Criptosistemas simétricos de flujo

stream ciphers

Lámina 1 Roberto Gómez C.

Clasificación métodos encripción simetricos

- Encripción en bloques

- Encripción en flujo

Lámina 2 Roberto Gómez C.

Encriptado en flujo

- En inglés: stream ciphers.
- Usa la llave como semilla de un generador de números pseudo-aleatorio.
- El resultado del generador es una secuencia de bits.
- La secuencia se suma módulo 2 con el texto claro (emisión) o con el criptograma (recepción)

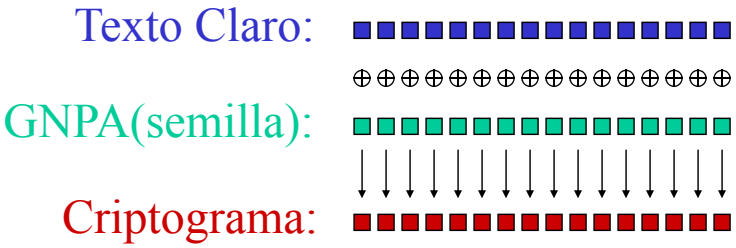
Lámina 3 Roberto Gómez C.

Características encripción flujo

- Generar una secuencia larga e imprevisible de dígitos binarios a partir de una llave corta elegida de forma aleatoria
- Es sencillo, rápido y seguro
- Es más seguro conforme más se aproxima la secuencia binaria generada a una autentica secuencia aleatoria

Lámina 4 Roberto Gómez C.

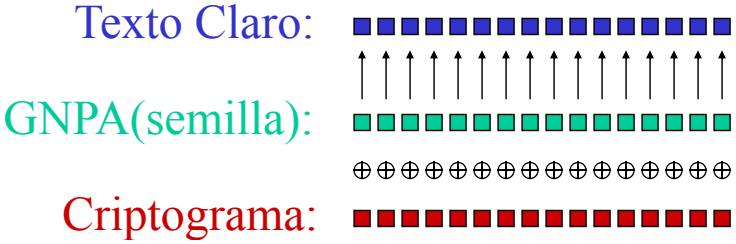
Encripción de criptosistemas de flujo



GNPA: Generador Números Pseudo-Aleatorios

Lámina 5 Roberto Gómez C.

Decripción de criptosistemas de flujo



GNPA: Generador Números Pseudo-Aleatorios

Lámina 6 Roberto Gómez C.

Elementos de encripción en flujo

- Vernam requiere un bit de llave por cada bit de texto claro, lo cual desbordaría capacidad canal
- Método cifrado en flujo:
  - emisor A
  - llave corta
  - algoritmo determinístico
  - receptor B

Lámina 7 Roberto Gómez C.

Esquema cifrado en flujo

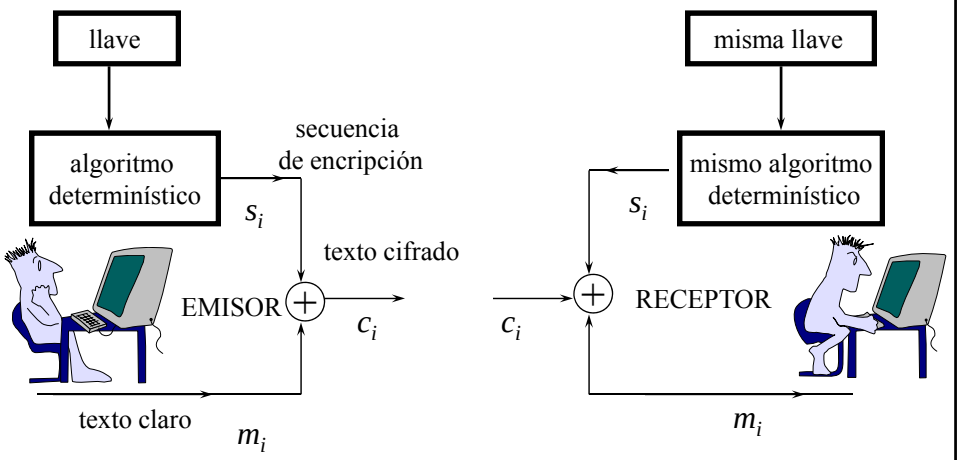


Lámina 8 Roberto Gómez C.

Ejemplo envío/recepción

Mensaje a enviar: HOLA
HOLA = 10010001000101100110010011111

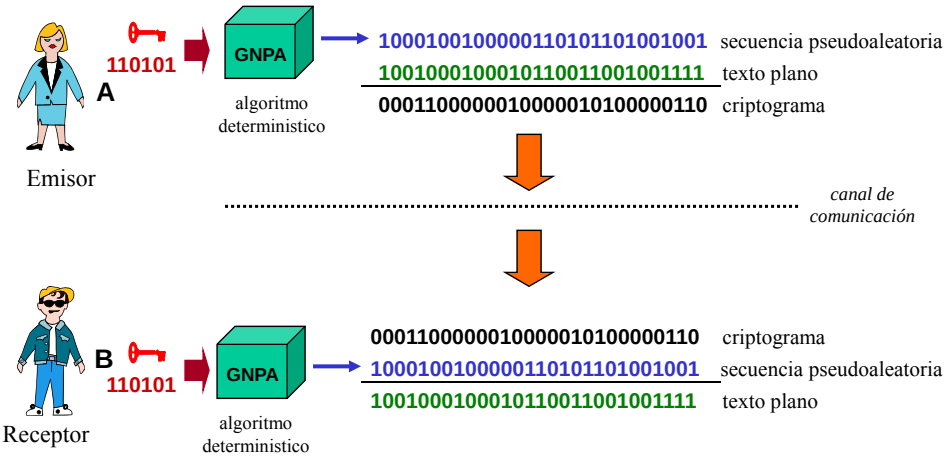


Lámina 9 Roberto Gómez C.

Generadores pseudoaleatorios

- Son algoritmos determinísticos que a partir de una llave corta (128 bits), conocida por emisor y receptor, generan simultáneamente una determinada secuencia de la longitud deseada.
- Estas secuencias nunca podrán ser autenticas secuencias aleatorias
- Son secuencias periódicas que deben ser lo más semejantes a una secuencia aleatoria

Lámina 10 Roberto Gómez C.

PRBG

- **Random Bit Generator**
  - dispositivo o algoritmo que genera una secuencia estadisticamente independientes y no relacionados (unbiased) digitos binarios
- **PseudoRandom Bit Generator(PRBG)**
  - algoritmo deterministico el cual, dado un secuencia binaria realmente aleatoria de longitud k, genera una secuencia binara de longitud $l >> k$, que parecer aleatoria
  - la entrada del PRBG es llamada la semilla, mientras que la salida del PRBG es llamada una secuencia pseudoaleatoria de bits (pseudorandom bit sequence)

Lámina 11 Roberto Gómez C.

Pruebas PRBG

- Se dice que un PRBG pasa todas las pruebas estadísticas de tiempo polinomial, si no existe un algoritmo que pueda distinguir correctamente entre la secuencia de salida de un generador y de una verdadera secuencia aleatoria con una probabilidad significativamente superior a ½.

Lámina 12 Roberto Gómez C.

NIST Statistical Test Suite

- Paquete de pruebas estadísticas desarrollado por la NIST para probar la aleatoreidad de (longitud arbitraria) secuencias binarias por generadores pseudo-aleatorios a nivel hardware o software.
- Pruebas se enfocan en una variedad de diferentes tipos de no aleatoreidad que pueden existir en una secuencia.
- Se trata de 16 pruebas

Lámina 13 Roberto Gómez C.

NIST Statistical Test Suite

1. Frequency (monobit)
2. Frequency within a block
3. Runs
4. Longest Run Of Ones in a block
5. Binary Matrix Rank
6. Discrete fourier transform (spectral)
7. Non-overlapping template matching
8. Overlapping template matching
9. Maurer's "Universal Statical"
10. Lempel-Ziv compression
11. Linear Complexity
12. Serial
13. Approximate Entropy
14. Cumulative Sums (Cusums)
15. Random excursions
16. Random excursions variant

http://csrc.nist.gov/rng/

Lámina 14 Roberto Gómez C.

Tipos de secuencias aleatorias

- Todos los generadores pseudoaleatorios producen secuencias finitas y periódicas de números empleando operaciones aritméticas y lógicas
- Lo único que se puede conseguir es que estas secuencias sean lo más largas posibles antes de comenzar a repetirse y que superen las pruebas estadísticas de aleatoriedad
- Entonces se distinguen:
  - Secuencias estadísticamente pseudoaleatorias
  - Secuencias criptográficamente pseudoaleatorias
  - Secuencias totalmente aleatorias

Lámina 15 Roberto Gómez C.

Secuencias estadísticamente aleatorias

- Secuencias que superan las pruebas estadísticas de aleatoriedad
  - ejemplo: generador congruencial lineal
- Problema:
  - son completamente predecibles
  - por lo que son poco útiles en Criptografía

Lámina 16 Roberto Gómez C.

Secuencias criptograficamente aleatorias

- Debe cumplir con la propiedad de ser impredicible.
- Debe ser computacionalmente intratable el problema de averiguar el siguiente número de secuencia, teniendo total conocimiento acerca de todos los números anteriores y del algoritmo de generación empleado.
- Existen varios generadores que cumplen lo anterior
  - no son suficientes debido a que necesitan de una semilla para inicializar el generados
  - si atacante logra saber la semilla, podría comprometer el sistema

Lámina 17 Roberto Gómez C.

Secuencias totalmente aleatorias

- No existe aleatoriedad cuando se habla de computadoras.
- Se puede decir que no existen en el universo sucesos cien por ciento aleatorios.
- Para efectos prácticos consideramos:
  - Secuencias aleatorias: una secuencia es totalmente aleatoria (o simplemente aleatoria) si no puede ser reproducida de manera fiable.

Lámina 18 Roberto Gómez C.

Objetivo generadores secuencias

- El objetivo no va a ser generar secuencias aleatorias puras, sino más bien secuencias impredecibles e irreproducibles.
- Entonces será suficiente con:
  - emplear un generador criptográficamente aleatorio
  - un mecanismo de recolección de bits aleatorios:
    - van a constituir la semilla totalmente aleatoria para alimentar el generador
    - un aspecto importante para la seguridad del sistema, es donde se va a almacenar la semilla empleada

Lámina 19 Roberto Gómez C.

Obteniendo bits aleatorios

- Existen valores obtenidos del hardware de la computadora que suelen proporcionar algunos bits de aleatoriedad.
- Lectura en un momento dado el valor de un reloj interno de alta precisión
  - primeras versiones Netscape
- uso de los número de serie de los componente físicos
  - estructura muy rígida y es posible adivinarlos
- Fuentes públicas de información, bits de un CD de audio,
  - los atacantes pueden disponer de ellas

Lámina 20 Roberto Gómez C.

Fuentes adecuadas bits aleatorios

- Tarjetas digitalizadores de sonido o vídeo
  - un dispositivo digitalizador de audio (o vídeo) sin ninguna entrada conectada, siempre que tenga ganancia suficiente, capta esencialmente ruido térmico, con una distribución aleatoria
- Unidades de disco
  - presentan pequeñas fluctuaciones en su velocidad de giro debido a turbulencias en el aire
  - si se dispone de un método para medir el tiempo de acceso de la unidad con suficiente precisión, se pueden obtener bits aleatorios de la calidad necesaria
- Distribuciones Linux ofrecen /dev/random

Lámina 21 Roberto Gómez C.

¿Existen otras opciones?

- Se puede efectuar la combinación de varias fuentes de información menos fiables.
  - leer el reloj del sistema, algún identificador de hardware, la fecha y hora locales, el estado de los registros de interrupciones del sistema etc.
- El número de bits que se obtendrán como resultado final del proceso ha de ser necesariamente menor que el número de bits recogido inicialmente
  - es necesario un mecanismo que realice lo anterior
  - se utilizan las denominadas funciones de mezcla fuertes

Lámina 22 Roberto Gómez C.

Funciones de mezcla fuertes

- Iniciales: FMF
- Función que toma dos o más fuentes de información y produce una salida en la que cada bit es una función compleja y no lineal de todos los bits de entrada
- Por termino medio, modificar un bit en la entrada debería alterar aproximadamente la mitad de los bits de salida
- Es posible emplear funciones criptográficas para construir este tipo de funciones
  - Algoritmos simétricos
  - Firmas digitales

Lámina 23 Roberto Gómez C.

Algoritmos simétricos y Funciones Mezcla Fuerte

- Algoritmo usa una llave de n bits y su entrada y salida son bloques de m bits
- Si se tienen n+m bits inicialmente, es posible codificar m bits usando como llave los n restantes
- Así se obtiene como salida un bloque de m bits con mejor aleatoriedad
- Por ejemplo, si se usa DES se puede reducir a 64 bits un bloque de 120 bits.

Lámina 24 Roberto Gómez C.

Huellas digitales y FMF

- Posible utilizar algoritmos de huellas digitales o compendio de mensajes.
- Algoritmos tipo hash que toman como entrada información de tamaño variable y dan como salida un string de n bits.
- Ejemplos:
  - MD5
  - SHA

Lámina 25 Roberto Gómez C.

Eliminando sesgos

- Que cierta información este "sesgada" significa que en la cadena hay mas ceros que unos o viceversa y esto no es deseable ya que para conseguir una semilla lo mas aleatoria posible es necesario que tengan igual probabilidad tanto el 0 como el 1.
- En la mayoría de los casos los bits obtenidos están sesgados: existen más unos que ceros o viceversa.
- Necesaria una fuente aleatoria no sesgada que presente igual probabilidad tanto para el 0 como para el 1
- Existen diferentes técnicas para ello:
  - Bits de paridad
  - Método de Von Neumman
  - Uso de funciones tipo hash (entropía)

Lámina 26 Roberto Gómez C.

Método de Von Neumann

- Este método consiste en ir examinando la secuencia de a 2 bits, eliminando los pares 00 y 11 e interpretando el 01 como 0 y el 10 como 1.
  - por lo que presenta la misma probabilidad tanto para 0 como para 1.
- Para comprobar la eficacia de este sistema basta con calcular que la probabilidad para 01 es igual que la de 10:
  - P(01)=P(10)=(0.5+d)(0.5-d),
  - siendo d el sesgo de la distribución inicial.
- El problema de este sistema es que no sabemos con anterioridad cuantos bits vamos a necesitar para obtener la cadena no sesgada.

Lámina 27 Roberto Gómez C.

XOR bits

- Aplicar una operación de XOR de varios bits consecutivos dependiendo de la paridad
- Si un bit aleatorio esta sesgado por 0 en un factor e, entonces la probabilidad de 0 puede ser escrita como
  - $P(0) = 0.5 + e$
- Un XOR con dos bits juntos, nos da
  - $P(0) = (0.5 + e)^2 + (0.5 - e)^2 = 0.5 + 2\,e^2$
- Aplicando el mismo cálculo el XOR de 4 bits proporciona:
  - $P(0) = 0.5 + 8e^4$
- Si se conoce el sesgo máximo aceptado en la aplicación, se puede calcular el número de bits a aplicar el XOR.

Lámina 28 Roberto Gómez C.

Uso de funciones hash

- Tomar varios eventos que contengan un poco de aleatoreidad
  - comandos ratón
  - numero de sector, hora y fecha, y buscar la latencia de cada operación del disco
  - posición actual del ratón
  - carga del CPU
  - tiempos de llegada de paquetes de red
  - entrada de un micrófono
- Tomar los valores obtenidos, "sumarlos" y aplicarles una función hash

Lámina 29 Roberto Gómez C.

Generadores pseudoaleatorios

- Generadores basados en congruencias lineales
- Registros de desplazamiento realimentados
  - Registros de desplazamiento realimentados no linealmente (NLFSR)
  - Registros de desplazamientos realimentados linealmente (LFSR)

Lámina 30 Roberto Gómez C.

Generador basado en congruencias lineales

- Basado en relaciones de recurrencia del tipo:
  - $X_{i+1} = (aX_i + b) \bmod m$
  - donde a,b, y m son parámetros generador (pueden usarse como llave secreta), y X_0= es la semilla
  - parámetros bien elegidos, X_{i+1}= no se repite en el intervalo [0, m-1]
    - i.e. periodo menor a m
  - Ejemplo valores iniciales por default:

$$a = 0x5DEECE66D$$

$$c = 0xB$$

$$m = 2^{48}$$

Lámina 31 Roberto Gómez C.

Características y ejemplo

- Propuesto por Lehmer (1949)
- Principal ventaja
  - numero de operaciones por por bit generado es pequeño
- Principal desventaja
  - las sucesiones que genera son predecibles
- Un primer ejemplo

a = 7 b = 0
m = 32 X_0 = 1
$X_{i+1} = 7\,X_i \bmod 32$
Secuencia: 7, 17, 23, 1, 7, 17, 23, …
Periodo 4

Lámina 32

Más ejemplos

- Segundo ejemplo

  a = 5 b = 0
  m = 32 $X_0 = 1$
  $X_{i+1} = 5 X_i \mod 32$
  Secuencia: 5, 25, 29, 17, 21, 9, 13, 1, …
  Periodo: 8

- Tercer ejemplo

  a = 5 b = 3
  m = 16 $X_0 = 1$
  $X_{i+1} = 5 X_i + 3 \mod 16$
  Secuencia: 1,8,11,10,5,12,15,14,9,0,3,2,13,4,7,6,1,8,...
  Periodo: 17

Lámina 33 Roberto Gómez C.

Comentarios generadores lineales

- La salida suele ser el número X_i/m, un número en coma flotante en el intervalo [0,1).
- Un buen generador lineal tendrá un periodo tan largo como sea posible (es decir m).
- Este máximo se alcanza si y sólo si se cumplen las tres siguientes condiciones [Knuth]
  1. b es primo con m
  2. a-1 es múltiplo de p para todo primo p que divida a m
  3. a-1 es múltiplo de 4 siempre que m sea múltiplo de 4
- Habitualmente se toma como m una potencia de 2 más de 1 de modo que se puede coger por ejemplo b=1 y a un número impar.

Lámina 34 Roberto Gómez C.

Comentarios generadores lineales

- No sólo este tipo de generador sino todos los de tipo polinómico congruencial, es decir:
  - $x = (a_n x^n + ... + a_1 x + a_0) \bmod m$
- son predecibles y por tanto inútiles para cualquier cifrado serio

Lámina 35 Roberto Gómez C.

Registros de desplazamiento realimentados

- FSR: Feedback Shift Registers
- Usados en criptología y teoría de códigos
- Basados en registros de corrimiento, que han servido a la criptología militar.
- Están constituidos de dos partes:
  - registro de corrimiento: secuencia de bits
  - función de retroalimentación
- Cuando se necesita un bit, todos los bits del registro de corrimiento son desplazados un bit a la derecha.
- El nuevo bit de la izquierda es calculado con la función de retroalimentación.

Lámina 36 Roberto Gómez C.

Esquema general FSR

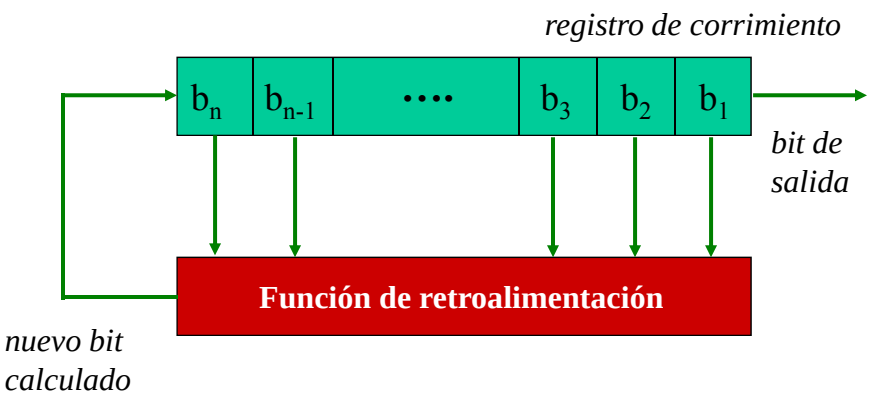


Lámina 37 Roberto Gómez C.

La función de retroalimentación

- De acuerdo a la función de retroalimentación podemos encontrar:
  - Registros de desplazamientos realimentados linealmente (LFSR)
  - Registros de desplazamiento realimentados no linealmente (NLFSR)
  - Registros de desplazamiento realimentados con carries (FCSR)
  - Combinaciones de los anteriores

Lámina 38 Roberto Gómez C.

Registros de desplazamientos realimentados linealmente

- LFSR: Linear Feedback Shift Register
- El más simple tipo de FSR es el linear feedback shift register LSFR.
- La función de retroalimentación es un XOR de algunos bits en el registro.
  - el conjunto de estos bits se le denomida tap register (secuencia de entrada)

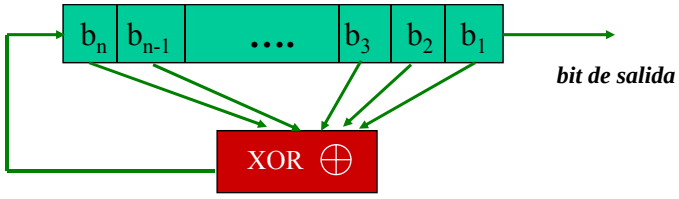
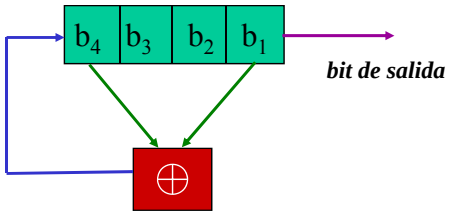


Lámina 39 Roberto Gómez C.

Ejemplo LSFR

- LFSR con bits de secuencia de entrada: b_4 b_1
- LFSR es inicializado con el valor 1111



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

- La secuencia de salida es: **1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 ...**

Lámina 40 Roberto Gómez C.

Periodicidad de los LFSR

- Un LSFR de *n* bits cuenta con 2^n-1 estados internos.
  - en teoria puede generar 2^n-1 secuencias pseudoaleatroias antes de repetirse.
- Solo LFSRs con una secuencia de entrada determinada se ciclarán a través de todos los 2^n-1 estados internos
  - estos son los periodos maximos de LFSR
  - la secuencia de salida se conoce como secuencia-m
- Para que un LSFR cuente con un periodo máximo:
  - polinomio formado por la secuencia de entrada más la constante 1 debe ser un polinomio primitivo *mod 2*
  - el grado del polinomio es la longitud del registro de corrimiento

Lámina 41 Roberto Gómez C.

Polinomios primitivos

- Un polinomio primitivo de grado *n*, es un polinomio irreductible que divide $x^{2\wedge(n-1)}+1$, pero no x^{d+1} para cualquier *d* que divide 2^n-1
- En general, no hay un metodo fácil para generar polinomios primitivos *mod 2* para un determinado grado.
  - la más fácil es elegir un polinomio al azar y probar si es primitivo
  - lo anterior es complicado
  - se cuenta con listas

Lámina 42 Roberto Gómez C.

Ejemplo polinomio primitivo

$$x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$$

- Primer número es la longitud del LFSR
- El último número siempre es cero y puede ignorarse
- Todos los números, a excepción del último especifican la secuencia de entrada
  - terminos de bajo grado en el polinomio corresponden a lugares cerca del lado izquierdo del registro

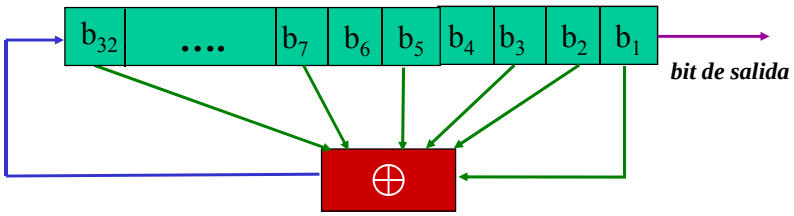


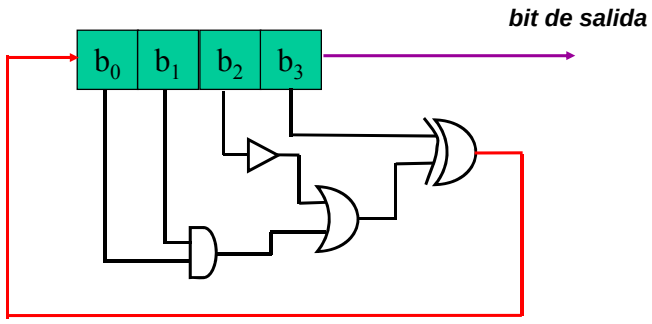
Lámina 43 Roberto Gómez C.

Registros de desplazamiento realimentados no linealmente

- NLFSR: Non Linear Feedback Shift Register
- Este tipo de generador presenta el problema de que no hay un metodo sistematico 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
  - criptográficamente hablando es muy peligroso
- Estos generadores son difíciles de implementar para una generación rápida de secuencias criptográficas.
- La función de realimentación puede ser cualquier cosa

Lámina 44 Roberto Gómez C.

Ejemplo NSFR: generador De Brujin



Secuencia entrada: 1101
Secuencia generada: 1011001010000111

Lámina 45 Roberto Gómez C.

Problemas de los NFSR

- Existen muchos sesgos, tales como más unos que ceros de los esperados en la secuencia de salida.
- El máximo periodo de secuencia puede ser mucho más bajo de lo esperado.
- El periodo de secuencia puede ser diferente para diferentes valores de inicio.
- La secuencia puede parecer aleatoria por un momento, pero después cae en un simple valor.

Lámina 46 Roberto Gómez C.

Otro ejemplo de NSFR

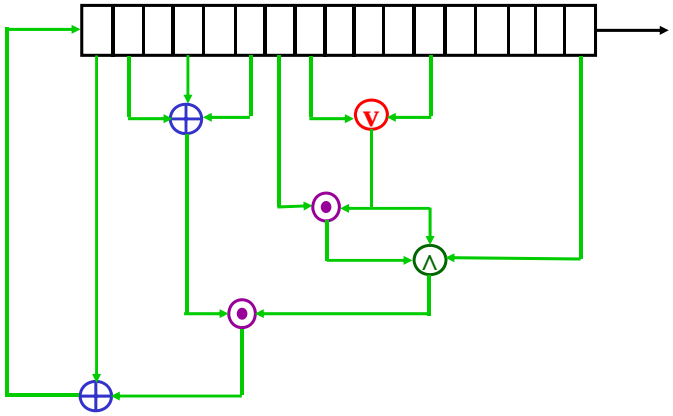


Lámina 47 Roberto Gómez C.

Registros de desplazamiento realimentados con carries

- FCSR: Feedback with Carry Shift Register
- Similares a los LFSR
  - ambos cuentan con un registro y una función de retroalimentación
- La diferencia es que el FCSR también cuenta con un registro de carry.
  - en lugar de realizar un xor de los bits de la secuencia de entrada, hay que sumar todos estos bits junto con el registro de carry
  - el resultado mod 2 se convierte en el nuevo bit
  - el resultado dividido por dos se convierte en el nuevo contenido del registro de carry

Lámina 48 Roberto Gómez C.

Feedback con Carry Shift Register

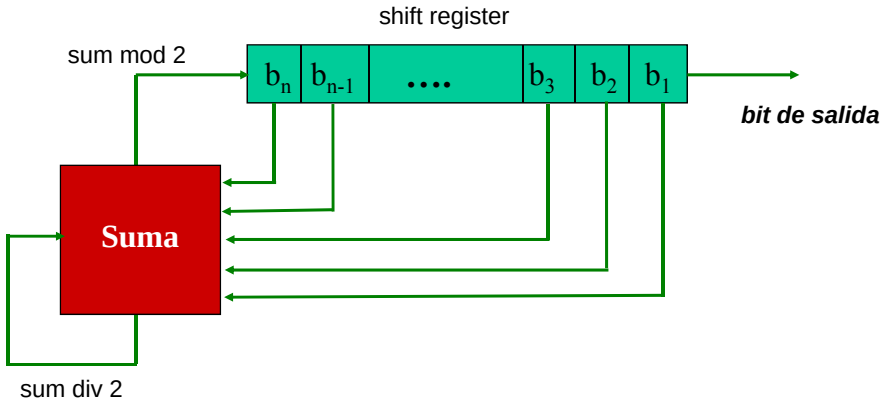
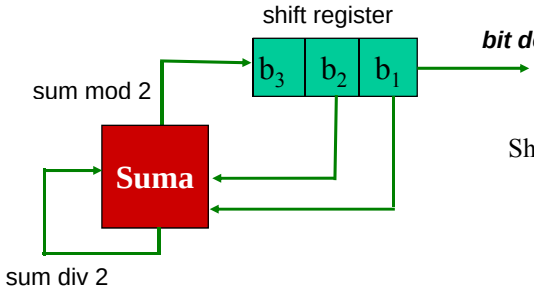


Lámina 49 Roberto Gómez C.

Ejemplo FCSR de 3 bits



- Bits secuencia entrada: b_2,b_1
- Secuencia inicial: 001
- Valor carry inicial: 0

Shift Register	Carry Register
001	0
100	0
010	0
101	0
110	0
111	0
011	1
101	1
010	1
001	1
000	1
100	0

Lámina 50 6391-48643 Roberto Gómez C.

Contruyendo criptosistemas de flujo con LFSRs

- Se toman uno a más LSFRs
  - de longitud diferente y con polinomios de realimentación diferentes
- La llave es el estado inicial de los LFSRs
- Cada vez que se requiera un bit, se realiza un shift sobre los LFSRs
- Las salidas de los LFSRs se introducen a una función de combinaciópn y todo el generador se conoce como generador de combinación
  - si la salida es una función de un simple LFSR, el generador se conoce como generador de filtro

Lámina 51 Roberto Gómez C.

Combinando generadores

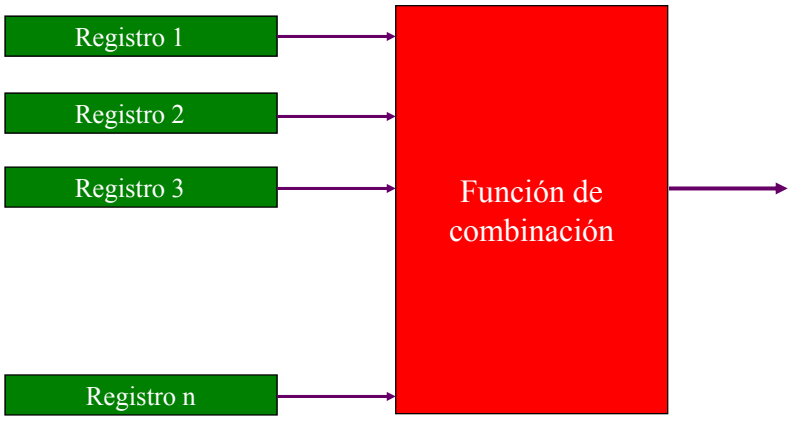


Lámina 52 Roberto Gómez C.

Complicando el generador

- Generadores controlados por reloj
  - Algunos generadores tienen los diferentes LFSRs activados a diferentes periodos de reloj.
  - A veces el activar un LFSR depende de la salida de otro.
- La mayoría de los criptosistemas de flujo son implementados en hardware,

Lámina 53 Roberto Gómez C.

Criptosistemas flujo basados en LFSR

- Generador Geffe
- Generador Jennings
- Generador Stop and Go Beth-Piper
- Generador Stop and Go Bilateral
- Generador Threshold
- Generadores Self-Decimated
- Generador Multispeed Inner-Product
- Generadores Summation
- DNRSG (Dynamic Random Sequence Generator)
- Generador en cascada de Gollman
- Generador Shrinking (shrink = contraer)
- Generador self-shrinking

Lámina 54 Roberto Gómez C.

Ejemplo Generador Geffe

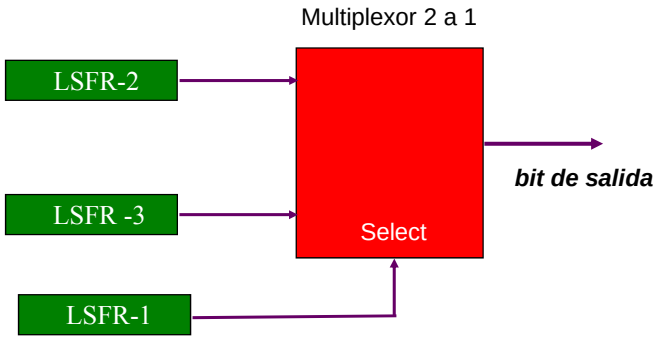
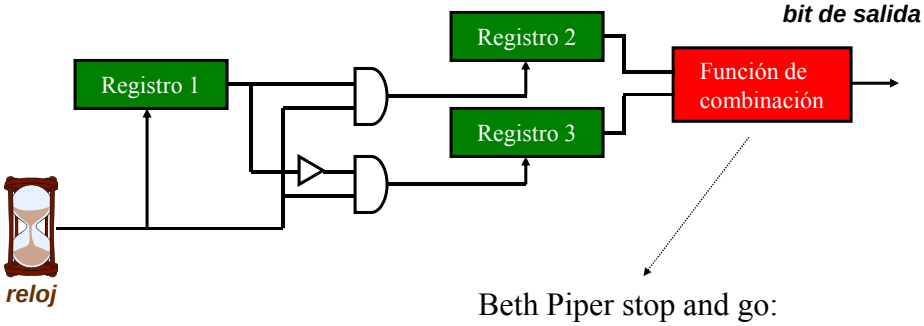


Lámina 55 Roberto Gómez C.

Ejemplo generador stop and go



Beth Piper stop and go:
funcion combinación: xor

Lámina 56 Roberto Gómez C.

Variantes generadores stop and go

- Generadores de tipo stop and go con FCSRs en lugar de LSFRs
- Adicionalmente la operación XOR puede ser reemplazado con una suma adicional con carry
  - Generador FCSR Stop and Go. Registro-1, registro-2 y registro-3 son FCSRs. La función de combinación es un xor
  - Generador FCSR/LFSR Stop and Go. Registro-1 es un FCSR y registros 2 y 3 son LSFRs. La operación de combinación es suma con carry.
  - Generador LFSR/FCSR Stop and Go. Registro-1 es un LFSR y registro-2 y 3 son FCSRs. La función de combinación es un xor.

Lámina 57 Roberto Gómez C.

Generador Threshold

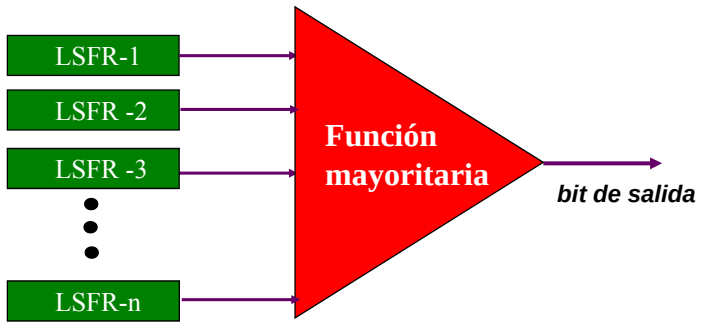


Lámina 58 Roberto Gómez C.

Cascada de Gollman

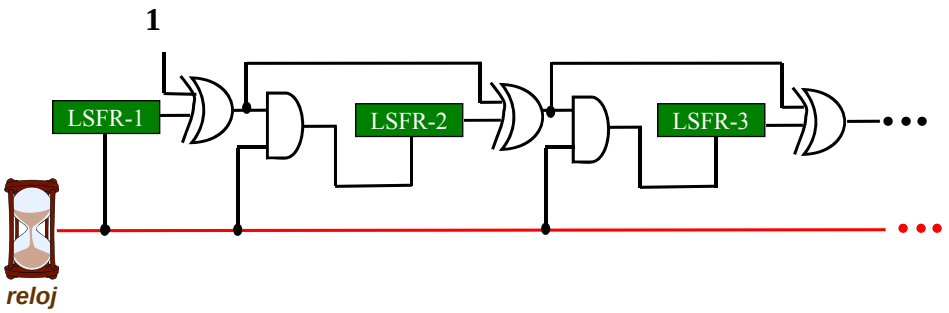


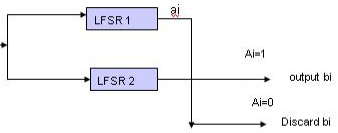
Lámina 59 Roberto Gómez C.

Generadores shrinking

- Shrink = encogimiento
- Publicado en 1993 por Don Coppersmith, Hugo Krawczyk y Yishay Mansour
- Se usan dos fuentes para crear una tercera fuente de bits pseudoaleatorios, de mejor calidad que las dos primeras
  - la secuencia de salida es una subsecuencia de la primera fuente
  - los elementos se eligen de acuerdo a las posiciones de bits 1 en la segunda secuencia
- La secuencia resultante es una versión “shrunken” de la primera

Lámina 60 Roberto Gómez C.

Calculando la secuencia salida en shrinking



- Primera secuencia A: a_0, a_1, a_2 ..
- Segunda secuencia S: s_0, s_1, s_2 …
- Tercera secuencia Z: z_0, z_1, z_2 …
- La tercera secuencia incluye los bits a_i para lo cuales si es 1, el resto de los bits de la primera secuencia son descartados
- Tanto A y S están bajo un reloj
  - si $S_i = 1$ entonces el bit de A es la salida del generador
  - sino el bit de la secuencia A es descartado y el reloj registra de nuevo.

Lámina 61 Roberto Gómez C.

Tipos generadores shrinking

- Existen cuatro tipos de generadores básicos que usan FCSRs:
  - Generador Shrinking FCSR
    - generador shrinking con FCSR en lugar de LSFR
  - Generador Shrinking FCSR/LFSR
    - generador shrinking con un LSFR “shrinking” un FCSR
  - Generador Shrinking LFSR/FCSR
    - generador shrinking con un FCSR “shrinking” un LFSR
  - Generador Self-Shrinking FCSR
    - generador self-shrinking con un FCSR en lugar de un LFSR

Lámina 62 Roberto Gómez C.

El algoritmo de cifra A5

- A5/1 y A5/2 propuestos los 80s para GSM (mandatorio para GSM, no así la autenticación)
- Mantenido en secreto hasta los 90s
- Ingeniería inversa y roto
- Recientemente se propuso A5/3 como un reemplazo
- Diferencias versiones:
  - A5/1: para uso local
  - A5/2: para exportación a ciertos países
- Algoritmos relacionados
  - A3: autenticación
  - A8: algoritmo de generación de llave de encripción

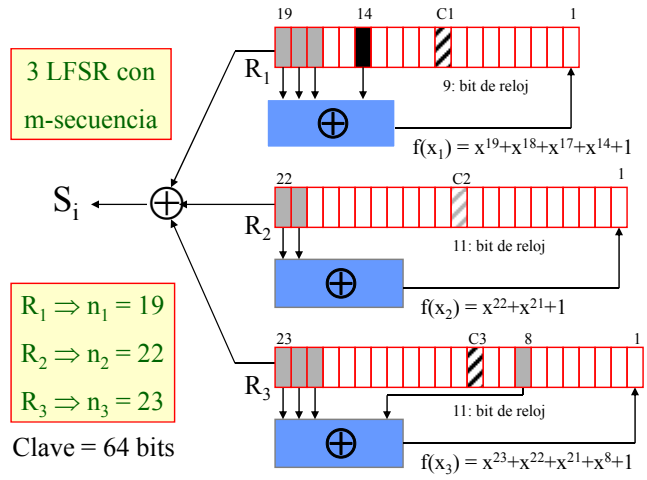
Lámina 63 Roberto Gómez C.

Descripción del algoritmo

- Consta de tres LFSRs controlador por un reloj
  - longitudes: 19, 22 y 23
- El control del reloj consiste en una función tipo "threshold", de los bits de en-medio de cada uno de los registros de corrimiento.
- La salida es el XOR de los tres LFSRs
- La suma lógica de los tres registros es de 64 bits
  - llave 64 bits es usada para inicializar los registros

Lámina 64 Roberto Gómez C.

Esquema del algoritmo de cifra A5/1



Una función mayoría entre C1, C2 y C3 hace que sólo los registros en los que coincide el bit con ese valor produzcan desplazamiento. En cada paso habrá dos o tres registros en movimiento.

Lámina 65 Roberto Gómez C.

Otros algoritmos de encripción para telefonía celular

	Confidencialidad	Autenticación	Llaves
US Análoga	Ninguna	Ninguna	Ninguna
US Digital	XOR mask & CMEA (ORYX)	CAVE	CAVE
GSM	A5/0, A5/2, or A5/1 (pronto: A5/3)	COMP128 (COMP128-2, 3DES-CBC-MAC)	COMP128 (igual)

Lámina 66 Roberto Gómez C.

El generador RC4

- RC4 es un criptograma de llave de tamaño variable desarrollado en 1987 por Ron Rivest para la RSA.
- Durante siete años su implementación fue privada.
- En septiembre 1994, alguien lo puso en la lista de correo Cypherpunks anonimamente.
- Lectores con copias legales de RC4 confirmaron su compatibilidad.
- RSA intento *poner de nuevo al genio en la botella*, pero fue muy tarde.

Lámina 67 Roberto Gómez C.

El mensaje

Newsgroups: sci.crypt,alt.security,comp.security.misc,alt.privacy
From: sterndark@netcom.com (David Sterndark)
Subject: RC4 Algorithm revealed.
Message-ID: <sternCvKL4B.Hyy@netcom.com>
Sender: sterndark@netcom.com
Organization: NETCOM On-line Communication Services (408 261-4700 guest)
Date: Wed, 14 Sep 1994 06:35:31 GMT

I am shocked, shocked, I tell you, shocked, to discover that the cypherpunks have illegaly and criminally revealed a crucial RSA trade secret and harmed the security of America by reverse engineering the RC4 algorithm and publishing it to the world.

On Saturday morning an anonymous cypherpunk wrote:

SUBJECT: RC4 Source Code

I've tested this. It is compatible with the RC4 object module that comes in the various RSA toolkits.

```
/* rc4.h */
typedef struct rc4_key
{
unsigned char state[256];
unsigned char x;
unsigned char y;
} rc4_key;
```

Lámina 68 Roberto Gómez C.

Características RC4

- La llaves está limitada a 40 bits debido a problemas de exportación
  - tiene capacidad de usar llaves entre 1 y 2048 bits
- Trabaja en dos fases:
  - la generación de la llave
  - la encripción

Lámina 69 Roberto Gómez C.

Funcionamiento del RC4

- RC4 trabaja en OFB (Output FeedBack mode).
- El flujo (stream) es independiente del texto en claro.
- Cuenta con una S-box de 8x8: S_0, S_1, S_2, … S_{255}, inicializados S_0=1, S_1=1, S_2=2, … S_{255}=255.
- Las entradas son una permutación de los números 0 a 255.
- La permutación es una función de la llave de tamaño variable.
- Cuenta con dos contadores: i e j, inicializados en cero.

Lámina 70 Roberto Gómez C.

Funcionamiento RC4

- La llave varia de 1 a 256 bytes para inicializar un tabla de 256 bytes.
  - generalmente limitada a 40 bits, debido a restricciones de exportación pero a veces es usado como una llave de 128 bits.
- Esta tabla se usa para generar un flujo pseudoaleatorio de bits el cual es XORed con el texto en claro para producir el criptograma.
- Cada elemento de la tabla es intercambiado al menos una vez.
- Tiene la capacidad de usar llaves entre 1 y 2048 bits.
- RC4 es usado en muchos paquetes comerciales como Lotus Notes, Oracle Secure SQL, WEP, etc

Lámina 71 Roberto Gómez C.

Pseudo-código RC4

Inicialización

```
S[0..255] = 0,1,…,255
K[0..255] = Key,Key,Key,…
for i = 0 to 255
        j = (j + S[i] + K[i]) mod 256
        swap S[i] and S[j]
```

Iteración (produciendo un byte al azar)

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap S[i] and S[j]
t = (S[i] + S[j]) mod 256
Output S[t]
```

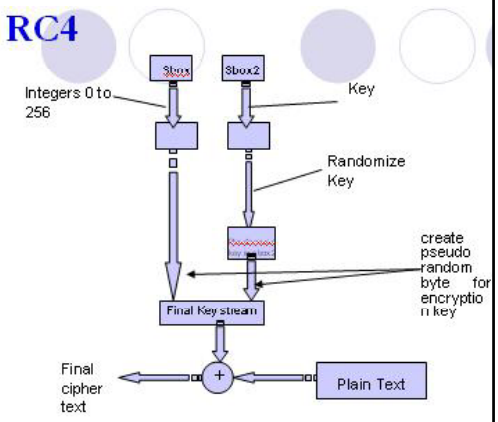


Lámina 72 Roberto Gómez C.

Fortalezas

- La dificultad de conocer donde se encuentra cualquier valor dentro de la tabla.
- La dificultad de conocer que elemento de la tabla es usado para seleccionar cada valor de la secuencia.
- Una llave RC4 es usada solo una vez
- Encripción es 10 veces más rápido que DES

Lámina 73 Roberto Gómez C.

Ejemplo uso RC4: WEP (cifrado)

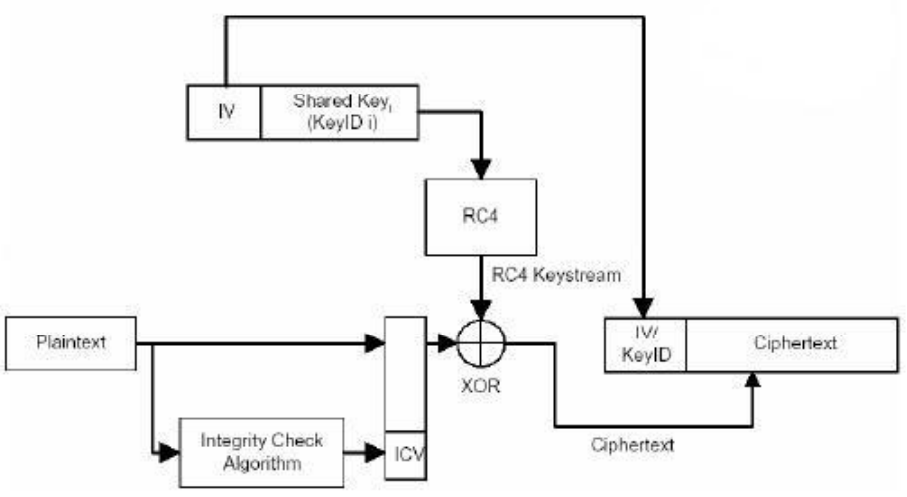


Lámina 74 Roberto Gómez C.

Ejemplo uso RC4: WEP (descifrado)

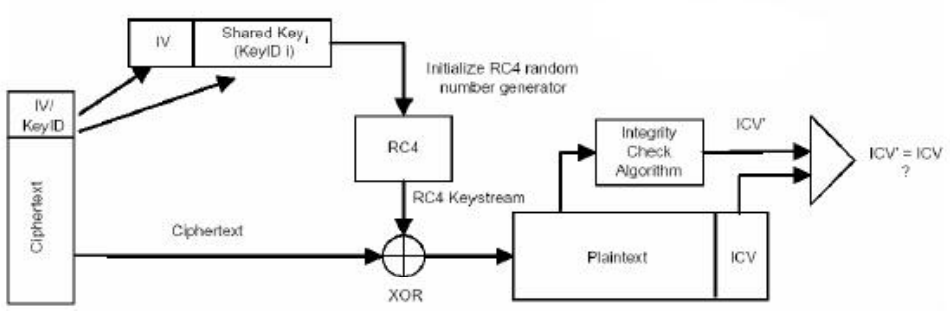


Lámina 75 Roberto Gómez C.

Criptosistema SEAL

- Software optimized Encryption Algorithm
- Diseñado por IBM por Phil Rogaway y Don Coppersmith en 1993
- Algoritmo optimizado para procesadores de 32 bits.
- Para correr requiere ocho registros de 32 bits y un caché de pocos kilobytes
- A través de una aparentemente lenta operación, SEAL preprocesa la operación de llaves en un conjunto de tablas.
  - después estas tablas son usadas para aumentar la velocidad de encripción y decripción

Lámina 76 Roberto Gómez C.

Pseudo-random function family

- SEAL no es un criptosistema de flujo tradicional
  - se trata de una familia de funciones pseudo-aleatorias
- Dada una llave de 160 bits, k y un número n de 32 bits
  - SEAL extiende n en un string de L bits k(n)
  - L puede tomar cualquier valor menor a 64 kilobytes
- Se supone que SEAL se aprovecha de la propiedad de que k es seleccionado al azar
  - entonces k(n) puede ser computacionalmente indistinguible de una función aleatorio de L bits de n

Lámina 77 Roberto Gómez C.

Ventaja SEAL

- Útil donde en aplicaciones donde criptosistemas de flujo tradicionales no lo son.
- La mayor parte criptosistemas flujo pueden generar una secuencia de bits en un solo sentido
  - conociendo la llave y una posición i, la única forma de determinar el $i^{ésimo}$ bit generado es generando todos los bits hasta el $i^{ésimo}$ bit buscado
- Con un criptositema como SEAL
  - se puede acceder a cualquier bit de la llave generada
- Uso: asegurar un disco duro
  - encriptar por, y cada, sectores de 512 bytes
  - con SEAL es posible encriptar el contenido del sector n, con un XOR del contenido con la la llave k(n)

Lámina 78 Roberto Gómez C.

Descripción SEAL

- Algoritmo controlado por tres tablas derivadas de las llaves: R, S y T
  - preprocesamiento mapea la llave k, a estos valores usando un procedimiento basado en SHA
- También usa cuatro registros de 32 bits: A,B,C y D
  - valores iniciales son determinados por n y las tablas R y T
  - estos registros son modificados a través de varias iteraciones, cada uno involucra 8 vueltas
  - en cada iteración 9 bits de un registro son usados para indexar la tabla T
  - contenido tabla se da un xor con el contenido de un segundo registro A,B,C y D.
  - primer registro es recorrido (shifted) 9 posiciones

Lámina 79 Roberto Gómez C.

Continuación descripción

- En algunas iteraciones el segundo registro es modificado a través de un xor con el ahora recorrido primer registro
- Después de 8 iteraciones de esto, A,B,C y D son añadidos al stream de la llave
- La iteración se completa añadiendo a A y a C valores adicionales dependiendo en n, n1, n2, n3 y n4
  - exactamente cual, depende en la paridad del numero de iteración

Lámina 80 Roberto Gómez C.

La inicialización

procedure Initialize$(n, \ell,\ A, B, C, D,\ n_1, n_2, n_3, n_4)$

$A \leftarrow n \oplus R[4\ell]$;
$B \leftarrow (n \ggg 8) \oplus R[4\ell+1]$;
$C \leftarrow (n \ggg 16) \oplus R[4\ell+2]$;
$D \leftarrow (n \ggg 24) \oplus R[4\ell+3]$;

for $j \leftarrow 1$ **to** 2 **do**
  $P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \ggg 9$;
  $P \leftarrow B$ & 0x7fc; $C \leftarrow C + T[P/4]$; $B \leftarrow B \ggg 9$;
  $P \leftarrow C$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \ggg 9$;
  $P \leftarrow D$ & 0x7fc; $A \leftarrow A + T[P/4]$; $D \leftarrow D \ggg 9$;

$(n_1,\ n_2,\ n_3,\ n_4) \leftarrow (D,\ B,\ A,\ C)$;

$P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \ggg 9$;
$P \leftarrow B$ & 0x7fc; $C \leftarrow C + T[P/4]$; $B \leftarrow B \ggg 9$;
$P \leftarrow C$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \ggg 9$;
$P \leftarrow D$ & 0x7fc; $A \leftarrow A + T[P/4]$; $D \leftarrow D \ggg 9$;

Lámina 81 Roberto Gómez C.

La función de encripción

function SEAL(a, n, L)

$y = \lambda$;
for $\ell \leftarrow 0$ **to** ∞ **do**
  Initialize$(n, \ell,\ A, B, C, D,\ n_1, n_2, n_3, n_4)$;
  for $i \leftarrow 1$ **to** 64 **do**
1 $P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \ggg 9$; $B \leftarrow B \oplus A$;
2 $Q \leftarrow B$ & 0x7fc; $C \leftarrow C \oplus T[Q/4]$; $B \leftarrow B \ggg 9$; $C \leftarrow C + B$;
3 $P \leftarrow (P + C)$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \ggg 9$; $D \leftarrow D \oplus C$;
4 $Q \leftarrow (Q + D)$ & 0x7fc; $A \leftarrow A \oplus T[Q/4]$; $D \leftarrow D \ggg 9$; $A \leftarrow A + D$;
5 $P \leftarrow (P + A)$ & 0x7fc; $B \leftarrow B \oplus T[P/4]$; $A \leftarrow A \ggg 9$;
6 $Q \leftarrow (Q + B)$ & 0x7fc; $C \leftarrow C + T[Q/4]$; $B \leftarrow B \ggg 9$;
7 $P \leftarrow (P + C)$ & 0x7fc; $D \leftarrow D \oplus T[P/4]$; $C \leftarrow C \ggg 9$;
8 $Q \leftarrow (Q + D)$ & 0x7fc; $A \leftarrow A + T[Q/4]$; $D \leftarrow D \ggg 9$;
9 $y \leftarrow y \parallel B + S[4i-4] \parallel C \oplus S[4i-3] \parallel D + S[4i-2] \parallel A \oplus S[4i-1]$;
10 **if** $|y| \geq L$ **then return** $(y_0 y_1 \ldots y_{L-1})$;
11 **if** $odd(i)$ **then** $(A,\ B,\ C,\ D) \leftarrow (A + n_1,\ B + n_2,\ C \oplus n_1,\ D \oplus n_2)$
      else $(A,\ B,\ C,\ D) \leftarrow (A + n_3,\ B + n_4,\ C \oplus n_3,\ D \oplus n_4)$;

Lámina 82 Roberto Gómez C.

Diagrama criptosistema SEAL

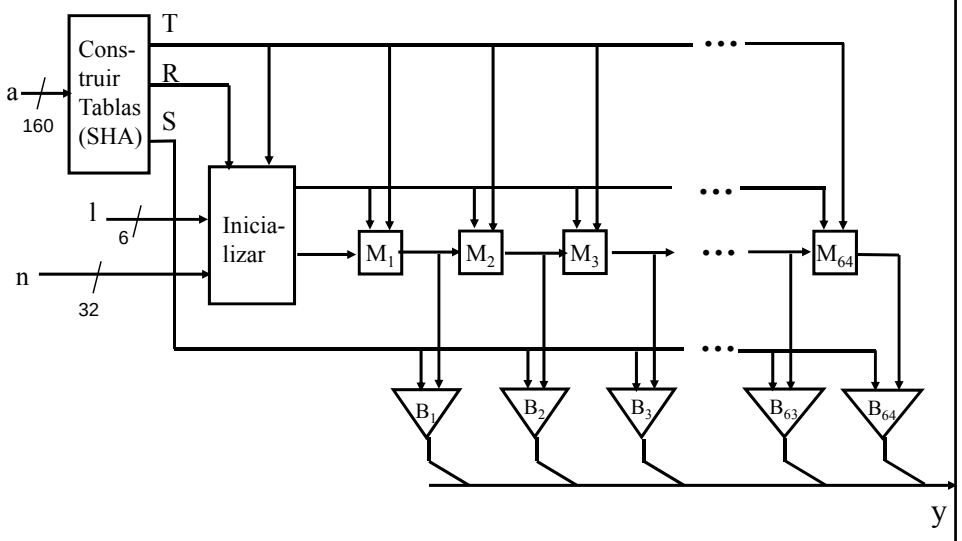


Lámina 83 Roberto Gómez C.

Ejemplo SEAL (1)

Llave a de 160 bits

```
67452301 efcdab89 98badcfe 10325476 c3d2e1f0,
```

$n = 0$x013577af, and $L = 32768$ bits. Table R consists of words $R[0], R[1], \ldots, R[15]$:

```
5021758d ce577c11 fa5bd5dd 366d1b93 182cff72 ac06d7c6
2683ead8 fabe3573 82a10c96 48c483bd ca92285c 71fe84c0
bd76b700 6fdcc20c 8dada151 4506dd64
```

The table T consists of words $T[0], T[1], \ldots, R[511]$:

```
92b404e5 56588ced 6c1acd4e bf053f68 09f73a93 cd5f176a
b863f14e 2b014a2f 4407e646 38665610 222d2f91 4d941a21
........ ........ ........ ........ ........ ........
3af3a4bf 021e4080 2a677d95 405c7db0 338e4b1e 19ccf158
```

Lámina 84 Roberto Gómez C.

Ejemplo SEAL (2)

The table S consists of words $S[0], S[1], \ldots, S[255]$:

```
907c1e3d ce71ef0a 48f559ef 2b7ab8bc 4557f4b8 033e9b05
4fde0efa 1a845f94 38512c3b d4b44591 53765dce 469efa02
........ ........ ........ ........ ........ ........
bd7dea87 fd036d87 53aa3013 ec60e282 1eaef8f9 0b5a0949
```

The output y of Algorithm 6.68 consists of 1024 words $y[0], y[1], \ldots, y[1023]$:

```
37a00595 9b84c49c a4be1e05 0673530f 0ac8389d c5878ec8
da6666d0 6da71328 1419bdf2 d258bebb b6a42a4d 8a311a72
........ ........ ........ ........ ........ ........
547dfde9 668d50b5 ba9e2567 413403c5 43120b5a ecf9d062
```

The XOR of the 1024 words of y is 0x098045fc.

Lámina 85 Roberto Gómez C.

Generador X9.17

- Propuesto por el Instituto Nacional de Estandares Norteamericano (NIST)
- A partir de una semilla inicial S_0 de 64 bits permite obtener secuencias de valores tambien de 64 bits
- El algoritmo es el siguiente:

$$g_n = DES(k, DES(k,t) \oplus s_n)$$
$$s_n = DES(k, DES(k,t) \oplus g_n)$$

  - k: llave aleatoria para la generación de cada secuencia
  - t: tiempo en el que cada valor es generado

Lámina 86 Roberto Gómez C.

Algoritmo X9.17

- Financial Institution Key Management (Wholesale) standard.
- Publicado por primera vez en 1985, reafirmado sin modificación en 1991, y actualizado en 1995.
- Este generador esta pensado como mecanismo para generar claves DES y vectores de inicialización, usando triple- DES como primitiva (podría usarse otro algoritmo de cifrado de bloques).
- También es ampliamente usado en banca y otras aplicaciones.
  - los generadores PRNG del PGP utilizan la especificación ANSI X9.17 que utiliza IDEA en vez de TripleDES

Lámina 87 Roberto Gómez C.

Pseudocódigo del algoritmo

- Entrada:
  - una semilla aleatoria y secreta de 64 bits
  - un entero m
  - una llave K del algoritmo 3-DES
- Salida:
  - m strings de 64 bits pseudo aleatorios: y1, y2, ... ym
- Algoritmo

1.) $q = E(K, Date\_Time)$

2.) For i from 1 to m do

2.1 $x_i = E(K, (q \oplus s)$

2.2 $s = E(K, (x_i \oplus q)$

3.) Return$(x_1, x_2, \ldots, x_m)$

Lámina 88 Roberto Gómez C.

Esquema X9.17

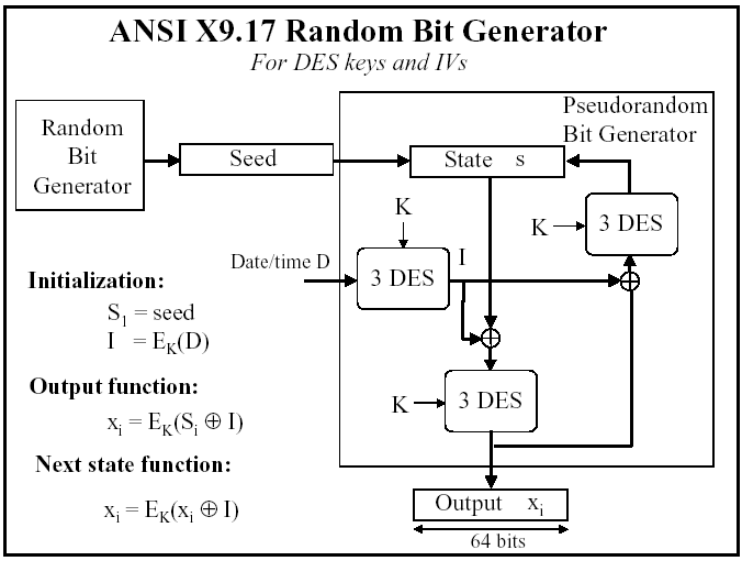


Lámina 89 Roberto Gómez C.

Generador Blum Blum Shub

- Recibe nombre de sus autores
- Algoritmo que más pruebas de resistencia ha superado, con la ventaja adicional de su gran simplicidad.
  - computacionalmente más costoso que el algoritmo X9.17
- Consiste en elegir dos números primos grandes: p y q que cumplan con:

$$p \equiv 3 (\mathrm{mod}\, 4)$$
$$q \equiv 3 (\mathrm{mod}\, 4)$$

- Sea entonces $n=pq$
- Elegir un número x aleatorio primo relativo con n (semilla inicial)

Lámina 90 Roberto Gómez C.

Continuación Blum Blum Shub

- Al contrario que *x* que debe ser mantenido en secreto, n puede ser público.
- Calculamos los valores s_i *(i = 1..k)* de la serie de la siguiente forma:

$$s_0 = (x^2)(\text{mod } n)$$
$$s_{i+1} = (s_i^2)(\text{mod } n)$$

- z_i es el bit menos significativo de s_i
- Salida: secuencia pseudoaleatoria de bits z_1, z_2, ... z_k de longitud *k*

Lámina 91 Roberto Gómez C.

Ejemplo Blum Blum Shub

- Valores iniciales
  - n=p·q=7·19=133
  - S=100
- Las salidas temporales son:
  - $X_0=100^2(\text{mod } 133)=25$ **(11001)**
  - $X_1=25^2(\text{mod } 133)=93$ **(1011101)**
  - $X_2=93^2(\text{mod } 133)=42$ **(101010)**
  - $X_3=42^2(\text{mod } 133)=16$ **(10000)**
  - $X_4=16^2(\text{mod } 133)=123$ **(1111011)**
- La secuencia es: **1,1,0,0,1,...**

Lámina 92 Roberto Gómez C.

Generador RSA

- Se trata de un generador de bits pseudoaleatorio
- Se dice que es es un generador de bits pseudoaleatorio criptograficamente seguro bajo la condición que la factorización de un numero n grande compuesto de dos numeros primos grandes, p y q, es intratable en un tiempo polinomial.
- El algoritmo proporciona como salida una secuencia pseudoaleatoria de bits z_1, z_2, ... z_k de longitud k

Lámina 93 Roberto Gómez C.

El algoritmo

1. Procedimiento de inicializacion

   Generar dos numeros primos secreto : p, q

   Calcular : $n = p \times q$ y $\varphi = (p-1) \times (q-1)$

   Seleccionar entero aleatorio tal que $1 < e < \varphi$ y $\text{mcd}(e, \varphi) = 1$

2. Seleccionar un entero y_0(semilla) tal que $y_0 \in [1, n]$
3. Calculo de la serie :

   For i from 1 to k do

   $y_i = (y_{i-1})^e \bmod n$

   z_i = bit menos significativo de y_i

Lámina 94 Roberto Gómez C.

Ejemplo secuencia generada

- Considerar valores siguientes:
  - n = 91261 = 263 x 347
  - e = 1547
  - y_0 = 75364
- Primeros 20 bits:
  - 10000111011110011000

i	yi	zi
0	75634	
1	31483	1
2	31238	0
3	51968	0
4	39796	0
5	28716	0
6	14089	1
7	5923	1
8	44891	1
9	62284	0
10	11889	1
11	43467	1
12	7125	1

Lámina 95 Roberto Gómez C.

Velocidad de algunos criptosistemas de flujo

Algoritmo	Velocidad encripción (kilobyets/seg)
A5	5
RC4	164
SEAL	381

Procesador prueba: 486SX a 33MHz

Lámina 96 Roberto Gómez C.

Otros criptosistemas de flujo

- Hughes XPD/KPD
- Nanoteq
- Rambutan
- Generadores aditivos
- Gifford
- Algoritmo M
- WEAK
- PKZIP

Lámina 97 Roberto Gómez C.

Resultado pruebas NIST

Statistical Test	G-SHA-1	G-DES	X9.17	BBS	MS	QCG II
Frequency	99.67%	99.00%	100.00%	99.00%	99.33%	99.00%
Block Frequency	99.33%	99.33%	98.67%	100.00%	99.00%	97.67%
Cusum Forward	99.00%	98.00%	97.67%	97.67%	98.00%	98.00%
Cusum Reverse	99.33%	97.67%	98.33%	98.33%	98.00%	98.33%
Runs	98.67%	98.33%	99.67%	99.33%	99.33%	99.67%
Longest Run Of Ones	98.67%	99.67%	99.67%	99.33%	99.67%	99.33%
Marsaglia's Rank	98.67%	98.67%	97.67%	100.00%	97.00%	99.33%
Spectral (DFT)	99.67%	99.33%	99.67%	99.33%	99.33%	100.00%
Nonoverlapping Template	99.00%	99.33%	99.00%	98.33%	99.00%	99.33%
Overlapping Template	98.33%	99.33%	98.00%	99.00%	99.67%	99.00%
Maurer's Universal	98.67%	98.67%	98.67%	99.00%	98.00%	99.00%
Approximate Entropy	99.00%	98.33%	99.33%	98.67%	100.00%	99.00%
Random Excursions	99.48%	97.37%	99.48%	100.00%	97.50%	98.91%
Lempel-Ziv Complexity	99.33%	99.67%	99.67%	99.33%	98.33%	99.67%
Linear Complexity	98.67%	98.33%	99.33%	98.67%	99.00%	99.00%

Lámina 98 Roberto Gómez C.

Una última tabla comparativa

Stream Cipher	Creation Date	Speed (cycles per byte)	(bits) Effective Key-Length	(bits) Initialization vector	(bits) Internal State	Attack Best Known	Attack Computational Complexity
A5/1	1989	Voice (W_{phone})	54	114	64	Active KPA OR KPA Time-Memory Tradeoff	~2 seconds OR $2^{39.91}$
A5/2	1989	Voice (W_{phone})	54	114	64?	Active	4.6 milliseconds
FISH	1993	Quite Fast (W_{SOBER})	Huge	?	?	Known-plaintext attack	2^{11}
Grain	Pre-2004	Fast	80	64	160	Key-Derivation	2^{43}
HC-256	Pre-2004	4 (W_{64})	256	256	65536	?	?
ISAAC	1996	2.375 (W_{bASE}) - 4.6875 (W_{32-bit})	8-8288 usually 40-256	N/A	8288	(2006) First-round Weak-internal-State-Derivation	4.67×10^{1240} (2001)
MUGI	1998-2002	?	128	128	1216	N/A (2002)	$\sim 2^{82}$
PANAMA	1998	2	256	1280	1216?	Hash Collisions (2001)	2^{82}
Phelix	Pre-2004	up to 8 (W_{ASE})	256 + a 128-bit Nonce	1280	?	Differential (2006)	2^{97}
Pike	1994	0.9 x FISH (W_{SOBER})	Huge	?	?	N/A (2004)	N/A (2004)
Py	Pre-2004	2.6	0-2048? usually 40-256?	64	8320	Cryptanalytic Theory (2006)	2^{75}
Rabbit	2003-Feb	3.7(W_{PII})-9.7(W_{ARLET})	128	64	512	N/A (2006)	N/A (2006)
RC4	1987	Impressive	8-2048 usually 40-256	8	2064	Shamir Initial-Bytes Key-Derivation OR KPA	2^{13} OR 2^{30}
Salsa20	Pre-2004	4.24 (W_{P4}) - 11.84 (W_{P4})	128 + a 64-bit Nonce	512	512 + 384 (key+IV+index)	Differential (2005)	N/A (2005)
Scream	2002	4 - 5 (W_{SAFER})	128 + a 128-bit Nonce	32?	64-bit round function	?	?
SEAL	1997	Very Fast (W_{32-bit})	?	32?	?	?	?
SNOW	Pre-2003	Very Good (W_{32-bit})	128 OR 256	32	?	?	?
SOBER-128	2003	?	up to 128	?	?	Message Forge	2^{-6}
SOSEMANUK	Pre-2004	Very Good (W_{32-bit})	128	128	?	?	?
Trivium	Pre-2004	4 (W_{ASE}) - 8 (W_{LO})	80	80	288	Brute force attack (2006)	2^{128}
Turing	2000-2003	5.5 (W_{ASE})	?	160	?	?	?
VEST	2005	42 (W_{ASIC}) - 64 (W_{FPGA})	Variable usually 80-256	Variable usually 80-256	256 - 800	N/A (2006)	N/A (2006)
WAKE	1993	Fast	?	?	8192	CPA & CCA	Vulnerable
Stream Cipher	Creation Date	Speed (cycles per byte)	(bits) Effective Key-Length	(bits) Initialization vector	(bits) Internal State	Attack Best Known	Attack Computational Complexity

Lámina 99 Roberto Gómez C.

Criptosistemas simétricos de flujo

stream ciphers

Lámina 100 Roberto Gómez C.