Biprocesador MicroBlaze para gestión de claves de grupos de usuarios

Jose M. Granado-Criado, Miguel A. Vega-Rodriguez, Juan M. Sanchez-Perez, Juan A. Gomez-Pulido

***Resumen*—PREMIO_JT. La gestión de claves de grupos de usuarios es una tarea crítica in aplicaciones multicast de contenido multimedia, como pueden ser la televisión por Internet, el pago por visión, la televisión satélite, etc. Estas claves deben ser recalculadas, encriptadas y redistribuidas cada vez que un usuario se dé de baja o de alta para evitar que usuarios no pertenecientes a un grupo concreto puedan acceder a los contenidos a dicho grupo. Este trabajo presenta un dual MicroBlaze System-on-Chip (SoC) de altas prestaciones para realizar la gestión de claves tan rápido como sea posible, reduciendo de esta forma la influencia de altas/bajas de usuarios en la recepción de los contenidos multimedia del resto de usuarios.**

***Palabras clave*—MicroBlaze, grupo de claves de usuario, AES, LKH, encriptación.**

I. Introducción

HOY en día, servicios como el pago por visión, la televisión por internet o la televisión satélite están en pleno apogeo. Este tipo de aplicaciones necesitan un alto esfuerzo computacional para garantizar que un usuario pueda acceder a los servicios contratados, los cuales pueden ser total o parcialmente los mismos que los contratados por otro usuario. Esta situación hace que los usuarios compartan unas claves (para los contenidos comunes) y no compartan otras.

Para gestionar estas claves, hay principalmente dos técnicas:

- En la primera técnica, la más básica, solamente hay una clave de grupo. Esta clave es utilizada para enviar contenidos encriptados a todos los miembros del grupo. El principal problema de esta técnica es que, cuando un usuario se da de baja, una nueva clave de grupo debe ser generada y, por tanto, encriptada con cada una de las claves de usuario de todos los usuarios del grupo. Esta tarea tiene un enorme coste computacional.
- La segunda técnica es la LKH (Logical Key Hierarchy) [6] y consiste en utilizar un árbol binario formando subgrupos de usuarios. En este caso, cuando un usuario se da de baja, solamente algunas claves serán recalculadas. Es más, el número de encriptaciones se reduce ya que se utilizan las claves de subgrupos para encriptar las nuevas claves generadas para todos los miembros de dicho subgrupo.

La tabla 1 muestra el número de encriptaciones necesarias para ambas técnicas. A primera vista, la técnica básica podría parecer mejor que la LKH ya que solamente necesita dos encriptaciones para el alta de un usuario. Sin embargo, para ver el número real de encriptaciones de cada técnica, vamos a calcular el número total de encriptaciones necesarias para un alta y una baja con el tamaño de grupo empleado en este trabajo, concretamente 8.388.608 usuarios, como veremos en la sección *Organicación de las Claves de Grupo*. Con este dato, el número total de encriptaciones necesarias para realizar un alta y una baja es de 8.388.609 (2 para el alta y 8.388.607 para la baja). Sin embargo, en la técnica LKH solamente necesitamos 90 encriptaciones (46 para el alta y 44 para la baja). Como vemos, LKH decrementa considerablemente el número de encriptaciones necesarias para una operación de alta y una de baja.

Este trabajo se estructura como sigue: La sección 2 muestra la arquitectura jerárquica de claves y el proceso de generación de claves. En la sección 3 describimos la organización de memoria empleada para almacenar el árbol de claves. A continuación, la sección 4 presenta la arquitectura hardware implementada. La sección 5 analiza los resultados finales comparándolos con los resultados obtenidos por otros trabajos encontrados en la literatura y, finalmente establecemos, en la sección 6, las conclusiones obtenidas y el trabajo futuro.

TABLA I

NÚMERO DE ENCRIPTACIONES DE LAS TÉCNICAS BÁSICA Y LKH

	Técnica básica	LKH
Alta	2	2 * log2 n
Baja	n – 1	2 * (log2 n – 1)
Complejidad media	O(n)	O(log n)

II. Descripción del problema

En esta sección se describen tanto la arquitectura LKH como el módulo de generación de claves.

A. Logical Key Hierarchy

La técnica LKH (Logical Key Hierarchy) consiste en dividir el grupo en subgrupos jerarquizados, cada uno de los cuales tiene su propia clave (denominada help-key) que estará compartida por todos los usuarios que pertenezcan a ese subgrupo. De esta forma la help-key raíz representa la clave de grupo y será esta la utilizada para encriptar los datos contratados. Este hecho hace que un usuario deba almacenar varias claves, concretamente su clave de identificación (K_i), la cual es conocida únicamente por dicho usuario, la clave de grupo (K_g), conocida por todos los usuarios del grupo, y todas las help-key ($K_{i\text{-}j}$) que se encuentren entre la clave de grupo y dicho usuario en el árbol de claves. Por ejemplo, la figura 1 muestra un LKH para 8 usuarios. En este caso, el usuario 5 almacenaría las claves K_g, $K_{4\text{-}7}$, $K_{4\text{-}5}$ y K_5.

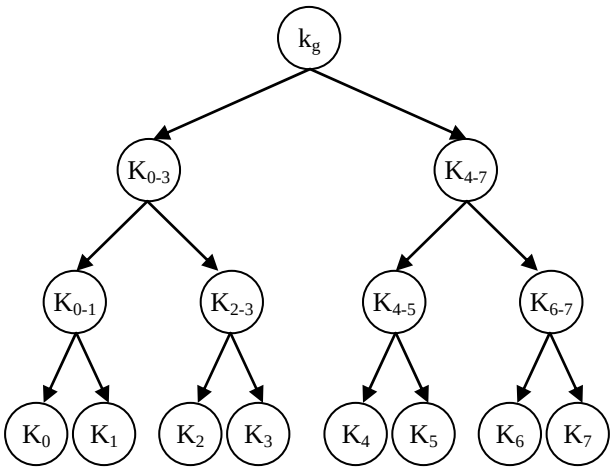

Fig. 1. Ejemplo de LKH

Cuando un usuario quiere unirse a un grupo, re realizan dos funciones: primeramente se calcula la clave de usuario y se regeneran la nueva clave de grupo y todas las help-keys de los nodos entre la clave de grupo y el usuario. Entonces, todas estas nuevas claves deben ser encriptadas y enviadas tanto al nuevo usuario como a todos aquellos a los que le afecte el cambio. Sin embargo, no es necesario utilizar las claves de usuario de todos los usuarios para encriptar estas nuevas claves, sino que se utilizarán las help-keys para encriptar cada una de las nuevas claves. Para entender mejor esta tarea, veamos un ejemplo: si un nuevo usuario se uniera en la posición 5 de la figura 1, se generarían las claves K_g^{nueva}, $K_{4\text{-}7}^{nueva}$, $K_{4\text{-}5}^{nueva}$ y K_5^{nueva}. Para este caso, las encriptaciones que se harían serían $E\_K_5^{nueva}(K_{4\text{-}5}^{nueva})$,

$E\_K_4(K_{4\text{-}5}^{nueva})$, $E\_K_{4\text{-}5}^{nueva}$ $(K_{4\text{-}7}^{nueva})$, $E\_K_{4\text{-}7}^{nueva}$ (K_g^{nueva}), donde $E\_K_n(K_m)$ indica que la clave K_m se encripta usando la clave K_n.

Por otra parte, cuando un usuario abandona un grupo, se realiza un proceso similar. La única diferencia es que, en este caso, no es necesario generar/encriptar la clave de un nuevo usuario.

Por otra parte, tanto en el alta como en la baja de usuarios, se actualiza el árbol de estados de claves o KST (Key State Tree). Este KST almacena el estado de todas las claves del árbol LKH. Sin embargo, cada nodo del KST no actualiza el estado de una clave, sino el de sus dos hijos, por lo que no es necesario almacenar el estado de las hojas (usuarios) del LKH ya que dicho estado nos lo dará el padre de los dos usuarios. De esta forma, el número de nodos del KST es igual al número de usuarios del grupo, ya que el número total de claves almacenadas es el doble del número de usuarios. Por tanto, el KST tendrá la mitad de nodos que el LKH y un nivel menos. Los posibles valores de estado (en binario) del KST son los siguientes:

- 00: No hay usuarios en ninguna de las ramas de este nodo.
- 01: Hay al menos un usuario en la rama derecha de este nodo.
- 10: Hay al menos un usuario en la rama izquierda de este nodo.
- 11: Hay al menos un usuario en cada una de las ramas de este nodo.

B. Generador de claves

Como hemos visto en la sección anterior, es necesario continuamente generar nuevas claves. Para realizar esta tarea empleamos el generador de claves ANSI X9.17 [4]. Este generador de claves utiliza un algoritmo de encriptación de clave simétrica para generar nuevas claves. En nuestro caso, dicho algoritmo criptográfico será el algoritmo AES de 128 bits [3]. Para la funcionalidad del generador se necesitan tres datos: la clave de generación (K_{gen}), el sello temporal (D) y la semilla inicial (S_0), todos ellos de 128 bits por utilizar la versión de 128 bits del algoritmo AES.

Para obtener una nueva clave, se deben realizar las siguientes operaciones:

1. $I=E\_K_{gen}(D)$ (solamente en el proceso de inicialización)
2. $K_i^{nueva}= E\_K_{gen}(I \text{ xor } S_i)$
3. $S_{i+1} = E\_K_{gen}(I \text{ xor } K_i^{nueva})$

Como podemos ver, necesitamos dos encriptaciones para generar una nueva clave. Si analizamos el ejemplo de la sección anterior, cuando un usuario se une a un LKH de ocho usuarios, se necesitan generar cuatro nuevas claves (una clave por cada nivel del árbol), es decir, 8 encriptaciones. Si extendemos este ejemplo a nuestro diseño con 8.388.608 usuarios, necesitaremos 23 nuevas claves, es decir, 46 encriptaciones por cada alta de usuario.

III. ORGANIZACIÓN DE MEMORIA

Esta sección muestra cómo están organizados el árbol de claves de grupo (Group Key Tree o GKT) el árbol de estados de clave (KST).

Para almacenar tanto el GKT como el KST utilizamos una memoria DDR3 de 512 MB. Utilizamos esta memoria porque DDR3 es una tecnología de memoria de alta velocidad y porque sus 512 MB nos permiten almacenar un gran número de usuarios. Algunos autores, sin embargo, utilizan memorias BlockRAM para almacenar el KST porque las BlockRAM son más rápidas que la memoria DDR3 [5]. Sin embargo, pierden en la cantidad de usuarios que puede almacenar su sistema: nuestro sistema puede almacenar un total de 8.388.608 usuarios frente a los 131.072 usuarios de [5].

Estos 8.388.608 usuarios, junto con sus claves de grupo, necesitan un total de 256 MB de memoria (cada elemento es una clave de 128 bits), y para el almacenamiento del estado de cada nodo se necesitan 8 MB de memoria. Esta última ocupación podría reducirse ya que un estado de clave únicamente ocupa 2 bits. Sin embargo, como el programa está codificado en C++, el KSM es una memoria con palabras de un byte. De esta forma, se podrían almacenar hasta 4 estados de clave en cada palabra del KSM, pero esta mejora conllevaría ciertos retrasos a la hora de obtener un estado concreto de una palabra con cuatro estados. Esta mejora sería interesante si se desarrolla un sistema con fuertes limitaciones de memoria pero, en nuestro caso, tanto el GKT como el KST pueden almacenarse sin ningún problema en la memoria DDR3. Además, si quisiéramos aumentar el número de usuarios (doblarlo), se necesitarían 512 MB de memoria para el GKT más 16 MB para el KST, o 4 MB si usamos la mejora de 4 estados por palabra. En cualquiera de los dos casos, se necesitaría aumentar la memoria DDR3. Por tanto, empleamos los promeros 256 MB de la memoria DDR3 para almacenar el GKT y los siguientes 8 MB para almacenar el KST.

Respecto a la organicación interna tanto del KST y del GKM, los autores en [5] almacenan el elemento raíz en la dirección más significativa de ambas estructuras. Este tipo de organización obliga a los autores a implementar funciones de conversión para calcular el identificador del KST en función del identificador del GKM y viceversa. En nuestro trabajo utilizamos la representación inversa, es decir, nuestro nodo raíz está almacenado en la dirección menos significativa de ambas memorias, por lo que el identificador de un elemento del KST es también su identificador en el GKM. La figura 2 muestra un ejemplo de organización GKM para 8 usuarios.

Level	Address	Key
0	0001	K_g
1	0010	$K_{0\text{-}3}$
	0011	$K_{4\text{-}7}$
2	0100	$K_{0\text{-}1}$
	0101	$K_{2\text{-}3}$
	0110	$K_{4\text{-}5}$
	0111	$K_{6\text{-}7}$
3	1000	K_0
	1001	K_1
	1010	K_2
	1011	K_3
	1100	K_4
	1101	K_5
	1110	K_6
	1111	K_7

Fig. 2. Organización de memoria para el GKT

IV. ARQUITECTURA HARDWARE

Esta sección describe todos los elementos que componen nuestro sistema. Para implementar dicho sistema hemos empleado una tarjeta de evaluación ML605 de Xilinx con una FPGA Virtex 6 XC6VLX240T [8].

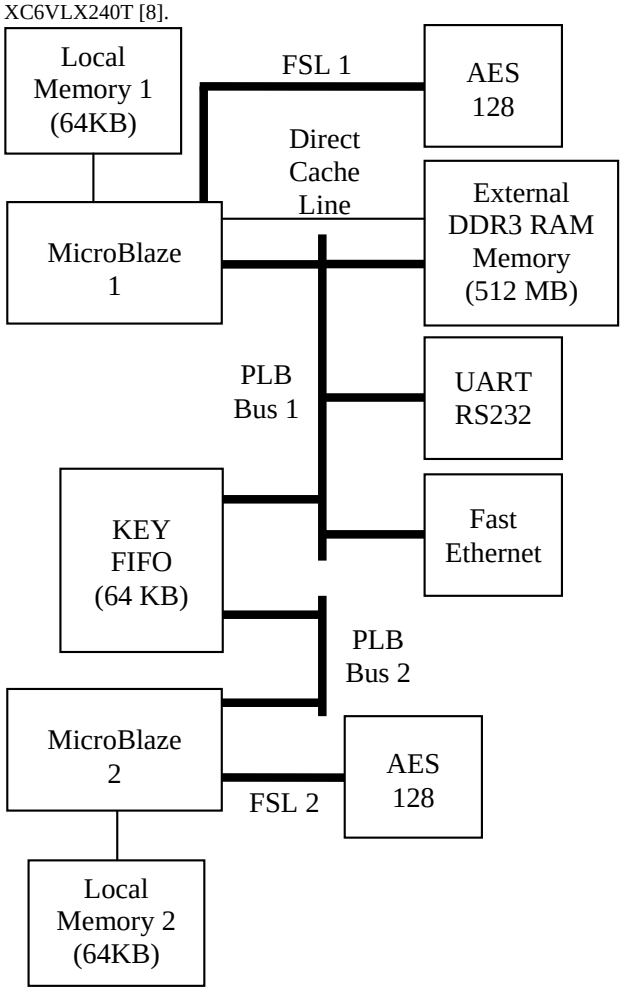

Fig. 3. Arquitectura Hardware del sistema

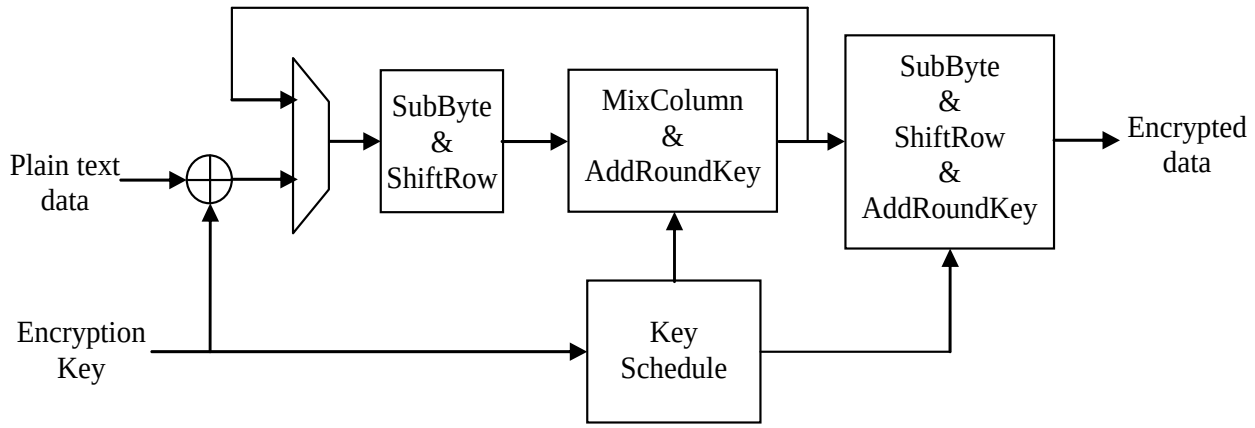

Fig. 4. Arquitectura AES

Como vemos en la figura 3, tenemos dos Microblaze v8.10.a [7], ambos trabajando a 200 MHz. El primero, Microblaze 1, es el procesador principal y realizará todas las tareas relativas a dar de alta y de baja usuarios. Además, este procesador encriptará las nuevas claves calculadas, las cuales son cargadas desde la memoria KEY FIFO, y enviadas a los usuarios por medio del elemento Fast Ethernet.

El Segundo procesador, MicroBlaze 2, tiene únicamente una tarea: crear nuevas claves. Para cada una de las claves calculadas, el MicroBlaze 2 almacena dichas claves en la memoria KEY FIFO. Este segundo procesador permite reducir el tiempo de ejecución que necesita el MicroBlaze 1 para dar de alta/baja usuarios en/del árbol de claves. Es importante recordad que se necesitan dos encriptaciones para crear una nueva clave (ver la sección *Descripción del problema*).

Respecto a la memoria KEY FIFO, se trata de una memoria de 64 KB implementada utilizando BlockRAMs (al ser las memorias BlockRAM dual-port, ambos procesadores pueden realizar lecturas y escrituras simultáneas) y se organiza como sigue: las primeras cuatro palabras de 32 bits almacenan los punteros de lectura y de escritura de la FIFO así como el número de lecturas y de escrituras realizadas. Elegimos esta estructura porque ambos MicroBlazes necesitan conocer estos cuatro datos para saber dónde deben leer/escribir y si la memoria está llena o vacía. A partir de la posición cuatro, se almacenan las nuevas claves calculadas. Como necesitamos cuatro palabras de 32 bits para almacenar una clave, la memoria KEY FIFO puede almacenar hasta 4.095 claves precalculadas.

El siguiente elemento crítico de nuestra arquitectura son los módulos AES. Estos módulos se encargan de realizar encriptaciones AES, pero, con el fin de reducir el tiempo de ejecución necesario para realizar cada encriptación, hemos realizado varias mejoras a la implementación AES estándar:

- Ejecución paralela: Cada uno de los bytes del *state* del AES se calcula en paralelo. Esta mejora implica que debemos implementar varias tablas S-Box, una para cada byte del *state*. Sin embargo, como las tablas S-Box están implementadas usando los bloques BlockRAM de la Virtex 6 que, como hemos dicho anteriormente son bloques dual-port, se puede utilizar una única S-Box para calcular dos bytes del *state*, por lo que únicamente necesitamos ocho tablas en lugar de 16.
- Implementación segmentada: Hemos realizado una implementación segmentada a nivel de fase del algoritmo AES. Esta segmentación tiene como propósito reducir el ciclo de reloj del módulo, ya que no se realizan encriptaciones segmentadas.
- Múltiples bloques cifrados con la misma clave: Hay que tener en cuenta que estos módulos AES no se encargan de encriptar grandes cantidades de datos, sino que únicamente encriptan datos de 128 bits y, en muchos casos, cada cifrado se realiza con claves diferentes, por lo que se han implementado teniendo en cuenta esta característica. Sin embargo, también se han diseñado de tal forma que puedan encriptarse varios bloques sin necesidad de cambiar la clave. Esto es muy útil en el proceso de generación de claves, donde todas las encriptaciones son hechas por la misma clave (K_{gen}). De esta forma conseguimos que, en el caso de encriptar una clave de usuario se necesitan enviar 256 bits al módulo, pero cuando se genera una nueva clave, únicamente es necesario enviar 128 bits.
- Implementación de la fase MixColumn: La fase MixColumn es la fase más costosa del algoritmo AES. Por este motivo, la hemos implementado por medio de la función xtime, la cual permite reducir el tiempo de ejecución de la fase a solamente un ciclo de reloj. Esto es gracias a que la función xtime reduce las operaciones de multiplicación 2^8GF a varias puertas XOR paralelas (Puede verse una explicación completa de la implementación de la fase MixColumn usando la función xtime en [2]).
- Combinación de fases: Para reducir el número de ciclos necesarios para encriptar un bloque, hemos combonado algunas fases del algoritmo. Concretamente, hemos realizado las siguientes combinaciones: En el bucle principal hemos combinado las fases SubByte y ShiftRow y MixColumn y AddRoundKey en una única fase; por otra parte, las tres últimas fases del algoritmo (las posteriores al bucle principal, SubByte, ShiftRow y AddRoundKey) las hemos combinado en una única fase. Gracias a estas mejoras, hemos reducido el número de ciclos de reloj de cuarenta en la versión estándar a solamente 20 en nuestra implementación de fases combinadas. La estructura completa de la implementación de fases combinadas del algoritmo AES puede verse en la figura 4.

De esta forma, y teniendo en cuenta que únicamente se encripta un bloque (recordemos que la segmentación es únicamente para reducir el ciclo de reloj), hemos conseguido un rendimiento de encriptación de 1’28 Gb/s. Por otra parte, los dos cores AES implementados están conectados a su MicroBlaze correspondiente por medio de conexiones FSLs (Fast Simple Links), que reducen el número de ciclos de comunicación respecto del bus PLB (Processor Local Bus).

Otro importante elemento es la memoria RAM DDR3 externa. Este es un elemento muy importante ya que va a almacenar completamente tanto el GKT y el KST, por lo que necesitamos y rápido acceso a memoria. Esta memoria está conectada al bus PLB pero, para garantizar el acceso más rápido posible, también está conectada al MicroBlaze 1 directamente por medio de una línea de caché de datos. La memoria RAM DDR3 trabaja a 400 MHz, es decir, el doble del ciclo de reloj del MicroBlaze 1.

Por último, hemos incluido dos módulos de comunicación al bus PLB 1: Un módulo RS232 UART y un módulo Fast Ethernet. El primer módulo nos permite comunicarnos con el MicroBlaze 1 para propósitos de experimentación, mientras que el segundo módulo permite enviar las claves a los usuarios correspondientes.

V. Resultados

En esta sección presentamos los resultados obtenidos en este trabajo y la comparación con otros autores.

Como dijimos anteriormente, hemos utilizado una tarjeta de evaluación de Xilinx ML605 con una FPGA Virtex 6 XC6VLX240T. La tabla 2 muestra la utilización de recursos tanto para el diseño completo como para cada módulo AES. Como podemos ver, empleamos muy pocos recursos tanto en el número de Slices ocupados como de módulos BlockRAMs. Concretamente utilizamos el 11% del total de Slices disponibles para implementar el diseño completo (el 1’1% para implementar cada uno de los módulos AES) y el 20% del total de los módulos BlockRAM disponibles (el 2’8% para cada

módulo AES). Esta baja ocupación nos permite implementar bastantes módulos hardware en trabajos futuros si fueran necesarios.

TABLA II

RESUMEN DE LA OCUPACIÓN DE RECURSOS UTILIZADOS

	Usados	Disponibles	Porcentaje de utilización
Diseño completo			
Número de Slices	4.216	37.680	11%
Numbero de BlockRAM	86	416	20%
Módulo AES			
Número de Slices	415	37.680	1.1%
Numbero de BlockRAM	16	416	3.8%

Por otra parte, la tabla 3 contiene los resultados obtenidos por nuestro diseño, concretamente los resultados relativos la mejor y la peor alta y baja. Estos datos están calculados usando un número total de 8.388.608 usuarios. Este número de usuarios puede ser un número realista de usuarios para aplicaciones de contenidos multimedia como la televisión por internet.

TABLA III

RESULTADOS OBTENIDOS

	Ciclos	Tiempo de ejecución (µs)
Mejor alta	844	4’22
Mejor baja	375	1’88
Peor alta	6.300	31’5
Peor baja	5.621	28’11

TABLA IV

COMPARATIVA DE RESULTADOS

Referencia	Tamaño del grupo	Tiempo de ejecución (ms)
Este trabajo	8.388.608	0’028
[5]	131.072	3’91
[6]	8.192	1’2
[5]	131.072	54’7
[1]	50	640

Finalmente, la tabla 4 compara nuestros resultados con los obtenidos por otros autores. Esta tabla muestra el tiempo de ejecución de la peor baja para varios trabajos así como su tamaño de grupo de usuarios. Como podemos ver, mejoramos claramente todos los tiempos de ejecución, pero es importante destacar que nuestro resultado es el tiempo de ejecución de la peor baja para 8.388.608 usuarios, mientras que el resto de trabajos mostrados en la tabla 4 son el tiempo de ejecución de la peor baja para grupos de usuarios mucho más pequeños (desde 50 hasta 131.072 usuarios). Esta diferencia en los tamaños de grupo convierte a nuestra implementación en una muy buena implementación, ya que no sólo mejoramos enormemente el tiempo de ejecución de trabajos previos, sino que, además, lo hacemos con un tamaño de grupo de usuarios mucho mayor que el resto.

VI. CONCLUSIONES

Este trabajo presenta un biprocesador MicroBlaze para realizar la gestión de claves para aplicaciones multicast de contenidos multimedia. Los resultados obtenidos muestran que nuestros resultados son mucho mejores que los obtenidos por otros autores, tanto en tiempo de ejecución como de tamaño de grupo de usuarios. Este buen resultado es conseguido gracias a la implementación realizada del módulo AES, el cual está implementado siguiendo diversas mejoras con el fin de reducir tanto el ciclo de reloj como el número de ciclos necesarios para encriptar un bloque de 128 bits (hemos reducido de 40 ciclos en la versión estándar a 20 ciclos en la versión mejorada), y por usar un segundo MicroBlaze para generar las nuevas claves en paralelo almacenándolas en una memoria FIFO implementada por medio de módulos BlockRAM.

Por otra parte, como trabajo futuro implementaremos una arquitectura segura completa. Esta arquitectura incluirá diversas funciones relativas a la seguridad, como son funciones MAC y HASH así como firmado digital por medio de algoritmos de clave pública. Además, se estudiará la posibilidad de añadir más procesadores MicroBlaze para mejorar el rendimiento del sistema (esto podemos hacerlo gracias a los pocos recursos utilizados en nuestra implementación.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por la Universidad de Extremadura a través del proyecto ACCVII-07.

REFERENCIAS

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, “On the Performance of Group Key Agreement Protocols” in ACM Trans. Information Systems Security, vol. 7, no. 3, pp. 457-488, 2004.

[2] J. M. Granado-Criado, M. A. Vega-Rodriguez, J. M. Sanchez-Perez and J. A. Gomez-Pulido, “A New Methodology to Implement the AES Algorithm using Partial and Dynamic Reconfiguration” in INTEGRATION, the VLSI Journal, Vol. 43, pp. 72–80, 2010.

[3] National Institute of Standards and Technology (NIST), “Advanced Encryption Standard (AES)” in Fed. Information Processing Standard 197, Nov. 2001.

[4] National Institute of Standards and Technology (NIST), “American National Standard for Financial Institution Key Management (Wholesale)” in Am. Banker Assoc., 1985.

[5] A. Shoufan and S. A. Huss, “High-Performance Rekeying Processor Architecture for Group Key Management” in IEEE Transactions on Computers, vol. 58, no. 10, October 2009, pp. 1421-1434.

[6] C.K. Wong, M. Gouda, and S.S. Lam, “Secure Group Communication Using Key Graph,” in IEEE/ACM Trans. Networking, vol. 8, no. 1, pp. 16-30, Feb. 2000.

[7] Xilinx, “MicroBlaze Processor Reference Guide (v12.0)”, 2011.

[8] Xilinx, “Virtex-6 Family Overview, DS150 (v2.4)”, 2012.