# Análisis e Implementación del Generador SNOW 3G Utilizado en las Comunicaciones 4G

J. Molina-Gil, P. Caballero-Gil
Departamento de Informática
Universidad de La Laguna
Email: jmmolina@ull.edu.es, pcaballe@ull.edu.es

A. Fúster-Sabater
Instituto de Física Aplicada.
Consejo Superior de Investigaciones Científicas
Email: amparo@iec.csic.es

***Resumen*—La cuarta generación de telefonía móvil, comercializada como tecnología 4G, y conocida como LTE (Long Term Evolution) o su versión mejorada LTE-A (LTE-Advanced), se está implantando con rapidez por todo el mundo. Teniendo en cuenta su carácter inalámbrico y móvil, la seguridad de estas comunicaciones resulta crucial. En este trabajo se presenta un estudio teórico y un análisis práctico del generador SNOW 3G, que constituye el núcleo de la integridad y la confidencialidad de las comunicaciones 4G. El objetivo es evaluar la implementación y funcionamiento de este generador en dispositivos móviles con el fin de proponer mejoras, principalmente en cuanto a eficiencia.**

***Palabras clave*—Criptografía de clave secreta (*Secret key cryptography*), Generador pseudoaleatorio (*Pseudorandom generator*), Cifrado en flujo (*Stream cipher*), SNOW 3G, 4G, LTE, LTE-A.**

## I. Introducción

El aumento en el consumo de datos móviles, y la aparición de numerosas aplicaciones y servicios de banda ancha son, junto a debilidades detectadas en la seguridad de las comunicaciones, los principales motivos que han conducido a la progresiva sustitución de la tecnología 3G o UMTS por la nueva tecnología 4G, conocida como LTE o LTE-A. El completo despliegue de las redes LTE comerciales en España es inminente, encontrándose actualmente en funcionamiento solo en algunas grandes ciudades. En cuanto a otros países, ya se ha implantado la tecnología 4G en: cuatro países de África, once de América, diecinueve de Asia, veintinueve de Europa, y dos de Oceanía.

En general, la evolución de cualquier sistema de telecomunicaciones suele implicar la mejora de sus características de seguridad gracias al aprendizaje a partir de las debilidades y los ataques sufridos por sus predecesores. Concretamente, la evolución de los sistemas de cifrado utilizados para proteger la confidencialidad en telefonía móvil se puede resumir de la forma siguiente. En primer lugar, el cifrado en flujo A5/1 y su versión A5/2 fueron utilizados en el estándar de telefonía 2G o GSM. En ambos cifrados se detectaron serias debilidades, razón por la cual el sistema de cifrado en bloque conocido como Kasumi los sustituyó en la tecnología 3G o UMTS. En 2010, el cifrado Kasumi fue atacado y roto con recursos computacionales muy modestos y en consecuencia, el sistema de cifrado tuvo que ser modificado de nuevo para la tecnología del nuevo estándar 4G o LTE, de manera que fue el cifrado en flujo SNOW 3G el que se propuso para la protección de la confidencialidad e integridad de las comunicaciones.

El objetivo de este trabajo es realizar un análisis del generador SNOW 3G, que constituye el núcleo tanto del algoritmo de confidencialidad UEA2 en LTE (EEA1 en LTE-A), como del de integridad UIA2 en LTE (EIA1 en LTE-A), [1]. Este generador proporciona una gran velocidad en la generación de datos, lo que lo hace en general muy apropiado para su uso en dispositivos con recursos limitados.

El contenido de este trabajo se organiza del siguiente modo. En la Sección II se incluye un resumen de trabajos relacionados con la temática. Los principales conceptos y notaciones utilizados se introducen en la Sección III. La Sección IV presenta algunos detalles de la implementación llevada acabo en la plataforma iOS, así como una evaluación de su rendimiento. Finalmente, la Sección V cierra este estudio con algunas conclusiones y trabajos futuros.

## II. Estado del Arte

Los predecesores del SNOW 3G fueron el SNOW 1.0 [2] y el SNOW 2.0 [3]. La versión original, SNOW 1.0, fue presentada al proyecto NESSIE, pero pronto se detectaron algunas debilidades y se lanzaron diversos ataques. Uno de ellos [4], con complejidad computacional de $O(2^{224})$, se basó en la posibilidad de recuperar la clave con solo conocer una salida del generador de longitud $2^{95}$. Otro criptoanálisis con similar complejidad computacional fue el ataque por diferenciación (distinguishing attack) [5], que también requería de una salida de tamaño $2^{95}$ .

Estos y otros ataques demostraron la existencia de algunas debilidades en el diseño del generador SNOW 1.0, por lo que surgió una nueva versión, más segura, denominada SNOW 2.0. Este es hoy en día uno de los dos cifrados en flujo elegidos para el estándar ISO/IEC IS 18033-4 [6]. Este generador se basa en unos principios de diseño similares a los del cifrado en bloque SOSEMANUK, que es uno de los cuatro cifrados finalistas dentro del perfil software seleccionados para el eSTREAM Portfolio [7].

Posteriormente, durante su evaluación por el European Telecommunications Standards Institute (ETSI), el diseño del SNOW 2.0 fue modificado para aumentar su resistencia a ataque algebraicos [8] de forma que dicha modificación dio lugar al SNOW 3G. El ETSI publicó un informe técnico acerca de su diseño [9], pero hasta el momento no se ha hecho pública ninguna evaluación completa del diseño del SNOW 3G.

ISBN: 978-84-9717-323-0

En las publicaciones existentes, el SNOW 3G se ha revelado como un generador con una gran resistencia a ataques por diferenciación lineal [10] [11], pero débil frente a otro tipos de ataques. Uno de los primeros y más sencillos intentos de criptoanálisis fue el ataque propuesto en [12]. Otro ataque [13], basado en la sincronización de la caché y en datos de tiempo empíricos, permitió recuperar el estado inicial en cuestión de segundos y sin necesidad de conocer ningún bit. Este tipo de ataque se basa en el hecho de que operaciones como las permutaciones y multiplicaciones por la constante $\alpha$ y su inversa, son implementadas realmente utilizando tablas de búsqueda. El trabajo presentado en [14] describe un estudio del mecanismo de resincronización del SNOW 3G usando ataques por colisión, con una complejidad de $O(2^8)$. Finalmente, el generador SNOW 3G ha sido sujeto de otros estudios de complejidad, tales como las publicaciones [15] y [16]. En este trabajo se proporciona un nuevo estudio, enfocado hacia los aspectos más prácticos de su implementación.

## III. Descripción Teórica del Generador Snow 3G

Los cifrados en flujo están basados en generadores pseudo-aleatorios cuyos bits de salida son operados con los bits del texto en claro mediante una XOR, para generar bit a bit el texto cifrado. Su principal ventaja es que en general permiten obtener el texto cifrado a una gran velocidad, lo que los hacen especialmente apropiados para comunicaciones que requieran eficiencia, y para dispositivos con recursos limitados, como por ejemplo los teléfonos móviles ya que la comunicación debe ser inmediata y disponen de una batería limitada.

El generador en flujo analizado en este trabajo tiene un estructura típica de generador no lineal basado en un registro de desplazamiento con realimentación lineal o LFSR (Linear Feedback Shift Register).

Los siguientes términos y notación se utilizan en este documento para describir la estructura e implementación del generador SNOW 3G:

$GF(2) = \{0, 1\}$ Cuerpo de Galois con dos elementos, 0 y 1.

$GF(2)[x]$ Anillo de polinomios en una indeterminada $x$, con coeficientes en $GF(2)$.

$p(x)$ Polinomio primitivo en $GF(2)[x]$.

$d$ Grado de un polinomio $p(x)$.

$GF(2^d)$ Cuerpo extendido de $GF(2)$, definido por un polinomio $p(x)$ de grado $d$, y con $2^d$ elementos.

$GF(2^d)[x]$ Anillo de polinomios en una indeterminada $x$ con coeficientes en $GF(2^d)$.

$\beta \in GF(2^8)$ Raíz del polinomio $x^8 + x^7 + x^5 + x^3 + 1$ perteneciente a $GF(2)[x]$.

$\alpha \in GF(2^{32})$ Raíz del polinomio $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$ perteneciente a $GF(2^8)[x]$.

$s_t$ estado de 32-bits perteneciente a un LFSR.

$=$ Operador de asignación.

$\oplus$ Operación XOR bit a bit.

$\boxplus$ Suma de enteros módulo $2^{32}$.

$||$ Concatenación de dos operandos.

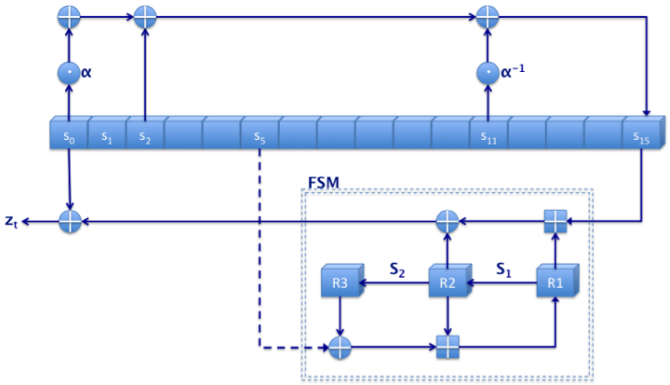


Figura 1. Generador SNOW 3G

La figura 1 muestra la estructura del generador SNOW 3G. Como puede verse, consta de dos partes principales: un LFSR y una máquina de estados finitos o FSM (Finite State Machine).

El LFSR consta de 16 estados $s_0, s_1, s_2, ..., s_{15}$, de forma que cada uno de ellos contiene 32 bits. La función de realimentación se define mediante un polinomio primitivo definido sobre un cuerpo extendido $GF(2^{32})$, e implica dos multiplicaciones, una por una constante $\alpha \in GF(2^{32})$ y otra por la inversa de $\alpha$. A continuación se muestra la expresión de dicha realimentación:

$$s_{t+16} = \alpha s_t \oplus s_{t+2} \oplus \alpha^{-1} s_{t+11}, \forall t \geq 0 \qquad (1)$$

La FSM constituye la parte no lineal del generador y se alimenta de dos valores de entrada procedentes del LFSR correspondientes a los estados $s_5$ y $s_{15}$. La FSM está formada por tres registros de 32 bits, R1, R2 y R3, y dos cajas de sustitución o S-boxes, S1 y S2, que se utilizan para actualizar los registros R2 y R3. Las dos S-Boxes S1 y S2 mapean los 32 bits de entrada a 32 bits de salida mediante la aplicación de varias combinaciones de una S-box básica sobre cada uno de los 4 bytes de entrada. La estructura de la caja S1 se basa en la S-box utilizada en el cifrado estándar AES (Advanced Encryption Standard), mientras que la S-box S2 fue especialmente diseñada para el SNOW 3G. Por último están las operaciones de mezcla que se utilizan en la FSM, donde se realiza una XOR bit a bit, y una suma de enteros módulo $2^{32}$.

El LFSR base del generador SNOW 3G puede usarse en dos modos de operación diferentes: de inicialización y de generación de secuencia cifrante. Por una parte, cuando opera en modo de inicialización, el generador se desplaza en cada pulso de reloj sin producir ninguna salida. Por otra parte, en modo de generación, tras cada pulso de reloj el generador se desplaza y produce una salida de una palabra de 32 bits. De hecho, se puede decir que el SNOW 3G es un generador orientado a palabras ya que produce una secuencia de salida de 32 en 32 bits, bajo el control de una clave de 128 bits y un vector de inicialización o IV (Initialization Vector) de 128 bits.

Con respecto a la implementación del SNOW 3G, que es el principal objeto de estudio de este trabajo, se pueden hacer varias observaciones.

En primer lugar, las dos multiplicaciones implicadas en el LFSR pueden ser implementadas como desplazamientos de bytes con una XOR con alguno de los $2^8$ patrones posibles, tal como se muestra a continuación. Dado que $\beta$ es la raíz del polinomio primitivo $x^8 + x^7 + x^5 + x^3 + 1$, el cuerpo extendido $GF(2^8)$ puede ser generado a partir de sucesivas potencias de $\beta$. Por tanto, el conjunto $\{0, 1, \beta, \beta^2, \beta^3, ..., \beta^{2^8-2}\}$ representa todo el cuerpo extendido $GF(2^8)$. De ahí tenemos que cualquier elemento de $GF(2^8)$ puede ser representado también mediante un polinomio en $GF(2)[x]$ de grado menor que 8, o bien con un byte cuyos bits se corresponden con los coeficientes de dicho polinomio. De esa manera, las operaciones en $GF(2^8)$ se corresponden con operaciones módulo el polinomio $x^8+x^7+x^5+x^3+1$. Esto significa que en particular, la multiplicación de dos elementos en $GF(2^8)$ resulta de la multiplicación de los dos polinomios correspondientes, posteriormente dividida por el polinomio $x^8 + x^7 + x^5 + x^3 + 1$ de manera que el resto es la salida resultante. La implementación de esta operación como una multiplicación binaria se describe a continuación. Teniendo en cuenta los dos bytes a multiplicar, para cada bit igual a uno en uno de los multiplicandos, se realizan varios desplazamientos a izquierda en el otro byte a multiplicar. Además, cada vez que el bit más a la izquierda del byte original antes del desplazamiento es 1, se realiza una XOR bit a bit con $A9_{16} = 10101001_2$, que es el byte correspondiente al polinomio $x^8+x^7+x^5+x^3+1$. El número de desplazamientos a izquierda a realizar viene dado por la posición de los bits iguales a 1 en el primer multiplicador.

Por otra parte, dado que $\alpha$ es una raíz del polinomio primitivo en $GF(2^8)[x]$, $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$, se puede generar el cuerpo extendido $GF(2^{32})$ como sucesivas potencias de $\alpha$, de forma que el conjunto $\{0, 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^{2^{32}-2}\}$ define todo el cuerpo $GF(2^{32})$. Por tanto se deduce que cualquier elemento de $GF(2^{32})$ puede ser representado mediante un polinomio en $GF(2^8)[x]$ de grado menor que 4, o bien con una palabra de 4 bytes correspondientes a los 4 coeficientes de dicho polinomio. De esa manera, las operaciones en $GF(2^{32})$ corresponden a operaciones con polinomios módulo $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$. Esto significa que en particular, la multiplicación de $\alpha$ por cualquier palabra de 4 bytes $(c_3, c_2, c_1, c_0) \in GF(2^{32})$ resulta de la multiplicación de $x$ por el polinomio $c_3x^3 + c_2x^2 + c_1x + c_0$, que es entonces dividida por el polinomio $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$, de forma que el resultado obtenido es el resto $(c_2+c_3\beta^{23})x[^3+(c_1+c_3\beta^{245})x^2+(c_0+c_3\beta^{48})x+c_3\beta^{239}$, o lo que es lo mismo, la palabra de 4 bytes $(c_2 + c_3\beta^{23}, c_1 + c_3\beta^{245}, c_0 + c_3\beta^{48}, c_3\beta^{239})$, $\forall c \in GF(2^8)$. De idéntica manera, la multiplicación de $\alpha^{-1}$ por cualquier palabra de 4 bytes $(c_3, c_2, c_1, c_0) \in GF(2^{32})$ resulta de la multiplicación de $x^{-1}$ por el polinomio $c_3x^3+c_2x^2+c_1x+c_0$, que es $c_3x^2 + c_2x + c_1 + c_0x^{-1}$. Como $xx^{-1} = 1$ y $\beta^{255} = 1, x^{-1}$ puede expresarse como $\beta^{255-239}x^3+\beta^{255-239+23}x^2+\beta^{255-239+245}x+\beta^{255-239+48} = \beta^{16}x^3+\beta^{39}x^2+\beta^6x+\beta^{64}$.

Tabla I
DISPOSITIVO UTILIZADO PARA LA EVALUACIÓN

| iPhone 3GS | | | |
|---|---|---|---|
| **Arquitectura** | **Frecuencia CPU** | **Cache L1I/L1D/L2** | **RAM** |
| Armv7-A | 600 MHz | 16Kb/16Kb/256Kb | 256MB |

Así, la salida resultante del producto es el resto $(c_0\beta^{16})x^3 + (c_3 + c_0\beta^{39})x^2 + (c_2 + c_0\beta^6)x + (c_1 + c_0\beta^{64})$, o lo que es lo mismo, la palabra de 4 bytes $(c_0\beta^{16}, c_3+c_0\beta^{39}, c_2+c_0\beta^6, c_1+c_0\beta^{64})$. En conclusión, una rápida implementación binaria de esta operación puede estar basada en tablas precalculadas de los valores $(c\beta^{16}, c\beta^{39}, c\beta^6, c\beta^{64})$, $\forall c \in GF(2^8)$.

## IV. IMPLEMENTACIÓN EN IOS Y EVALUACIÓN

Esta sección recoge un estudio y comparación de diferentes implementaciones software con el fin de concluir cuál es la implementación óptima en dispositivos con recursos limitados, como son los teléfonos móviles. Se ha analizado la implementación del SNOW 3G en plataformas móviles, en concreto para la plataforma iOS, siendo Objective C el lenguaje de programación utilizado. En particular, se han realizado diversos estudios en un iPhone 3GS, cuyas características principales se presentan en la Tabla I.

El primer aspecto a tener en cuenta es que los LFSRs han sido tradicionalmente diseñados para operar sobre el cuerpo de Galois $GF(2)$, lo que es muy apropiado para implementaciones hardware. Sin embargo, cuando se trata de implementaciones software, los LFSRs en los que cada etapa contiene un solo bit tienen una eficiencia menor. Teniendo en cuenta que la mayoría de los microprocesadores en teléfonos móviles tienen una longitud de palabra de 32 bits, la implementación del LFSR del SNOW 3G se espera que sea más eficiente que la de esos otros generadores, ya que en el SNOW 3G el LFSR se define sobre el cuerpo extendido $GF(2^{32})$. Teniendo esto en cuenta, se puede afirmar que el hecho de que la implementación del SNOW 3G se realice teniendo en cuenta el cuerpo $GF(2^{32})$, resultará más adecuado para la arquitectura que actualmente soportan los teléfonos móviles.

El segundo aspecto a considerar está relacionado con las operaciones aritméticas que se deben realizar en el generador, y específicamente, la multiplicación en el cuerpo extendido, porque la función de realimentación en el SNOW 3G implica diversas sumas y multiplicaciones, siendo la multiplicación la operación con mayor coste computacional.

Todos los resultados mostrados fueron obtenidos utilizando Instruments, que es un analizador y visualizador del rendimiento de aplicaciones en código OS X e iOS. Es una herramienta flexible y potente, que permite realizar el seguimiento de uno o más procesos y examinar los datos obtenidos.

Los datos proporcionados corresponden a una media de 10 ejecuciones de las pruebas realizadas, en las cuales se generaron $10^7$ bytes de secuencia cifrante utilizando la plataforma mencionada. La Tabla II muestra el tiempo total (en milisegundos) de cada función correspondiente a la implementación del SNOW 3G.

Tabla II
FUNCIONES CON MÉTODO RECURSIVO

| Resumen | | |
|---|---|---|
| **Función** | **Tiempo (ms)** | **%** |
| MULxPow | 29054,9 | 92,88 |
| ClockLFSRKeyStreamMode | 572 | 1,77 |
| DIValpha | 356,6 | 1,1 |
| main | 264,7 | 0,8 |
| MULalpha | 326,8 | 0,99 |
| GenerateKeystream | 243,8 | 0,73 |
| ClockFSM | 180,3 | 0,54 |
| S2 | 128,1 | 0,34 |
| S1 | 129,9 | 0,37 |
| Generator | 1,3 | 0 |
| **Total** | 30258,5 | 100 |

Tabla III
FUNCIONES CON TABLAS PRECOMPUTADAS

| Coste Computacional | | |
|---|---|---|
| **Función** | **Tiempo (ms)** | **%** |
| ClockLFSRKeyStreamMode | 347,3 | 28,69 |
| main | 277,2 | 22,35 |
| ClockFSM | 182,2 | 14,95 |
| S1 | 146 | 12,01 |
| S2 | 138 | 11,3 |
| GenerateKeystream | 107,3 | 8,84 |
| Generator | 1,3 | 0,04 |
| **Total** | 1199,4 | 100 |

Los resultados muestran claramente que la multiplicación es la función más costosa. La segunda función más costosa es el desplazamiento del LFSR, que se realiza en cada pulso de reloj. A continuación se presentan dos técnicas diferentes para llevar a cabo la multiplicación, así como diferentes formas de implementar el desplazamiento del LFSR propuestas en [11], con el objetivo de descubrir cuál es la implementación óptima.

### *IV-A. Multiplicación*

Tras realizar la implementación del SNOW 3G tal como está propuesta en [1], la Tabla II muestra los resultados de tiempo consumido por cada función. Como puede verse, la función MULxPow utilizada para multiplicar por $\alpha$ y por $\alpha^{-1}$ es la que más tiempo consume.

En la implementación, la multiplicación puede ser programada como una serie de desplazamientos de bytes recursivos y XORs condicionales, o bien como una búsqueda en un tabla con resultados precalculados. Además, en cada pulso de reloj del LFSR, el polinomio de realimentación usa dos funciones $MUL_\alpha$ y $DIV_\alpha$, que se definen como se muestra a continuación:

$$MUL_\alpha = \\ MUL_xPOW(c, 23, 0xA9)||MUL_xPOW(c, 245, 0xA9) \\ MUL_xPOW(c, 48, 0xA9)||MUL_xPOW(c, 239, 0xA9)$$

$$DIV_\alpha = \\ MUL_xPOW(c, 16, 0xA9)||MUL_xPOW(c, 39, 0xA9) \\ MUL_xPOW(c, 6, 0xA9)||MUL_xPOW(c, 64, 0xA9)$$

La primera propuesta de implementación de la multiplicación resulta más apropiada para sistemas con recursos de memoria limitados, ya que no requiere de espacio de almacenamiento. Sin embargo, como se puede ver en la Tabla II, esta implementación tiene un coste computacional significativo.

La segunda propuesta, que implica el uso de tablas precalculadas, proporciona resultados óptimos, tal como se puede ver en la Tabla III. Tal como muestran los resultados, esta implementación se puede considerar la más rápida para la multiplicación y supone una mejora del 96 % respecto al método recursivo en cuando a tiempo consumido. Sin embargo, uno de los mayores problemas de esta propuesta es la necesidad de almacenamiento, lo que puede ser un problema en dispositivos con recursos limitados. En particular, para SNOW 3G, la tabla consta de 256 elementos de 32 bits cada uno, lo que supone un total de 32*256 bits. Además, la implementación implica dos tablas, una para la función $MUL_\alpha$ y otra para $DIV_\alpha$ lo que significa un total de 2048 bytes. Esta cantidad no supone un gran problema dadas las características de los teléfonos móviles actuales, por lo que por eficiencia, la conclusión obtenida es que este método parece bastante adecuado para los dispositivos analizados.

### *IV-B. LFSR*

Las estructuras de los LFSRs son en general bastante difíciles de implementar en software de forma eficiente. La razón principal es el desplazamiento de los 16 estados en cada pulso de reloj. Este desplazamiento en una implementación hardware, se realiza de manera simultánea por lo que el proceso completo puede ser realizado en un simple pulso de reloj. Sin embargo, en una implementación software, el proceso es iterativo y por lo tanto costoso.

Como se ve en la Tabla III, una vez optimizada la multiplicación, la función ClockLFSRKeyStreamMode es la que más tiempo consume. Con el fin de buscar una mejora de los tiempo consumidos por esta función hemos utilizado diferentes técnicas de optimización descritas en [17], junto con técnicas secuenciales (hardcode) propuestas en la especificación del SNOW 3G.

El método hardcode consiste en la incorporación de los datos directamente en el código fuente en vez de utilizar bucles e índices como hacen el resto de propuestas. El coste de este método de hardcode se corresponde con 15 asignaciones secuenciales. Esta técnica, a pesar de ser más larga, parece que requiere menos tiempo. A continuación se muestra la implementación de este método.

```
void ClockLFSRKeyStreamMode()
 u32 v = (((LFSR_S0 << 8) & 0xffffff00 )^
 (MULalpha((u8)((LFSR_S0>>24) & 0xff))) ^
 (LFSR_S2) ^
 ((LFSR_S11 >> 8) & 0x00ffffff)
```

Tabla IV
EJECUCIÓN CON DIFERENTES MÉTODOS DE IMPLEMENTACIÓN PARA EL LFSR

| Funciones | Tradicional | HardCode | Búfer Circular | Ventanas Deslizantes | Desdoblamiento de Bucles |
|---|---|---|---|---|---|
| ClockLFSRKeyStreamMode | 491,1 | 342,5 | 834 | 184,1 | 291,3 |
| Generator | 1,4 | 1,3 | 1,3 | 1,8 | 1,1 |
| GenerateKeystream | 65,8 | 65,8 | 198,3 | 88,2 | 68,4 |
| main | 246,6 | 306,8 | 296,8 | 297 | 294,4 |
| **Tiempo total** | 804,9 | 716,4 | 1330,4 | 571,1 | 655,2 |

```
(DIValpha((u8)((LFSR_S11) & 0xff)))
);
LFSR_S0 = LFSR_S1;
LFSR_S1 = LFSR_S2;
LFSR_S2 = LFSR_S3;
LFSR_S3 = LFSR_S4;
LFSR_S4 = LFSR_S5;
LFSR_S5 = LFSR_S6;
LFSR_S6 = LFSR_S7;
LFSR_S7 = LFSR_S8;
LFSR_S8 = LFSR_S9;
LFSR_S9 = LFSR_S10;
LFSR_S10 = LFSR_S11;
LFSR_S11 = LFSR_S12;
LFSR_S12 = LFSR_S13;
LFSR_S13 = LFSR_S14;
LFSR_S14 = LFSR_S15;
LFSR_S15 = v;
```

El análisis llevado a cabo incluye una comparación de diferentes técnicas de implementación de un LFSR. Para ello se ha utilizado el método basado en tablas de multiplicaciones precalculadas descrito como óptimo en la sección anterior, y se realiza cada experimento para $10^7$ bytes de salida generados por el LFSR del SNOW 3G.

Los valores obtenidos se resumen en la Tabla IV, que muestra el tiempo consumido por las funciones que implican el desplazamiento del LFSR y sus operaciones, así como el tiempo total consumido por cada una de las diferentes propuestas.

Los resultados muestran que el método de hardcode propuesto en la especificación del SNOW 3G no es la mejor opción. Aunque representa un 11 % de mejora sobre el método tradicional, es el método de ventanas deslizantes el que presenta los mejores resultados, con un 29 % de mejora respecto al método tradicional y un 20 % respecto a la propuesta de hardcode. Por otra parte, la peor propuesta es la de búfer circular. Como puede deducirse de los tiempos obtenidos, el método no es aplicable debido a que la actualización de diferentes índices implica el uso de aritmética modular, lo que no resulta muy eficiente con ese método.

Por tanto, se podría concluir que una óptima implementación del LFSR implica el uso de tablas precalculadas para la multiplicación y la técnica de ventanas deslizantes para el desplazamiento del LFSR.

Sin embargo, esta nueva propuesta de implementación del LFSR puede afectar a otra parte del SNOW 3G como puede ser la FSM. Por esta razón, el principal objetivo ahora es determinar si la mejora en la implementación del LFSR puede afectar negativamente a otras partes del código, o bien concluir cuál es la mejora total que se puede llegar a alcanzar.

Tabla V
EJECUCIÓN EN MODO OPTIMIZADO

| Coste Computacional | | |
|---|---|---|
| **Función** | **Tiempo (ms)** | **%** |
| ClockLFSRKeyStreamMode | 184,7 | 15,28 |
| main | 282,3 | 22,23 |
| ClockFSM | 195,4 | 17,68 |
| S1 | 135,9 | 12,65 |
| S2 | 163,4 | 16,33 |
| GenerateKeystream | 118,2 | 10,95 |
| Generator | 1,2 | 1,07 |
| **Total** | 1081,1 | 100 |

Para analizarlo, se ha implementado el SNOW 3G con estas dos posibles mejoras. En la Tabla V se muestra el resumen de los resultados obtenidos. Si se comparan los resultados con los de la Tabla III, se puede ver claramente que esta implementación mejora los tiempos para las funciones $ClockLFSRKeyStreamMode, S1$ y $GenerateKey$. Sin embargo, otras funciones como $ClockFSM, S2, GenerateKeystream$ han aumentado ligeramente sus tiempos. La función con el peor resultado es $S2$, que ha incrementado su valor en un 26 % respecto a la propuesta previa. Por otra parte, la mejora más importante ha sido para la función $ClockLFSRKeyStreamMode$, con un 47 %. Todos estos resultados suponen en su conjunto una mejora del 10 % respecto a la propuesta de la especificación.

## V. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha presentado un análisis, tanto desde un punto de vista teórico como práctico, del generador que se utiliza para la protección de la confidencialidad y la integridad en la generación 4G de telefonía móvil. En particular, después de una introducción teórica del generador SNOW 3G, y varios estudios de implementación del generador en la plataforma móvil iOS, se ha realizado una comparación entre diferentes propuestas, y se han obtenido conclusiones interesantes sobre cómo mejorar la eficiencia de su implementación a través de la optimización del software.

Dado que este es un trabajo en progreso, todavía hay muchos problemas abiertos, tales como el análisis de parámetros no analizados en este trabajo, la utilización de diferentes arquitecturas, así como un estudio comparativo entre ellas. También otro trabajo futuro es la propuesta de una versión ligera del generador SNOW 3G para dispositivos con recursos limitados, y el análisis de propiedades teóricas del generador.

## AGRADECIMIENTOS

Investigación financiada por el MINECO y la fundación Europea FEDER mediante los proyectos TIN2011-25452 e IPT-2012-0585-370000.

## REFERENCIAS

[1] ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2, "SNOW 3G Specification, version 1.1", http://www.3gpp.org/ftp/, 2006.

[2] P. Ekdahl, T. Johansson "SNOW - a new stream cipher", *Proceedings of First Open NESSIE Workshop*, 2000.

[3] P. Ekdahl, T. Johansson "A New Version of the Stream Cipher SNOW", *Proceedings of Selected Areas in Cryptography*, LNCS 2595, pp. 47-61, 2003.

[4] P. Hawkes, G.G. Rose, "Guess-and-determine attacks on SNOW", *Proceedings of Selected Areas in Cryptography*, LNCS 2595, pp. 37-46, 2003.

[5] D. Coppersmith, S. Halevi, C. Jutla, "Cryptanalysis of stream ciphers with linear masking", Proceedings of CRYPTO, LNCS 2442, pp. 515-532, 2002.

[6] ISO/IEC 18033-4:2005. Information technology - Security techniques - Encryption algo-rithms - Part 4: Stream ciphers. http://www.iso.org/iso/home/store/catalogue_ics/

[7] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, A. Gouget, H. Sibert, "Sosemanuk, a fast software-oriented stream cipher eSTREAM, ECRYPT Stream Cipher", *ECRYPT-Network of Excellence in Cryptology, Call for stream Cipher Primitives-Phase 2*, http://www.ecrypt. eu. org/stream, 2005.

[8] O. Billet, H. Gilbert, "Resistance of SNOW 2.0 Against Algebraic Attacks", *Proceedings of CT-RSA,* LNCS 3376, pp. 19-28, 2005.

[9] ETSI/SAGE Technical report: Specification of the 3GPP Confidentiality and Integrity Al-gorithms UEA2 & UIA2. Document 5: Design and Evaluation Report, Version 1.1, September 2006.

[10] O. Nyberg, J. Wall "Improved Linear Distinguishers for SNOW 2.0", *Proceedings of Fast Software Encryption,* LNCS 4047, pp. 144-162, 2006.

[11] D. Watanabe, A. Biryukov, C. De Canniere, "A Distinguishing Attack of SNOW 2.0 with Linear Masking Method", *Proceedings of Selected Areas in Cryptography*, LNCS 3006, pp. 222-233, 2004.

[12] B. Debraize, I.M. Corbella, "Fault Analysis of the Stream Cipher Snow 3G", *Proceedings of Workshop on Fault Diagnosis and Tolerance in Cryptography*, IEEE, pp. 103-110, 2009.

[13] B. Brumley, R. Hakala, K. Nyberg, B. Sovio, "Consecutive S-box Lookups: A Timing Attack on SNO 3G", *Proceedings of Information and Communications Security*, LNCS 6476, pp. 171-185, 2010.

[14] A. Biryukov, D. Priemuth-Schmid, B. Zhang, "Multiset collision attacks on reduced-round SNOW 3G and SNOW 3G", *Proceedings of Applied Cryptography and Network Security*, pp. 139-153, 2010.

[15] G. Orhanou, S. El Hajji, Y. Bentaleb,"NOW 3G stream cipher operation and complexity study", *Contemporary Engineering Sciences-Hikari Ltd*, 3(3), pp. 97-111, 2010.

[16] P. Kitsos, G. Selimis, O. Koufopavlou,"High performance ASIC implementation of the SNOW 3G stream cipher", *IFIP/IEEE VLSI-SOC,* 2008.

[17] O. Delgado-Mohatar, A. Fúster-Sabater,"Software Implementation of Linear Feedback Shift Registers over Extended Fields,' *Proceedings of CISIS/ICEUTE/SOCO Special Sessions,* pp. 117-126, 2012.