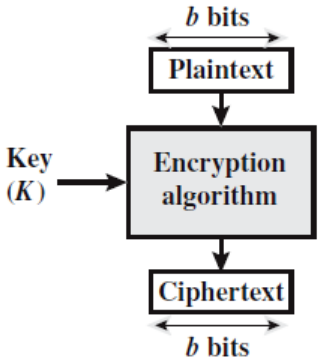
Cifrador en flujo

- Toma el texto en claro elemento a elemento cifrándolo **a modo de cadena**
- Aplica una **transformación variable** que **depende** de la clave y **del estado del cifrador**
- La mayoría se basa en el esquema Vernam

Cifradores en bloque y en flujo



(a) Stream cipher using algorithmic bit-stream generator



(b) Block cipher

Números Aleatorios

Uso de números aleatorios

- Tienen muchas aplicaciones en criptografía:
  - Distribución de claves
  - Esquemas de autentificación recíproca
  - Generación de claves de sesión
  - Generación de claves para RSA
  - Generación de secuencias para cifrado en flujo
- Características que ha de cumplir una secuencia aleatoria:
  - Aleatoriedad
    - Distribución estadística uniforme de los elementos a lo largo de la secuencia
    - Independencia, el valor de los elementos de la secuencia no depende de valores anteriores
    - Existen muchos tests para establecer la no-aleatoriedad pero ninguno que demuestre la aleatoriedad
  - Impredecibilidad
    - Resulta imposible establecer el siguiente bit con una probabilidad significativamente superior a 1/2

Tipos de generadores aleatorios

- Realmente aleatorios:
  - Basados en procesos físicos
  - Costosos de implementar y mantener
  - Requieren un post-proceso extenso (muy lentos)
  - No deterministas
- Pseudoaleatorios:
  - Basados en algoritmos deterministas
  - Muy eficientes
  - Parecen realmente aleatorios en la práctica (indistinguibles)
  - *Estos son los que interesan en la mayoría de los casos*

Generadores Pseudoaleatorios

- Toman un valor aleatorio pequeño (*semilla*) y generan una secuencia pseudoaleatoria mucho más larga.

- La semilla determina la secuencia producida
  (permite repetir la secuencia al emplear la misma semilla).

- La secuencia producida es, en la práctica, indistinguible de una secuencia realmente aleatoria.

- Aspectos de seguridad a tener en cuenta
  - **Período**: la longitud de la secuencia antes de empezar a repetirse.
  - **Tamaño de semilla**: el número de semillas posibles (≈espacio de claves).
  - **Impredecibilidad**: no podemos predecir el siguiente bit a pesar de conocer los anteriores.

Requisitos de un PRNG

- Aleatoriedad:
- Los tests buscan:
  - **Uniformidad**: la secuencia es aleatoria a lo largo de su longitud.
  - **Escalabilidad**: la secuencia es aleatoria también al extender su longitud.
  - **Consistencia**: la secuencia es aleatoria para todas las posibles semillas.
- Tests habituales:
  - **Frecuencia**: distribución de patrones.
  - **Runs**: grupos de 1's o 0's consecutivos.
  - **Poker**: elementos de 8 bit, 16 bit, etc.
  - **Test de Maurer**: asociado a la compresibilidad.
  - **TestU01**: suite de tests *(buena pero lenta)*
  - **PracRand**: suite de tests *(más rápida y moderna)*

- Impredecibilidad:
  - Predictiva
    - Imposible determinar el siguiente bit a pesar de conocer los anteriores
  - Retroactiva
    - Imposible determinar la semilla a partir de los bits producidos
- Semilla:
  - Ha de ser **impredecible** para que la secuencia también lo sea
  - Ha de ser un valor **aleatorio** *(el PRNG extiende dicha aleatoriedad a una secuencia mucho más larga)*

Diseño de PRNG

- Algoritmos a propósito
  - Congruencial Lineal
  - LFSRs
  - Matriciales
  - RC4
- Uso de primitivas existentes
  - Cifradores en bloque
  - Cifradores asimétricos (clave pública)
  - Funciones hash y MAC

Generador congruencial lineal

- Propuesto por Lehmer en 1951
- Se parametriza con 4 números
  - m módulo
  - a coeficiente
  - c incremento
  - X_0 semilla

$$X_{n+1} = (a \cdot X_n + c) \bmod m$$

- Se suele utilizar (típico en rand de programación):
  - $a = 7^5 = 16807$
  - $c = 0$
  - $m = 2^{31} - 1$
- Sólo son necesarios 4 valores consecutivos para romperlo *(mediante un sistema de ecuaciones lineal)*
- Útil en estadística y simulación pero ***¡no en criptografía!***

Generador Blum Blum Shub

- Propuesto en 1986 por Blum, Blum y Shub
- Es **demostrablemente seguro**
  (se basa en problemas matemáticos bien conocidos, análogos a los utilizados en criptografía de clave pública)
- Se eligen dos primos muy grandes p y q tales que

$$p \equiv q \equiv 3(\mathrm{mod}4)$$
$$n = pq$$

$$X_0 = s^2 \bmod n$$
$$X_i = (X_{i-1})^2 \bmod n$$
$$B_i = X_i \bmod 2$$

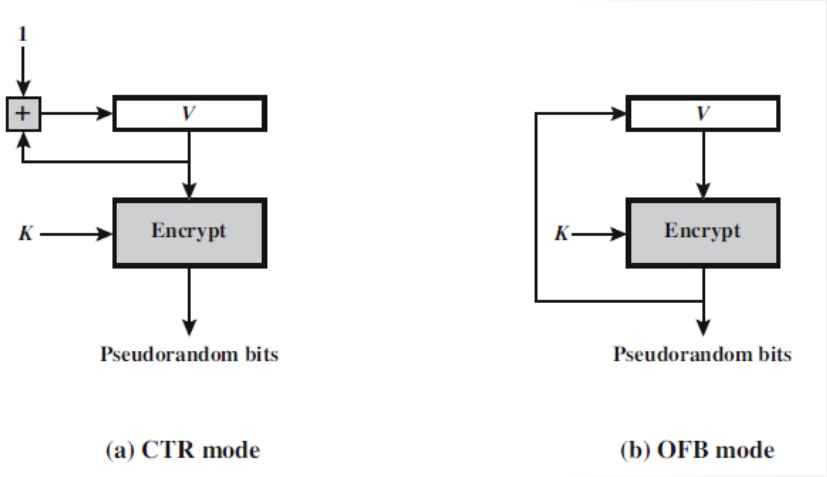
- La secuencia consiste en los distintos bits B_i, mientras que s es la semilla.
- No existe, en la actualidad, un algoritmo en **tiempo polinomial** que permita establecer el siguiente bit
- Es **extremadamente lento** por lo que su uso queda restringido a secuencias muy cortas:
  - Claves, valores pequeños, desafíos, etc.

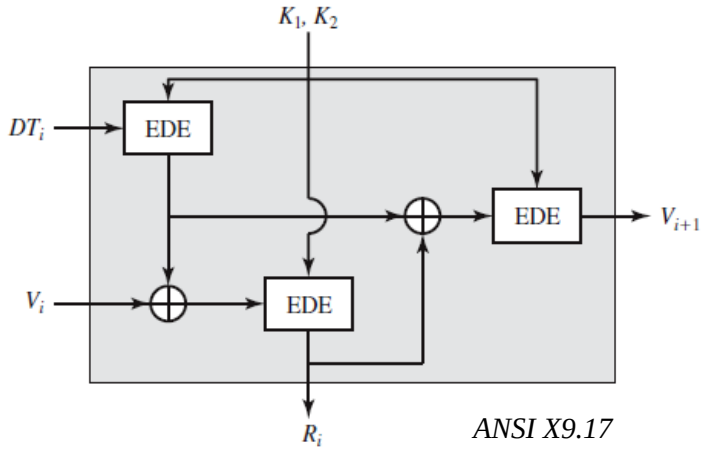
PRNG basados en cifradores en bloque

- Uso de cifradores en bloque en modos CTR y OFB
  - Práctico, reutiliza el cifrador en bloque ya implementado

- El estándar ANSI X9.17
  - Muy seguro
  - Finanzas y PGP
  - Hace uso de triple DES.
  - Se puede adaptar a otras primitivas más modernas

(los cifradores en bloque se tratan en el siguiente tema)





Cifradores en flujo

Se basan en generadores pseudoaleatorios (PRNG)

Cifradores en flujo

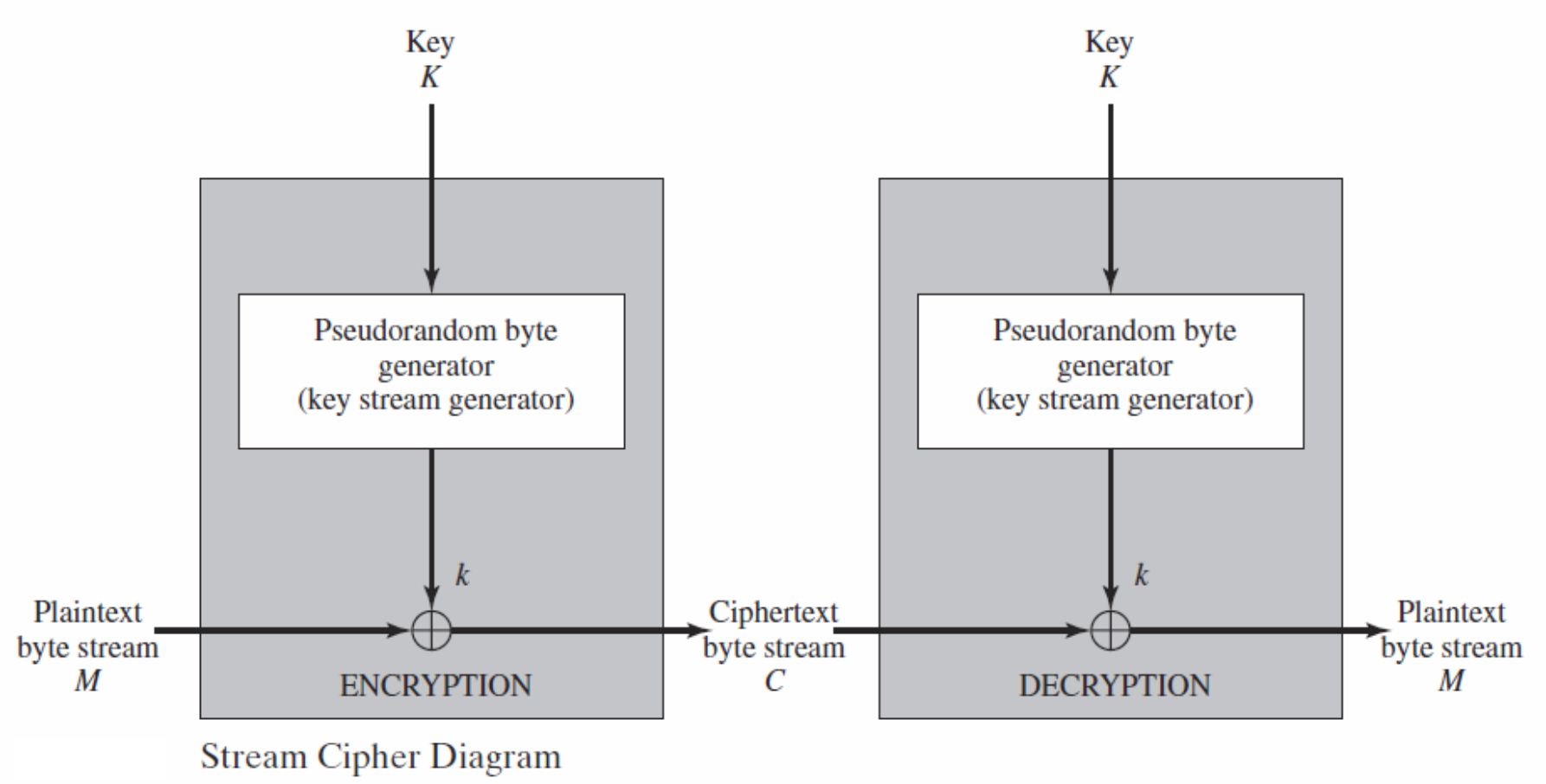
$$c_i = m_i \oplus k_i$$

$$m_i = c_i \oplus k_i$$

$$m_i \oplus k_i \oplus k_i = m_i$$

- Se basan en hacer el **XOR** entre el texto en claro y una secuencia cifrante.
- **Descifrar es cifrar dos veces** con la misma secuencia cifrante, puesto que la operación XOR se anula.
- En la práctica, se usa un **generador pseudoaleatorio** para obtener la secuencia cifrante.
- La **semilla** del PRNG actúa como **clave**.
- Al ser un algoritmo determinista, **sólo es necesario compartir la clave** (semilla) en lugar de toda la secuencia cifrante (que no sería viable)

Cifradores en flujo



Stream Cipher Diagram

Cifradores en flujo

- Síncronos:
  - La secuencia cifrante se genera de forma independiente.
  - El emisor y receptor han de estar sincronizados.
  - Un bit borrado provoca un error de sincronía.
  - Un bit alterado no afecta a los demás bits.
- Autosincronizantes:
  - La secuencia cifrante es una función de la clave y el texto cifrado.
  - El emisor y receptor se sincronizan de forma automática.
  - Cualquier bit erróneo provoca una propagación de errores.

*Los cifradores en flujo **son casi todos síncronos**, si bien algunos pueden ser modificados para ser autosincronizantes*

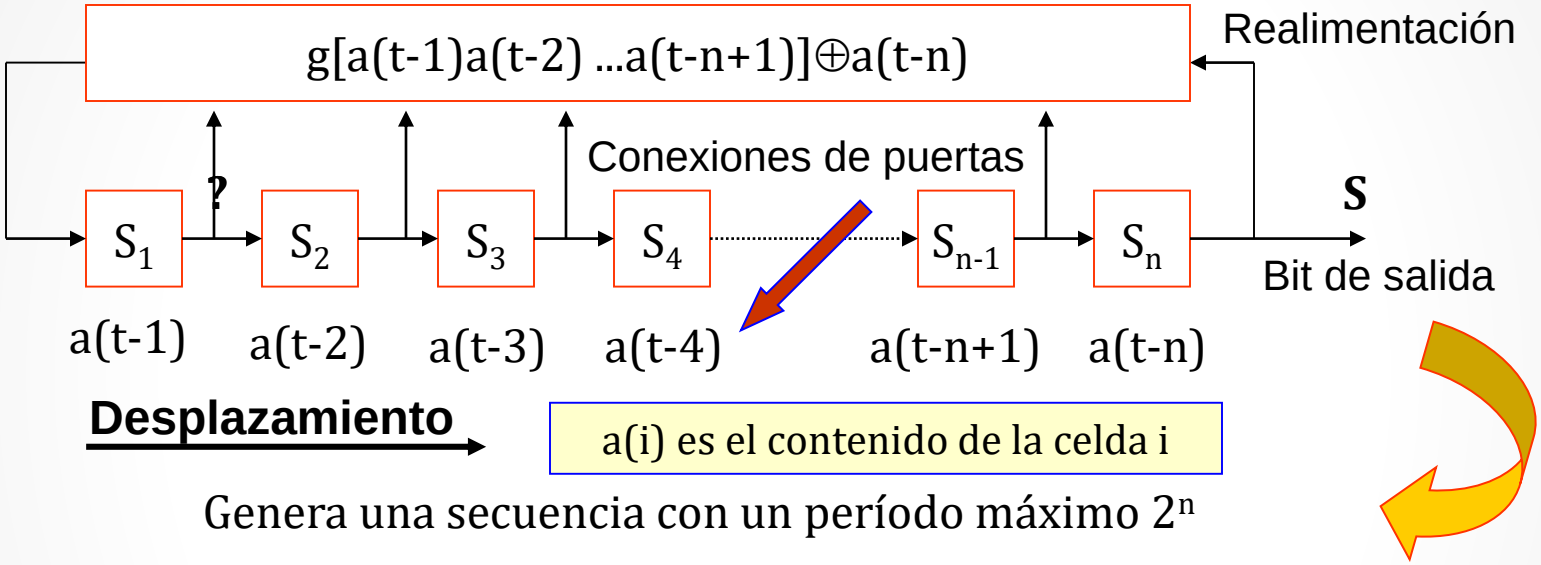
LFSRs

• • •

Registros de desplazamiento con retroalimentación lineal

LFSRs

S(i) es la semilla de cada celda i: es un bit 0 ó 1



a(i) es el contenido de la celda i

Genera una secuencia con un período máximo 2^n

Según la función de retroalimentación:

Registros de **D**esplazamiento **R**ealimentados **N**o **L**inealmente → NLFSR

Registros de **D**esplazamiento **R**ealimentados **L**inealmente → **LFSR**

Más utilizados

LFSRs

- Son muy populares para la generación de secuencias pseudoaleatorias
- Su función de retroalimentación es lineal (XOR de bits)
- Su modelización es sencilla y su implementación en hardware también
- Tienen un período máximo posible de 2^n-1, siendo n el tamaño del registro

- Las secuencias generadas cumplen los requisitos
  - Período
  - Distribución estadística
  - Facilidad de implementación
- Fallan en la impredecibilidad
  - Complejidad lineal
  - Dados 2n bits sucesivos se puede predecir el resto de la secuencia
- Se modifican para hacerlos seguros
  - Filtrado no lineal
  - Combinación no lineal

LFSRs

Filtrado no lineal

- Difícil de analizar y acotar
- Se parte de un LFSR con período máximo
- Se diseña una función no lineal que se aplica a la salida para obtener una complejidad lineal idónea
- No es la opción más habitual

Combinación no lineal

- Sencillo de analizar
- Combina varios LFSR con período máximo
- Selector
  - Geffe
- Control de reloj
  - Beth / Piper
  - Gollman
- Multireloj
  - Massey / Rueppel

RC4

RC4

- Diseñado en 1987 por Ron Rivest (RSA)
- Clave de tamaño variable, operaciones sobre bytes
  (ideal para implementaciones software)
- Basado en una permutación aleatoria
- Tiene un período cercano a 10^{100}
- Es increíblemente rápido en software (y hardware)

- Se ha utilizado en
  - SSL/TLS
  - WEP/WPA
  - Etc.
- Mantenido en secreto por RSA
  - En 1994 se publicó de forma anónima en Cypherpunks
- Es sorprendentemente sencillo y fácil de recordar

RC4

- Se basa en una s-box de 8x8
  (un array de 256 bytes)
  que depende de la clave
  (de hasta 256 bytes de longitud)

```
/* inicialización */
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod keylen];

/* Permutación inicial de S */
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);
```

```
/* Generación de secuencia */
i, j = 0;
while (true)
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    k = S[t];
```

RC4

- Existen publicaciones analizando métodos de ataque
  - Si bien no son prácticos en la mayoría de los casos, hacen dudar de RC4 y **ha sido prohibido en los estándares de SSL/TLS**

- En el caso de WEP (WiFi), se encontró una vulnerabilidad en la forma en la que se generan las claves para RC4
  - No afecta a RC4 en sí, es un problema del protocolo
  - Resalta la dificultad de diseñar sistemas seguros a pesar de emplear primitivas criptográficas (en principio) seguras

- Existe un ataque descubierto en la Royal Holloway (Londres)
  - Necesita 2^{24} conexiones (puede suponer un problema con TLS, que hace muchas conexiones en poco tiempo)

- Existen múltiples variantes que buscan más seguridad pero ninguna ha alcanzado la popularidad de RC4: RC4A, VMPC, RC4+, Spritz,…

Salsa20

Salsa20

- Diseñado por Dan Bernstein y envíado a eSTREAM
- Se basa en una función pseudoaleatoria que emplea:
  - Suma en 32bits
  - XOR entre bits
  - Rotaciones
- Obtiene un valor de 512bits a partir de:
  - Clave de 256bits
  - Nonce de 64bits
  - Posición de 64bits
- Se puede obtener cualquier parte de la secuencia sin generar los bits previos
  (ideal para cifrado de disco o acceso aleatorio)
- De 4 a 14 ciclos por byte y de dominio público

Salsa20

- Variantes:
  - Salsa20 -> 20 rondas
    - Salsa20/8 *(8 rondas)*
    - Salsa20/12 *(12 rondas)*
  - ChaCha
    - Intenta incrementar la difusión en cada ronda
    - Mejorar el rendimiento
    - Modifica la función de ronda
    - A pesar de ser una versión mejorada es menos popular
- Elección en eSTREAM
  - Obtuvo grandes votaciones en las fáses 1 y 2.
  - No avanzó a la fase 3 porque se consideró no apto para entornos de hardware restringido
- Criptoanálisis
  - No existen ataques conocidos para Salsa20/12 o Salsa20 completo
  - Existe un ataque que rompe la versión de 8 rondas

Otros cifradores en flujo

- A5/1 – A5/2 (1989, GSM 2G)
- SEAL (1997, IBM)
- Phelix (2004, eStream)
- HC128 (2004, eStream)
- SNOW 3G (2006, 3GPP)
- Spritz (2014, Rivest)

Ampliación

Otros materiales

- Se puede consultar el capítulo 11 del libro de Lucena
  (en los materiales de UACloud)

- También se puede consultar los capítulos 5 y 6 de *"Handbook of Applied Cryptography"*
  (más avanzado y en inglés)

Cuestiones

- Busca información online acerca de otros cifradores en flujo (ver transparencia anterior). Un buen punto de partida puede ser *Wikipedia*.

- ¿Qué cifrador en flujo elegirías en la actualidad, atendiendo a la seguridad y el rendimiento?