Generación de Números Pseudoaleatorios usados en Sistemas Criptográficos

José de Jesús Ángel Ángel
jesus@seguridata.com.mx

SeguriDATA
www.seguridata.com

Generación de Números Pseudoaleatorios usados en Sistemas Criptográficos

José de Jesús Ángel Ángel
jesus@seguridata.com

Resumen

En este número encontrar una descripción de los rasgos más importantes sobre la generación de números pseudoaleatorios, principalmente, cuales son las condiciones que debe de cumplir un dispositivo que genera números pseudoaleatorios.

En gran parte de los sistemas criptográficos usados actualmente se hace necesario generar números aleatorios, sin embargo, es conocido que es una tarea difcil de llevar a cabo, por lo que se opta por generar números pseudoaleatorios, es decir, números que están cerca de ser aleatorios. Se dice que un dispositivo o algoritmo genera números pseudoaleatorios si contiene un proceso determinístico, el cual toma como entrada un número que se supone aleatorio, llamado semilla y tiene como salida una secesión de números "casi" aleatoria.

En lo anterior nos encontramos con dos problemas: el primero cómo generar una semilla aleatoria y el segundo cómo probar que la sucesión obtenida es "casi" aleatoria. El propósito de este artículo es proporcionarle algunas sugerencias para resolverlos.

1. Generación de Semillas

Para poder considerar que un número es aleatorio, el conjunto de donde es tomado debe de cumplir los requisitos de espacio equiprobable, es decir, que todo elemento tenga la misma probabilidad de ser elegido y que la elección de uno no dependa de la elección del otro. Para poder lograr esto, se debe de hacer a un lado la posible intervención humana y hacer uso de generadores de aleatoriedad lo más natural posible.

Gran parte del éxito de un diseñador de sistemas criptográficos se respalda en sus generadores de números aleatorios, éstos pueden estar basados en "hardware" o en "software". En el primer caso se toman a eventos físicos como generadores aleatorios, por ejemplo, el tiempo de emisión entre partículas radioactivas, la distribución térmica de semiconductores o resistores, la inestabilidad de un oscilador, la potencia del sonido en micrófonos o la intensidad de video de una cámara, etcétera.

Para garantizar que el generador aleatorio sea seguro, debe de ser ajeno a cualquier atacante. Hoy en da se han creado dispositivos VLSI basados en capacitores y osciladores.

La mayor parte de dispositivos para generar una semilla aleatoria son por explicación obvia, basados en software. En este caso se requiere una vez más que el número generado cumpla las mismas condiciones que antes. Aquí también se hace uso de eventos que estén lo más alejado de la intervención humana como por ejemplo: el sistema de reloj, el tiempo entre cada activación del teclado o el ratón, el contenido de buffers de entrada o salida, entre otros.

Es importante hacer notar que mientras más recursos se utilicen en la generación de las semillas es más probable que nos acerquemos a un buen generador de números aleatorios. Después que se ha decidido cuáles eventos deben de ser usados, se recomienda que la combinación de ellos sea la concatenación, ya que esto permite que los eventos sean independientes y as la posibilidad de que sea encontrado el origen por un atacante disminuya considerablemente.

Lo anterior sugiere que debe de existir una medida de calidad de un generador de semillas aleatorias. En efecto, existen varias formas de probar si la calidad de este tipo de dispositivos es aceptable. Esto lo encontrar más ampliamente en la última sección

2. Generación de números pseudoaleatorios

Una vez que se obtiene una semilla, se procede a generar la cadena de bits pseudoaleatoria, es decir una cadena que es "casi" aleatoria, la generación de esta cadena debe ser rápida, por tal razón no es práctico usar el método anterior repetidamente.

Sin embargo, es más fácil encontrar una forma que genere la cadena pseudoaleatoria a partir de la semilla. Se ha podido demostrar que cualquier función de un sólo sentido se puede utilizar para generar cadenas pseudoaleatorias.

Podemos definir como un generador de números pseudoaleatorios a un dispositivo que recibe como entrada una semilla *s*, que se supone aleatoria, y da como salida una cadena

de bits de longitud *n*. Existen varios ejemplos estándar: el propuesto en ANSI X9.17 que usa como función de un sólo sentido a DES (FIPS 186).

Existen otros tipos de generadores de números pseudoaleatorios, que en la práctica son más lentos y sin embargo, cuando se resuelve el problema de la aritmética modular o en general la aritmética de un campo finito, es muy recomendable su uso. Estos generadores hacen uso de funciones del tipo RSA ([8]), o el Gamal ([3]) basando su aleatoriedad en la presunta intractabilidad de problemas como la factorización de un número, producto de dos números primos diferentes, y del problema del logaritmo discreto, por ejemplo, sobre una curva elíptica definida en un campo finito de característica 2.

Veamos algunos ejemplos de este tipo de generadores:

Algoritmo generador de bits pseudoaleatorio

Entrada:

Dos primos *p,q* , elegir *e*, tal que $\mathrm{mcd}(e, \phi) = 1$ donde $\phi = (p-1)(q-1)$.

Una semilla $x_0 \in [1, n-1]$

Algoritmo:

a) Para *j*=1 hasta *k*:

a1) $x_j \leftarrow (x_{j-1})^e \mod n$

a2) $z_j \leftarrow$ el menor bit significativo de x_j

Salida:
La secesión $z_1, z_2, \ldots, zk.$

Existen algunas variantes a este tipo de algoritmos como la de Micali-Schnorr ([7]), y la de Blum-Blum-Shub ([1]).

Se puede mostrar que este tipo de generadores son criptográficamente seguros, es decir, que pasan las pruebas estadísticas sobre aleatoriedad que a continuación se describen.

3. Pruebas de aleatoriedad sobre dispositivos generadores de números pseudoaleatorios

En esta sección podrá ver la forma de "probar" que un generador de números pseudoaleatorios tenga buena calidad.

El problema de la aleatoriedad es esencialmente teórico, sin embargo, se puede acercar la justificación de que un espacio es equiprobable probando que no se cumplen las características más claras de no-aleatoriedad, es decir, condiciones necesarias, aunque no suficientes. La práctica dice que esto basta para que la cadena pseudoaleatoria sea muy cercana a una cadena verdaderamente aleatoria.

La mayoría de generadores pseudoaleatorios ignoran cadenas de bits muy particulares, es decir, formalmente disminuye el espacio considerado, sin embargo, se puede probar que sólo alcanzan un 0.01×10^{-5} % del total si la cadena es del tamaño similar a las llaves DES.

En la literatura conocida ([4]), existen varias pruebas estadísticas que han sido utilizadas para probar la aleatoriedad de un generador de números pseudoaleatorios, estas pruebas verificaban por ejemplo, que las cadenas pseudoaleatorias tengan un número "igual" de ceros y unos. Contando en número de ceros y unos se calcula una estadística de prueba χ^2 y finalmente se acepta la hipótesis de que la diferencia es insignificante si χ^2 muestral cae dentro del intervalo de aceptación. Existen otras pruebas que generalizan la anterior, es decir, de algún modo miden que la distribución de ceros y unos no está cargada.

En 1990 ([5]), U. M. Maurer propone su "Universal Statistical Test for Random Bit Generators", y demuestra que ésta detecta además todos los defectos de no aleatoriedad que detectan las anteriores pruebas. Además de ser muy simple de implementar.

Enseguida se muestra en qué consiste la prueba de Maurer. Tenemos como entrada a la sucesión $s= s_0, s_1, \ldots, s_{n-1}$ y como salida a la estadística de prueba X que se distribuye normalmente con media μ y varianza $\sigma^2 = c(L,K,)^2 \quad (\sigma_1)^2 \quad / K$ donde $c(L,K) \approx 0.7 - (0.8/L) + (1.6 + (12.8/L))K^{(-4/L)}$ para $K \geq 2^L$. Entonces $Z = (X - \mu)/\sigma$ se distribuye normalmente con parámetros N(0, 1).

Una tabla obtenida de [5], para μ y $(\sigma_1)^2$ se muestra enseguida:

L	μ	$(\sigma_1)^2$	L	μ	$(\sigma_1)^2$
1	0.7326495	0.690	9	8.1764248	3.311
2	1.5374383	1.338	10	9.1723243	3.356
3	2.4016068	1.901	11	10.170032	3.384
4	3.3112247	2.358	12	11.168765	3.401
5	4.2534266	2.705	13	12.168070	3.410
6	5.2177052	2.954	14	13.167693	3.416
7	6.1962507	3.125	15	14.167488	3.419
8	7.1836656	3.238	16	15.167399	3.421

La estadística de prueba X se obtiene como:

$$X = \sum_{Q+1}^{Q+K} \log(A_i) \quad \ldots\ldots\ldots\ldots (1)$$

Lo anterior se puede ver de la siguiente manera: para probar que un dispositivo satisface la prueba de Maurer, seguiremos los siguientes pasos:

1) En condiciones normales el dispositivo tendrá que generar una cadena aleatoria *s* de longitud *n* donde *n* es como mínimo alrededor 2 millones y como máximo de 66 millones. Después se ver cómo vara la longitud de esta cadena.

2) Se divide la cadena *s* en bloques de longitud L bits, donde L vara de 1 a 16. Se recomienda que es suficiente hacer la prueba para $8 \leq L \leq 16$. El residuo de *n* entre L se puede ignorar.

3) Ahora se divide el número de bloques en dos partes: los primeros Q bloques y los restantes K bloques. Q debe ser elegido de al menos $10 \cdot 2^L$ bloques, como existen 2^L bloques diferentes de longitud L, se supone que en una cadena de 10 veces este número aparecer al menos un bloque de los 2^L diferentes. Q ser la cadena de iniciación.

4) Los restantes K bloques serán la parte de prueba, se recomienda que Q sea al menos $1000 \cdot 2^L$, es decir esperando que el dispositivo genere 1000 veces cada bloque diferente.

5) La prueba se realiza por medio de la tabla T[j], en esta tabla se guardar la última aparición del bloque j. Precisamente la tabla T, se inicializa con los primeros Q bloques haciendo de

$1 \leq i \leq Q$

$T[b_i] = i$, donde b_i es el número cuya representación binaria está en el bloque i. Posteriormente se calcula $A_i = i - T[b_i]$, que nos proporciona la estadística de prueba definida en la ecuación (1).

He aquí, el algoritmo:

Entrada: una sucesión $s = s_1, s_2, \ldots, s_n$.

a) Desde i =1 hasta Q, asignar $T[b_i] = i$ (Inicializa la tabla T)

b) sum = 0

c) Desde i = Q+1 hasta Q+K hacer

c1) sum ¬ sum + log (i-T[bi])

c2) $T[b_i] \leftarrow i$

d) X ¬ sum / K
Salida: la estadística de prueba X.

Ejemplo muestra:

Si s = **00110110 01110010100010001111000100111010011101001101101000**
En este caso L = 2, Q = 4, entonces la insalivación de la tabla queda como:

$T[00_2] = 1$
$T[11_2] = 2$
$T[01_2] = 3$
$T[10_2] = 4$

Como K = 25, entonces un resumen del ciclo para obtener a X es el siguiente:

i	log(*i-T[b $_j$]*)	Actualiza sum	Actualiza
5	log (5-3)	sum=sum + log (2)	T [01] = 5
6	log (6-2)	sum=sum + log (4)	T [01] = 6
7	log (7-1)	sum=sum + log (6)	T [01] = 7
8	log (8-4)	sum=sum + log (4)	T [01] = 8
9	log (9-8)	sum=sum + log (1)	T [01] = 9
10	log (10-7)	sum=sum + log (3)	T [01] = 10
	⋮	⋮	⋮
27	log (27-20)	sum=sum + log (7)	T [01] = 27
28	log (28-27)	sum=sum + log (1)	T [01] = 28
29	log (29-24)	sum=sum + log (5)	T [01] = 29

Lo que resulta de aquí es la estadística de prueba X = 0.499109, usando los parámetros antes definidos esta variable tiene una media $\mu = 1.5374383$, con desviación estándar que se puede calcular según las anteriores fórmulas, entonces $\sigma = 0.723644$ y esto significa que la variable Z (normal estándar) es Z = -1.43486, lo que permite deducir que la cadena es pseudoaleatoria, con una confiabilidad del 95%, ya que -1.64 $\leq$ -1.43. Termina entonces la prueba.

Para finalizar este artículo mencionaremos que en la práctica real, los parámetros recomendados anteriormente implican que el generador de números pseudoaleatorios produzca una cadena de 2068480 bits al menos para L = 8 y de 66 191 360 bits en la prueba en el caso de L = 16. El pasar esta prueba el dispositivo garantizar la calidad requerida por cualquier exigencia de alta seguridad criptográfica conocida hasta el momento ([2]).

Bibliografía

1 **Blum, L. Blum M., Shub M.**, ***"A simple unpredictable pseudorandom number generator"***, SIAM Journal on Computing 15, pp 364-383, 1986.

2 *FIPS 140,* ***"Security requirements for cryptographic modules"***, Federal Information Processing Standards Publication 140 - I, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 1994.

3 **Kaliski Jr. B. S.** *"**Elliptic curves and cryptography: a pseudorandom bit generator and other tools"***, Ph D thesis, MIT Department of Electrical Engineering and Computer Science, 1988.

4 ***Knuth D.E.,*** **"The Art of Computer Programming - Seminumerical Algorithms, Vol 2"**, Addison Wesley Reading, 1973.

5 **Maurer U.M..** ***"A Universal Statistical Test for Random Bit Generators"***, Advances in Cryptology CRIPTO '90, LNCS 537, pp 409-420, 1991.

6 **Menezes A.J., van Oorschot P.C., Vanstone S.A.,** ***"HandBook of Applied Cryptography"***, CRC Press 1997.

7 ***Micali S., Schnorr C.P., "Efficient, perfect polynomial random number generators"***, Journal of Cryptology 3, pp 157-172, 1991.

8 **Shamir A.** ***"On the generation of cryptographically strong pseudorandom sequences"***, ACM Transaction on Computer Systems, 1, 1983.

SeguriDATA, Seguridad Privada, S.A. de C.V.
http://www.seguridata.com