Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Lecturas en Ciencias de la Computación
ISSN 1316-6239

⊕**P**⊕

Un algoritmo aleatorio de cifrado simétrico por bloques

R.A. Pastoriza.

RT 2007-10

Centro IOMMA

Caracas, Septiembre, 2007.

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de investigación de Operaciones y Modelos Matemáticos Aplicados

⊕P⊕

Un algoritmo aleatorio de cifrado simétrico por bloques

v.2

R. A. Pastoriza
rpastoriza@cantv.net

Caracas, septiembre 2007

Índice

1. Introducción

Los métodos de cifrado simétricos por bloque [6] Menezes y otros, se apoyan casi en su mayoría en transformaciones determinísticas, fijas pero suficientemente complicadas, de la cadena – string binario - de entrada, el texto natural. Su fortaleza se suele basar en la adecuada mezcla de operaciones lógicas -o exclusivos-, multiplicaciones, sustituciones y corrimientos de bits.

En este trabajo se presenta como idea central, el núcleo del algoritmo, que las transformaciones de cifrado, de un bloque de texto natural se hagan de manera aleatoria. O sea que cada bloque sufre una transformación aleatoria propia de él. Este enfoque conduce directamente a solventar esa característica desconcertante de los cifradores determinísticos que es ni mas ni menos la de preservar estructuras subyacentes a los textos de entrada, hecho visible en imágenes cifradas modo ECB y que lleva a la recomendación usual de usar el cifrado por bloques en modo encadenado CBC u otros similares.

Tales transformaciones aleatorias se apoyan en la disponibilidad y fortaleza de un generador aleatorio GNSA_CS, que sea una fuente de seudo azar confiable, reproducible, con excelentes propiedades aleatorias y además criptográficamente seguro. Esta última propiedad es la que ciertamente no abunda en la extensa literatura de generación de números seudo aleatorios. Se proporciona además la descripción de generador de números seudo aleatorios – GNSA -, a nuestro entender criptográficamente seguro, por diseño mas no por demostraciones o pruebas. Damos entonces este hecho como descontado.

Otra consideración de diseño es tratar de evitar la cercanía de la clave de las transformaciones que sufre el texto, en la idea de dificultar, entre otros ataques posibles, los que se hacen vía pares de texto natural, y clave escogidos. Estos requerimientos conducen eventualmente a la necesidad de medir, cuantificar, la difusión de un cierto cifrador[1].

Las experiencias realizadas han llevado al autor a bosquejar una arquitectura

[1] Ver [1] A. Briceño

que pretende reflejar adecuadamente esas preocupaciones. En la Fig.1 siguiente se ilustran estas ideas.

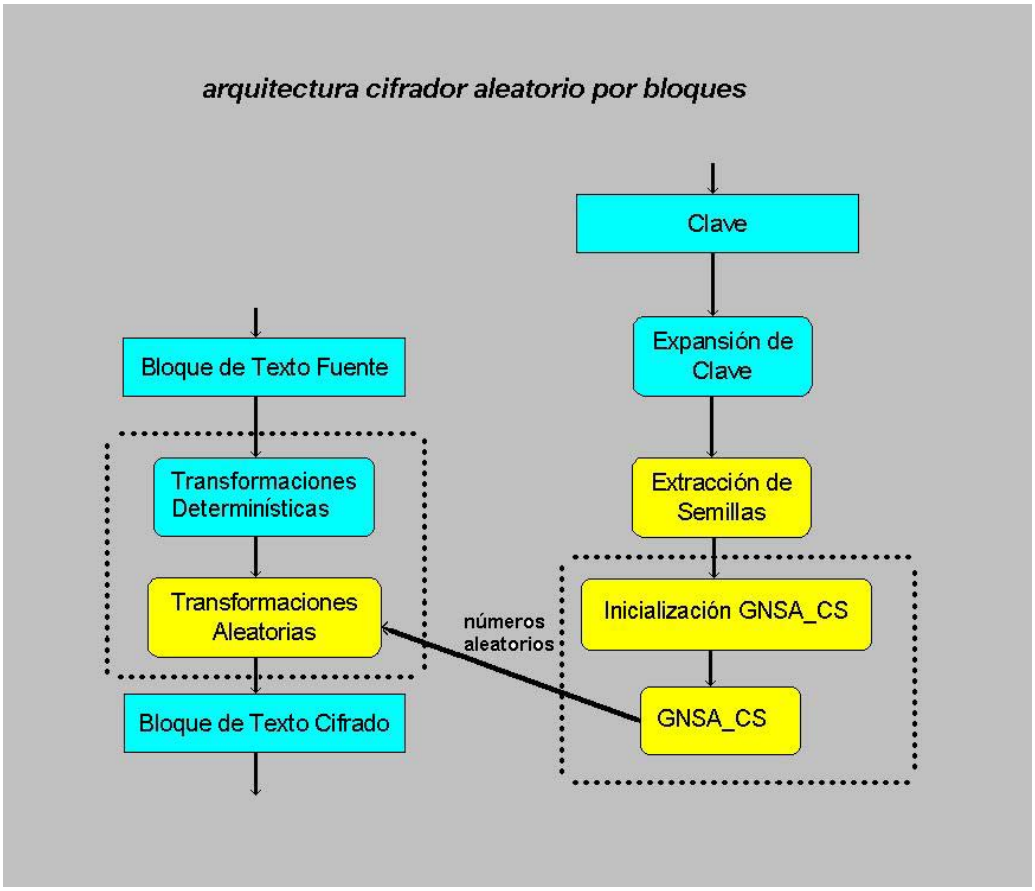

Fig.1

Dentro de la arquitectura genérica antes descripta para un cifrador aleatorio, las transformaciones determinísticas o aleatorias pueden ser de diversa naturaleza: productos matriciales, operaciones lógicas, cajas de sustitución, permutaciones, sustituciones, rotaciones, etc..

El camino del azar pasa por la construcción de un pool de bits, la extracción aleatoria de semillas del pool, la inicialización del GNSA_CS, y su posterior uso como fuente de azar.

Aquí se describe ⊕P⊕ un método de cifrado simétrico por bloques, de naturaleza aleatoria, basado en sucesivas transformaciones aleatorias

elementales de la cadena binaria de entrada.

El texto natural de entrada destinado a cifrar se divide en la cantidad necesaria de bloques de acuerdo con las longitudes de bloque determinadas, sean estos de igual longitud o no.

2. Algoritmo ⊕P⊕

La idea central del algoritmo propuesto es tratar sucesivamente, de una manera similar pero aleatoria, cada bloque de entrada de una cierta longitud.

La descripción que sigue del algoritmo se limita a la descripción de las transformaciones destinadas a cifrar/descifrar el bloque de entrada.

El tratamiento de un bloque de texto natural binario Z con tamaño *long_bloque* o sea con *long_bloque* bits, se realiza, en el sentido clásico, de la siguiente manera. Primero se realiza un proceso que trata de contribuir a una buena *difusión*, para posteriormente llevar acabo transformaciones destinadas a lograr una buena *confusión* en el algoritmo.

1.*Difusión*

El bloque de entrada *Z* se multiplica por una matriz triangular superior binaria de orden *long_bloque* llamada D^+; con la adición y multiplicación matriciales heredadas de las operaciones binarias en el campo de Galois de orden 2.

El vector resultante Z1 de la operación anterior se multiplica, usando la mismas operaciones binarias que antes, por una matriz triangular inferior binaria D^-, con la dimensiones adecuadas.

El bloque de salida resultante es Z2.

2. *Confusión*

Se construye un vector binario aleatorio A, con la longitud adecuada, de manera tal que esté a una distancia promedio C(*long_block* , *long_block/2)* [2] del texto preprocesado Z2.

Se calcula Y=A⊕Z2.

Este vector Y resultante es permutado aleatoriamente produciendo el vector W.

[2] C(n,m) es el número combinatorio de n elementos tomados de a m.

Nuevamente se construye otro vector binario aleatorio B, con la longitud adecuada, de manera tal que también esté a distancia promedio C(*long_block , long_block/2)* del vector W.

Finalmente se calcula el texto cifrado como ZC=B⊕W.

En resumen que el algoritmo bosquejado, con las debidas precauciones es invertible, delineándose así el correspondiente algoritmo desencifrador.

En la Fig.2 se presenta una ilustración del algoritmo aleatorio de cifrado simétrico por bloques ⊕P⊕.

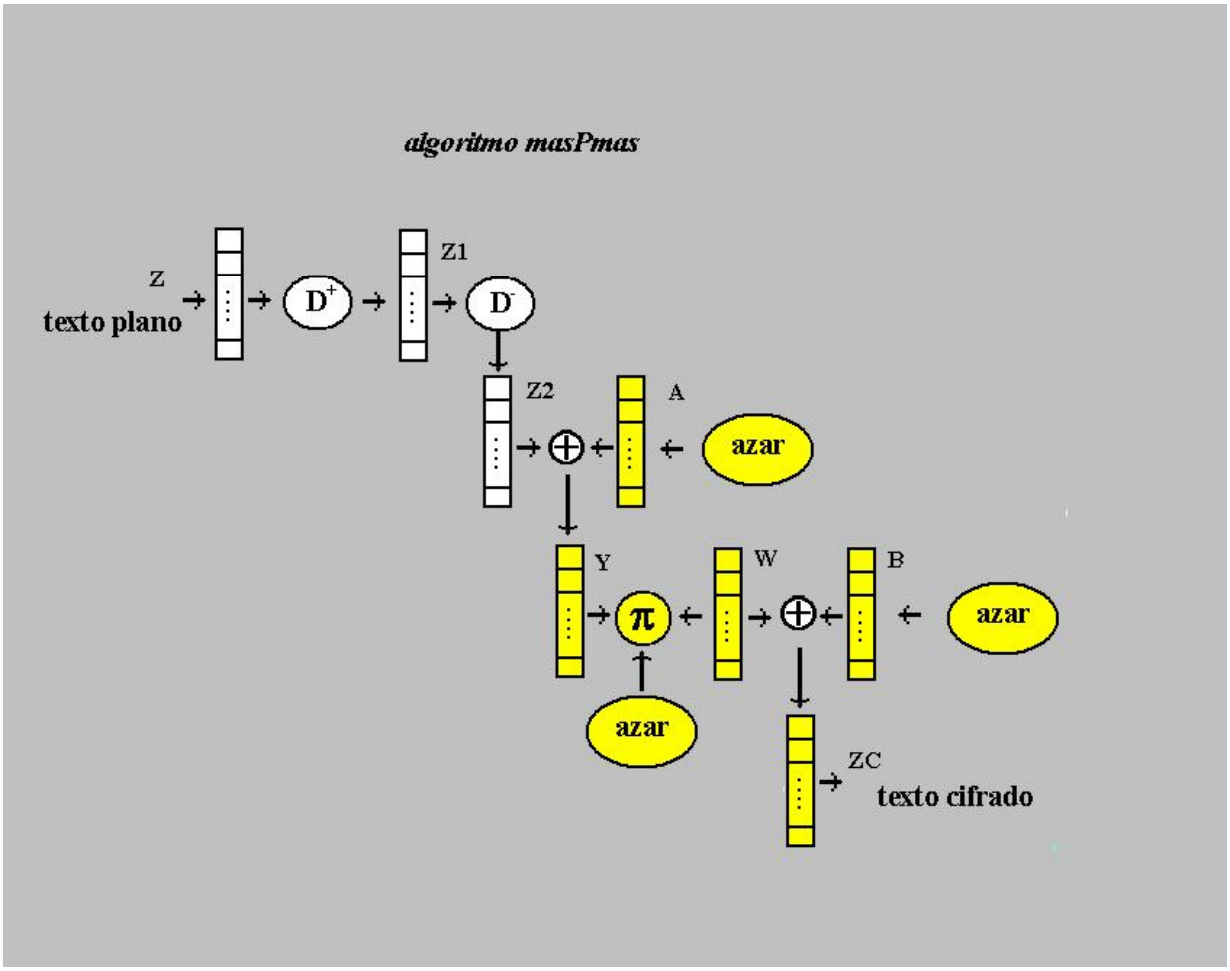

Fig.2

azar en el gráfico, es un sucesión de números seudoaleatorios generados por un generador de números seudoaleatorios criptográficamente seguro GNSA_CS.

Un esquema un poco más formal de los respectivos algoritmos aleatorios cifrador/desencifrador de bloque simétrico esbozados es:

2.1. Algoritmo cifrador E

Entradas: bloque de texto natural binario Z, *long_bloque*, azar

Difusión

1. $Z1 \leftarrow D^{+}Z$
2. $Z2 \leftarrow D^{-}Z1$

Confusión

3. $A \leftarrow F(azar)$
4. $Y \leftarrow A \oplus Z2$
5. $W \leftarrow \pi(azar, Y)$
6. $B \leftarrow F(azar)$
7. $ZC \leftarrow B \oplus W$

Salida: ZC bloque de texto cifrado binario

2.2. *Algoritmo desencifrador* E^{-1}

Entradas: bloque de texto cifrado ZC, *long_bloque*, azar

1. azar inverso ← azar
2. B ← F(azar inverso)

3. $W \leftarrow B \oplus ZC$
4. $Y \leftarrow \pi^{-1}$(azar inverso,W)
5. $A \leftarrow$ F(azar inverso)
6. $Z2 \leftarrow A \oplus Y$
7. $Z1 \leftarrow (D^{-})^{-1} Z2$
8. $Z \leftarrow (D^{+})Z1$

Salida: Z bloque de texto natural

En la descripción anterior:

Z es el bloque string/cadena binaria de entrada a cifrar, y ZC el bloque string binario cifrado de salida. Ambos con una longitud de *long_bloque*.
D^{+} es la matriz binaria triangular superior con la dimensión adecuada.
D^{-} es la matriz binaria triangular inferior con la dimensión adecuada.
azar es la secuencia de números seudoaleatorios del GNSA_CS inicializado con cierta semilla/s.
F(azar) es la construcción aleatoria de un vector binario X.
$\oplus$ es la operación de XOR entre dos vectores binarios.
π (azar, H) es la permutación aleatoria de los elementos del vector binario H.
El símbolo $^{-1}$ indica en cada caso la respectiva transformación inversa.

Dado que los algoritmos de generación de números aleatorios en esencia no son invertibles computacionalmente, para calcular las funciones inversas que se requieren en E^{-1}se debe calcular primero la sucesión de números seudoaleatorios resultante – azar – usada para cifrar y luego emplearla para descifrar en el orden inverso - azar inverso -.

En resumen el algoritmo cifrador E, para una dimensión *long_bloque* fija del bloque, al igual que el algoritmo desencifrador E^{-1} están caracterizados sólo por los números arrojados por el generador de números aleatorios GNSA_CS que se emplee. El estado del generador GNSA_CS es consecuencia de la/s semilla/s que se utilicen para su inicialización.

3. Expansión de la clave

La idea presentada en este punto es como proporcionar una fuente de azar reproducible asociada de manera unívoca, y no invertible, a la clave ***K*** que el usuario introduce. Todo el proceso de construcción de la clave expandida ***k_exp*** debe ser tal que garantice obtener una fuente de entropía, para su posterior uso en las transformaciones aleatorias del algoritmo ⊕P⊕., a la vez que sea repetible y no invertible. La clave expandida ***k_exp*** tendrá 1024 bits de longitud. Esta fuente de entropía debe ser tal que un muestreo aleatorio de sus bits luzca como aleatorio. O sea pase pruebas empíricas de aleatoriedad, por ejemplo Tuftest según [4] Marsaglia y Tseng .

La motivación de tomar estas consideraciones de diseño son: a) Evadir, en lo posible, los ataques vía diccionario sobre el espacio de las claves, y b) Ampliar el espacio de claves para dificultar un ataque por búsqueda exhaustiva, dirigida o no, inteligente o no, en el espacio de claves, c) Distanciar la clave ***K*** de su uso en el proceso de cifrado, y d) Los valores muestreados aleatoriamente de ***k_exp*** luzcan como seudoaleatorios .

La descripción del procedimiento desarrollado para construir la clave expandida discurre así:

La clave ***K*** usada en el algoritmo ⊕P⊕. tiene, a lo sumo, 256 bits de longitud y se proporciona como entrada. Si la clave ***K*** suministrada no tiene esa longitud, se la rellena con 0's y 1's alternados hasta completar los 256 bits; llamando a esta cadena clave *Ke*.

Luego, se calcula Ke^c la negación (o complemento a 1) de la clave *Ke*, también de 256 bits. Se construye, entonces, la concatenación de ambas cadenas para formar el valor $Ke//Ke^c$ de 512 bits.

El hash criptográfico de este último valor $hash(Ke//Ke^c)$ conforma una subclave expandida de 512 bits (lo que significa que la función Hash Criptográfica empleada debe ofrecer como salida un hash de 512 bits)[3].

A esta subclave expandida se le calcula su negación, que también es de 512 bits y la inserción alternativa de la subclave expandida con su negación

[3] SHA512, Whirlpool son posibles elecciones para implantar la expansión de la clave..

conforma la *clave expandida* ***k_exp*** del algoritmo ⊕P⊕., que así por construcción es de 1024 bits.

Dado que la construcción de la clave expandida se hace con la subclave expandida y su negación, se logra equiparar la cantidad de 0's y 1's presentes en la *clave expandida* ***k_exp***.

4. Extracción de semillas

El vector binario ***k_exp*** servirá como un conjunto/buffer binario del cual extraer, aleatoriamente por muestreo sin reemplazo, ver [6] Pastoriza, las semillas que se requieren para operar las transformaciones aleatorias de ⊕P⊕.

Se emplearán dos generadores de números seudo aleatorios, denominados aquí GNSA y GNSA_CS. El primero – GNSA - una vez inicializado se utilizará para inicializar aleatoriamente las semillas del segundo GNSA_CS, aquel destinado a proporcionar los números aleatorios usados en la transformaciones aleatorias del cifrado.

La primera acción será muestrear al azar en el conjunto de los 1024 bits de ***k_exp***; el subconjunto muestreado resultante, con la cantidad necesaria de bits, interpretados como enteros serán las semillas iniciales del GNSA_CS.

Inicialmente se extraerá de ***k_exp,*** con una cierta regla fija, un vector binario con la longitud adecuada como para que esos bits sirvan como semilla/s inicial/es del GNSA. A continuación otra extracción fija servirá para establecer el número inicial de rondas del GNSA durante el calentamiento de este generador.

Entonces el GNSA, ya debidamente inicializado y calentado, se lo utilizará para realizar el muestreo aleatorio de los bits de ***k_exp***. Los bits resultantes de este muestreo, en cantidad suficiente, se los interpretará como los números enteros necesarios para inicializar el GNSA_CS.

Estos valores enteros/semillas serán usados para inicializar el GNSA_CS cuyos sucesivos valores generados se emplearán en las transformaciones aleatorias del tipo A⊕Z y permutación π integrantes del cifrador ⊕P⊕.

5. Generación de los valores seudoaleatorios

La obtención de un GNSA_CS se apoyará en emplear generadores con buenas propiedades seudo aleatorias de uniformidad e independencia debidamente probadas.

Uno de tales generadores es [3] KISS de Marsaglia. No es el único, se podrían emplear el [5] MT19937 de Matsumoto y Nishima, el ran2 de [8] Press y otros, [2] ISAAC de Jenkins. Todos con muy buenas o excelentes propiedades de seudo aleatoriedad, ISAAC incluso es criptográficamente seguro.

Experiencias realizadas sugieren que al diseñar un generador de números seudo aleatorios criptográficamente seguro se pueden considerar apropiados los criterios:

1. Usar 2 GNSA con buenas propiedades aleatorias para generar 2 enteros sin signo de 32 bits *n1* y *n2.* Inicializarlos de manera independiente.
2. Construir entrelazando *n1* y *n2,* un número doble precisión *u.*
3. Usar una *tabla de mezcla* para almacenar y/o *decimar* los valores aleatorios *u.*

La arquitectura genérica, de un tal generador aparece ilustrada en la Fig.3 a continuación.

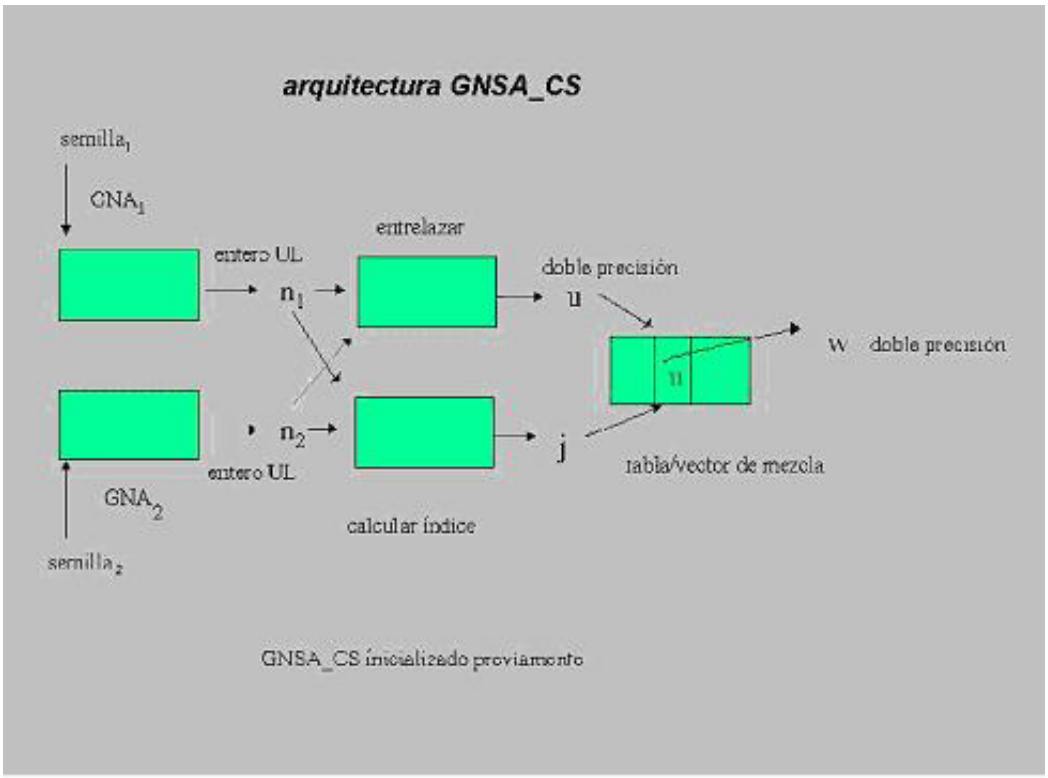
arquitectura GNSA_CS
semilla1
GNA1
entero UL
n1
entrelazar
doble precisión
u
v
w doble precisión
n2
entero UL
j
tabla/vector de mezcla
GNA2
calcular índice
semilla2
GNSA_CS inicializado previamente

Fig.3

6. Notas de implementación

6.1. longitud del bloque

Dado que las transformaciones del algoritmo no están limitadas en sus dimensiones, incluso las que son matriciales, ver párrafo siguiente, es posible tomar como longitud de los bloques un mismo valor para todos ellos, de utilidad cuando por ejemplo se desea cifrar en modo encadenado CBC; o bien definir una longitud individual por cada bloque que se cifra, aspecto que conduce a la necesidad transmitir estas longitudes. Una ventaja de este último esquema es que no se requiere rellenar con 0 y 1 -padding- el texto natural para alcanzar el tamaño del último bloque.

6.2. D^+ D^-

Los productos matriciales indicados se pueden realizar sencillamente sin necesidad de almacenar las matrices. En efecto:

Sea D^+ una matriz binaria triangular superior de dimensión *n*. X es un vector binario de dimensión *n*.

Entonces el producto matricial $Y = D^+ \otimes X$, donde la operación matricial $\otimes$ es la extensión natural de las operaciones binarias $\otimes$ y $\oplus$ en el campo GF(2), se puede expresar como la recurrencia siguiente:

$$y_k = y_{k+1} \oplus x_k \quad \text{con } y_{n+1} = 0, \text{con } \; k = n,...,1$$

De manera similar el producto $X = (D^+)^{-1}Y$ necesario para el proceso de descifrado es:

$$x_k = y_{k+1} \oplus y_k \quad \text{con } y_{n+1} = 0, \text{con } \; k = n,...,1$$

Para la operaciones matriciales con la matriz binaria triangular inferior D^- , $Y = D^- \otimes X$ y $X = (D^-)^{-1}Y$ de manera equivalente se tienen:

$$y_k = y_{k-1} \oplus x_k \quad \text{con } y_0 = 0, \text{con } \; k = 1,..., n$$

y para la inversa

$x_k = y_{k-1} \oplus y_k \quad \text{con } y_0 = 0, \text{con } \; k = 1,...,n$

6.3. AmasZ

Las dos operaciones del algoritmo

A ← F(azar)
Y ← A⊕Z2

se pueden realizar de forma conjunta.

Para un cierto vector binario *b* de longitud *n*, en el conjunto de los 2^n vectores del espacio se puede establecer una partición donde cada clase de equivalencia $A_k(b)$ contiene los vectores binarios que están a una distancia *k* del vector *b*. Esto es contiene a todos los vectores que difieren de él en *k* bits, empleando la métrica de Hamming.

Dado un vector binario cualesquiera $b \in \{0,1\}^n$ la máxima difusión para ese vector se obtiene con alguna transformación T cuya imagen $T(b) \in A_{n/2}$.

De donde la construcción al azar del vector A se hará de manera tal que la mitad de sus bits difieran de los de Z2. En promedio el vector resultante estará a distancia *n/2*. De esa manera Y será un vector con *n/2* componentes idénticos a los de Z2, y los otros *n/2* componentes serán los valores complementarios de los de Z2.

El algoritmo AmasZ resume esas consideraciones. Como un punto adicional cabe señalar que este algoritmo es su propio inverso.

Algoritmo AmasZ
/* obtener un vector *y* aleatorio, en promedio, a distancia Hamming *n/2* del vector *z**/

Entradas: *n, z, semilla*
/ n dimensión del vector, z vector binario, semilla valor inicial del GNSA_CS unif */*

Proceso
For i=1,n do
    /* generar un valor *u* aleatorio uniforme [0,1] */
    u *unif([0,1],semilla)*
    /* cambiar al azar el i-esimo componente de z */
    si(u ≤ 0.5) y(i) z(i)
      caso contrario y(i) xor(z(i),1)
endfor
return y(1),…,y(n)

Salidas: *y*

6.4. Permutación aleatoria

El método de intercambio es uno de los más eficientes computacionalmente para muestrear valores aleatorios sin reemplazo de una población finita, ver [7] Pastoriza. Cuando el tamaño de la muestra coincide con el tamaño de la población naturalmente se obtiene una permutación aleatoria.

Supongamos que los objetos a permutar están dados en forma de un arreglo a(1),…,a(n). Se toma la permutación identidad como el arreglo base de la población.

Con esa estructura se pueden intercambiar los objetos, entonces el muestreo aleatorio es simple. Basta elegir un objeto uniformemente al azar, dentro de los no seleccionados al momento, e intercambiarlo con el último objeto disponible.

En la práctica se permutan los índices del arreglo de índices y luego el de los objetos por ellos señalados.

Algoritmo Permutación aleatoria

Entradas: *n, semilla*
/* obtener una permutación aleatoria del conjunto {1,2,…,n} */
/* *n* tamaño del arreglo; *semilla* valor inicial del GNSA_CS unif */

Proceso
/* Establecer la permutación identidad */
For i=1,n do
    a(i) ← i
endfor

/* Intercambiar valores */
For i=n,1,-1 do
    u ← unif([0,1],semilla)
    *j ← u*i+1*
    x ← a(j)
  a(j) ← a(i)
  a(i) ← x
endfor
return a(1),…,a(n)

Salidas: *a(1),…,a(n)*
/* permutación de los índices */

6.5. Expansión de la clave

La figura Fig.4 da una visión esquemática del algoritmo.

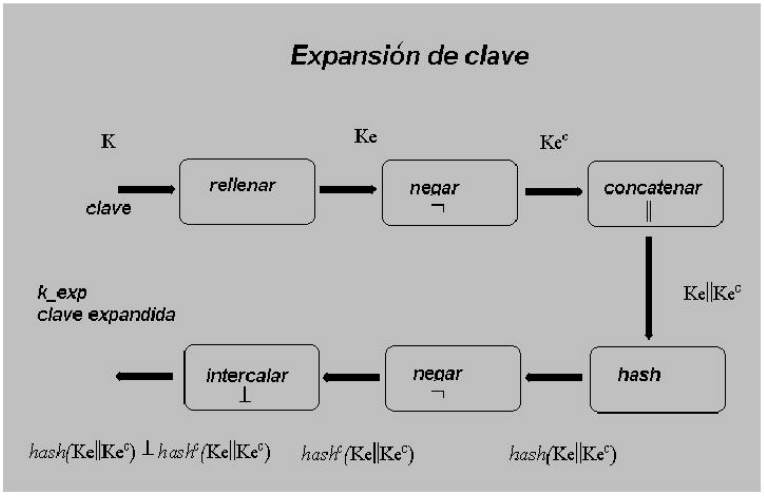

Fig.4

Donde hash es una primitiva criptográfica oneway, por ejemplo SHA512 o Whirpool. Y las restantes $\neg$, $\|$, $\perp$ son operaciones entre vectores binarios.

6.6. Extracción de semillas

La extracción de las semillas necesarias para inicializar GNSA_CS se realizará como se dijo antes mediante un muestreo aleatorio de la clave expandida *k_exp*.

La rutina que implementa el muestreo aleatorio, al igual que en la permutación aleatoria, será el método del intercambio.

Como fuente de azar para el muestreo aleatorio se utilizará un GNSA

especifico. Aquí se sugiere utilizar la función ran2 de [8] Press y otros. ran2 combina un congruencial lineal de Park y Miller con un SHR3, las secuencias de corrimiento de bits de Marsaglia, y como tal es un generador irreversible con buenas propiedades aleatorias.

Ciertas pruebas empíricas, prueba del cupón, realizadas con un congruencial lineal sugieren que se requieren alrededor de 25 números consecutivos para obtener una primera colección de dígitos. Valor acorde con el teórico. En consecuencia se sugiere emplear como mínimo 25/50 valores para calentar el generador GNSA.

6.7. GNSA_CS

La arquitectura diseñada requiere disponer de dos GNSA. Aquí se sugiere adoptar [3] Kiss99 de Marsaglia. Es simple, eficiente, con excelentes propiedades aleatorias, combina al sumar un congruencial lineal, un SHR3 de corrimiento de registros y dos multiplicaciones con acarreo. Y como tal es un generador irreversible.

La distribución del tiempo de permanencia en una tabla de mezcla sigue la ley geométrica

$$\text{Prob}[X=k] = p\,(1-p)^{k-1} \text{ con } k=1,2,.....$$

Con *p* la probabilidad de elegir una celda, todas igualmente probables.

Si el tamaño de la tabla de mezcla de los enteros generados es por lo menos 57, o sea $p \approx 0.0175$, entonces un valor bajo conduce a un árbol aleatorio extenso ante un análisis retrospectivo de los números eventualmente capturados.

La construcción de *w* un número en doble precisión es la descripta en [5] Matsumoto y Nishima. Extraer 27 bits de un entero *n1* con 32 bits y 26 bits en el otro *n2*; y hacer los corrimientos necesarios para conformar 53 bits propios de la mantisa de un doble precisión en el formato IEE754.

6.8.Modos de operación

Se pueden implantar tanto un modo de operación derivado de tomar igual longitud de bloque, que permite su modo encadenado CBC; o bien adoptar longitudes distintas.

Independientemente de lo anterior se puede recurrir a implantar el modo *paranoico*, esto es reinicializar el azar cada vez que se trata un bloque. Hecho que se puede realizar ya que los bits extraídos al azar de *k_exp* pasan las pruebas de aleatoriedad por construcción. Modo este inspirado en el hecho que, el eventual descifrado de un bloque, no proporcione información sobre el descifrado de los otros bloques ya que la fuente del azar es otra distinta e independiente de la del eventual bloque descifrado.

7. Conclusiones

Se propone una arquitectura genérica para cifradores aleatorios por bloques. Se introduce ⊕P⊕ algoritmo de cifrado por bloques de naturaleza aleatoria. También un GNSA_CS, que debe ser sometido a criptoanálisis para comprobar su alegada fortaleza criptográfica.

La fortaleza criptográfica del algoritmo ⊕P⊕ es un punto abierto y queda como un tópico a estudiar. El análisis preliminar realizado y las características incorporadas en el diseño permiten ser optimista en cuanto a las fortalezas criptográficas de ambas construcciones.

Dado que en este algoritmo simétrico no hay restricciones acerca del tamaño de los bloques que se deseen cifrar se pueden utilizar con longitudes de bloque grandes, por ejemplo 64/128/256/512/1024 bits u otras desiguales.

Referencias

[1] A. Briceño, Herramienta para la cuantificación de la difusión en algoritmos de encriptamiento simétrico por bloque. TEG. UCV, Facultad de Ingeniería, Escuela de Ingeniería Eléctrica, Postgrado en Comunicaciones y Redes de Comunicación de Datos. 2006.

[2] JENKINS, B. 1996. ISAAC. in fast software encryption. In *Proceedings of the 3rd International Workshop*. Cambridge, UK, D. Gollmann, Ed. Lecture Notes in Computer Science, vol. 1039. Springer-Verlag, 41–49. http://burtleburtle.net/bob/rand/isaacafa.html.

[3] G. Marsaglia, , KISS_f90.htm

[4] G. Marsaglia, y W. W Tsang, Some difficult-to-pass tests of randomness.

[5] M.Matsumoto, y T.Nishimura, Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulations. 1998.*

[6] A. MENEZES, P. van OORSCHOT, S. VANSTONE(1996), *Handbook*

of Applied Cryptography, CRC Press, New York, chap. 7.

[7] R. A. PASTORIZA, Muestreo sin reemplazo, RT 01-2005, UCV, Facultad de Ciencias, Escuela de Computación, 2005.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery(1996), *Numerical Recipes in Fortran 90. The Art of Parallel Scientific Computing, Volume 2 of Fortran Numerical Recipes*, Cambridge University Press, New York, chap. B7.