Sistemas de cifrado en flujo.

Introducción.

La diferencia fundamental entre el cifrado en bloque y el cifrado en flujo es que en el primero la información se cifra en bloques de un determinado tamaño, mientras que en el segundo el cifrado se realiza bit a bit, siendo en general mucho más rápido. Los sistemas de cifrado en flujo hacen un uso muy elevado de los sistemas generadores de números pseudoaleatorios. En estos sistemas un determinado algoritmo genera un conjunto de bits con una apariencia totalmente aleatoria. Estos se combinan con el texto fuente mediante un conjunto de operaciones para obtener el texto cifrado.

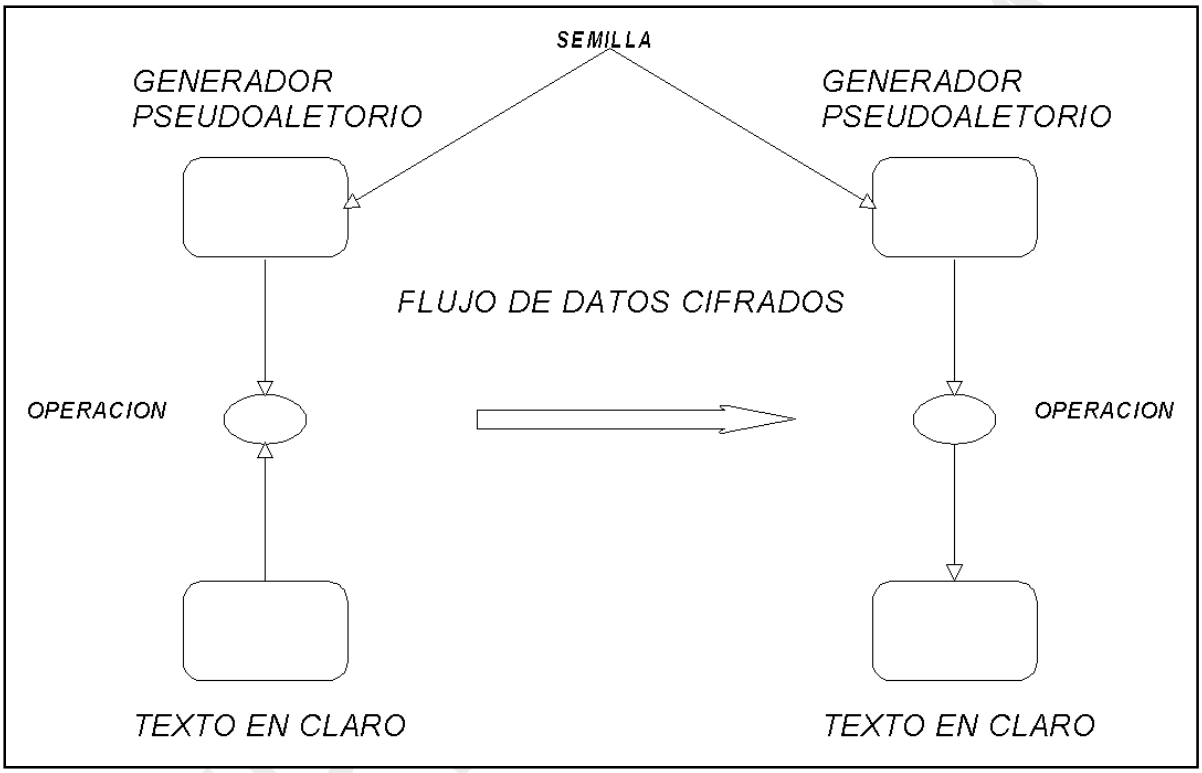
Los sistemas de cifrado en flujo pretenden acercarse lo más posible al teórico *secreto perfecto* definido por Shannon y cuya expresión es el denominado *one-time pad*. El problema viene dado por el hecho de que en el *one-time pad* o cifrado de un solo uso el tamaño de la clave debe ser como mínimo igual al del mensaje en claro a transmitir, lo que lo invalida para aplicaciones de comunicaciones. En lugar de utilizar claves de un tamaño muy elevado, se pretende generarlas mediante un conjunto de funciones denominadas generadores de números pseudoaleatorios, ya que los ordenadores al ser dispositivos estrictamente secuenciales son incapaces de generar números aleatorios puros. Por otra parte en algunos casos las secuencias aleatorias generadas deben poder ser reproducibles, es decir, debe poder generarse el mismo conjunto de números aleatorios en dos puntos distintos, cosa que es impracticable con la utilización de generadores aleatorios puros. Algunos autores [FOS97] consideran, sin embargo, que el tamaño de la clave no es problema porque existen métodos de generación de números aleatorios en el mundo físico y dispositivos de almacenamiento de bajo coste que permiten el almacenamiento de claves aleatorias puras lo suficientemente largas. A pesar de ello creemos que tales métodos no son aplicables a sistemas de comunicaciones y presentan la misma desventaja que los libros de códigos, es decir, la existencia física de un medio que almacena la clave, susceptible de ser duplicado, perdido o robado.

Los sistemas de cifrado en flujo presentan una serie de ventajas con respecto a los cifradores en bloque. En primer lugar son mucho más rápidos, generalmente mucho más simples y fácilmente implementables en hardware.

Un esquema de un sistema de cifrado en flujo es el mostrado en la imagen siguiente. La operación a la que se hace referencia puede ser cualquier conjunto de operaciones, aunque generalmente se suele utilizar el OR exclusivo como operación. Como vemos en el diagrama siguiente, el punto problemático es la generación de números aleatorios. Un ataque a un sistema de este tipo solo puede ser realizado en base a poder predecir la secuencia de números que generará el sistema. En este caso pueden ocurrir tres cosas:

a) Se conoce el algoritmo de generación de números aleatorios. Esta situación, aunque parezca lo contrario es bastante posible, la mayoría de los sistemas de generación de números aleatorios son utilizados para generar números en aplicaciones comerciales típicas, por ejemplo para protocolos en navegadores. Al ser el producto en general de fácil obtención, siempre es posible utilizar técnicas de ingeniería inversa que permitan la obtención del algoritmo. En este caso la seguridad depende de la calidad de la semilla. Un ataque de este tipo es el que realizaron unos estudiantes franceses al protocolo del navegador de Netscape. Si bien el generador de números pseudoaleatorios era de gran calidad, la generación de la semilla inicial era pobre y permitía hacer conjeturas sobre el valor inicial en un conjunto reducido de valores.

b) No se conoce el algoritmo de generación de números pseudoaleatorios ni la semilla que la genera. En este caso la única esperanza del criptoanalista es la de que el sistema tenga un periodo corto e intentar deducirlo.
c) Se conoce la semilla, pero no el tipo de generador. Es posible realizar conjeturas sobre el tipo de generador utilizado, y si se monitoriza la salida del sistema, es posible obtener la información deseada.



Conceptos básicos.

La mayoría de los generadores de números aleatorios, en particular todos los obtenidos por métodos matemáticos, pueden generar una secuencia infinita de números, sin embargo, aunque esta secuencia pueda ser infinita, será con toda probabilidad periódica. Es decir, a partir de un determinado momento la secuencia se repetirá. Al número de elementos entre repeticiones se le denomina el periodo de la secuencia y al conjunto de elementos que forman un periodo se le denomina un ciclo. Evidentemente se pretende obtener generadores que tengan un periodo máximo, idealmente infinito, para simular el one-time pad. En realidad, y tal como se plantea en [BEK82], el concepto de pseudoaleatoriedad y su aplicación es diferente en criptografía de lo que pueda serlo en otras disciplinas científicas. En otras disciplinas se pretende obtener generadores de números aleatorios que sigan una determinada distribución estadística acorde con el experimento que se pretende simular. En criptografía, más que la propia aleatoriedad, principal cualidad debe ser la impredicibilidad de la secuencia. Esta impredicibilidad es la que da la fortaleza al sistema.

Más formalmente podemos decir que dada una secuencia $S = s_1 s_2 .. s_n$ con periodicidad *p*, tenemos siempre que $s_i = s_{i+p} \forall i \in [1, m-p]$, o inversamente definimos una secuencia *S* como periódica si existe un *p>0* tal que $s_i = s_{i+p} \forall i \in [1, m-p]$. Definimos un ciclo de la secuencia *S* de período *p* a una subsecuencia cualquiera de longitud *p* que cumpla la propiedad anterior, definiéndose el primer ciclo de la secuencia como el ciclo *generador* de la misma. En el caso de que un conjunto finito de términos preceda a la aparición del primer ciclo se dice que la secuencia tiene un *retraso* que viene definido por ese conjunto de términos. Podemos definir también la función de autocorrelación de la secuencia para una traslación *t* como $\sigma(t) = \frac{A-D}{p}$, siendo *A* el número de posiciones en las que las secuencias s_m y s_{m+t} coinciden y *D* el numero de posiciones en las que no coinciden. Evidentemente tenemos que *A+D = p* y que $\sigma(t+p) = \sigma(t)$.

Postulados de Golomb.

En [GOL82] se indican las propiedades que debe cumplir una secuencia binaria de período *p*. Estos postulados de Golomb como se les conoce son:

1) Si *p* es par el ciclo de longitud *p* debe contener la misma cantidad de unos y ceros. En el caso de que *p* sea impar, la diferencia entre unos y ceros no puede ser mayor que uno.
2) En un ciclo de longitud *p*, la mitad de las series de componentes idénticos (rachas) tiene longitud 1, una cuarta parte tiene longitud 2, una octava parte tiene longitud 3, y en general, para cada *i* para el cual hay al menos 2^{i+1} series de componentes iguales, $\frac{1}{2^i}$ tienen longitud *i*. Además se cumple que para cada una de esas longitudes hay tantas series de ceros como de unos.
3) La autocorrelación de una secuencia de período *p* se mantiene constante para cada valor de *t*, cuando $t \neq 0$. La función de autocorrelación se define como:

$$C(t) = \frac{1}{p}\sum_{n=1}^{p} a_n a_{n+t} = \begin{cases} 1 & \text{si } t = 0 \\ -\frac{1}{p} & \text{si } t \neq 0 \end{cases}$$

La razón de estos postulados es clara. El primero refleja la propiedad de que las entradas tengan la misma probabilidad de aparición al tener la misma cantidad de elementos de cada clase. En la segunda se pretende igualar la probabilidad de aparición de bigramas, trigramas, etc. El último postulado pretende evitar ataques de tipo Kasiski, es decir, evitar la aparición de secuencias binarias en las cuales haya una cierta periodicidad de aparición de patrones.

Números aleatorios.

Los números aleatorios son números con dos propiedades estadísticas importantes: uniformidad e independencia. Se puede decir que un número aleatorio a_i es una muestra

independiente de una distribución continua entre 0 y 1. De esta manera decimos que su distribución de probabilidad viene dada por:

$$f(x) = \begin{cases} 1, \text{con } 0 \le x \le 1 \\ 0, \text{en cualquier otro caso.} \end{cases}$$

El valor esperado de cada a_i viene dado por:

$$E(a) = \int_0^1 x dx = \frac{x^2}{2}\Big|_0^1 = \frac{1}{2}$$

Y la varianza viene dada por:

$$V(a) = \int_0^1 x^2 dx - [E(R)]^2 = \frac{x^3}{3}\Big|_0^1 - \left(\frac{1}{2}\right)^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$$

Como consecuencia de las propiedades de uniformidad e independencia tenemos que:

1. Si el intervalo [0,1] está dividido en *n* clases o subintervalos de igual longitud, el número esperado de observaciones en cada intervalo es $\frac{N}{n}$, siendo *N* el número total de observaciones.
2. La probabilidad de observas un valor en un intervalo independiente en particular es independiente de los valores observados anteriormente.

Números pseudoaleatorios.

Se trata de números que pretenden producir una secuencia de números entre 0 y 1 que simulan, tan fielmente como sea posible, las propiedades ideales de uniformidad e independencia. Sin embargo, los números pseudoaleatorios no son aleatorios propiamente dichos y presentan algunos problemas, entre los que podemos incluir[i]:

1. Los números generados pueden no estar uniformemente distribuidos.
2. Los números generados pueden ser discretos en lugar de continuos.
3. La media de los números generados puede ser muy alta o muy baja.
4. La varianza de los números generados puede ser muy alta o muy baja.
5. Puede haber dependencias entre los números. Por ejemplo:
    a. Autocorrelación entre números.
    b. Números sucesivos mucho mayores o menores que sus adyacentes.
    c. Varios números mayores que la media, seguidos por varios números menores que la media.

Existen varios métodos de generación de números aleatorios, siendo alguno de ellos inadecuado para la utilización en criptografía por su falta de seguridad, utilizándose ampliamente en otros campos como el de la simulación. En [BAN96] se indican los errores a evitar en una función de generación de números pseudoaleatorios, así como las propiedades más deseables de su implementación en un ordenador.

Para determinar lo buenas que son las secuencias generadas por un método existen varios test, que veremos más adelante. Sin embargo, ya que la mayoría de los números son generados mediante ordenadores, es importante que el método utilizado cumpla una serie de condiciones. Entre ellas:

1. La rutina debe ser rápida.
2. Debe ser portable.

3. Debe tener un periodo muy largo.
4. Debe ser replicable.
5. Debe cumplir lo más fielmente posible las propiedades estadísticas ideales de uniformidad e independencia.
6. En su uso en aplicaciones criptográficas debe cumplir con la propiedad de la impredecibilidad. Es decir, conocidos varios símbolos de salida $s_0, s_1, ..., s_{n-1}$, éstos no deben aportar información suficiente para poder predecir el valor del siguiente símbolo de salida s_n.

Para su utilización en criptografía debemos añadir tres propiedades [SOR99] que ya hemos comentado anteriormente:

1) La longitud de la clave debe ser grande para evitar la prueba exhaustiva de todas las claves.
2) El periodo debe ser largo, idealmente, tan largo como el mayor de los mensajes a transmitir.
3) La secuencia generada debe ser impredecible. Es decir, dados los *n* - 1 elementos anteriores, debe ser imposible predecir el valor del n-ésimo.

Fred Piper en [PIP82] añade una cuarta propiedad. El sistema debe parecer no lineal. La mayoría de los ataques criptoanalíticos pretenden obtener una formulación lineal a un problema criptográfico.

Métodos de generación de números aleatorios.

Generadores congruenciales.

Se trata de uno de los métodos más usados para generar números aleatorios dada su facilidad de implementación y la posibilidad de obtener secuencias con un periodo muy largo. Son muy rápidos y con unas propiedades estadísticas muy buenas, pero su seguridad queda en entredicho al depender cada valor solo del anterior y de que la seguridad del algoritmo no es especialmente buena, tal como demostró Jim Reeds en su día[ii]. Fue propuesto por Lehmer en 1951 y produce una secuencia de enteros $x_0, x_1, .., x_n$ entre 0 y $m-1$ siguiendo la relación:

$$x_{n+1} = (ax_n + c) \bmod m, \quad \text{con } i = 0,1,2,..,$$

Este caso se suele denominar *generador congruencial afín*, mientras que en el caso en que *c* = 0, se suele denominar *generador congruencial lineal* y viene definido por la siguiente formula. Los valores iniciales x_0, *a*, *c* y *m* se denominan *semilla*, constante multiplicadora, incremento y modulo respectivamente. Su elección afecta drásticamente a sus propiedades estadísticas y su periodo máximo como veremos más adelante.

Se Si *a*, *b*, *m* y x_0 se escogen adecuadamente se pueden conseguir generadores *completos* o de *secuencia máxima*. En [BAN96] se dan las siguientes elecciones de los parámetros antes indicados:

1) Si $m = 2^x$ y $b \neq 0$, el máximo periodo posible es 2^x y se obtiene si *b* es relativamente primo con *m* y $a = 1 + 4k$ con *k* entero.
2) Si $m = 2^x$ y $b = 0$, el máximo periodo posibles es 2^{x-2} y se obtiene cuando la semilla X_0 es par y el multiplicador $a = 3 + 8k$ o $a = 5 + 8k$ con *k* entero.

3) Si *m* es un número primo y *b* = 0, el máximo periodo posible es *m* – 1 y se obtiene cuando el multiplicador *a* cumple la propiedad de que el menor *k* que cumple que $a^k - 1$ es divisible por *m* es *k* = *m* – 1.

Un ejemplo ayudará a entender el proceso. Supongamos que tenemos el siguiente generador congruencial:

$$x_{n+1} = (17x_n + 37) \bmod 99$$

Los primeros números generados suponiendo que empezamos con una $x_0 = 3$ serían:

$$x_1 = (17.3 + 37) \bmod 99 = 88$$
$$x_2 = (17.88 + 37) \bmod 99 = 48$$
$$x_3 = (17.48 + 37) \bmod 99 = 61$$

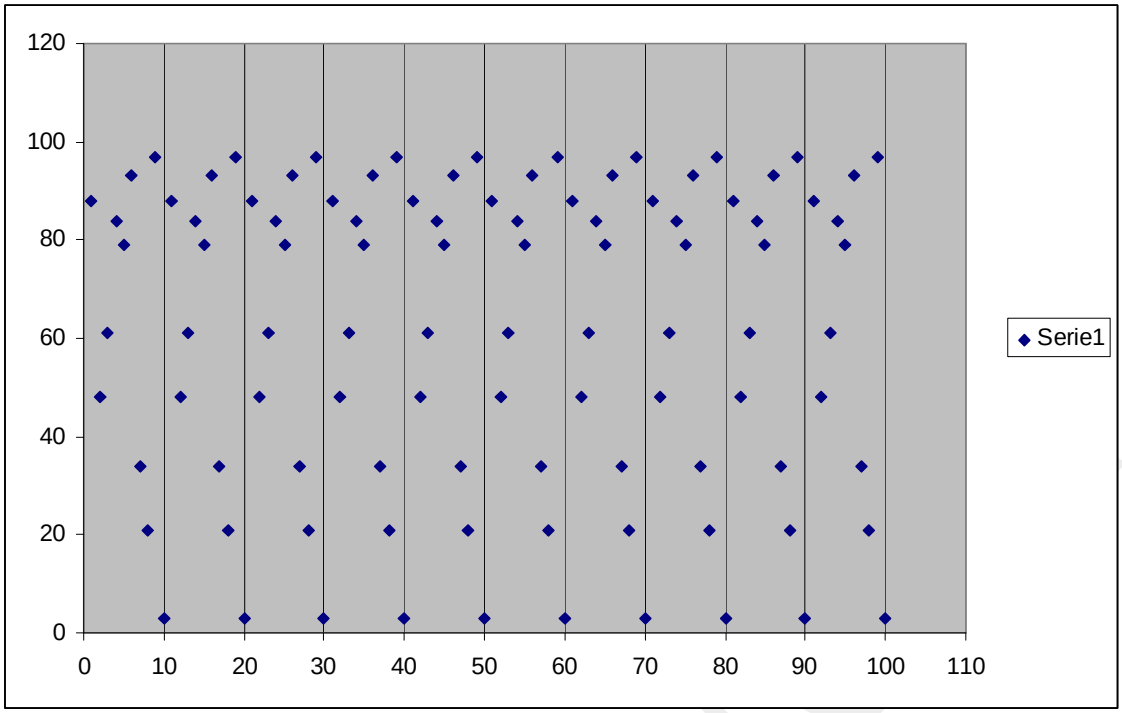
Los números generados estarían en el intervalo [0, m], sin embargo, en muchos casos necesitamos convertir esos números a números decimales. En ese caso se obtienen por ejemplo simplemente dividiendo por $m+1$. En nuestro caso los números resultantes serían:

$$x_1 = {}^{88}\!/_{100} = 0{,}88$$
$$x_2 = {}^{48}\!/_{100} = 0{,}48$$
$$x_3 = {}^{61}\!/_{100} = 0{,}61$$

Como hemos dicho la elección de los parámetros x_0, *a*, *c* y *m* afecta de una forma drástica al funcionamiento del generador, en particular a sus propiedades estadísticas. El ejemplo anterior es una muestra de ello. Los primeros 100 números generados son:

88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3**,88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3**,88,48,61,84,79,93,34,21,97,3,**88,48,61,84,79,93,34,21,97,3.**

Como vemos, el periodo en esta serie es de diez, con lo que a partir del onceavo elemento la serie se repite. En el siguiente gráfico podemos apreciar muy claramente esta repetición.



La elección de los valores *a*, *c*, y *m* es fundamental para conseguir un periodo grande, como veremos más adelante. Lo que nos interesa inicialmente es ver cuan uniformes e independientes son los valores $x_0, x_1, .., x_n$ generados. También, y no menos importantes, son la obtención de una máxima densidad y, evidentemente, un periodo máximo. A pesar de que los valores generados son discretos, si escogemos un *m* muy grande, por ejemplo 2^{48}, un valor común en generadores de lenguajes de simulación, podremos considerar a niveles prácticos la función como continua y asumir como densidad máxima el hecho de que no haya intervalos grandes en los que la función no tome valores.

Para obtener un periodo máximo en estos generadores es condición necesaria y suficiente que se cumplan las siguientes condiciones[iii]:

1. El $mcd(m,c) = 1$
2. $\forall q$ primo tal que q/m se cumple que $q/(a-1)$
3. Si $4/m$ entonces se cumple que $4/(a-1)$.

En el caso de generadores congruenciales lineales el periodo máximo se consigue si:

1. $mcd(x_0, m) = 1$
2. *a* es un elemento primitivo modulo *m.*

Un elemento se dice que es primitivo modulo *m* cuando después de *m* cálculos ha generado todos los elementos del conjunto finito [0, *m*-1]. Para que un número *a* sea un elemento primitivo modulo p^e se debe cumplir alguna de las siguientes condiciones[iv]:

1. $p^e = 2$ y *a* es impar.
2. $p^e = 4$ y $a \bmod 4 = 3$.
3. $p^e = 8$ y $a \bmod 8 = 3, 5$ o 7.
4. $p = 2, e \geq 4$ y $a \bmod 8 = 3, 5$.

5. p es impar, $e = 1, a \neq 0 \bmod p$ y $a^{\frac{(p-1)}{q}} \neq 1 \bmod p \; \forall q$ primo divisor de $p-1$.
6. p es impar, $e > 1, a$ cumple con lo establecido en el apartado anterior y además cumple $a^{(p-1)} \neq 1 \bmod p^2$.

En [BAN01] se dan unas recetas más sencillas para obtener generadores congruenciales lineales con periodo máximo[v]:

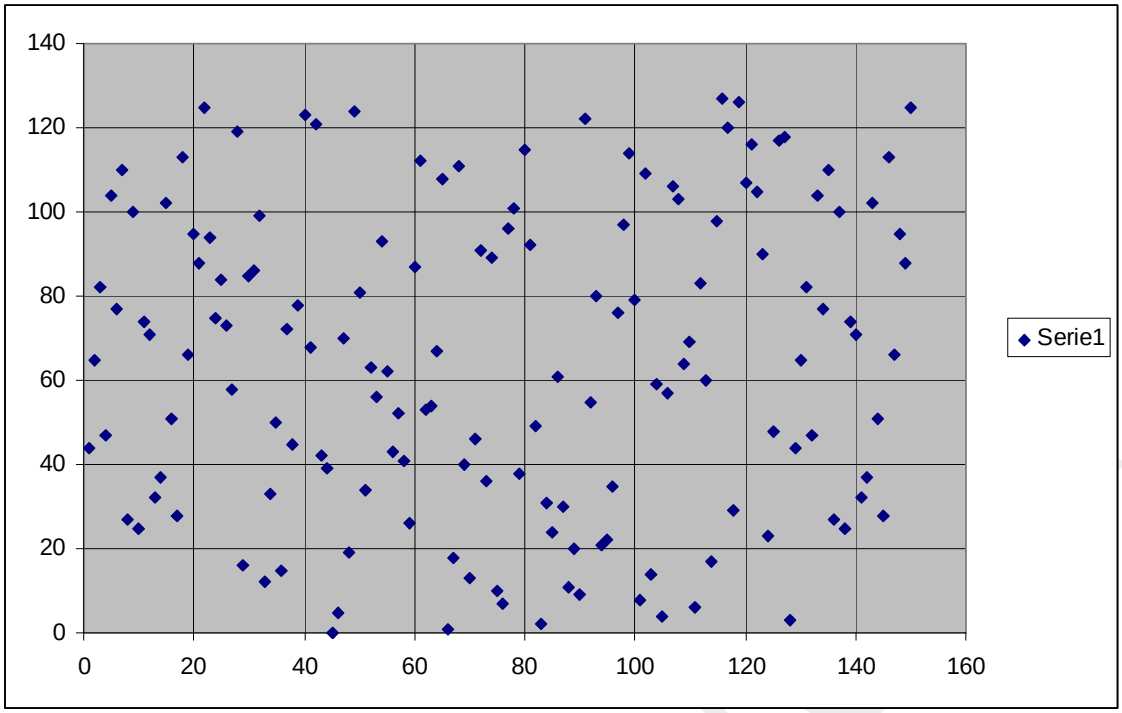
1. Para una potencia de 2, por ejemplo 2^b, y con $c \neq 0$, el periodo máximo 2^b se obtiene si c es relativamente primo con 2^b y $a = 1 + 4k$, con k entero.
2. Para una potencia de 2, por ejemplo 2^b, y con $c \neq 0$, se obtiene un periodo máximo de 2^{b-2} si x_0 es impar y $a = 3 + 8k$ o $a = 5 + 8k$ para algún $k = 0,1,...$
3. Para m, un numero primo, y $c = 0$ se obtiene un periodo igual a $m-1$ si a cumple la propiedad de que el entero k más pequeño tal que $a^k - 1$ es divisible por m es $k = m-1$.

Veamos un ejemplo utilizando la receta 1 anterior. Escogemos k = 3, b = 7, c = 5 con lo que $m = 2^7 = 128$ y a = 13. La formula resultante sería: $x_i = (ax_{i-1} + c) \bmod m$.

Los primeros 150 valores que obtenemos a partir de un valor inicial $x_0 = 3$ son:

44, 65, 82, 47, 104, 77, 110, 27, 100, 25, 74, 71, 32, 37, 102, 51, 28, 113, 66, 95, 88, 125, 94, 75, 84, 73, 58, 119, 16, 85, 86, 99, 12, 33, 50, 15, 72, 45, 78, 123, 68, 121, 42, 39, 0, 5, 70, 19, 124, 81, 34, 63, 56, 93, 62, 43, 52, 41, 26, 87, 112, 53, 54, 67, 108, 1, 18, 111, 40, 13, 46, 91, 36, 89, 10, 7, 96, 101, 38, 115, 92, 49, 2, 31, 24, 61, 30, 11, 20, 9, 122, 55, 80, 21, 22, 35, 76, 97, 114, 79, 8, 109, 14, 59, 4, 57, 106, 103, 64, 69, 6, 83, 60, 17, 98, 127, 120, 29, 126, 107, 116, 105, 90, 23, 48, 117, 118, 3, **44, 65, 82, 47, 104, 77, 110, 27, 100, 25, 74, 71, 32, 37, 102, 51, 28, 113, 66, 95, 88, 125.**

Como vemos, en este caso el periodo es máximo (128) y si observamos el gráfico siguiente podemos observar que los resultados obtenidos son, a primera vista, aleatorios.



Se han propuesto métodos para eliminar la dependencia exclusiva del último elemento, la más común es la utilización de generadores cuadráticos y cúbicos, pero todas estas implementaciones se han demostrado inseguras [SCH94].

Ejemplo: Programa en freebasic para calcular una cantidad determinada de números aleatorios con un generador congruencial. El resultado se graba en el fichero *afin.txt* dentro de la carpeta *cripto* en el disco duro *c*.

```
'Programa para calcular números aleatorios con un generador congruencial
Dim As LongInt i
Dim As longint a
Dim As LongInt m
Dim As LongInt c
Dim As longint x
Dim As longint iteraciones
Input "Introduzca la constante:", c
Input "Introduzca el multiplicador:", a
Input "Introduzca el modulo:", m
Input "Introduzca el valor inicial:", x
Input "Introduzca iteraciones:", iteraciones
Open "c:\cripto\afin.txt" For Output As #1
For i = 0 To iteraciones
        x=(a*x+c)Mod(m)
        print #1, x; ";"
Next
Close #1
end
```

Uso de los números pseudoaleatorios en criptografía.

Es evidente que la utilización de estos números en criptografía, partiendo del cumplimiento de las condiciones anteriormente expuestas, es de una gran utilidad. Su utilización es sencilla. Si los vamos a utilizar como un generador pseudoaleatorio de bits simplemente debemos convertir el número obtenido por el método que queramos en un bit o grupo de bits. Por ejemplo, podemos hacer una sustitución del número por un 1 binario en el caso de que el número resultante sea impar y por un 0 en caso contrario. También podemos hacer lo mismo con los dos números centrales, o con el primer y último dígito del número obtenido. Por ejemplo, con los números obtenidos en el ejemplo anterior (44, 65, 82, 47, 104, 77, 110, 27, 100, 25) obtendríamos la siguiente tira de bits en el primer caso: 0101010101, y la siguiente en el caso de utilizar el primer y último número para generar bits: 00010001101110011001.

Otra posibilidad, en el caso de cifrar números directamente, es sumar sin acarreo al número resultante de codificar la letra. Este método es ampliamente utilizado en ambientes diplomáticos y militares en combinación con libros de códigos. Supongamos que utilizamos una tabla como la siguiente para codificar las letras del alfabeto:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
09	15	23	12	14	65	87	95	45	44	75	51	61	39	57	03	13	88	71	85	62	17	33	55	77	19	99

Tabla 1.

Un mensaje como “EL ATAQUE DE LOS TANQUES EMPEZARÁ DE MADRUGADA” quedaría como:

14510961098817141214510385620939881714851461131499097109121461091271178709120 9

Tal como podemos ver en la tabla siguiente:

E	L	A	T	A	Q	U	E	D	E	L	O	S	T	A	N	Q	U	E	S	E	M	P	E	Z	A	R	A	D	E	M	A	D	R	U	G	A	D	A
14	51	09	61	09	88	17	14	12	14	51	03	85	62	09	39	88	17	14	85	14	61	13	14	99	09	71	09	12	14	61	09	12	71	17	87	09	12	09

Vamos a cifrarlo utilizando el siguiente generador congruencial:

$$x_{n+1} = 3571x_n + 5377 \bmod 8973 \text{ con la semilla inicial } x_0 = 3213$$

Los primeros números de la serie serían:

3213	2533	5936	8607	8449	0557	2418	8029	8201	3276
3181	4910	5745	8494	8711	2967	3421	0542	2691	4855

Iremos sumando los dígitos obtenidos sin acarreo con el mensaje codificado obteniendo el siguiente resultado:

E	L	A	T	A	Q	U	E	D	E	L	O	S	T	A	N	Q	U	E	S	E	M	P	E	Z	A	R	A	D	E	M	A	D	R	U	G	A	D	A
14	51	09	61	09	88	17	14	12	14	51	03	85	62	09	39	88	17	14	85	14	61	13	14	99	09	71	09	12	14	61	09	12	71	17	87	09	12	09
32	13	25	33	59	36	86	07	84	49	05	57	24	18	80	29	82	01	32	76	31	81	49	10	57	45	84	94	87	11	29	67	34	21	05	42	26	91	78
46	64	24	94	58	14	93	11	96	53	56	50	09	70	89	58	60	18	46	51	45	42	52	24	46	44	55	93	99	25	80	66	46	92	12	29	25	03	77

Para descifrar simplemente tendremos que poner debajo del mensaje cifrado los números generados con el generador congruencial e ir restando sin acarreo. Una vez obtenido el resultado vamos a la tabla de sustitución y cambiamos cada letra por su valor.

46	64	24	94	58	14	93	11	96	53	56	50	09	70	89	58	60	18	46	51	45	42	52	24	46	44	55	93	99	25	80	66	46	92	12	29	25	03	77
32	13	25	33	59	36	86	07	84	49	05	57	24	18	80	29	82	01	32	76	31	81	49	10	57	45	84	94	87	11	29	67	34	21	05	42	26	91	78
14	51	09	61	09	88	17	14	12	14	41	03	85	62	09	39	88	17	14	85	14	61	13	14	99	09	71	09	12	14	61	09	12	71	17	87	09	12	09
E	L	A	T	A	Q	U	E	D	E	L	O	S	T	A	N	Q	U	E	S	E	M	P	E	Z	A	R	A	D	E	M	A	D	R	U	G	A	D	A

El ataque de Reeds a los generadores congruenciales.

Supongamos que nuestros servicios de escucha han captado el mensaje cifrado anterior y pasan el mensaje al equipo de criptoanálisis, que lo único que conocen, por el estudio de mensajes similares, es que los números utilizados en el generador congruencial son de cuatro dígitos y que la tabla de codificación es la tabla 1[1]. Además se está esperando un ataque con tanques, con lo que una frase probable en el mensaje sería precisamente LOS TANQUES. Vamos probando el siguiente esquema con todos los grupos de doce dígitos hasta que llegamos al que verdaderamente contiene la palabra buscada:

Palabra probable	L	O	S	T	A	N	Q	U	E	S
Cifrado	56	50	09	70	89	58	60	18	46	51
Valor palabra	51	03	85	62	09	39	88	17	14	85
Clave	05	57	24	18	80	29	82	01	32	76

Los números que se generan son: 0557, 2418, 8029, 8201 y 3276. Tenemos por la definición de generador congruencial que:

$$2418 \equiv 557x + b \bmod M$$
$$8029 \equiv 2418x + b \bmod M$$
$$8201 \equiv 8029x + b \bmod M$$
$$3276 \equiv 8201x + b \bmod M$$

Restando las ecuaciones sobre la primera tenemos que

$$5611 \equiv 1861x \bmod M$$
$$5783 \equiv 7472x \bmod M$$
$$858 \equiv 7644x \bmod M$$

Intentaremos ahora eliminar *x*. 1861 y 7472 son relativamente primos con lo que podemos multiplicar la primera ecuación por 7472 y la segunda por 1861. La tercera no nos es necesaria. Una vez hecho esto tendríamos:

$$41925392 \equiv 13905392x \bmod M$$
$$10762163 \equiv 13905392x \bmod M$$

[1] La obtención de las tablas de codificación básicas es muy frecuente en ambientes de guerra, bien por captura, el caso más general, bien por comparación entre varios mensajes cifrados y su equivalente en claro.

Con lo que $31163229 \equiv 0 \bmod M$, es decir, *M* divide a éste número. Factorizamos pues el número y nos dará que es un producto de 3x3x23x151x997. Cogemos todas las posibles combinaciones de números inferiores a 10000 ya que sabemos que el generador congruencial da como resultado números de cuatro dígitos. En nuestro caso puede ser:

	3	9
23	69	207
151	453	1359
997	2991	8973
3473		

Es decir, que los candidatos para *M* son: 3, 9, 23, 69, 151, 207, 453, 997, 1359, 2991, 3473, 8973. Parece lo más lógico que el número sea el mayor posible, en este caso el 8973, que es la solución correcta. Tenemos que encontrar ahora el valor de *x*, para ello cogemos la primera ecuación del grupo de 3 ($5611 \equiv 1861x \bmod 8973$) ya que sus factores 8973 y 1861 son relativamente primos, lo que garantiza que existe una solución única. El inverso multiplicativo de 1861 mod 8973 es 3973, con lo que calculamos el resultado y obtenemos que el valor de *x* es 3571, lo que sabemos es correcto. Solo nos falta encontrar *b* que podemos calcular simplemente utilizando la primera ecuación inicial $2418 \equiv 557x + b \bmod M$ con lo que tenemos que $b = 2418 - 557 \times 3571 \bmod 8973 = 2418 - 6014 \bmod 8973 = 5377 \bmod 8973$.

Generadores congruenciales lineales combinados.

El problema de los generadores congruenciales lineales es el periodo, que en general es pequeño para las necesidades actuales, sin contar con la poca seguridad que presentan. Es necesario pues encontrar generadores con periodos más largos. Una aproximación que ha funcionado bien desde el punto de vista de la longitud de periodo es la combinación de dos o más generadores congruenciales, sin embargo, como veremos, en la mayoría de los casos no se consigue un aumento significativo de la seguridad.

Método de L'Ecuyer.

Este método fue presentado por L'Ecuyer en 1988 [ECU88]. La idea consiste en utilizar la suma de varios generadores congruenciales independientes para obtener un resultado cuyo periodo sea función de los periodos de dichos generadores.

Para ver como podemos combinar los generadores nos basamos en el siguiente resultado:

Sean $X_{i,1}, X_{i,2}, ..., X_{i,k}$ variables aleatorias discretas, no necesariamente igualmente distribuidas, pero una de ellas sí forzosamente distribuida uniformemente en el intervalo $[0, m_i - 2]$. Tenemos entonces que $X_i = \left(\sum_{j=1}^{k} X_{i,j} \right)$ esta uniformemente distribuida en dicho intervalo. Basándonos en este resultado, si tenemos el conjunto $X_{i,1}, X_{i,2}, ..., X_{i,k}$ de las i-ésimas salidas de *k* diferentes generadores congruenciales multiplicativos, donde el j-ésimo generador tiene un módulo primo m_j y el multiplicador a_j se escoge de manera que el periodo sea máximo, es decir, $m_j - 1$. De esta manera el conjunto generado por el j-ésimo generador está aproximadamente uniformemente distribuido en $[1, j-1]$ y consecuentemente

$X_{i,j}-1$ está uniformemente distribuido en $[0, j-2]$, con lo que se le aplica el resultado anterior. L'Ecuyer sugiere la utilización de un generador de la forma:

$$X_i = \left(\sum_{j=1}^{k} (-1)^{j-1} X_{i,j} \right) \bmod m_i - 1$$

$$\text{con } R_i = \begin{cases} \dfrac{X_i}{m_1} \text{ con } X_i > 0 \\ \dfrac{m_i - 1}{m_1} \text{ con } X_i = 0 \end{cases}$$

El periodo máximo en un generador de este estilo sería:

$$P = \frac{(m_1 - 1)(m_2 - 1)..(m_k - 1)}{2^{k-1}}$$

Para un ordenador de 32 bits L'Ecuyer recomienda un generador con $k = 2$, $m_1 = 214783563$, $a_1 = 40014$, $m_2 = 2147483399$ y $a_2 = 40692$. Este generador tiene un periodo de $\dfrac{(m_1 - 1)(m_2 - 1)}{2}$.

Test de aleatoriedad.

Como ya hemos indicado anteriormente, los generadores de números pseudoaleatorios son capaces de generar secuencias que pueden confundirse con secuencias aleatorias puras, sin embargo, estas secuencias tienen todas un periodo finito, en general bastante elevado. En aplicaciones criptográficas este periodo suele ser superior a 10^{50} [BEK82]. Necesitamos alguna herramienta que nos permita determinar si efectivamente las secuencias generadas son aleatorias. Para ello se suelen aplicar técnicas estadísticas que con un nivel de significación α nos permiten decidir que la secuencia tiene un $(100-\alpha)\%$ de confianza de ser aleatoria. La mayoría de los generadores actuales son generadores aleatorios de bits. Para medir su aleatoriedad se suelen utilizar los tests incluidos en el FIPS 140-1 "*Security requirements for cryptographic modules*" en los que se comprueban 20000 bits aleatorios. Los test más frecuentemente utilizados son:

1) *Test de frecuencia o* χ^2. Consiste en la aplicación del test χ^2 con un nivel de significación α. Se calcula $\chi^2 = \dfrac{(n_0 - n_1)^2}{n}$, donde n_0 es el número de ceros en la secuencia y n_1 el número de unos, y se contrasta con una tabla de la distribución χ^2 para el nivel de significación deseado, si el valor calculado es menor que el obtenido en la tabla diremos que la secuencia es correcta, en caso contrario es rechazada. En el FIPS-401 se considera válido si el número de bits está entre 9654 y 10346.

2) *Test de series.* Se utiliza para determinar que las probabilidades de transición sean correctas. Sean n_{00}, n_{01}, n_{10} y n_{11} la cantidad de pares 00, 01, 10 y 11 en la secuencia

de bits en observación. Tenemos que $n_{00}+n_{01}+n_{10}+n_{11}=n-1$ y queremos además que $n_{00}=n_{01}=n_{10}=n_{11}\approx\frac{n-1}{4}$, se demuestra que $\frac{4}{n-1}\sum_{i=0}^{1}\sum_{j=0}^{1}(n_{ij})^2-\frac{2}{n}\sum_{i=0}^{1}(n_i)^2+1$ se distribuye aproximadamente como una χ^2 con dos grados de libertad para $n\geq 21$, con lo cual se calcula el valor anterior y se procede como en el test anterior.

3) *Test del poker*. Se trata de calcular las frecuencias de cada tipo de sección de longitud *m*. Sean $f_0, f_1,..., f_{2^m-1}$ las frecuencias, tenemos que $\sum_{i=0}^{2^m-1} f_i = F = \left[\frac{n}{m}\right]$. La función $\frac{2^m}{F}\sum_{i=0}^{2^m-1}(f_i)^2 - F$ se distribuye como una χ^2 con 2^m-1 grados de libertad. Nótese que para *m* = 1 tenemos el test de series. Existe una variante del test en el que se evalúa la función $\frac{2^m}{F}\sum_{i=0}^{m}\frac{(x_i)^2}{\binom{m}{i}}-F$ que se distribuye como una χ^2 con *m* grados de libertad, siendo x_i los bloques de longitud *m* tales que están formados por *i* unos y *m* - *i* ceros. En el FIPS-401 se dividen los 20000 bits en grupos de 4 dígitos. Cada grupo de 4 bits (*)

4) *Test de autocorrelación*. Sea $a_1, a_2,.., a_n$ la secuencia de bits a evaluar. Definimos $A(d)=\sum_{i=1}^{n-d} a_i \oplus a_{i+d}$ con $0\leq d\leq n-1$ y d un entero fijo tal que $1\leq d\leq\left\lfloor\frac{n}{2}\right\rfloor$. Utilizamos la expresión $2\frac{\left(A(d)-\frac{n-d}{2}\right)}{\sqrt{n-d}}$ que sigue aproximadamente la distribución Normal con media 1 y varianza cero para valores de $n-d\geq 10$.

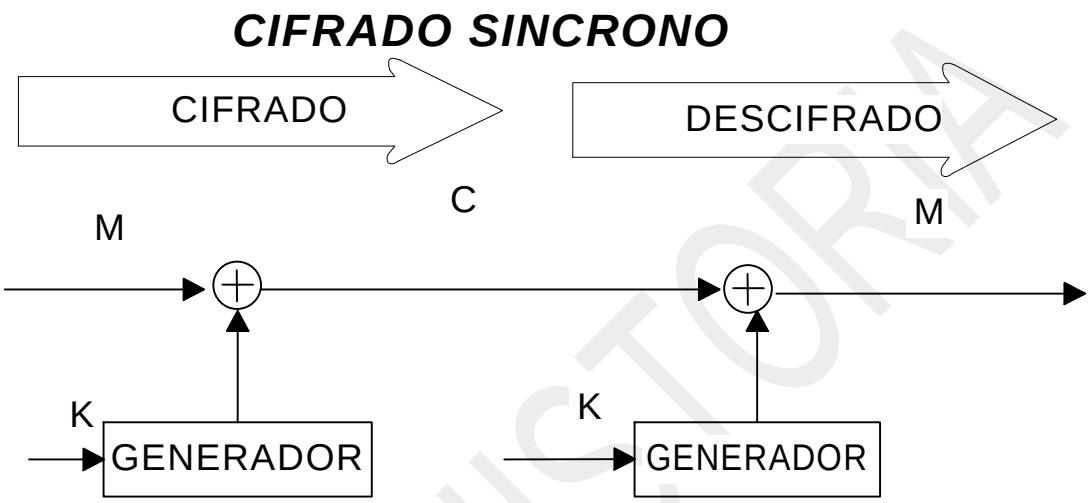
5) *Test de rachas.* Sea r_{0i} el número de rachas de ceros de longitud *i*, y su equivalente en unos r_{1i}. El número total de rachas de ceros y unos definidos como r_o y r_1 es $r_0=\sum_{i=1}^{n} r_{0i}$, y $r_1=\sum_{i=1}^{n} r_{1i}$, la expresión $\sum_{i=1}^{k}\frac{(r_{0i}-e_i)^2}{e_i}+\sum_{i=1}^{k}\frac{(r_{1i}-e_i)^2}{e_i}$ con $e_i=\frac{(n-i+3)}{2^{i+2}}$ y *k* el mayor entero *i* para el cual $e_i\geq 5$, se distribuye como una χ^2 con $2.k-2$ grados de libertad.

Funcionamiento de los sistemas de cifrado en flujo.

Los sistemas de cifrado en flujo presentan dos modos de funcionamiento, cifrado síncrono y autosincronizante. En el cifrado síncrono el generador de números aleatorios no depende del mensaje a cifrar, con lo que el emisor y el receptor deben sincronizar la clave antes de empezar la transmisión y en el caso de pérdida de un bit se pierde el sincronismo.

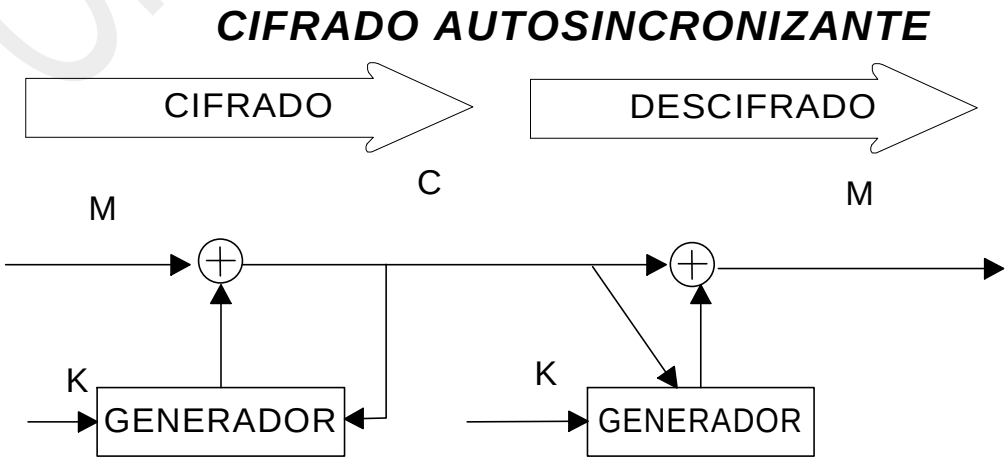
La propiedades más relevantes de los sistemas síncronos son[SOR99]:

1) No existe dependencia con el mensaje a cifrar.
2) Los errores en transmisión no se propagan.
3) La secuencia cifrante es periódica.
4) La pérdida o inserción de un bit en el mensaje produce la pérdida del sincronismo.



En los cifrados autosincronizantes, la generación de la clave es función del mensaje utilizado anteriormente, esto tiene la ventaja de permitir la sincronización automática en situaciones de pérdida de sincronismo. Las propiedades más destacadas de los sistemas autosincronizantes son:

1) El estado siguiente depende de los últimos *n* caracteres del mensaje.
2) En caso de un error de transmisión, éste se propaga *n* bits.
3) La pérdida o inserción de un bit en el mensaje provoca la perdida de sincronismo durante *n* bits.
4) La secuencia cifrante es aperiódica.



A continuación se exponen algunos de los métodos de generación de números aleatorios más comunes.

Generadores de Tausworthe.

Son generadores de bits que tienen la forma $b_i = (c_1.b_{i-1} + c_2.b_{i-2} + ... + c_q.b_{i-q}) \bmod 2$. Las constantes c_i solo pueden tomar los valores 0 y 1. Permiten obtener secuencias muy largas con propiedades estadísticas muy buenas.

Generador de Blum-Micali.

Se trata de un algoritmo muy seguro basado en la dificultad de calcular logaritmos discretos. Sean *a* y *p* números primos con *p* grande. Se escoge una semilla inicial x_0 y se calculan los números de la siguiente manera:

1) Calcular $x_{i+1} = a^{x_i} \bmod p$.
2) Escoger $\begin{cases} 1 \text{ si } \log_a x < \dfrac{p-1}{2} \\ 0 \text{ en cualquier otro caso} \end{cases}$

Generador RSA.

El generador RSA, basa su seguridad en la dificultad de factorizar un número, al igual que su homónimo para cifrado. La obtención de números aleatorios se realiza de la siguiente manera. Se escogen dos números primos *p* y *q*. Se calcula su producto *n* y se escoge un número *b* tal que sea relativamente primo a *n*. Al igual que en el algoritmo de cifrado, *n* y *b* son públicos mientras que *p* y *q* son secretos.
Se escoge una semilla inicial x_0 y se obtienen los de números aleatorios mediante aplicación de la siguiente formula:

$$x_{i+1} = x_i^b \bmod n .$$

Si el número obtenido es par, se escoge un 0 como el siguiente bit de la secuencia, en caso contrario se escoge un 1.

DES en modo OFB.

El DES en modo OFB puede utilizarse para la generación de números aleatorios. La salida de 64 bits presenta unas buenas propiedades estadísticas.

Generador Blum Blum Shub o de residuos cuadráticos.

Se trata quizás del algoritmo más popular, no solo por la sonoridad de su nombre, sino también por su eficiencia y seguridad. Según Schneier [SCH94] se trata del algoritmo más sencillo y más eficiente, si bien es relativamente lento comparado con otros generadores, es

excelente para aplicaciones de alta seguridad. Su fortaleza, al igual que en el caso anterior, se basa en la dificultad de factorizar números grandes.

Sean p y q dos números primos grandes y $n = p.q$. Se escoge un número entero x relativamente primo con n. Si denotamos por s_i a los números de la secuencia pseudoaleatoria tenemos que

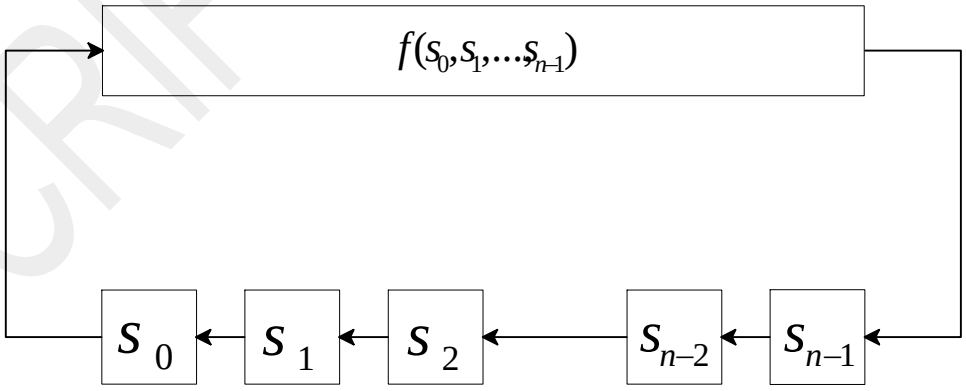
$$x_i = \begin{cases} x^2 \bmod n & \text{si } i = 0 \\ x_{i-1} \bmod n & \text{si } i > 0 \end{cases} \qquad s_i = x_i \bmod 2$$

Algoritmos genéticos.

La utilización de algoritmos genéticos para la obtención de secuencias pseudoaleatorias viene definida por su misma esencia. Su principal ventaja es la de que los números son cuasi aleatorios y que su aleatoriedad puede controlarse a través de la función de aptitud y de la proporción de mutación asignada. Sus principales inconvenientes son el hecho de necesitar una función generatriz de números aleatorios para los procesos de mutación y su relativa lentitud. La seguridad del sistema en este caso depende del conocimiento del valor inicial de la población, de los tantos por ciento asignados al cruzamiento y a la mutación, la función de generación de números aleatorios y su semilla.

Registros de desplazamiento.

Un registro de desplazamiento es un dispositivo de almacenamiento de n bits que denotaremos $s_0, s_1, .., s_{n-1}$, estos elementos se van desplazando y variando su contenido en función de la existencia o no de una función de realimentación. El valor de estos elementos en un tiempo t se denomina el *estado* del registro. Si bien existen varios tipos de registros de desplazamiento, en criptografía se utilizan básicamente los dos tipos, los registros con realimentación lineal (Linear Feedback Shift Register) y los registros con realimentación no lineal (Non Linear Feedback Shift Register). Las operaciones utilizadas en ambos son la O exclusiva denotada por $\oplus$ y la Y denotada por el punto.



En el caso de que la función $f(s_0, s_1, ..., s_{n-1})$ esté definida por un polinomio sobre un campo de Galois en módulo 2, que denotaremos como CG(2), se dice que el registro de desplazamiento es lineal y los coeficientes c_i los denominamos coeficientes de realimentación. En los apartados siguientes correspondientes a registros de desplazamiento,

consideramos que las operaciones se realizan en CG(2) y utilizaremos indistintamente los símbolos + y $\oplus$ para denotar la suma exclusiva.

Registros de desplazamiento con realimentación lineal(LFSR).

En un registro de desplazamiento con realimentación lineal podemos representar la función f como $f(s_0, s_1, ..., s_{n-1}) = c_o.s_o \oplus c_1.s_1 \oplus ... \oplus c_{n-1}s_{n-1}$ y dado un instante *t*, el valor de cualquier estado queda fijado para un instante posterior *t*+1 de la siguiente manera:

$$s_i(t+1) = \begin{cases} s_{i+1}(t) \text{ con } 0 \le i \le n-2 \\ \sum_{i=0}^{n-1} c_i s_i(t) \text{ con } i = n-1 \end{cases}$$

Vista la definición de LFSR, la primera pregunta es ¿Cual es el tamaño máximo del periodo en un LFSR?. El tamaño máximo del periodo es de $2^n - 1$, ya que la secuencia de estados 0,0,...,0 debe eliminarse pues genera continuamente la misma salida.

Polinomios primitivos.

Definimos como polinomio característico de cualquier LFSR el definido por $f(x) = c_0 + c_1 x + ... + c_{n-1}x^{n-1} + x^n$. Se dice que un polinomio característico es irreducible si no existen *g(x)* y *q(x)* de grado menor o igual a n tales que *f(x) = g(x).q(x).* La importancia de los polinomios irreducibles es que puede demostrarse que cualquier polinomio de grado *n* puede expresarse como el producto de dos polinomios irreducibles. Un caso especial de estos polinomios son los denominados primitivos, que tienen la propiedad de que todas las secuencias no nulas generadas por él tienen un periodo de $2^n - 1$.

Se define el polinomio recíproco de un polinomio $f(x) = \sum_{i=0}^{n} a_i x^i$ de grado *n* como $f^*(x) = \sum_{i=0}^{n} a_i x^{n-i} = x^n h\left(\frac{1}{x}\right)$. Los polinomios recíprocos tienen una serie de propiedades interesantes:

1) El grado del polinomio $f^*(x)$ es como máximo *n*, siendo *n* exactamente si $h_0 = 1$.
2) Para cualquier polinomio característico se cumple siempre que el grado del polinomio es el mismo que el de su recíproco.
3) $(f^*(x))^* = f(x) \Leftrightarrow f(0) = 1.$
4) $(f_1(x).f_2(x))^* = f_1^*(x).f_2^*(x).$

Dados los polinomio *f(x),g(x)* y *h(x)* si se cumple que *h(x) = f(x).g(x),* se dice que *g(x)* y *f(x)* dividen a *h(x).* Un polinomio cuyos únicos divisores son el 1 y él mismo se dice que es un polinomio *irreducible*. Se define el máximo común divisor de dos polinomios *f(x), g(x),* un tercer polinomio *h(x)* tal que *h(x)* divide a ambos y si otro polinomio que cumpla esta propiedad, también divide a *h(x).* Análogamente se define el mínimo común múltiplo de dos polinomios *f(x), g(x)* como el mínimo polinomio que divide a ambos. En el caso de que el $\text{mcd}(f(x), g(x)) = 1$, se dice que ambos polinomio son relativamente primos.

Definimos la función generatriz de una secuencia infinita como $G(x) = \sum_{i=0}^{\infty} a_i x_i$. Si *G(x)* es la función generatriz de la secuencia de salida s_t, *G(x)* quedará definido completamente por su estado inicial y su polinomio característico.

A continuación presentamos algunos teoremas relativos a los polinomios en los CG(2), las demostraciones de los mismos pueden consultarse en [BEK82].

1) Sea $f(x) = c_0 + c_1 x + ... + c_{n-1} x^{n-1} + x^n$ un polinomio en CG(2) y $\Omega(f)$ el espacio de soluciones de *f*, supongamos además que el conjunto $\{s_i\} \in \Omega(f)$. Si $G(x) = \sum_{i=0}^{\infty} s_i x_i$, entonces se cumple que $G(x) = \frac{\phi(x)}{f^*(x)}$, con $\phi(x) = \sum_{i=1}^{n} (c_i x^{n-i} (\sum_{j=0}^{i-1} s_j x_j))$ y $c_n = 1$.

Como vemos, $\phi(x)$ queda determinado completamente por f*(x)* y el conjunto $\{s_i\}$ que representa el estado inicial, una vez obtenido $\phi(x)$ podemos calcular *G(x)*, con lo cual tenemos definida la secuencia de los s_i.

2) Sean *f(x)* y *g(x)* dos polinomios y sea *h(x)* = m.c.d(*f(x)*, *g(x)*), entonces se cumple que $\frac{f(x)}{h(x)}$ y $\frac{g(x)}{h(x)}$ son relativamente primos.

3) Sean *f(x)*, *g(x)* y *h(x)* polinomios tales que *h(x)* divide al producto de *f(x)* y *g(x)* y tales que el m.c.d.(*h(x)*, *f(x)*) = 1, se cumple que *h(x)* divide a *g(x)*.

4) Sea *f(x)* un polinomio con exponente *e*, si $\{s_i\} \in \Omega(f)$, se cumple que el periodo de $\{s_i\}$ divide a *e*.

5) Sea *f(x)* un polinomio irreducible con exponente *e*, con $\{s_i\} \in \Omega(f)$ y $\{s_i\} \neq \{0\}$, el periodo de $\{s_i\}$es *e*. Como consecuencia de esta propiedad, notemos que si un polinomio irreducible tiene grado *n* y exponente $2^n - 1$, cualquier secuencia no nula generada por él será de periodo máximo, es decir $2^n - 1$.

6) Sea *f(x)* un polinomio de grado *n*. Si $\{s_i\} \in \Omega(f)$ y $\{s_i\} \neq \{0\}$, entonces $\{s_i\}$ es de periodo máximo si y solo si *f(x)* es primitivo.

Con las herramientas necesarias para la construcción de polinomios de periodo máximo, nos queda por determinar la cantidad de polinomios primitivos de un determinado grado. Para ello se utiliza la función de Euler, que ya hemos visto al presentar el sistema RSA. Para un determinado *n*, el número de polinomios de ese grado sobre CG(2) viene definido por la formula $\lambda(n) = \frac{\phi(2^n - 1)}{n}$.

Por último indicar que cualquier secuencia de periodo máximo generada por un LFSR, cumple los postulados de Golomb, con lo que podemos determinar que inicialmente pueden

utilizarse en algoritmos criptográficos. Sin embargo, las secuencias criptográficas generadas por LFSR son relativamente sencillas de criptoanalizar, ya que dada una secuencia de 2*n* dígitos, siendo *n* el grado del polinomio primitivo podemos fácilmente obtener el estado inicial y sus coeficientes. La importancia principal de las LFSR es que cualquier secuencia periódica puede generarse con un LFSR no singular, denominándose *complejidad lineal* la longitud del mínimo LFSR capaz de generarla. Para determinar el LFSR capaz de generar una secuencia dada, se utiliza el algoritmo siguiente.

Algoritmo de Massey-Berlekamp.

El algoritmo de Massey-Berlekamp, es un algoritmo capaz de determinar la complejidad lineal de una secuencia binaria finita de longitud n en $O(n^2)$ operaciones de bits. El esquema parte de fijar un LFSR de una longitud dada e ir comprobando si añadiendo un bit de la secuencia inicial el LFSR sigue siendo capaz de generar la secuencia, en caso contrario se aumenta la longitud del LFSR y se sigue con el mismo proceso. Un esquema del algoritmo obtenido de [MEN98] es el siguiente:

/* L es la complejidad lineal y C(D) el polinomio característico de la misma */

Función Massey-Berlekamp

Definir vectores C(D), B(D), T(D);

$C(D) \leftarrow 1;$

$L \leftarrow 0;$

$m \leftarrow -1;$

$B(D) \leftarrow 1;$

$N \leftarrow 0;$

Mientras $N < n$ *hacer*

$$d \leftarrow (s_N + \sum_{i=1}^{L} c_i sN - i) \bmod 2$$

$$\text{Si } d = 1 \text{ hacer} \begin{cases} T(D) \leftarrow C(D) \\ C(D) \leftarrow C(D) + B(D).D^{N-m} \\ \text{Si } L \leq \dfrac{N}{2} \text{ hacer} \begin{cases} L \leftarrow N + 1 - L \\ m \leftarrow N \\ B(D) \leftarrow T(D) \end{cases} \end{cases}$$

$N \leftarrow N + 1$

devolver (N).

Combinadores no lineales.

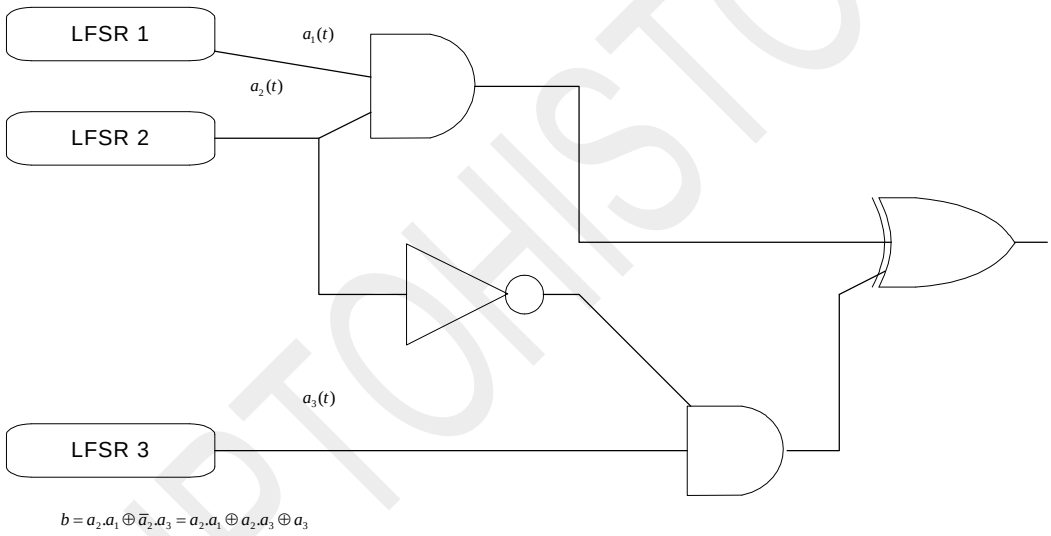
Como hemos visto, la linealidad es uno de los principales enemigos de la seguridad. Es por ello que pretendemos la eliminación de la misma en aras a obtener un sistema criptográficamente seguro. Existen dos formas básicas de reducir la linealidad en los LFSR:

1) Utilizar una función de realimentación no lineal.
2) Utilizar uno o varios LFSR.

El problema de la primera aproximación es que no existen, por el momento, conocimientos matemáticos suficientes para poder determinar la seguridad relativa de un sistema. En el segundo caso la idea es combinar varios LFSR mediante una función o dispositivo que permita generar secuencias de mayor calidad desde el punto de vista criptográfico. A continuación vemos algunos de los más comunes.

Generador de Geffe.

En este generador la idea consiste en combinar tres generadores lineales de grados n_1, n_2, n_3 cuyos polinomios sean relativamente primos mediante una función no lineal de forma que el periodo de la serie resultante sea el producto del periodo de los tres LFSR. La complejidad lineal del generador viene dada por $n_1 n_2 + n_1 n_3 + n_2$.

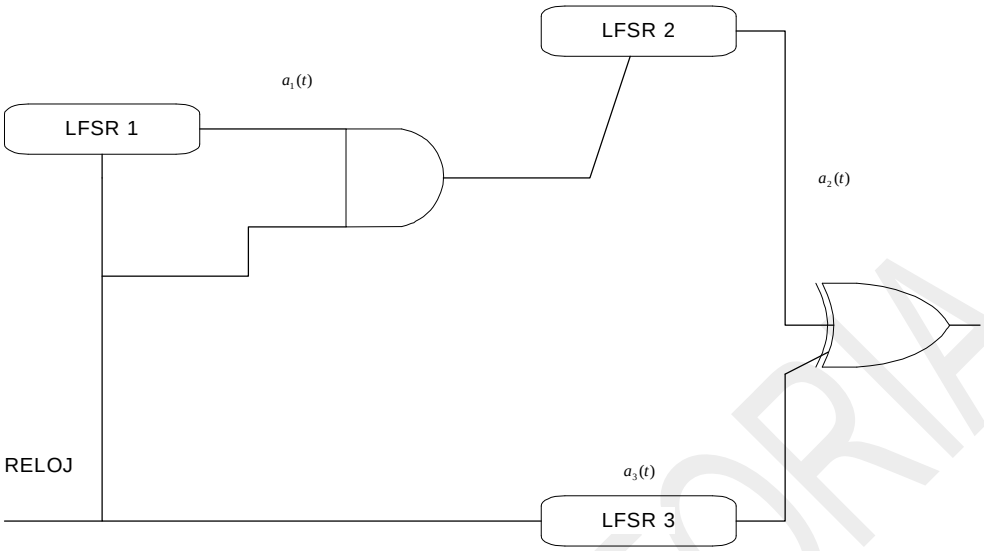


Aunque estadísticamente los valores generados son buenos, los correspondientes a la autocorrelación son pobres. Sea s la salida del generador, tenemos que $p(s = a_1) = p(a_2 = 1) + p(a_2 = 0).p(a_3 = a_1) = \frac{1}{2} + \frac{1}{2}.\frac{1}{2} = \frac{3}{4}$. El cálculo de $p(s = a_3)$ nos da el mismo resultado. Este generador es pues criptográficamente inseguro y sucumbe ante ataques por correlación[MEN98].

Generador de Beth-Piper.

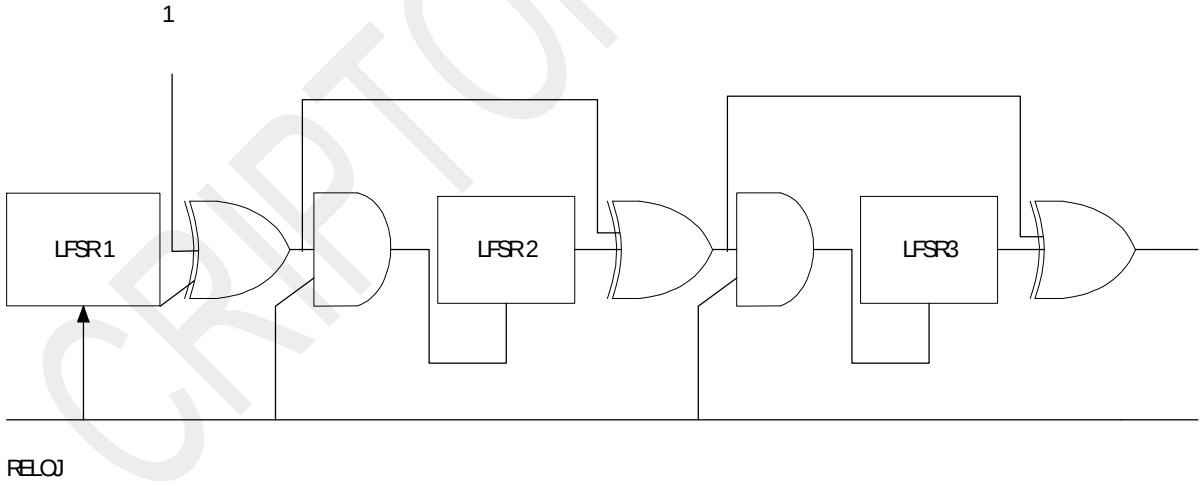
La idea básica en este generador consiste en utilizar la salida de un LFSR para controlar la entrada de un segundo LFSR, de forma que este último solo cambie su estado en el caso de que la salida del primero sea un 1. Si definimos como n_1, n_2, n_3 los grados de los polinomios implicados, el sistema tiene una complejidad lineal de $\left(2^{n_1} - 1\right)n_2 + n_3$, pero al

igual que el generador de Geffe su seguridad es baja y puede ser atacado de la misma manera[SCH94].



Generador de Gollman.

Se trata de una variante del generador de Beth-Piper que comparte con él una complejidad lineal muy elevada, pero se diferencia en que por el momento no se conoce ningún ataque válido contra él. Básicamente consiste en un conjunto de *n* LFSR conectados en serie de forma que la salida de un LFSR controle el reloj del siguiente LFSR. Si todos los LFSR tienen la misma longitud *l*, la complejidad lineal vendrá dada por $l(2^l-1)^{n-1}$.



RC4.

El algoritmo de cifrado RC4 fue diseñado por Ron Rivest en 1987 por encargo de RSA Data Security Inc. Se trata de un algoritmo de longitud variable de clave muy rápido y compacto, con un periodo superior a 10^{100} . Si bien era un algoritmo propietario cuyo código se mantenía secreto por la RSA, en Septiembre de 1994, el código fuente fue introducido anónimamente en una lista de correo Cypherpunk.

Se trata de un algoritmo muy sencillo. Utiliza una caja *S* de 256 entradas, que son una permutación de los números del 0 al 255 en función de la clave. Utiliza dos contadores *i* y *j*,

que inicialmente están a cero. Sea *m* un byte del mensaje, el proceso de generar un byte cifrado *c* es como sigue:

$$i = (i + j) \bmod 256$$
$$j = (j + S_i) \bmod 256$$
$$\textit{Intercambiar } S_i \text{ y } S_j$$
$$t = (S_i + S_j) \bmod 256$$
$$K = S_t$$
$$c = m \oplus K$$

La inicialización de la caja *S* se hace rellenando inicialmente la misma con los números del 1 al 255, de forma que $S_0 = 0,...,S_{255} = 255$. Paralelamente se inicializa un vector K de claves de 256 bytes con la clave, repitiéndola si es necesario. Una vez hecho esto se combinan ambos vectores para obtener la caja *S* definitiva de la siguiente manera:

$$j = 0$$
$$i = 0$$
$$\textit{Hacer } 256 \textit{ veces}$$
$$j = (j + S_i + K_j) \bmod 256$$
$$\textit{Intercambiar } S_i \text{ y } S_j$$
$$i = i + 1$$
$$\textit{FinHacer}$$

CONTINUARÁ

[i] [BAN01, 256-257].
[ii] [REE77].
[iii] [SOR99,19]
[iv] Obtenidos de [SOR99, 19], la fuente primaria es [KNU69, 9-33].
[v] [BAN01, 259-260].