Secuencias pseudoaleatorias de números

Este documento no pretende ser un curso exhaustivo, en el mejor de los casos, únicamente puede considerarse como un conjunto de notas complementarias en alguna asignatura. Por supuesto, todo documento es susceptible de mejora, cualquier sugerencia o comunicación de error u omisión será bienvenida.

Una aproximación para el cifrado en flujo considera la generación de una secuencia de números aleatorios como clave que se combina con el mensaje (xor u otra función). Esta aproximación no es tan distinta de aproximaciones basadas en LFSRs si se considera que el generador de la secuencia pseudoaleatoria que constituirá la clave no se genera bit a bit sino en bloques de un determinado número n de bits (y por lo tanto un número acotado por 2^{n-1}).

El proyecto ECRYPT de la Unión Europea planteaba el desarrollo de sistemas de cifrado en flujo (eSTREAM Portfolios) para su implementación bien software (Portfolio 1) o hardware (Portfolio 2). El Proyecto eStream incluye Salsa20/12 como uno de los cuatro sistemas ganadores incluídos en su Portfolio 1.

Salsa20

Método propuesto por Daniel J. Bernstein que plantea su método como una larga secuencia de operaciones sencillas en lugar de una secuencia corta de operaciones complejas. Las operaciones utilizadas (suma módulo 32, xor y desplazamientos cíclicos) buscan impedir predicción y dispersar pequeñas modificaciones entre los bits de mayor y menor orden.

Bernstein argumenta que su propuesta permite obtener el mismo grado de dispersión obtenido por otros métodos pero sin la necesidad de utilizar hardware dedicado. El sistema está pensado para (intentar) reducir las filtraciones que provocaría el uso de operaciones complejas o tablas de búsqueda (S-boxes) que podrían utilizarse en ataques de canal lateral. El sistema utiliza una clave de 128 o 256 bits, y utiliza dos nonces para generar bloques pseudoaleatorios de 512 bits. El autor descarta el uso de bloques mayores para no reducir el rendimiento. El uso de dos nonces, (uno en realidad es un contador de bloque) permite obtener una determinada fracción de la secuencia sin necesidad de generar toda ella desde el principio.

Todo el esquema se articula en torno a una función Salsa (que el autor describe como de resumen) que considera la clave y los dos nonces utilizados en el proceso de generación. El bloque resultante es combinado con la información utilizada como entrada. La Figura 1 ilustra este proceso.

La implementación de la función Salsa20 recibe la clave:

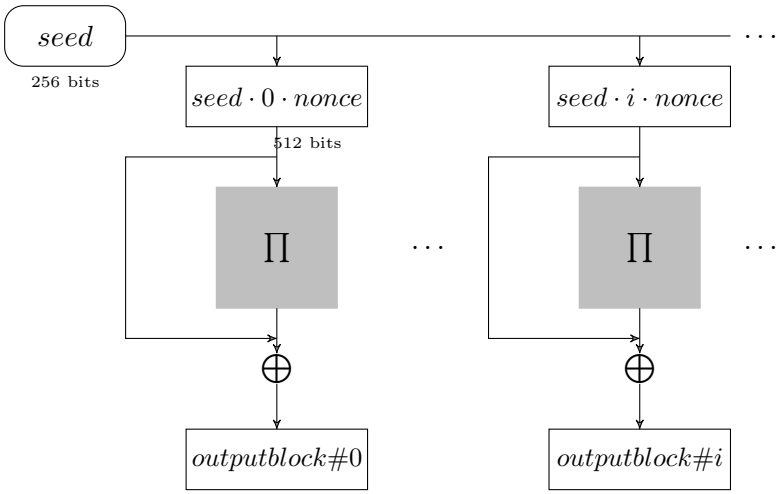

Figura 1: Salsa20 como generador pseudoaleatorio de números.

$$k_0, k_1, \ldots, k_7 \in \{0,1\}^{32}$$

el nonce y el contador de bloque:

$$\begin{array}{ll} j_0, j_1 \in \{0,1\}^{32} & \text{Contador de posición} \\ n_0, n_1 \in \{0,1\}^{32} & \text{Nonce} \end{array}$$

y cuatro constantes:

$$\begin{array}{ll} c_0 = 61707865_{HEX} & c_1 = 3320646E_{HEX} \\ c_2 = 79622D32_{HEX} & c_3 = 6B206574_{HEX} \end{array}$$

Para describir de forma intuitiva las operaciones es interesante considerar toda esta información dispuesta en una matriz cuadrada, pero nombrando cada una de las celdas de acuerdo con un esquema particular:

$$\omega = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \Longleftarrow \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & n_0 & n_1 \\ j_0 & j_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}$$

de modo que, por ejemplo, x_8 hace referencia inicialmente a los bits menos significativos del contador de bloque. El proceso se articula en 10 dobles rounds que modifican el contenido de cada celda de la matriz. El proceso de modificación se implementa en *cuartos de round* que consideran distintas partes de la matriz en cada llamada.

Algoritmo 1 Salsa20

Entrada: $x_1x_2x_3 \dots x_{15} \in \{0,1\}^{512}$ //$x_i \in \{0,1\}^{32}$
Entrada: Salsa20 $QuarterRound(a, b, c, d, t)$
1: **Método**
2: **Para** $i = 1$ **hasta** 10 **hacer**
3: $QuarterRound(x_0, x_4, x_8, x_{12}, 1)$ //Round impar
4: $QuarterRound(x_1, x_5, x_9, x_{13}, 2)$
5: $QuarterRound(x_2, x_6, x_{10}, x_{14}, 3)$
6: $QuarterRound(x_3, x_7, x_{11}, x_{15}, 4)$
7: $QuarterRound(x_0, x_1, x_2, x_3, 1)$ //Round par
8: $QuarterRound(x_4, x_5, x_6, x_7, 2)$
9: $QuarterRound(x_8, x_9, x_{10}, x_{11}, 3)$
10: $QuarterRound(x_{12}, x_{13}, x_{14}, x_{15}, 4)$
11: **FinPara**
12: **FinMétodo.**

El Algoritmo 1 muestra el proceso. Intuitivamente, Salsa20 recorre la matriz considerando primero las columnas (rounds impares) y posteriormente considerando las filas (rounds pares). La modificación de las celdas se realiza en cada uno de los cuartos de round (Algoritmo 2) que considera exclusivamente operaciones de suma módulo 32 (denotadas por $\boxplus$), xor (denotadas por $\oplus$) y desplazamiento circular (denotadas por $<<<$). El quinto parámetro permite un tratamiento circular de los parámetros en cada llamada.

Algoritmo 2 QuarterRound de Salsa

Entrada: $a, b, c, d \in \{0,1\}^{32}$
Entrada: $t \in \{1, 2, 3, 4\}$
1: **Método**
2: Ejecuta las operaciones de forma circular empezando por la t-esima.
3:
(i) $b \oplus = (a \boxplus d) <<< 7;$
(ii) $c \oplus = (b \boxplus a) <<< 9;$
(iii) $d \oplus = (c \boxplus b) <<< 13;$
(iv) $a \oplus = (d \boxplus c) <<< 18;$

El comité eSTREAM sugiere el uso de Salsa12 (seis iteraciones en lugar de las 10 del bucle en el Algoritmo 1). Con objeto de equilibrar las necesidades de seguridad y velocidad se han propuesto variantes con menor número de iteraciones, aunque el autor encuentra que 20 iteraciones es un compromiso de equilibrio entre ambas medidas. Actualmente no se conocen ataques eficientes a versiones *mayores* de este metodo.

ChaCha20

Versión de Salsa20 que propone un nuevo cuarto de round que busca incrementar la dispersión. Mantiene la misma estructura de Salsa20, pero con un nonce mayor (que pasa a ser de 96 bits) un contador de bloque menor (de 32 bits) y distinta disposición incial de la clave y las constantes, todas ellas del mismo tamaño:

$$\omega = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \Longleftarrow \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ j_0 & n_0 & n_1 & n_2 \end{pmatrix}$$

Algoritmo 3 ChaCha20

Entrada: $x_1x_2x_3 \dots x_{15} \in \{0,1\}^{512}$ $//x_i \in \{0,1\}^{32}$
Entrada: ChaCha20 $QuarterRound(a,b,c,d)$
1: **Método**
2: **Para** $i = 1$ **hasta** 10 **hacer**
3: $QuarterRound(x_0, x_4, x_8, x_{12})$ //Rounds impares
4: $QuarterRound(x_1, x_5, x_9, x_{13})$;
5: $QuarterRound(x_2, x_6, x_{10}, x_{14})$
6: $QuarterRound(x_3, x_7, x_{11}, x_{15})$;
7: $QuarterRound(x_0, x_5, x_{10}, x_{15})$ //Rounds pares
8: $QuarterRound(x_1, x_6, x_{11}, x_{12})$;
9: $QuarterRound(x_2, x_7, x_8, x_{13})$
10: $QuarterRound(x_3, x_4, x_9, x_{14})$;
11: **FinPara**
12: **FinMétodo.**

El Algoritmo 3 muestra el proceso estandar (RFC-8439). De nuevo puede verse el proceso como un recorrido de la matriz utilizando distintas celdas en cada llamada a un cuarto de round. En este caso, ChaCha20 recorre la matriz considerando primero las columnas (al igual que Salsa20) y posteriormente considerando las diagonales (rounds pares). Cada llamada a un cuarto de round modifica cada celda dos veces (Salsa20 modificaba cada celda una única vez). El Algoritmo **??** describe el cuarto de round, que de nuevo, considera exclusivamente operaciones de suma módulo 32, xor y de desplazamiento circular.

La adopción de ChaCha20 por parte de Google como sutituto de RC4 en TLS ha hecho que se proponga como estandar en IKE e IPsec (RFC 7634). Se ha publicado la propuesta de estandar para TLS (RFC-7905).

Algoritmo 4 QuarterRound de ChaCha

Entrada: $a, b, c, d \in \{0,1\}^{32}$

1: **Método**
2: $a \boxplus = b$
3: $d \oplus = a$
4: $d <<< 16$
5: $c \boxplus = d$
6: $b \oplus = c$
7: $b <<< 12$
8: $a \boxplus = b$
9: $d \oplus = a$
10: $d <<< 8$
11: $c \boxplus = d$
12: $b \oplus = c$
13: $b <<< 7$
14: **FinMétodo.**