Implementación Software de Registros de Desplazamiento sobre Cuerpos Extendidos

Oscar Delgado Mohatar
Universidad Internacional de Castilla y León
C/ Calzadas 5, 09004 Burgos, España
Email: oscar.delgado@unicyl.es

Amparo Fúster Sabater
Instituto de Seguridad de la Información, C.S.I.C.
C/ Serrano 144, 28006 Madrid, España
Email: amparo@iec.csic.es

***Abstract*—En este trabajo se analiza el diseño e implementación software de LFSR definidos sobre cuerpos extendidos $GF(2^n)$ en lugar de su definición tradicional sobre el cuerpo binario $GF(2)$. De esta forma, se pretende aprovechar la estructura subyacente del procesador en el que se ejecutan y obtener implementaciones más eficientes. Se presentan resultados numéricos de este estudio realizado sobre diferentes cuerpos extendidos y diferentes plataformas de muy distintas prestaciones. Los beneficios son claros para todas las posibles aplicaciones de LFSR, en particular para aplicaciones criptográficas.**

I. Introducción

Tradicionalmente los Registros de Desplazamiento con Realimentación Lineal (Linear Feedback Shift Registers, LFSR) se utilizan como generadores de secuencia pseudoaleatoria con aplicación tanto en comunicaciones (sistemas de TV digital o señal de posicionamiento GPS) como en criptografía (bloques básicos para cifradores en flujo).

En la práctica, estas estructuras funcionan sobre el cuerpo binario $GF(2)$ y proporcionan un bit de salida a cada pulso de reloj. De hecho, todo el substrato matemático de los LFSR se encuentra definido sobre este cuerpo [8]. Aunque esta aproximación pueda ser adecuada en implementaciones hardware nunca lo será en implementaciones software. Dado que el tamaño de la longitud de palabra en los microprocesadores varía de los 8 bits en los menos potentes hasta los 64 bits en los de mayor potencia, realizar implementaciones orientadas al bit resulta claramente ineficiente.

El objetivo de este trabajo es, por tanto, utilizar cuerpos finitos más adecuados para la arquitectura de los actuales microprocesadores. En este sentido, las elecciones naturales son los cuerpos de Galois extendidos, con 2^n elementos, donde n está relacionado con el tamaño de los registros del propio microprocesador, que normalmente será 8, 16 ó 32 bits. De esta forma, los elementos del cuerpo encajan perfectamente en una unidad de almacenamiento, de manera que pueden ser manipulados eficientemente. Al mismo tiempo, la tasa de generación de secuencia de salida también se ve incrementada: hasta 8, 16 ó 32 bits a cada pulso de reloj.

Desafortunadamente, las operaciones aritméticas sobre cuerpos de Galois extendidos son computacionalmente muy costosas [1], especialmente cuando se comparan con la suma binaria en $GF(2)$ que se implementa con una simple operación XOR. Por tanto, para acelerar en lo posible estas operaciones, la mayoría de las implementaciones software que trabajan en el cuerpo $GF(2^8)$ utilizan tablas precomputadas [3], [4], donde la entrada de dos operandos muestra el resultado de dicha operación. Sin embargo, pasar de $GF(2^8)$ al cuerpo inmediatamente superior $GF(2^{16})$ tiene un impacto muy significativo en el tamaño de dichas tablas, que estaría por encima de la capacidad de memoria de la mayoría de los sistemas. En entornos con memoria limitada, como redes de sensores o MANets, el procedimiento es claramente inviable.

Por tanto, en este trabajo se propone:

1) El estudio de técnicas que realicen operaciones aritméticas (particularmente la multiplicación) en un cuerpo de Galois extendido $GF(2^n)$.
2) La adaptación de dichas técnicas a los LFSR de uso criptogrfico.

Con este análisis, se pretende extraer diversas conclusiones y responder a ciertas preguntas como qué técnica es más adecuada para su uso en LFSR, qué grado máximo de mejora cabe esperar de cada una de ellas y cómo dichas técnicas se ven afectadas por las características de las plataformas hardware sobre las que se ejecutan.

II. LFSR sobre $GF(2^n)$

Tradicionalmente los LFSR, componentes básicos de muchos cifradores en flujo, se han definido sobre el cuerpo de Galois binario. Sin embargo, nada impide que estos registros puedan trabajar sobre cuerpos de mayor tamaño, aprovechando la estructura subyacente del procesador en el que se ejecutan. La finalidad de este trabajo es definir los LFSR sobre cuerpos de Galois extendidos, $GF(2^n)$, en lugar de sobre el cuerpo tradicional $GF(2)$. La validez de esta propuesta se fundamenta en que la secuencia generada por un LFSR sobre un cuerpo extendido preserva las mismas propiedades que las secuencias generadas por los LFSR tradicionales [13], de forma que siguen siendo aptos para su uso criptográfico. Los cuerpos algebraicos compuestos constituyen, por tanto, una pieza esencial en la definición de los nuevos registros de desplazamiento, que denominaremos en lo sucesivo registros de desplazamiento extendidos.

A. Registros de desplazamiento extendidos

Los registros de desplazamiento extendidos se definen de igual manera que los tradicionales. A continuación, se exponen

algunos conceptos básicos.

Definición 1. Un registro de desplazamiento lineal extendido de longitud L es una máquina de estados finita que actúa sobre un cuerpo finito $\mathcal{F}_q$, de característica p, donde $q = p^e$ para algún $e > 1$. En nuestro caso, $q = 2^n$ y, por tanto, $\mathcal{F}_q = GF(2^n)$. El LFSR cuenta con L celdas de memoria $r_{L-1}, \ldots, r_1, r_0$, numeradas de derecha a izquierda, donde cada una de ellas contiene elementos de $GF(2^n)$. En un instante de tiempo cualquiera t el contenido del registro conforma su estado y se denota por $S_t = (s_{t+L-1}, s_{t+L-2}, \ldots, s_t)$. El estado en el instante de tiempo cero, S_0, se denomina *estado inicial* del LFSR. En cada unidad de tiempo (pulso de reloj en las implementaciones hardware), el registro actualiza su estado. De esta forma, el estado S_{t+1} se deriva del estado S_t de la siguiente forma:

1) El contenido de la celda r_{L-1} se convierte en la salida de la unidad de tiempo t y entra a formar parte de la secuencia generada.
2) El contenido de la celda r_i se desplaza a la celda r_{i+1} para todo i, $0 \leq i \leq L-2$.
3) El nuevo contenido de la celda r_0 es el denominado elemento de realimentación, que se calcula de acuerdo a la siguiente función de realimentación:

$$s_{t+L} = \sum_{i=0}^{L-1} c_i \, s_{t+i}, \;\; c_i \in GF(2^n), \tag{1}$$

donde los coeficientes $c_0, c_1, \ldots, c_{L-1}$ se denominan *coeficientes de realimentación* y la yuxtaposición de variables representa la multiplicación en $GF(2^n)$.

Estos coeficientes definen el polinomio generador o *polinomio característico* del LFSR:

$$p(x) = x^L + c_{L-1}\, x^{L-1} + \cdots + c_1\, x + c_0\,. \tag{2}$$

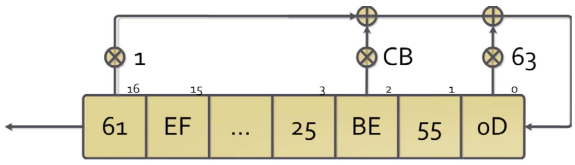

Fig. 1. LFSR extendido de 17 etapas definido sobre $GF(2^8)$

Como es sabido, si este polinomio es primitivo, entonces el periodo T de la secuencia generada por el LFSR es máximo de valor $T = 2^L - 1$. En la Figura 1 puede encontrarse un ejemplo de un LFSR extendido de 17 etapas definido sobre $GF(2^8)$. En esta nueva arquitectura los contenidos de las celdas, s_{t+i}, y los coeficientes de realimentación ya no son bits sino elementos de $GF(2^8)$ expresados en notación hexadecimal. Este hecho afectará sobre todo a las operaciones de suma y multiplicación que deben realizarse sobre ellos.

TABLE I
PARÁMETROS DE LAS FUNCIONES DE REALIMENTACIÓN UTILIZADAS EN ESTE TRABAJO

Polinomio	α	β
$P_8(x)$	$C6$	67
$P_{16}(x)$	$19B7$	$013C$
$P_{32}(x)$	$F21DA317$	$E28C895D$

B. Aritmética en $GF(2^n)$

Los elementos del cuerpo $GF(2^n)$ se representan habitualmente como polinomios de grado menor que n con coeficientes en $GF(2)$. De esta forma, por ejemplo, el elemento a de $GF(2^4)$, notado a = 0111, puede representarse por el polinomio $a(x) = x^2 + x + 1$. Con esta representación polinomial, las operaciones aritméticas se llevan a cabo haciendo uso de un polinomio irreducible $R(x)$ de grado n que se define sobre el cuerpo binario de la siguiente forma:

$$R(x) = x^n + d_{n-1}x^{n-1} + \cdots + d_1 x + 1, \; d_i \in GF(2). \tag{3}$$

En este escenario, la operación suma sigue realizándose de la misma forma que en $GF(2)$, es decir con una simple operación XOR. Por el contrario, la multiplicación exige que el resultado sea reducido módulo $R(x)$. En las siguientes secciones se analizan estas operaciones básicas con mayor profundidad.

C. Funciones de realimentación

Para el estudio, experimentación y posterior análisis del diseño de los LFSR extendidos, se ha elegido un conjunto de polinomios primitivos, $P_8(x)$, $P_{16}(x)$ y $P_{32}(x)$, definidos sobre los cuerpos $GF(2^8)$, $GF(2^{16})$ y $GF(2^{32})$ respectivamente. De forma genérica, los polinomios de realimentación tienen la forma:

$$P(x) = x^{17} + x^{15} + \alpha\, x^2 + \beta, \tag{4}$$

con $\alpha, \beta \in GF(2^n)$ y $\alpha, \beta \neq 0$. Por su parte, las funciones de realimentación correspondientes a dichos polinomios primitivos son:

$$s_{t+17} = s_{t+15} + (\alpha\, s_{t+2}) + (\beta\, s_t)\,, \tag{5}$$

donde + denota la suma sobre el cuerpo $GF(2^n)$ y, como se dijo anteriormente, la yuxtaposición de variables denota la multiplicación sobre dicho cuerpo. Los valores concretos de α, β para cada polinomio y función de realimentación se muestran en la Tabla I.

III. MULTIPLICACIÓN EN $GF(2^n)$

La operación multiplicación en cuerpos de Galois extendidos es, con diferencia, la operación algebraica más costosa desde un punto de vista computacional. Es por eso que la prestación final de una implementación depende del método específico de multiplicación utilizado. A continuación se analizan y comparan diferentes métodos de multiplicación en cuerpos extendidos.

TABLE II
TAMAÑOS DE LA TABLA DE RESULTADOS PRECOMPUTADOS

n	Tamaño tabla resultante	Tamaño en bytes
8	256 KB	2^{18}
10	4096 KB	2^{22}
12	64 MB	2^{26}
14	1 GB	2^{30}
16	16 GB	$2^{34} \approx 10^{10}$
...	...	...
32	64 EB	$2^{66} \approx 10^{18}$

A. Método MulTABLE

Cuando la dimensión n del cuerpo extendido $GF(2^n)$ es pequeña, el procedimiento más rápido para multiplicar dos elementos es precomputar los resultados y almacenarlos en una tabla. En general, la tabla de resultados tiene un tamaño de $2^{(2n+2)}$ bytes, de forma que este método sólo resulta aplicable para valores de n razonablemente pequeños. En la Tabla II se muestra el crecimiento del tamaño de la tabla de resultados para distintos valores de n. Como puede apreciarse, en la práctica, el único valor razonable es n = 8. En este caso el tamaño de la tabla es de 256 KB. Para n = 16, la tabla ocuparía ya 16 GB, y para n = 32 la tabla alcanzaría un tamaño astronómico de 64 EB (!) (1 EB = 10^{18} bytes).

B. Método LogTABLE

Cuando el método anterior no puede utilizarse, el siguiente método efectivo para realizar estas multiplicaciones consiste en hacer uso de tablas de logaritmos y logaritmos inversos, tal y como se describe en [9]. Utilizando estas tablas y algo de aritmética básica, se pueden multiplicar dos elementos sumando sus correspondientes logaritmos y tomando luego el logaritmo inverso, para así producir el resultado final. En efecto,

$$a \cdot b = invlogTable[(logTable[a] + logTable[b])],$$

de forma que reducimos la operación multiplicación a tres búsquedas en una tabla y una operación suma, lo que resulta mucho más eficiente. La primera de las tablas necesarias, que se denominará $logTable$, se define para valores entre 1 y $2^n - 1$, conteniendo para cada elemento del cuerpo su correspondiente logaritmo. Por otro lado la segunda tabla, llamada $invlogTable$, se define para los índices 0 a $2^n - 1$ y, de forma recíproca, realiza una correspondencia entre estos valores y su logaritmo inverso. Claramente, $logTable[invlogTable[i]] = i$, y $invlogTable[logTable[i]] = i$. Este tipo de tablas ocupan $2^{(n+2)}$ bytes cada una, en lugar de los $2^{(2n+2)}$ del método anterior. De esta forma, ahora para n = 8 y n = 16, sólo son necesarios 2 KB y 0.5 MB, respectivamente. Sin embargo, el crecimiento exponencial es implacable y para n = 32 el tamaño necesario ya se acerca a 32 GB, lo que está fuera del alcance de la mayoría de los computadores modernos.

C. Método SHIFT

El método LogTABLE, aunque supone una mejora muy importante respecto al presentado en primer lugar, tiene sin embargo unas necesidades de memoria poco realistas para n = 32. En este caso, cuando las tablas no son prácticas, siempre queda el recurso de la multiplicación $a \cdot b$ de propósito general. Esta se implementa de manera eficiente convirtiendo el segundo factor, b, en una matriz de bits de dimensión $n \times n$ y multiplicándola por un vector de bits correspondiente al primer factor, a, para así obtener el vector producto [14]. Aunque es significativamente más lento que los dos métodos anteriores, es un procedimiento que funciona siempre, para cualquier tamaño de n y necesita unos requerimientos de memoria mínimos.

D. Método SPLITW8

Por último, para el caso especial de n = 32, se puede utilizar un método válido exclusivamente para este valor de n. El proceso comienza creando siete tablas de 256 KB cada una. Estas tablas se utilizan para multiplicar los números de 32 bits, dividiéndolos en cuatro partes de 8 bits cada una, y realizando entonces 16 multiplicaciones y sumas binarias para obtener el resultado final. Haciendo uso de esta optimización, el algoritmo es 7 veces más rápido que utilizando el método SHIFT para el caso concreto de n = 32.

E. Arquitecturas analizadas

Las pruebas se han llevado a cabo en diferentes arquitecturas con el fin de obtener los resultados más realistas posibles. La Tabla III muestra las características de cada una de las arquitecturas utilizadas. La elección de estas plataformas se ha realizado procurando que resultasen lo más representativas posibles. Por esta razón, se han incluido desde dispositivos con una exigua capacidad de cómputo como microcontroladores de 8 bits, pasando por computadores ya desfasados como un Pentium III, hasta ordenadores actuales como un Intel Core 2 o un AMD Athlon.

TABLE III
LISTA DE ARQUITECTURAS UTILIZADAS DURANTE LA EVALUACIÓN DE LAS PROPUESTAS

Arquitectura	Frecuencia CPU	Cache L1I/L1D/L2	Memoria RAM
AMD Athlon 64 Dual Core Processor 4200+	2.2Ghz	(64KB)/512KB	2GB
Intel Core 2 Duo	2Ghz	32KB/32KB/4MB	1GB
Intel Pentium III	450Mhz	-	192MB
Microcontrolador ATmega 168	16Mhz	-	14KB

F. Análisis de métodos de multiplicación

Los experimentos realizados consistieron en generar una secuencia de 10^7 bytes con las distintas plataformas y sobre los diferentes cuerpos incluyendo el cuerpo binario $GF(2)$. Se evaluaron las métricas habituales: tiempo de ejecución total (en segundos) y tasa de generación de secuencia de salida (en Mega bytes por segundo). En la Tabla IV se representan dichos valores junto con un factor de mejora γ. Este factor compara el tiempo de generación de la secuencia de prueba cuando se trabaja sobre cuerpos extendidos o sobre el cuerpo binario. Un factor de mejora de 2 significa que el LFSR implementado

TABLE IV
TIEMPOS DE EJECUCIÓN, TASA DE SALIDA Y FACTOR DE MEJORA PARA LAS DIFERENTES PLATAFORMAS

	$GF(2)$		
Arquitectura	Tiempo (s)	Salida (MB/s)	Factor γ
ATmega 168	4450	0.002	-
Pentium III	24.37	0.41	-
Dual Core	1.58	6.32	-
Athlon	1.14	8.77	-
	$GF(2^8)$		
Arquitectura	Tiempo (s)	Salida (MB/s)	Factor γ
ATmega 168	1605	0.006	3
Pentium III	3.89	2.57	6.25
Dual Core	0.56	21.27	2.80
Athlon	0.59	16.94	1.93
	$GF(2^{16})$		
Arquitectura	Tiempo (s)	Salida (MB/s)	Factor γ
ATmega 168	-	-	-
Pentium III	1.49	6.71	16.35
Dual Core	0.24	41.66	6.60
Athlon	0.33	30.30	3.45
	$GF(2^{32})$		
Arquitectura	Tiempo (s)	Salida (MB/s)	Factor γ
ATmega 168	-	-	-
Pentium III	1.79	5.58	13.57
Dual Core	0.27	37.03	5.85
Athlon	0.37	27.02	3.08

sobre el correspondiente cuerpo extendido es 2 veces más rápido que cuando se implementa sobre $GF(2)$. El método de multiplicación utilizado en cada caso fue el más rápido para cada plataforma. Como puede apreciarse, la introducción de los cuerpos extendidos proporciona importantes mejoras respecto a la implementación tradicional de LFSR sobre el cuerpo binario.

En la Tabla V aparecen los resultados, en millones de multiplicaciones por segundo, de los diferentes métodos analizados. Un guión indica que el método no es aplicable para esa combinación de tamaño de n y arquitectura. Por otro lado, los valores numéricos que aparecen como subíndices evalúan cómo se comporta cada método para un cuerpo dado respecto al método más rápido. Éste resulta ser MulTABLE en todos los casos y es el que se ha considerado como referencia. Así, por ejemplo, considerando la arquitectura Dual Core y el cuerpo extendido $GF(2^8)$ puede observarse que el método MulTABLE permite llevar a cabo algo más de 110 millones de multiplicaciones por segundo, mientras que el método LogTABLE únicamente alcanza los 94.11 millones. El subíndice que aparece en este último método, 0.85, corresponde al cociente entre los dos valores anteriores y proporciona una relación entre el rendimiento de los dos métodos. De esta forma, si el valor es menor que 1, el segundo método será más lento; concretamente 0.85 veces en este caso. De la misma forma, si el valor es mayor que 1, el segundo método será más rápido en la proporción indicada.

La primera de las plataformas analizadas corresponde a un microcontrolador de 8 bits, de gama baja o media, típico de una red de sensores. Este cuenta con escasos recursos computacionales, tanto de CPU como de memoria, por lo que la única posible elección es el método SHIFT. Por su parte, los métodos MulTABLE y LogTABLE, ambos basados en el uso de tablas, quedan claramente fuera del alcance de un dispositivo con apenas 14 KB de memoria disponible. En cualquier caso, esta plataforma consigue realizar casi 800 multiplicaciones por segundo lo que supone una cifra nada despreciable en un arquitectura con recursos computacionales tan limitados.

En las arquitecturas tipo PC puede observarse que los resultados absolutos son proporcionales a su capacidad computacional. Lógicamente la plataforma Dual Core puede realizar aproximadamente el triple de operaciones por segundo que un Pentium III. A pesar de ello, puede observarse que las diferencias de rendimiento entre los distintos métodos se mantienen entre arquitecturas. En otras palabras, la proporción de cuánto mejor o peor es un método con respecto a otro se mantiene entre las diferentes plataformas. Este hecho es fácil de comprobar observando que, por ejemplo, el coeficiente de evolución del método LogTABLE toma los valores 0.87, 0.85 y 0.87 en las diferentes arquitecturas, lo que supone una desviación típica de tan sólo $\sigma = 0.011$. Por otro lado, el método SHIFT presenta una variación en este coeficiente algo mayor, de 0.03, aunque también este parámetro pueda considerarse como un valor estable.

A la vista de estos resultados puede deducirse que las notables diferencias en las arquitecturas, tanto en capacidad computacional como en memoria RAM, no tienen una influencia determinante en el rendimiento de los diferentes métodos, que presentan comportamientos muy estables. Evidentemente, el rendimiento absoluto es muy diferente entre arquitecturas, pero esta magnitud "escala" de la misma forma en cada una de ellas. Otro hecho significativo, que se desprende del análisis de los datos presentados, es que el rendimiento decrece rápidamente conforme aumenta la dimensión del cuerpo; de forma que, por ejemplo, la multiplicación en $GF(2^{32})$ es unas 10 veces más lenta que en $GF(2^8)$ en todas las arquitecturas.

Las conclusiones finales que pueden extraerse de los experimentos realizados resultan, a la vista de los datos, bastante claras. Trabajando en el cuerpo $GF(2^8)$ el método más eficiente para la multiplicación de elementos en $GF(2^n)$ resulta sin duda MulTABLE, que utiliza tablas precomputadas de 256 KB. El tamaño de estas tablas, sin embargo, las hace prohibitivas cuando pasamos al cuerpo $GF(2^{16})$. En este caso deben utilizarse otro tipo de tablas, las utilizadas por el método LogTABLE. Éstas, sin embargo, requieren pasos adicionales, lo que provoca que dicho método vea decrementado ligeramente su rendimiento. Finalmente, en $GF(2^{32})$, el método más adecuado es SPLITW8, que implementa una mejora sobre el método general SHIFT, aún cuando requiera una memoria total de 1792 KB para almacenar las tablas necesarias.

IV. MEJORAS EN LA IMPLEMENTACIÓN

Como se ha comentado anteriormente, los LFSR son estructuras especialmente difíciles de implementar de forma eficiente en software. La razón principal es el desplazamiento necesario,

TABLE V
RESULTADOS DE LOS DIFERENTES MÉTODOS DE MULTIPLICACIÓN EN DIVERSAS ARQUITECTURAS

Arquitectura	Método de multiplicación			
ATmega168	MulTABLE	LogTABLE	SHIFT	SPLITW8
$GF(2^8)$	-	-	0.00079	-
$GF(2^{16})$	-	-	-	-
$GF(2^{32})$	-	-	-	-
PentiumIII	MulTABLE	LogTABLE	SHIFT	SPLITW8
$GF(2^8)$	31.73	$27.77/_{0.87}$	$1.39/_{0.04}$	-
$GF(2^{16})$	-	4.53	$0.55/_{0.12}$	-
$GF(2^{32})$	-	-	0.18	$0.87/_{4.83}$
Dual Core	MulTABLE	LogTABLE	SHIFT	SPLITW8
$GF(2^8)$	110.03	$94.11/_{0.85}$	$4.40/_{0.03}$	-
$GF(2^{16})$	-	60.25	$0.55/_{0.02}$	-
$GF(2^{32})$	-	-	0.47	$8.03/_{17.08}$
Athlon	MulTABLE	LogTABLE	SHIFT	SPLITW8
$GF(2^8)$	89.62	$78.76/_{0.87}$	$5.52/_{0.06}$	-
$GF(2^{16})$	-	29.05	$2.25/_{0.07}$	-
$GF(2^{32})$	-	-	0.82	$3.41/_{4.15}$

a cada pulso de reloj, del contenido de cada una de sus celdas. Este desplazamiento ocurre de forma simultánea en las implementaciones hardware, de forma que todo el proceso puede llevarse a cabo en un único pulso. No ocurre lo mismo, sin embargo, en las versiones software donde el proceso se convierte en iterativo y costoso. En este caso se necesita para un registro de L etapas un total de $L-1$ desplazamientos. Por tanto, es también esencial utilizar cualquier técnica de implementación en software que ayude a mejorar el rendimiento final. En las siguientes subsecciones, se analizarán tres de estas técnicas de implementación.

A. Búfer circular

Esta técnica, conocida y utilizada ampliamente en otros ámbitos de la algorítmica [2], [15], es especialmente adecuada para su aplicación a LFSR pues permite evitar los costosos $L-1$ desplazamientos. Su funcionamiento es muy sencillo y consiste básicamente en utilizar varios punteros, que se mueven a lo largo del registro, en lugar de mover los propios datos. Cuando los punteros llegan al final del búfer, vuelven al principio, dando así la impresión de un búfer circular. Los punteros necesarios para el uso de esta técnica aplicada a LFSR son los siguientes: un puntero de inicio que corresponde a la celda r_{L-1} que genera la salida en cada ciclo, un puntero de fin de registro que corresponde a la celda r_0 que recibe la realimentación en cada ciclo y una serie de punteros de realimentación que corresponden a los coeficientes no nulos del polinomio característico. Actualizar los diferentes índices necesarios para el funcionamiento del LFSR implica el uso de aritmética módulo la longitud del registro. Si L es una potencia de 2, esta operación es muy eficiente en los actuales microprocesadores, pero en caso contrario no lo es tanto. Para evitar este inconveniente, puede utilizarse una variante de esta técnica, que se presenta a continuación.

B. Ventana deslizante

La técnica de la ventana deslizante es especialmente conocida por su uso en el protocolo TCP/IP, donde se encarga del control del flujo de datos [12]. Utilizando los mismos principios de la técnica anterior es posible, sin embargo, evitar alguno de sus inconvenientes haciendo uso de un búfer de longitud doble de la necesaria. Cuando se escribe un nuevo valor en el búfer, se hace simultáneamente en dos lugares diferentes. El puntero comienza en la mitad del búfer de longitud doble y cuando alcanza el final vuelve al medio de nuevo. De esta forma, se puede acceder a todos los valores intermedios en posiciones fijas a la derecha del puntero. Como ventaja adicional, este esquema permite que el registro puede tener una longitud arbitraria, sin restricciones, y a pesar de ello resultar muy eficiente en su funcionamiento.

C. Desdoblamiento de bucle

Por último, la técnica de desdoblamiento de bucle (Loop unrolling o loop unwinding en inglés) es un método de optimización muy utilizado en diversos ámbitos de la informática, que pretende mejorar el tiempo de ejecución de aquellas porciones de código que contengan bucles [7], [11]. Dado que la implementación de un registro de desplazamiento en software es básicamente la repetición del cuerpo de un bucle, esta técnica parece, a priori, encajar de forma natural con los LFSR.

El desdoblamiento de bucle funciona replicando el cuerpo original del bucle principal varias veces, ajustando el código que se encarga de la comprobación del fin del bucle y eliminando las instrucciones de salto que resulten redundantes. Aunque pueda parecer ilógico, esta técnica consigue en la mayoría de los casos que el código optimizado, a pesar de ser más largo, requiera menos tiempo de ejecución. La eficacia de este método de optimización se debe fundamentalmente a dos razones básicas:

- El número de instrucciones de saltos se reduce y, en consecuencia, la variable índice que controla el bucle se modifica un número menor de veces.
- En casi cualquier arquitectura moderna diseñada con un alto grado de paralelismo, esta técnica puede obtener mejores prestaciones de ejecución para las instrucciones del interior del bucle.

Aunque es difícil de cuantificar y depende en gran medida de cada caso concreto, en general esta técnica permite mejoras en el tiempo de ejecución que oscilan entre el 10 y el 30%, véase [5].

En la Tabla VI aparecen los resultados de aplicación a la plataforma Dual Core de estas técnicas software para cada uno de los cuerpos considerados incluido el cuerpo binario. También se representa el Factor δ que compara los tiempos de ejecución invertidos en cada una de las configuraciones sin mejoras y con cada una de las mejoras de implementación.

V. CONCLUSIONES

En este trabajo se ha propuesto el uso de cuerpos algebraicos extendidos como base para el diseño de LFSR. Aunque

TABLE VI
RESULTADOS DE LAS DIFERENTES TÉCNICAS DE MEJORA Y FACTOR δ

Técnica de mejora		$GF(2)$	$GF(2^8)$	$GF(2^{16})$	$GF(2^{32})$
Ninguna	Salida (MB/s)	6.32	21.27	41.66	37.03
	Factor δ	-	-	-	-
Búfer circular	Salida (MB/s)	9.70	22.72	58.82	43.47
	Factor δ	1.53	1.06	1.41	1.17
Ventana deslizante	Salida (MB/s)	24.39	27.02	62.5	47.61
	Factor δ	3.85	1.27	1.50	1.28
Desdoblamiento de bucles	Salida (MB/s)	66.67	38.46	76.92	55.55
	Factor δ	10.54	1.81	1.84	1.50

tradicionalmente estas estructuras se han definido sobre el cuerpo de Galois binario nada impide hacerlo sobre cuerpos de dimensión superior. De esta forma, se pretende aprovechar la estructura subyacente del procesador en el que se ejecutan y obtener así implementaciones software mucho más eficientes.

Los resultados numéricos muestran cómo estas técnicas permiten obtener incrementos en el rendimiento final realmente significativos. Comparados con los métodos tradicionales, es posible conseguir factores de mejora de hasta 10.15, que suponen un aumento en la eficiencia de más de un 900% ($\% = (\delta - 1) \times 100$).

El estudio ha revelado también otros hechos importantes y llamativos. El primero es que, tal y como se esperaba, el rendimiento obtenido por el LFSR aumenta conforme lo hace el tamaño del cuerpo base $GF(2^n)$ utilizado. Sin embargo, esta mejora no es lineal, como podría indicar la intuición, sino que sufre un pequeño decremento en $GF(2^{32})$ respecto a $GF(2^{16})$. Las razones que explican este fenómeno están relacionadas con el equilibrio entre beneficio proporcionado por el cuerpo base y el coste computacional de sus operaciones. Básicamente, cuando el tamaño del cuerpo aumenta, también lo hace el coste de sus operaciones de multiplicación y esto repercute, directamente, en la función de realimentación del LFSR. De esta forma, la ventaja de obtener en cada ciclo de funcionamiento del LFSR una salida de mayor tamaño, 8, 16 ó 32 bits, se pierde gradualmente. El límite, de acuerdo con los datos obtenidos, parece estar en $GF(2^{16})$ que es el cuerpo que mejor rendimiento obtuvo en los experimentos realizados. Por tanto, a pesar de que los cuerpos extendidos han sido propuestos para su uso en LFSR, este trabajo demuestra que su aplicación no siempre resulta tan rentable. Cuerpos mayores que $GF(2^{16})$ parecen no ser tan eficaces, pues el coste de sus operaciones internas es tan alto que se pierden sus ventajas.

Por otro lado, se ha mostrado cómo otras técnicas de mejora, como búferes circulares o desdoblamiento de bucles se adaptan perfectamente a la implementación de LFSR, mejorando muy significativamente la tasa de generación de salida de los mismos. Los resultados concluyen, también, que estas técnicas resultan más eficaces en cuerpos de pequeño tamaño, como $GF(2^8)$, $GF(2^{16})$, incluso $GF(2)$ y que van perdiendo eficacia para cuerpos mayores.

Los experimentos realizados concluyen, por tanto, que la forma más eficiente de implementar registros de desplazamiento en software es trabajar en el cuerpo extendido $GF(2^{16})$, combinando esta técnica de cuerpos extendidos con la técnica del desdoblamiento de bucles.

REFERENCES

[1] R. Avanzi, N. Theriault, "Effects of Optimizations for Software Implementations of Small Binary Field Arithmetic", in Proc. of WAIFI, pp. 69–84, 2007.

[2] T. Bijlsma, M. Bekooij, Pierre Jansen, and G. Smit, "Communication between nested loop programs via circular buffers in an embedded multiprocessor system", in SCOPES 08: Proceedings of the 11th international workshop on Software and compilers for embedded systems, pp. 33–42, 2008.

[3] S. Chowdhury and S. Maitra, "Efficient Software Implementation of Linear Feedback Shift Registers", in Proc. of International Conference in Cryptology in India - INDOCRYPT 01, Lecture Notes in Computer Science 2247, pp. 397–407, 2001.

[4] S. Chowdhury and S. Maitra, "Efficient Software Implementation of LFSR and Boolean Function and Its Application in Nonlinear Combiner Model", in Proc. of Applied Cryptography and Network Security - ACNS 2003, Lecture Notes in Computer Science 2846, pp. 387–402, 2003.

[5] J.W. Davidson and S. Jinturkar, "An Aggressive Approach to Loop Unrolling", Tech. Rep., Charlottesville, VA, USA, 2001.

[6] P. Deepthi, D.S. John and P. Sathidevi, "Design and analysis of a highly secure stream cipher based on linear feedback shift register", *Comput. Electr. Eng.*, vol. 35, no. 2, pp. 235–243, 2009.

[7] O. Dragomir, T.P. Stefanov and K. Bertels, "Loop Unrolling and Shifting for Reconfigurable Architectures", in Proceedings of the 18th International Conference on Field Programmable Logic and Applications (FPL08), September 2008.

[8] S.W. Golomb, Shift Register Sequences. Laguna Hills, CA, USA: Aegean Park Press, 1981.

[9] K. Greenan, E. Miller and T. Schwarz, "Optimizing Galois field arithmetic for diverse processor architectures and applications", en: E. Miller, C. Williamson, editors. Proc. of MASCOTS, IEEE Computer Society, pp.257–266, 2008.

[10] P. Hawkes and G.G. Rose, "Guess-and-Determine Attacks on SNOW", in Proc. of the 9th Annual International Workshop on Selected Areas in Cryptography - SAC02, Lecture Notes in Computer Science 2595, pp. 37–46, 2003.

[11] J.C. Huang and T. Leng, "Generalized Loop-Unrolling: A Method for Program Speedup", Application-Specific Software Engineering and Technology, IEEE Workshop on Field Programmable Logic, pp. 244–249, 1999.

[12] C.K. Koc, "Analysis of sliding window techniques for exponentiation", Computers and Mathematics with Applications, vol. 30, pp. 17–24, 1995.

[13] C. Paar, "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields", PhD Thesis, Institute for Experimental Mathematics, University of Essen, Germany, 1994.

[14] J.S. Plank, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Storage Applications", Tech. Rep. CS-05-569, University of Tennessee, December 2005.

[15] Y. Tsujita, "Effective Remote MPI-I/O on a Parallel Virtual File System using a Circular Buffer: A Case Study for Optimization", in IASTED PDCS, pp. 490–495, 2005.