# Implementing Cryptographic Pairings on ARM Dual-Core Processors

R. Cuiman, A. Cabrera, and S. Sánchez-Solano

***Abstract*—In this paper, we explore the parallelization capabilities of the ARM processing system embedded in a Zynq device for a software implementation of the optimal Ate pairing. First, the use of the NEON coprocessor was evaluated. It was found that on ARM v7 Cortex-A9 processors the computation of the optimal Ate pairing based on NEON does not perform better than an optimized ARM-assembly equivalent implementation. Therefore, we moved to explore the parallelization of pairing computation using a dual-core processing approach. By organizing operations of line evaluation and point arithmetic formulas to have little data dependency, it was possible to schedule independent operations to be perfomed simultaneously in separate cores of an ARM dual-core Cortex-A9 processor. The same principle was applied in the arithmetic procedures of the extension fields. In this way, our software is able to perform $25.6\%$ and $6.6\%$ faster than the best two implementations previously reported on ARM Cortex-A9 processors.**

***Index Terms*—Pairing-based cryptography, Elliptic curves, Bilinear pairings, ARM Cortex-A9, NEON, Dual-core processing.**

## I. Introducción

La construcción de protocolos criptográficos basados en emparejamientos bilineales ha motivado el desarrollo de diversas investigaciones relacionadas con el cálculo eficiente de los mismos. Actualmente, la mejor opción a considerar en la práctica es el emparejamiento Ate óptimo (Ate-O) definido sobre alguna de las familias de curvas elípticas con propiedades favorables para el cálculo de emparejamientos [1]–[3]. Las dos etapas principales del cálculo del emparejamiento Ate-O son el bucle de Miller y la exponenciación final, las cuales a su vez involucran operaciones aritméticas sobre campos finitos.

Se han reportado numerosas implementaciones del emparejamiento Ate-O en la literatura, la mayoría destinadas a computadoras personales [4]–[8]. Sin embargo, la creciente presencia de dispositivos móviles en nuestro quehacer cotidiano ha despertado un gran interés por la implementación de emparejamientos bilineales en procesadores ARM. Entre los ejemplos más notables se encuentran las realizadas en [9], [10] y [11]. La implementación desarrollada en [9] invierte aproximadamente $51010 \times 10^3$ ciclos de reloj en el cálculo del emparejamiento Ate-O en coordenadas afines para un nivel de seguridad de $128$ bits. Por su parte, el mejor resultado obtenido en [10] para un mismo nivel de seguridad fue de $11886 \times 10^3$ ciclos de reloj, fruto de una implementación optimizada en lenguaje ensamblador utilizando coordenadas proyectivas estándares. La biblioteca desarrollada en [11] supera los resultados de tiempo anteriores al valerse de la posibilidad que tienen las instrucciones NEON [12] para ejecutar cálculos en paralelo.

En ninguna de las investigaciones anteriores se consideró explícitamente el empleo de procesamiento multinúcleo. Por esa razón, nuestro objetivo inicial consistía en continuar el trabajo de [11] al combinar el empleo de instrucciones NEON con la capacidad añadida de realizar operaciones independientes en núcleos separados de un procesador ARM doble núcleo. La idea era utilizar NEON para construir primitivas eficientes para las operaciones de multiplicación y cuadrado en la extensión de campo de segundo grado $\mathbb{F}_{p^2}$, para luego ejecutar dos operaciones independientes de $\mathbb{F}_{p^2}$ de manera simultánea, cada una en un núcleo diferente del procesador. No obstante, se encontró que en un procesador ARM Cortex-A9 el cálculo simultáneo de dos multiplicaciones de múltiple precisión utilizando NEON resultó ser menos eficiente que calcular de manera consecutiva dos multiplicaciones de múltiple precisión codificadas en lenguaje ensamblador de ARM. Por tanto, el enfoque de procesamiento doble núcleo fue utilizado para realizar operaciones de $\mathbb{F}_{p^2}$ en paralelo, pero implementando la aritmética de múltiple precisión subyacente en lenguaje ensamblador en lugar de emplear NEON. Como resultado, nuestros tiempos de ejecución para el bucle de Miller son menores en $41\%$ y $24\%$ a los obtenidos por [10] y [11] respectivamente, lo cual se traduce en mejoras respectivas de $25.6\%$ y $6.6\%$ en el rendimiento total del cálculo del emparejamiento Ate-O para un nivel de seguridad de $128$ bits.

El resto del trabajo se encuentra organizado de la siguiente manera: La Sección II proporciona algunos aspectos preliminares sobre el emparejamiento Ate-O en curvas de Barreto-Naehrig y la Sección III presenta la torre de extensiones empleada para la representación de los campos finitos subyacentes. En la Sección IV se discute la implementación de las operaciones aritméticas en la extensión de campo $\mathbb{F}_{p^2}$ tanto en lenguaje ensamblador de ARM como utilizando instrucciones NEON. La Sección V aborda el enfoque de procesamiento doble núcleo para paralelizar operaciones aritméticas en el contexto del emparejamientos Ate-O. Seguidamente, la Sección VI analiza el impacto del procesamiento doble núcleo en el costo computacional del emparejamiento Ate-O y compara nuestros resultados contra los obtenidos en trabajos previos. Finalmente, se ofrecen las consideraciones finales del trabajo en la Sección VII.

R. Cuiman, Departmento de Automática y Computación, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, Cuba, raudel@automatica.cujae.edu.cu.

A. J. Cabrera, Departmento de Automática y Computación, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, Cuba, alex@automatica.cujae.edu.cu.

S. Sánchez-Solano, Instituto de Microelectrónica de Sevilla, IMSE-CNM, CSIC/Universidad de Sevilla, Sevilla, España, santiago@imse-cnm.csic.es.

## II. Emparejamiento Ate-O en Curvas de Barreto-Naehrig

Las curvas de Barreto-Naehrig [2] constituyen una de las familias de curvas elípticas más populares para el cálculo eficiente de emparejamientos bilineales. Dichas curvas se encuentran definidas por la ecuación $E(\mathbb{F}_p) : y^2 = x^3 + b$, poseen orden primo $r = \#E(\mathbb{F}_p)$ y grado de incrustación $k = 12$. La característica $p$ del campo finito subyacente y el orden de la curva $r$ son parametrizados mediante los siguientes polinomios:

$$\begin{aligned} p &= p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1 \\ r &= r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1 \end{aligned} \quad (1)$$

El parámetro $z \in \mathbb{Z}$ debe ser seleccionado de manera que $p = p(z)$ y $r = r(z)$ sean ambos números primos. Además, es conveniente seleccionar el valor de $z$ de manera que el primo $p$ resultante permita construir la extensión de campo $\mathbb{F}_{p^{12}}$ con propiedades favorables para realizar operaciones aritméticas de manera eficiente [13]. En este sentido, para un nivel de seguridad de 128 bits, la mejor opción es $z = -4080000000000001_{16}$ [14]. Sin embargo, quizás la propiedad más útil de las curvas de Barreto-Naehrig radica en la existencia de una curva $E'$ isomórfica con $E$ que permite trasladar gran parte de las operaciones involucradas en el cálculo del emparejamiento desde $\mathbb{F}_{p^{12}}$ hacia la extensión de menor grado $\mathbb{F}_{p^2}$.

La ecuación de $E'$ es $E'(\mathbb{F}_{p^2}) : y'^2 = x'^3 + b'$, donde $b' = b\xi^{-1}$ o $b' = b\xi^{-5}$ en dependencia de cuál de los dos valores conduce a que el orden $\#E'(\mathbb{F}_{p^2})$ de $E'$ sea divisible entre $r$. El elemento $\xi$ es un no-residuo cuadrático y no-residuo cúbico en $\mathbb{F}^*_{p^2}$ de modo que el binomio $W^6 - \xi$ es irreducible sobre $\mathbb{F}_{p^2}[W]$. Sea $\alpha \in \mathbb{F}_{p^{12}}$ una raíz de $W^6 - \xi$, el mapa $\psi : E'(\mathbb{F}_{p^2}) \mapsto E(\mathbb{F}_{p^{12}})$ que permite transformar puntos de $E'(\mathbb{F}_{p^2})$ en puntos de $E(\mathbb{F}_{p^2})$ es $(x', y') \mapsto (\alpha^2 x', \alpha^3 y')$ [15]. En caso que $\xi$ sea utilizado además para construir la extensión de campo $\mathbb{F}_{p^{12}}$, entonces evaluar $\psi(Q')$ tiene un costo bastante pequeño pues se convierte básicamente en un reordenamiento de coeficientes en la representación en $\mathbb{F}_{p^{12}}$ de las coordenadas $(x', y')$ de $Q'$ unido a alguna posible multiplicación por $\xi$, la cual se puede realizar mediante una suma y una resta en $\mathbb{F}_{p^2}$.

Denótese como $E(\mathbb{F}_{p^{12}})[r]$ a los subgrupos de torsión-r de la curva. O sea, $E(\mathbb{F}_{p^{12}})[r] = \{P \in E : [r]P = \mathcal{O}\}$, donde $[a]P$ representa la multiplicación del punto $P$ por el escalar $a \in \mathbb{Z}$ y $\mathcal{O}$ es el punto en el infinito [16]. Téngase $P \in E(\mathbb{F}_p)$, $Q \in E(\mathbb{F}_{p^{12}})[r]$ y $Q' \in E'(\mathbb{F}_{p^2})[r]$, donde $Q = \psi(Q')$. El emparejamiento Ate-O [5], [17] sobre curvas de Barreto-Naehrig consiste en el mapa bilineal $a_{\text{opt}} : E'(\mathbb{F}_{p^2})[r] \times E(\mathbb{F}_p) \mapsto \mu_r$ dado por,

$$\begin{aligned} a_{\text{opt}}(Q', P) = (f_{6z+2,\psi(Q')}(P) \cdot l_{[6z+2]\psi(Q'),\pi_p(\psi(Q'))}(P) \cdot \\ l_{[6z+2]\psi(Q')+\pi_p(\psi(Q')),-\pi_p^2(\psi(Q'))}(P))^{(p^{12}-1)/r}, \end{aligned} \quad (2)$$

donde $z$ es el parámetro de Barreto-Naehrig y $l_{Q_1,Q_2}(P) \in \mathbb{F}_{p^{12}}$ es la ecuación de la línea correspondiente a la suma de los puntos $Q_1, Q_2 \in G_2$, evaluada en el punto $P$. La ecuación (2) asegura que $a_{\text{opt}}(Q', P) \in \mu_r$, donde $\mu_r \subset \mathbb{F}^*_{p^{12}}$ es el grupo de raices unitarias de grado $r$.

### A. Algoritmo de Miller

El Algoritmo 1 reproduce la secuencia de pasos del algoritmo de Miller [18] adaptado para el cómputo del emparejamiento Ate-O sobre curvas de Barreto-Naehrig. Gracias a la existencia del isomorfismo $\psi$, el cálculo de la tangente $l_{\psi(T'),\psi(T')}$ junto con el doblado de punto $[2]T'$; así como de la línea secante $l_{\psi(T'),\psi(Q')}$ con la suma de puntos $T' + Q'$ se llevan a cabo en $\mathbb{F}_{p^2}$. Las operaciones de elevar al cuadrado la variable de Miller $f$, al igual que la multiplicación de $f$ por los resultados de evaluar las líneas anteriores tienen lugar en $\mathbb{F}_{p^{12}}$. No obstante, dado que la mitad de los coeficientes de $l_{\psi(T'),\psi(T')}(P), l_{\psi(T'),\psi(Q')}(P) \in \mathbb{F}_{p^{12}}$ son cero, dichas multiplicaciones se realizan mediante un caso especial denominado *multiplicación dispersa* que resulta más eficiente que una multiplicación genérica en $\mathbb{F}_{p^{12}}$.

Algoritmo 1. Algoritmo de Miller para el emparejamiento Ate-O sobre curvas de Barreto-Naehrig.

**Entradas:** $P \in G_1$, $Q' \in G_2'$, $\ell = 6z + 2 = \sum_{i=0}^{\lceil \log_2(\ell) \rceil} \ell_i 2^i$
**Salidas:** $a_{\text{opt}}(Q', P)$

1: $f \leftarrow 1$, $T' \leftarrow Q'$
2: **for** $i = \lfloor \log_2(\ell) \rfloor$-1 **downto** 0 **do**
3: $\quad f \leftarrow f^2$
4: $\quad f \leftarrow f \cdot l_{\psi(T'),\psi(T')}(P)$
5: $\quad T' \leftarrow [2]T'$
6: $\quad$ **if** $\ell_i = 1$ **then**
7: $\quad\quad f \leftarrow f \cdot l_{\psi(T'),\psi(Q')}(P)$
8: $\quad\quad T' \leftarrow T' + Q'$
9: $\quad$ **end if**
10: **end for**
11: $Q'_1 \leftarrow \psi^{-1}(\pi_p(\psi(Q')))$
12: $f \leftarrow f \cdot l_{\psi(T'),\psi(Q'_1)}(P)$
13: $T' \leftarrow T' + Q'_1$
14: $Q'_1 \leftarrow -\pi_p(Q'_1)$
15: $f \leftarrow f \cdot l_{\psi(T),\psi(Q'_1)}(P)$
16: $f \leftarrow f^{(p^{12}-1)/r}$
17: Devolver $f$.

*1) Exponenciación final:* El paso 16 del Algoritmo 1 se refiere a la etapa de exponenciación final. Para acelerar el cálculo de la misma, el exponente $(p^{12} - 1)/r$ se descompone en una parte *fácil* y otra *difícil* según se muestra a continuación:

$$(p^{12} - 1)/r = \underbrace{(p^6 - 1) \cdot (p^2 + 1)}_{\text{exponente fácil}} \cdot \underbrace{[(p^4 - p^2 + 1)/r]}_{\text{exponente difícil}}$$

El cálculo de $f^{(p^6-1)\cdot(p^2+1)}$ es relativamente simple. Requiere una inversa y dos multiplicaciones en $\mathbb{F}_{p^{12}}$ así como una exponenciación a la potencia de $p^2$, la cual puede realizarse casi libre de costo mediante la aplicación del endomorfismo de Frobenius $\pi_p : E \mapsto E : (x, y) \mapsto (x^p, y^p)$ en $\mathbb{F}_{p^{12}}$.

Para la parte difícil se utilizó la estrategia propuesta en [19], la cual representa el exponente $(p^4 - p^2 + 1)/r$ en base $p$ como $\chi_3 p^3 + \chi_2 p^2 + \chi_1 p + \chi_0$. Los coeficientes $\chi_i$ son expresados

en función del parámetro $z$ según:

$$\chi_0(z) = -36z^3 - 30z^2 - 18z - 2$$
$$\chi_1(z) = -36z^3 - 18z^2 - 12z + 1$$
$$\chi_2(z) = 6z^2 + 1$$
$$\chi_3(z) = 1$$

Tomando $m \in \mathbb{F}_{p^{12}}$ como el resutado de la parte fácil, entonces $m^{(\chi_3 p^3 + \chi_2 p^2 + \chi_1 p + \chi_0)}$ se convierte en:

$$\underbrace{[m^p \cdot m^{p^2} \cdot m^{p^3}]}_{y_0} \cdot \underbrace{[1/m]}_{y_1}^2 \cdot \underbrace{[(m^{z^2})^{p^2}]}_{y_2}^6 \cdot \underbrace{[1/(m^z)^p]}_{y_3}^{12} \cdot$$
$$\underbrace{[1/(m^z \cdot (m^{z^2})^p)]}_{y_4}^{18} \cdot \underbrace{[1/m^{z^2}]}_{y_5}^{30} \cdot \underbrace{[1/(m^{z^3} \cdot (m^{z^3})^p)]}_{y_6}^{36}$$

Finalmente, la expresión $y_0 \cdot y_1^2 \cdot y_2^6 \cdot y_3^{12} \cdot y_4^{18} \cdot y_5^{30} \cdot y_6^{36}$ puede ser evaluada de manera eficiente mediante una cadena de adicicón de pequeña longitud [20]. Los términos $m^p$, $m^{p^2}$ y $m^{p^3}$ son calculados fácilmente mediante el endomorfismo de Frobenius. Por su parte, para los términos $m^z$, $m^{z^2}$ y $m^{z^3}$ se utilizó el enfoque convencional de exponenciación binaria de izquierda a derecha.

## III. Torre de Extensiones de Campos

Koblitz y Menezes [21] fueron los primeros en advertir que construir $\mathbb{F}_{p^k}$ como una torre de extensiones de campos permitía acelerar notablemente la aritmética en $\mathbb{F}_{p^k}$. Cuando se trabaja con curvas de Barreto-Naehrig la torre debe comenzar con una extensión cuadrática sobre $\mathbb{F}_p$ dado que la curva isomórfica se encuentra definida sobre $\mathbb{F}_{p^2}$. La torre puede continuar con una extensión cuadrática sobre $\mathbb{F}_{p^2}$ para construir $\mathbb{F}_{p^4}$ y finalizar entonces con una extensión cúbica sobre $\mathbb{F}_{p^4}$ para obtener $\mathbb{F}_{p^{12}}$; o, por el contrario, construir $\mathbb{F}_{p^6}$ como una extensión cúbica de $\mathbb{F}_{p^2}$ y $\mathbb{F}_{p^{12}}$ como una extensión cuadrática sobre $\mathbb{F}_{p^6}$. Esta segunda alternativa ha sido ampliamente utilizada en la literatura [4]–[6], [9], [10]. En particular, de acuerdo a las recomendaciones realizadas en [22], este trabajo adopta la siguiente torre:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta), \text{donde } \beta = -1$$
$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi), \text{donde } \xi = u + 1$$
$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$$

Un elemento $a \in \mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta)$ se representa como $a = a_{[0]} + a_{[1]}u$, donde $a_{[0]}, a_{[1]} \in \mathbb{F}_p$. El par de elementos $b \in \mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$ y $c \in \mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$ se representan a través de los polinomios $b = b_{[0]} + b_{[1]}v + b_{[2]}v_2$ y $c = c_{[0]} + c_{[1]}w$ respectivamente, donde $b_{[0]}, b_{[1]}, b_{[2]} \in \mathbb{F}_{p^2}$ mientras que $c_{[0]}, c_{[1]} \in \mathbb{F}_{p^6}$. De este modo, $c \in \mathbb{F}_{p^{12}}$ puede representarse también mediante las ecuaciones (3) o (4). Los coeficientes $c_{[00]} \ldots c_{[12]}$ en la ecuación (3) corresponden a elementos en $\mathbb{F}_{p^2}$ mientras que los coeficientes $c_{[000]} \ldots c_{[121]}$ en la ecuación (4) se encuentran en $\mathbb{F}_p$.

$$c = c_{[00]} + c_{[01]}v + c_{[02]}v^2 + (c_{[10]} + c_{[11]}v + c_{[12]}v^2)w \quad (3)$$

$$\begin{aligned} c = {} & (c_{[000]} + c_{[001]}u) + (c_{[010]} + c_{[011]}u)v \\ & + (c_{[020]} + c_{[021]}u)v^2 + [(c_{[100]} + c_{[101]}u) \\ & + (c_{[110]} + c_{[111]}u)v + (c_{[120]} + c_{[121]}u)v^2]w \end{aligned} \quad (4)$$

De ahora en adelante se empleará la notación anterior para hacer referencia a coeficientes específicos dentro de la representación polinómica de elementos en $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ o $\mathbb{F}_{p^{12}}$.

## IV. Implementación de la Aritmética en $\mathbb{F}_{p^2}$

Antes de abordar los detalles de las implementaciones realizadas en este trabajo, es oportuno mencionar que durante la fase experimental del mismo se utilizó la placa de desarrollo ZedBoard como plataforma de implementación. La misma cuenta con un dispositivo XC7Z020 de la serie Zynq-7000 de Xilinx, que contiene un sistema de procesamiento basado en un procesador ARM Cortex-A9 doble núcleo excitado a una frecuencia de 667 MHz [23]. Es válido señalar entonces que las muestras de tiempo fueron adquiridas por medio del temporizador global de 64 bits de la familia Cortex-A9, el cual es excitado a la mitad de la frecuencia del procesador. Igualmente, los tiempos promedio que se presentan de aquí en adelante fueron calculados en base a 10000 repeticiones independientes de la operación bajo prueba en cada caso.

### A. Aritmética en $\mathbb{F}_{p^2}$ utilizando Ensamblador de ARM

Como se mencionaba en la Sección II-A, la mayoría de las operaciones aritméticas utilizadas en el algoritmo de Miller para el emparejamiento Ate-O sobre curvas de Barreto-Naehrig tienen lugar en $\mathbb{F}_{p^2}$ y $\mathbb{F}_{p^{12}}$. Además, vale destacar que, debido a la construcción de la torre de extensiones de campos $\mathbb{F}_p \rightarrow \mathbb{F}_{p^2} \rightarrow \mathbb{F}_{p^6} \rightarrow \mathbb{F}_{p^{12}}$, la aritmética en $\mathbb{F}_{p^{12}}$ se compone de operaciones aritméticas en $\mathbb{F}_{p^6}$ que a su vez están conformadas por operaciones aritméticas en $\mathbb{F}_{p^2}$. Por ende, para acelerar el cálculo del emparejamiento Ate-O sobre curvas de Barreto-Naehrig, resulta conveniente, en primera instancia, implementar las operaciones aritméticas en $\mathbb{F}_{p^2}$ de la forma más eficiente posible.

En el caso de la multiplicación en $\mathbb{F}_{p^2}$ se utilizó el método de Karatsuba [24]. El Algoritmo 2 muestra la secuencia de operaciones correspondientes. Gracias a que el parámetro $z$ seleccionado en la Sección II produce un primo $p$ de 254 bits, es posible utilizar la técnica de reducción demorada [25]. En consecuencia, todas las multiplicaciones, sumas y restas del Algoritmo 2 corresponden a operaciones convencionales de la aritmética de números enteros. Las reducciones modulares son pospuestas y realizadas en un solo paso al final del algoritmo. En cuanto a la operación de elevar al cuadrado, el Algoritmo 3 muestra el procedimiento correspondiente. Dicho procedimiento tiene su base en la fórmula que se emplea para elevar al cuadrado números complejos [26]. Nótese que la técnica de reducción demorada también se utiliza en este caso.

Las operaciones que gobiernan el tiempo de ejecución de los algoritmos de multiplicación y cuadrado en $\mathbb{F}_{p^2}$ son la multiplicación de números enteros y la reducción modular. En el caso que ocupa este trabajo se trata de multiplicaciones de números enteros de 256 bits y reducciones de números enteros de 512 bits módulo $p$. Por tal motivo, la primera alternativa en pos de una implementación eficiente de la aritmética en $\mathbb{F}_{p^2}$ fue desarrollar un par de subrutinas optimizadas en lenguaje ensamblador para la multiplicación y la

Algoritmo 2. Multiplicación de Karatsuba en $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2+1)$ utilizando reducción demorada.

**Entradas:** Elementos $a = a_{[0]} + a_{[1]}u, b = b_{[0]} + b_{[1]}u \in \mathbb{F}_{p^2}$
**Salidas:** $c = a \cdot b = c_{[0]} + c_{[1]}u \in \mathbb{F}_{p^2}$
1: $t_1 = a_{[0]} + a_{[1]}$
2: $t_2 = b_{[0]} + b_{[1]}$
3: $t_3 = t_1 \cdot t_2$
4: $t_1 = a_{[1]} \cdot b_{[1]}$; $t_2 = a_{[0]} \cdot b_{[0]}$
5: $t_3 = t_3 - t_1$
6: $t_3 = t_3 - t_2$
7: $t_2 = t_2 - t_1$
8: $c_{[1]} = t_3 \bmod p$; $c_{[0]} = t_2 \bmod p$

Algoritmo 3. Cuadrado en $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2+1)$ utilizando reducción demorada.

**Entradas:** Elemento $a = a_{[0]} + a_{[1]}u \in \mathbb{F}_{p^2}$
**Salidas:** $c = a^2 = c_{[0]} + c_{[1]}u \in \mathbb{F}_{p^2}$
1: $t_1 = a_{[0]} + a_{[1]}$
2: $t_2 = a_{[0]} - a_{[1]}$
3: $t_3 = a_{[0]} \cdot a_{[1]}$; $t_4 = t_1 \cdot t_2$
4: $t_3 = 2t_3$
5: $c_{[1]} = t_3 \bmod p$; $c_{[0]} = t_4 \bmod p$

reducción respectivamente denominadas *asm_mp_mul_256* y *asm_mp_reduct_512*.

La subrutina *asm_mp_mul_256* es una implementación del algoritmo de multiplicación de múltiple precisión [27] en la que se han desenrollado los bucles para obtener un mejor rendimiento. La Fig. 1 muestra fragmentos del código fuente de la iteración #1, la iteración #2 y la iteración #8. Las iteraciones de la tercera a la séptima junto con la segunda y la octava son muy similares entre sí, la diferencia radica en la palabra del multiplicando que se utiliza, así como las palabras del producto que se calculan en cada caso. El multiplicando, el multipicador y el producto se ubican en memoria a partir de las direcciones apuntadas por los registros del procesador r1, r2 y r0 respectivamente. Las palabras del multiplicador son inicialmente almacenadas en los registros r4-r11 para evitar leerlas nuevamente en cada iteración y reducir así la cantidad necesaria de accesos a memoria. No obstante, el mayor provecho en la implementación de la subrutina *asm_mp_mul_256* se extrae de un uso intensivo de la instrucción umaal del repertorio ARM que permite efectuar en un solo paso la operación $(C_{\text{out}}, V) = U + a \cdot b + C_{\text{in}}$, la cual constituye el cálculo central del algoritmo de multiplicación de múltiple precisión.

La subrutina *asm_mp_reduct_512* implementa el método de reducción de Montgomery [28] de acuerdo a la variante SOS (*Separated Operand Scanning*) propuesta en [29]. Debido a la similitud de este algoritmo con el de multiplicación de múltiple precisión en cuanto a estructura y a las operaciones involucradas, en su implementación se siguió el mismo principio de desenrollar los bucles e igualmente se utilizó de manera intensiva la instrucción umaal.

La primeras dos filas de la Tabla I muestran los tiempos de ejecución de *asm_mp_mul_256* y *asm_mp_reduct_512* respectivamente. Seguidamente aparecen los tiempos obtenidos para la multiplicación en $\mathbb{F}_p$, así como para la multiplicación y el cuadrado en $\mathbb{F}_{p^2}$ al utilizar como base dichas subrutinas.

TABLA I
TIEMPOS PARA LA ARITMÉTICA EN $\mathbb{F}_p$ Y $\mathbb{F}_{p^2}$ BASADA EN LAS SUBRUTINAS *asm_mp_mul_256* Y *asm_mp_reduct_512*

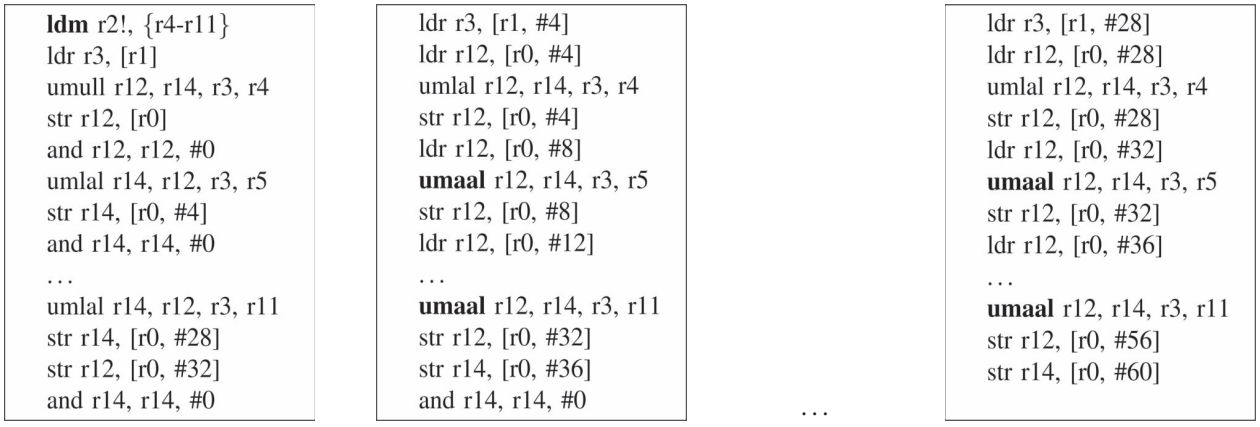
| Operación | Tiempos ($10^3$ ciclos de reloj) |
|---|---|
| Multiplicación de 256 bits | 0.23 |
| Reducción de 512 bits | 0.37 |
| Multiplicación en $\mathbb{F}_p$ | 0.60 |
| Multiplicación en $\mathbb{F}_{p^2}$ | 1.68 |
| Cuadrado en $\mathbb{F}_{p^2}$ | 1.35 |

### *B. Aritmética en $\mathbb{F}_{p^2}$ utilizando NEON*

El par de multiplicaciones del paso 4, así como las dos reducciones modulares del paso 8 del Algoritmo 2 han sido escritas deliberadamente en una sola línea para enfatizar que se trata de operaciones sin dependencia de datos que pueden por tanto ser ejecutadas simultáneamente. Lo mismo ocurre con los pasos 3 y 5 del Algoritmo 3. Esto fue aprovechado en [11] al explotar la capacidad que posee el repertorio de instrucciones NEON para ejecutar operaciones en paralelo. En consecuencia, con el objetivo de acelerar la aritmética en $\mathbb{F}_{p^2}$, el citado trabajo describe la implementación de dos subrutinas para calcular simultáneamente dos multiplicaciones de múltiple precisión y dos reducciones de Montgomery respectivamente. Siguiendo ese mismo enfoque, en este trabajo fueron implementadas un par de subrutinas similares a las proporcionadas por la biblioteca desarrollada en [11] (https://sandia.cs.cinvestav.mx/projects/3-neon-abe). No obstante, en nuestro caso se encontró conveniente realizar las modificaciones descritas a continuación, las cuales permitieron reducir el uso de memoria y además, en el caso específico de la multiplicación, obtener una mejora en cuanto a tiempo de ejecución. Las modificaciones fueron las siguientes:

- Reemplazo de las llamadas al intrínseco vld1_u32 por llamadas a vld1_lane_u32 en ambos procedimientos (multiplicación de números enteros y reducción de Montgomery). Esto permitió eliminar cuatro arreglos de 16 palabras de 32 bits, así como evitar la necesidad de entrelazar las palabras de los operandos de entrada.
- Reemplazo del intrínseco vld1q_u64 por el intrínseco vmovq_n_u64 que resulta ser una alternativa más eficiente para cargar la constante 0x00000000FFFFFFFFUL utilizada en subsecuentes llamadas a vandq_u64 durante el procedimeinto de multiplicación.
- Empleo del intrínseco vreinterpretq_u32_u64 en lugar de vmovn_u64 ya que vreinterpretq_u32_u64 no genera código equivalente en ensamblador y por tanto su empleo es libre de costo.
- Al inicio de cada iteración se eliminó la limpieza innecesaria del primer elemento del arreglo temporal que almacena los resultados intermedios del producto.

Como resultado, nuestra implementación ahorra 256 bytes de memoria y reduce el tiempo de ejecución promedio de la




(a) Iteración #1 (b) Iteración #2 (c) Iteración #8

Fig. 1. Fragmento de la subrutina *asm_mp_mul_256*: multiplicación de múltiple precisión de 256 bits en lenguaje ensamblador de ARM.

multiplicación en aproximadamente 42 ciclos de reloj. Este no es un aumento de rendimiento significativo si se considera una sola multiplicación, pero se convierte en una mejora más notable cuando se tienen en cuenta todas las multiplicaciones que son necesarias durante el cálculo de un emparejamiento.

La Tabla II presenta los resultados de tiempo para la aritmética basada en NEON. La columna "factor" se refiere a la razón entre el tiempo de ejecución de una operación basada en NEON y el correspondiente a dos ejecuciones consecutivas de la operación equivalente implementada en lenguaje ensamblador. Esta relación puede verse como una medida de cuánta mejora proporcionaría NEON en cada caso. El objetivo deseado es obtener factores de mejora por debajo de 1: mientras menor sea el valor, mejor sería el rendimiento obtenido. Sorpresivamente, los resultados no fueron los esperados. Nótese que es mejor realizar dos multiplicaciones de múltiple precisión consecutivas o dos reducciones modulares consecutivas en ensamblador que realizarlas simultáneamente mediante el uso de NEON, lo cual implica que suceda exactamente lo mismo con la multiplicación en $\mathbb{F}_p$. Estos resultados se atribuyen a que no existe un equivalente a la instrucción umaal de ARM en el repertorio de instrucciones NEON. Por tanto, para calcular operaciones del tipo $(C_{\text{out}}, V) = U + a \cdot b + C_{\text{in}}$ es necesario utilizar tres intrínsecos diferentes: vmlal_u32, vaddq_u64 y vshrq_n_u64. Los dos primeros son los que ejecutan en realidad la operación de multiplicación y acumulación, mientras que el tercero se encarga de recuperar el acarreo a ser propagado sucesivamente.

TABLA II
TIEMPOS PARA LA MULTIPLICACIÓN DE MÚLTIPLE PRECISIÓN, LA REDUCCIÓN MÓDULO $p$ Y LA MULTIPLICACIÓN EN $\mathbb{F}_p$ UTILIZANDO NEON

| Operación | Tiempos ($10^3$ ciclos de reloj) | factor |
|---|---|---|
| Multiplicación de 256 bits | 0.50 | 1.09 |
| Reducción de 512 bits | 0.76 | 1.03 |
| Multiplicación en $\mathbb{F}_p$ | 1.26 | 1.05 |

Tal como refleja la Tabla III, los tiempos de ejecución de las operaciones en $\mathbb{F}_{p^2}$ basadas en NEON son impactados negativamente por los resultados adversos mostrados en la Tabla II. Tanto la multiplicación como el cuadrado en $\mathbb{F}_{p^2}$ basados en NEON son alrededor de un 3% más lentos que sus contrapartes en lenguaje ensamblador.

TABLA III
TIEMPOS PARA LA ARITMÉTICA EN $\mathbb{F}_{p^2}$ BASADA EN NEON

| Operación | Tiempos ($10^3$ ciclos de reloj) |
|---|---|
| Multiplicación en $\mathbb{F}_{p^2}$ | 1.74 |
| Cuadrado en $\mathbb{F}_{p^2}$ | 1.40 |

## V. ARITMÉTICA CON PROCESAMIENTO DOBLE NÚCLEO

Esta sección evalúa el enfoque de procesamiento doble núcleo para acelerar la aritmética de campo involucrada en el cálculo del emparejamiento Ate-O. La idea es extraer el mayor grado de paralelismo posible al organizar las operaciones involucradas en varios niveles del cálculo del emparejamiento para que tengan poca dependencia de datos entre ellas. De esta manera, aquellas operaciones independientes podrán ser distribuidas adecuadamente para ser ejecutadas en paralelo en núcleos diferentes del procesador.

### A. Cálculo de Líneas y Aritmética de Puntos

Tanto para el cálculo de líneas como para la aritmética de puntos se utilizaron coordenadas Jacobianas. En el caso de las curvas de Barreto-Naehrig, dado $T' = (X_{T'} : Y_{T'} : Z_{T'})$ en coordenadas Jacobianas, las ecuaciones (5) y (6) permiten calcular el punto $R' = [2]T' = (X_{R'} : Y_{R'} : Z_{R'})$ y la tangente $l_{\psi(T'),\psi(T')}(P)$ respectivamente. El punto $P = (x_P, y_P)$ se utiliza en coordenadas afines.

$$\begin{aligned} X_{R'} &= 9X_{T'}^4 - 8X_{T'}Y_{T'}^2 \\ Y_{R'} &= 3X_{T'}^2(4X_{T'}Y_{T'}^2 - X_{R'}) - 8Y_{T'}^4 \\ Z_{R'} &= 2Y_{T'}Z_{T'} \end{aligned} \tag{5}$$

$$\begin{aligned} l_{\psi(T'),\psi(T')}(P) = {} & 2Y_{T'}(y_P Z_{T'}^3 - Y_{T'}\alpha^3) \\ & + 3X_{T'}^2(X_{T'}\alpha^3 - x_P Z_{T'}^2\alpha) \end{aligned} \tag{6}$$

De manera similar, dados $T' = (X_{T'} : Y_{T'} : Z_{T'})$ y $Q' = (X_{Q'} : Y_{Q'} : Z_{Q'})$, la ecuaciones (7) y (8) calculan el punto $R' = T' + Q' = (X_{R'} : Y_{R'} : Z_{R'})$ y la línea secante $l_{\psi(T'),\psi(Q')}(P)$ repectivamente.

$$\begin{aligned} X_{R'} &= (Y_{Q'}Z_{T'}^3 - Y_{T'}Z_{Q'}^3)^2 \\ &\quad - (X_{Q'}Z_{T'}^2 - X_{T'}Z_{Q'}^2)^2(X_{Q'}Z_{T'}^2 + X_{T'}Z_{Q'}^2) \\ Y_{R'} &= (Y_{Q'}Z_{T'}^3 - Y_{T'}Z_{Q'}^3)[X_{T'}Z_{Q'}^2(X_{Q'}Z_{T'}^2 - X_{T'}Z_{Q'}^2)^2 \\ &\quad - X_{R'}] - Y_{T'}Z_{Q'}^3(X_{Q'}Z_{T'}^2 - X_{T'}Z_{Q'}^2)^3 \\ Z_{R'} &= Z_{T'}Z_{Q'}(X_{Q'}Z_{T'}^2 - X_{T'}Z_{Q'}^2) \end{aligned} \tag{7}$$

$$\begin{aligned} &l_{\psi(T'),\psi(Q')}(P) = \\ &\quad (y_P Z_{Q'}Z_{T'}^3 - Y_{T'}Z_{Q'}\alpha^3)(X_{Q'}Z_{T'}^2 - X_{T'}Z_{Q'}^2) + \\ &\qquad (X_{T'}\alpha^3 - x_P Z_{T'}^2\alpha)(Y_{Q'}Z_{T'}^3 - Y_{T'}Z_{Q'}^3) \end{aligned} \tag{8}$$

Siguiendo las recomendaciones de [30], las ecuaciones (5) y (6) se han combinado en un único procedimiento para el cálculo de tangentes y doblado de puntos. El Algoritmo 4 muestra una forma de realizar esto a un costo de siete multiplicaciones, cinco cuadrados y cuatro sumas en $\mathbb{F}_{p^2}$ junto con cuatro multiplicaciones en $\mathbb{F}_p$. La notación definida en la Sección III es utilizada para hacer referencia a coeficientes específicos de la representación polinómica de elementos en la torre de campos. Nótese que las operaciones han sido organizadas para reducir la dependencia de datos y extraer el mayor grado de paralelismo posible. En correspondencia, aquellas multiplicaciones y cuadrados que han sido escritos en una sola línea pueden ser realizados simultáneamente.

Las ecuaciones (7) y (8) también se pudieron combinar en un procedimiento similar para el cálculo de líneas secantes y suma de puntos al costo de 18 multiplicaciones, cuatro cuadrados y siete sumas en $\mathbb{F}_{p^2}$ junto a cuatro multiplicaciones en $\mathbb{F}_p$. Lo importante a destacar en nuestro análisis es que, del mismo modo que se organizaron las operaciones del Algoritmo 4 para minimizar la dependencia de datos, en este caso las 18 multiplicaciones y los cuatro cuadrados en $\mathbb{F}_{p^2}$ así como las cuatro multiplicaciones en $\mathbb{F}_p$ se agruparon en 9, 2 y 2 pares de operaciones independientes respectivamente.

### *B. Aritmética en $\mathbb{F}_{p^{12}}$*

La aritmética en $\mathbb{F}_{p^{12}}$ también se beneficia de la posibilidad de paralelizar las operaciones subyacentes en $\mathbb{F}_{p^2}$. Por ejemplo, una multiplicación en $\mathbb{F}_{p^{12}}$ mediante el método de Karatsuba involucra tres multiplicaciones en $\mathbb{F}_{p^6}$ las que contienen a su vez seis multiplicaciones en $\mathbb{F}_{p^2}$. Estas últimas pueden organizarse en tres grupos de dos multiplicaciones independientes cada uno. De este modo, las 18 multiplicaciones en $\mathbb{F}_{p^2}$ involucradas en una multiplicación en $\mathbb{F}_{p^{12}}$ pueden realizarse mediante nueve pares de multiplicaciones simultáneas en $\mathbb{F}_{p^2}$. Algo similar ocurre con el cuadrado y la multiplicación dispersa en $\mathbb{F}_{p^{12}}$. En el primer caso existe un total de 12 multiplicaciones en $\mathbb{F}_{p^2}$ que se pueden agrupar en seis pares de multiplicaciones independientes. En cuanto a la multiplicación dispersa, 12 de las 13 multiplicaciones en $\mathbb{F}_{p^2}$ pueden agruparse igualmente en seis pares.

Algoritmo 4. Procedimiento para el cálculo combinado de líneas tangentes y doblado de puntos en coordenadas Jacobianas.

**Entradas:** $T' = (X_{T'} : Y_{T'} : Z_{T'})$ en coordenadas Jacobianas y $P = (x_P, y_P)$ en coordenadas afines.
**Salidas:** $l_{\psi(T'),\psi(T')}(P) \in \mathbb{F}_{p^{12}}$ y $R' = [2]T'$.

1: $l = 0$ ▷ $l \in \mathbb{F}_{p^{12}}$
2: $v_1 = 2Y_{T'}$
3: $u_1 = X_{T'}^2$; $t_1 = Z_{T'}^2$
4: $t_2 = t_1 \cdot Z_{T'}$
5: $u_1 = 3u_1$
6: $l_{[100]} = x_P \cdot t_{1_{[0]}}$; $l_{[101]} = x_P \cdot t_{1_{[1]}}$ ▷ multiplicación $\mathbb{F}_p$
7: $l_{[000]} = y_P \cdot t_{2_{[0]}}$; $l_{[001]} = y_P \cdot t_{2_{[1]}}$ ▷ multiplicación $\mathbb{F}_p$
8: $l_{[10]} = -l_{[10]}$
9: $l_{[11]} = -Y_{T'}$
10: $l_{[10]} = l_{[10]} \cdot u_1$; $t_1 = X_{T'} \cdot u_1$
11: $l_{[11]} = l_{[11]} \cdot v_1$; $l_{[00]} = l_{[00]} \cdot v_1$
12: $l_{[11]} = l_{[11]} + t_1$
13: $t_1 = v_1^2$
14: $Z_{R'} = Z_{T'} \cdot v_1$; $t_2 = t_1 \cdot X_{T'}$
15: $v_2 = 2t_2$
16: $X_{R'} = u_1^2$; $t_1 = t_1^2$
17: $t_1 = t_1/2$
18: $X_{R'} = X_{R'} - v_2$
19: $Y_{R'} = t_2 - X_{R'}$
20: $Y_{R'} = Y_{R'} \cdot u_1$
21: $Y_{R'} = Y_{R'} - t_1$
22: Devolver $l$ y $R' = (X_{R'} : Y_{R'} : Z_{R'})$.

### *C. Paralelizando Operaciones en $\mathbb{F}_p$ y $\mathbb{F}_{p^2}$*

Distribuir en núcleos diferentes aquellas operaciones con posibilidad de ser ejecutadas en paralelo puede contribuir a reducir el tiempo de cálculo del emparejamiento Ate-O.

Desde el punto de vista algorítmico, la estrategia de procesamiento doble núcleo implementada en este trabajo es sencilla. Consiste en un protocolo de comando-respuesta a través de una zona de memoria compartida. El núcleo principal (núcleo-0) se encarga de controlar el flujo de operaciones mientras que el núcleo secundario (núcleo-1) actúa como coprocesador aritmético. En la Fig. 2 se ilustra el flujo típico de operaciones. El núcleo-0 toma los operandos destinados al núcleo-1 y los escribe en la memoria compartida, luego escribe el comando adecuado y seguidamente comienza a procesar sus propios datos. Una vez que el núcleo-1 detecta el comando invoca a la función correspondiente para procesar los operandos almacenados en la memoria compartida. Al finalizar, el núcleo-1 activa una bandera de reconocimiento. Por su parte, tras culminar su fase de procesamiento, el núcleo-0 se mantiene esperando por la activación de la bandera de reconocimiento para leer los resultados del núcleo-1 que permanecen en la memoria compartida. En este punto, ambos núcleos se encuentran listos para comenzar la ejecución de una nueva secuencia de operaciones.

La zona de memoria compartida fue implementada mediante la Memoria *On-chip* (OCM, por sus siglas en inglés) disponible en el sistema de procesamiento del Zynq [31]. Para ello se utilizó la configuración recomendada en [32]. En

particular, fue deshabilitado el acceso de cache a la OCM en ambos núcleos en pos de garantizar que los accesos a la memoria compartida se realizaran de forma determinista.

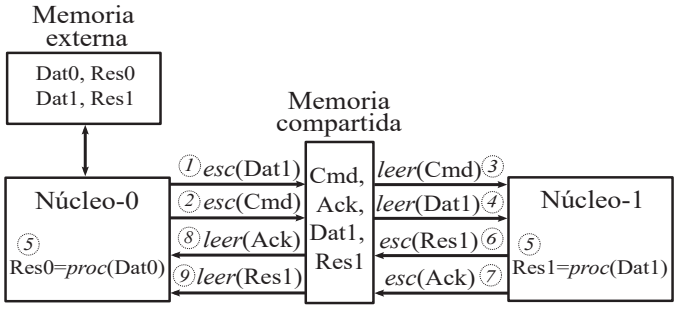



Fig. 2. Esquema de procesamiento doble núcleo.

En base al esquema de procesamiento anterior, dos multiplicaciones independientes en $\mathbb{F}_p$ son ejecutadas simultáneamente cada una en un núcleo diferente. En total, como se muestra en la Tabla IV, se invierten alrededor de $1020$ ciclos de reloj, lo cual es un $15\%$ más rápido que dos multiplicaciones consecutivas ejecutadas en un mismo núcleo. Esto puede ser aprovechado para acelerar las multiplicaciones en $\mathbb{F}_p$ involucradas en los procedimientos de cálculo de líneas y aritmética de puntos. En cuanto a las operaciones en $\mathbb{F}_{p^2}$, nótese que realizar dos multiplicaciones simultáneas es aproximadamente un $26\%$ más rápido que dos multiplicaciones consecutivas ejecutadas en un mismo núcleo. Lo mismo ocurre en el caso del cuadrado en $\mathbb{F}_{p^2}$ donde la mejora de rendimiento promedio se eleva a un $30\%$.

TABLA IV
TIEMPOS PARA LA ARITMÉTICA EN $\mathbb{F}_p$ Y $\mathbb{F}_{p^2}$ UTILIZANDO PROCESAMIENTO DOBLE NÚCLEO

| Operación | Tiempos ($10^3$ ciclos de reloj) | factor |
|---|---|---|
| Multiplicación en $\mathbb{F}_p$ | 1.02 | 0.85 |
| Multiplicación en $\mathbb{F}_{p^2}$ | 2.47 | 0.74 |
| Cuadrado en $\mathbb{F}_{p^2}$ | 1.90 | 0.70 |

## VI. RESULTADOS DE IMPLEMENTACIÓN DEL EMPAREJAMIENTO ATE ÓPTIMO

Esta sección evalúa el impacto de las estrategias de implementación discutidas anteriormente en el tiempo de ejecución del emparejamiento Ate-O. Represéntese el costo computacional de una multiplicación en $\mathbb{F}_p$ como $M$ y los costos de una multiplicación, un cuadrado, una suma, una multiplicación por $\xi$ y una inversión en $\mathbb{F}_{p^2}$ como $\tilde{M}$, $\tilde{C}$, $\tilde{S}$, $\tilde{M}_\xi$ e $\tilde{I}$ respectivamente. Como se mencionaba en la Sección V-A, el costo total de evaluar la tangente junto con el doblado de un punto de curva es de $4M + 7\tilde{M} + 5\tilde{C} + 4\tilde{S}$ mientras que la evaluación de la secante con la suma de puntos tiene un costo de $4M + 18\tilde{M} + 4\tilde{C} + 7\tilde{S}$. Las restantes operaciones del bucle de Miller son las de elevar al cuadrado la variable de Miller y la multiplicación dispersa de la variable de Miller por los resultados de evaluación de las líneas (ya sean secantes o tangentes), ambas operaciones pertenecientes a la extensión de campo $\mathbb{F}_{p^{12}}$ (véase la Sección II-A).

Debido a que la extensión de campo $\mathbb{F}_{p^{12}}$ es una extensión cuadrática sobre $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, la fórmula para elevar al cuadrado números complejos puede ser adaptada para realizar operaciones de cuadrado en $\mathbb{F}_{p^{12}}$ a un costo de $12\tilde{M} + 42\tilde{S} + 6\tilde{M}_\xi$. Por su parte, la multiplicación dispersa tiene un costo de $13\tilde{M} + 25\tilde{S} + 3\tilde{M}_\xi$, el cual es $5\tilde{M} + 35\tilde{S} + 4\tilde{M}_\xi$ menor que el costo de una multiplicación genérica en $\mathbb{F}_{p^{12}}$.

Para estimar el costo total del bucle de Miller se debe tener en cuenta que el parámetro $z$ seleccionado en la Sección II es un número entero de $63$ bits con un peso de Hamming igual a $3$. Debido a ello, la longitud $\ell = 6z + 2$ del bucle de Miller resulta en un número entero de $65$ bits con un peso de Hamming igual a $5$. Esto implica que durante la ejecución del bucle de Miller se realizarán $64$ evaluaciones de tangentes y doblado de puntos, cuatro evaluaciones de líneas secantes y suma de puntos, así como $64$ cuadrados y $68$ multiplicaciones dispersas en $\mathbb{F}_{p^{12}}$. Por tanto, el costo total del bucle de Miller sería aproximadamente de $272M + 2172\tilde{M} + 336\tilde{C} + 4672\tilde{S} + 588\tilde{M}_\xi$.

Tras el bucle de Miller y antes de la exponenciación final se realiza un ajuste de la variable de Miller $f$ en el que se encuentran involucradas otras dos multiplicaciones dispersas en $\mathbb{F}_{p^{12}}$, así como una suma de puntos y el cálculo de dos endomorfismos de Frobenius en $\mathbb{F}_{p^2}$. En total estas operaciones poseen un costo de $8M + 59\tilde{M} + 6\tilde{C} + 60\tilde{S} + 6\tilde{M}_\xi$.

La otra operación que domina el tiempo de cálculo del emparejamiento Ate-O es la parte difícil de la exponenciación final. Lo más costoso de calcular en este caso es la exponenciación por $z$ de elementos en $\mathbb{F}_{p^{12}}$ necesaria para hallar los términos $m^z$, $m^{z^2}$ y $m^{z^3}$ (véase la Sección II-A1). Utilizando el método binario de izquierda a derecha, cada exponenciación por $z$ se calcula mediante $62$ cuadrados y dos multiplicaciones en $\mathbb{F}_{p^{12}}$. Debido a que los elementos resultantes de la parte fácil de la exponenciación final son elementos unitarios [33], el costo de una operación cuadrado en $\mathbb{F}_{p^{12}}$ se reduce a $7\tilde{M} + 4\tilde{C} + 27\tilde{S} + 4\tilde{M}_\xi$. Esto conlleva a que una exponenciación por $z$ tenga un costo de $470\tilde{M} + 248\tilde{C} + 1794\tilde{S} + 262\tilde{M}_\xi$, lo cual, unido al costo de la parte fácil ($62\tilde{M} + 12\tilde{C} + 179\tilde{S} + 25\tilde{M}_\xi + \tilde{I}$), nos permite brindar un estimado total para la exponenciación final de $1772\tilde{M} + 772\tilde{C} + 6449\tilde{S} + 918\tilde{M}_\xi + \tilde{I}$.

Considérese ahora el efecto de los tiempos mostrados en la Tabla IV. Téngase en cuenta que el costo de dos operaciones independientes y por tanto paralelizables puede ser multiplicado por el factor de mejora correspondiente. Por ejemplo, en el caso del procedimiento para la evaluación de la línea tangente y el doblado de puntos, las cuatro multiplicaciones de $\mathbb{F}_p$ involucradas fueron agrupadas en dos pares de dos multiplicaciones independientes cada uno. De la misma forma, seis de las siete multiplicaciones y cuatro de los cinco cuadrados de $\mathbb{F}_{p^2}$ fueron agrupados en tres y dos pares respectivamente. Por tanto, el costo total de dicho procedimiento se reduce a:

$$\begin{aligned} &0.85 \cdot 4M + (\tilde{M} + 0.74 \cdot 6\tilde{M}) + (\tilde{C} + 0.70 \cdot 4\tilde{C}) + 4\tilde{S} \\ &= 3.4M + 5.44\tilde{M} + 3.8\tilde{C} + 4\tilde{S} \end{aligned}$$

Realizando un análisis similar con las métricas restantes es posible obtener nuevos estimados para los costos del bucle

de Miller, el ajuste de $f$ y la exponenciación final, los cuales se muestran en la Tabla V. Nótese que las mejoras obtenidas son equivalentes a $40.8M + 538.24\tilde{M} + 81.6\tilde{C}$ en el caso del bucle de Miller, $1.2M + 15.34\tilde{M} + 2.8\tilde{C}$ para el ajuste de $f$ y $411.76\tilde{M} + 231\tilde{C}$ para la exponenciación final. Esto implica que, en total, la implementación del emparejamiento Ate-O basada en aritmética de campo acelerada con procesamiento doble núcleo se estime que invierta alrededor de $2155 \times 10^3$ ciclos de reloj menos en calcular un emparejamiento que la variante implementada en un solo núcleo.

TABLA V
COSTOS ESTIMADOS PARA LAS OPERACIONES PRINCIPALES DEL EMPAREJAMIENTO ATE-O UTILIZANDO PROCESAMIENTO DOBLE NÚCLEO

| Operación | Costo |
|---|---|
| Bucle de Miller | $231.2M + 1633.76\tilde{M} + 254.4\tilde{C} + 4672\tilde{S} + 588\tilde{M}_\xi$ |
| Ajuste de $f$ | $6.8M + 43.66\tilde{M} + 4.2\tilde{C} + 60\tilde{S} + 6\tilde{M}_\xi$ |
| Exp. final | $1360.24\tilde{M} + 541\tilde{C} + 6449\tilde{S} + 918\tilde{M}_\xi + \tilde{I}$ |

La Tabla VI proporciona una comparación de nuestros resultados experimentales contra los mejores tiempos obtenidos en [10] y [11] para procesadores ARM Cortex-A9. Obsérvese que nuestros tiempos para la exponenciación final son superiores a los reportados en los otros trabajos. Esto se debe a que la variante implementada en el marco de esta investigación (ver Sección II-A1) es menos eficiente que las empleadas en [10] y [11]. No obstante, cabe señalar que es posible mejorar los resultados de [10] y [11] en este aspecto al aplicar el enfoque de procesamiento doble núcleo en una reimplementación de la exponenciación final incorporando, para la operación cuadrado en $\mathbb{F}_{p^{12}}$, el método propuesto en [34] y empleado en [11], o el descrito en [35] y recomendado en [36]. Por el momento, para evaluar la efectividad del enfoque de procesamiento doble núcleo, se hace énfasis en comparar los resultados asociados al bucle de Miller que sí se encuentra implementado con varias de las técnicas que conforman el estado del arte actual.

TABLA VI
TIEMPOS PARA EL CÁLCULO DEL EMPAREJAMIENTO ATE-O SOBRE CURVAS DE BARRETO-NAEHRIG DE 254 BITS Y COMPARACIÓN CON LOS RESULTADOS DE TRABAJOS PREVIOS

| Trabajo | Lenguaje | Tiempos ($10^3$ ciclos de reloj) | | |
|---|---|---|---|---|
| | | Bucle de Miller | Exp. final | $a_{\text{opt}}$ |
| [10][a] | ASM | 7376 | 4510 | 11886 |
| [11][b] | NEON | 5758 | 3794 | 9477 |
| | NEON | 5715 | 5679 | 11404 |
| Este trabajo[c] | ASM (1-núcleo) | 5548 | 5523 | 11112 |
| | ASM (2-núcleo) | 4375 | 4456 | 8849 |

[a]Galaxy Nexus (ARM v7) TI OMAP 4460 Cortex-A9 a 1.2 GHz
[b]Galaxy Note (ARM v7) Exynos 4 Cortex-A9 a 1.4 GHz
[c]Zynq ZC7Z020 (ARM v7) Cortex-A9 a 667 MHz

Como se aprecia en la Tabla VI, nuestra implementación del bucle de Miller basada en NEON así como la desarrollada en [11] poseen un rendimiento similar. La mejora conseguida en este trabajo se debe a las optimizaciones descritas en la Sección IV-B. Por otra parte, obsérvese que el rendimiento de nuestra implementación mononúcleo en lenguaje ensamblador supera a los de ambas implementaciones basadas en NEON, la de [11] y la de este trabajo. Esto se atribuye al empleo de la instrucción umaal de ARM la cual no tiene equivalente en el repertorio NEON. Aun cuando nuestra exponenciación final posee un rendimiento un poco menor, nótese que los mejores tiempos para el emparejamiento Ate-O provienen de nuestra implementación basada en procesamiento doble núcleo. La misma logra calcular un emparejamiento en $2263 \times 10^3$ ciclos de reloj menos que la variante en ensamblador ejecutada en un solo núcleo, una diferencia incluso mayor que la mejora de $2155 \times 10^3$ ciclos de reloj estimada anteriormente.

## VII. CONCLUSIONES

El buen desempeño de los protocolos criptográficos basados en emparejamientos bilineales está estrechamente relacionado a la obtención de un rendimiento adecuado en el cálculo del emparejamiento. Aprovechar características específicas de la plataforma de implementación seleccionada puede contribuir notablemente a dicho propósito. En tal sentido, este trabajo abordó la implementación en procesadores ARM del emparejamiento Ate óptimo sobre curvas de Barreto-Naehrig para un nivel de seguridad de 128 bits. Se exploraron dos variantes de implementación: repertorio de instrucciones NEON y procesamiento doble núcleo. A diferencia de lo esperado, se mostró que en un procesador ARM v7 Cortex-A9 la implementación basada en NEON no superó en cuanto a rendimiento a una implementación mononúcleo optimizada en lenguaje ensamblador. Por tanto, para aprovechar el alto grado de paralelismo presente en varios niveles del cálculo del emparejamiento, fue utilizado el enfoque de procesamiento doble núcleo descrito en la Sección V. Como resultado, el software desarrollado en este trabajo calcula un emparejamiento Ate óptimo en un $25.6\%$ y un $6.6\%$ más rápido que las implementaciones desarrolladas en [10] y [11] respectivamente aun cuando nuestra etapa de exponenciación final es menos eficiente. Considerando solamente los tiempos correspondientes al bucle de Miller, las mejoras respectivas son de $41\%$ y $24\%$. De ahí que exista un margen de mejora adicional a partir de la reimplementación de la exponenciación final utilizando un método similar al empleado en [11] junto al enfoque de procesamiento doble núcleo descrito en este trabajo.

## REFERENCIAS

[1] P. S. L. M Barreto, B. Lynn, and M. Scott, "Constructing Elliptic Curves with Prescribed Embedding Degrees," in *Security in Communication Networks - SCN 2002*, Lecture Notes in Computer Science, vol. 2576, pp. 257-267.
[2] P. S. L. M Barreto and M. Naehrig, "Pairing-Friendly Elliptic Curves of Prime Order," in *Selected Areas in Cryptography - SAC 2005*, Lecture Notes in Computer Science, vol. 3897, pp.319-331.
[3] E. Kachisa, E. Schaefer, and M. Scott, "Constructing Brezing-Weng Pairing-Friendly Elliptic Curves Using Elements in the Cyclotomic Field," in *Pairing-Based Cryptography - Pairing 2008*, Lecture Notes in Computer Science, vol. 5209, pp.126-135.
[4] J. L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, "High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves," in *Pairing-Based Cryptography - Pairing 2010*, Lecture Notes in Computer Science, vol. 6487, pp.21-39.
[5] M. Naehrig, R. Niederhagen, and P. Schwabe, "New software speed records for cryptographic pairings," in *Progress in Cryptology - LATINCRYPT 2010*, Lecture Notes in Computer Science, vol. 6212, pp.109-123.

[6] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, "Faster Explicit Formulas for Computing Pairings over Ordinary Curves," in *Advances in Cryptology - EUROCRYPT 2011*, Lecture Notes in Computer Science, vol. 6632, pp. 48-68.

[7] S. Mitsunari, "A Fast Implementation of the Optimal Ate Pairing over Bn curve on Intel Haswell," *IACR Cryptology ePrint Archive*, 2013:362, 2013.

[8] M. A. Khandaker, Y. Nanjo, L. Ghammam, S. Duquesne, Y. Nogami, and Y. Kodera, "Efficient Optimal Ate Pairing at 128-bit Security Level," in *18th International Conference on Cryptology - IndoCrypt 2017*, Chennai, India, 2017, pp.186-205.

[9] T. Acar, K. Lauter, M. Naehrig, and D. Shumow, "Affine Pairings on ARM," *IACR Cryptology ePrint Archive*, 2011:243, 2011.

[10] G. Grewal, R. Azarderakhsh, P. Longa, S. Hu, and D. Jao, "Efficient Implementation of Bilinear Pairings on ARM processors," *IACR Cryptology ePrint Archive*, 2012:408, 2012.

[11] A. H. Sánchez and F. Rodríguez-Henríquez, "NEON Implementation of an Attribute-Based Encryption Scheme," in *Applied Cryptography and Network Security - ACNS 2013*, Lecture Notes in Computer Science, vol. 7954, pp.322-338.

[12] *NEON Programmer's Guide*. ARM, Cambridge, U.K., 2013.

[13] S. Duquesne, N. E. Mrabet, S. Haloui, F. Rondepierre, "Choosing and generating parameters for low level pairing implementation on BN curves," *IACR Cryptology ePrint Archive*, 2015:1212, 2015.

[14] Y. Nogami, M. Akane, Y. Sakemi, H. Kato, and Y. Morikawa, "Integer Variable $\chi$-Based Ate Pairing," in *Pairing-Based Cryptography - Pairing 2008*, Lecture Notes in Computer Science, vol. 5209, pp.178-191.

[15] M. Naehrig, "Constructive and Computational Aspects of Cryptographic Pairings," Ph.D. dissertation, Eindhoven University of Technology, Netherlands, 2009.

[16] C. Costello, "Fast Formulas for Computing Cryptographic Pairings," Ph.D. dissertation, Queensland University of Technology, Brisbane, Australia, 2012.

[17] F. Vercauteren, "Optimal Pairings," *IEEE Trans. Information Theory*, vol. 56, no. 1, pp.455-461, 2009.

[18] V. S. Miller, "Short Programs for functions on Curves," unpublished

[19] M. Scott, N. Benger, M. Charlemagne, L. J. Domínguez-Pérez, and E. J. Kachisa, "On the final exponentiation for calculating pairings on ordinary elliptic curves," in *Pairing-Based Cryptography - Pairing 2009*, Lecture Notes in Computer Science, vol. 5671, pp.78-88.

[20] J. Olivos, "On Vectorial Addition Chains," *Journal of Algorithms*, vol. 2, no. 1, pp.13-21, 1981.

[21] N. Koblitz and A. Menezes, "Pairing-based cryptography at high security levels," in *IMA Int. Conf.*, Lecture Notes in Computer Science, vol. 3796, pp.13-36.

[22] N. Benger and M. Scott, "Constructing Tower Extensions of Finite Fields for Implementation of Pairing-Based Cryptography," in *Arithmetic of Finite Fields - WAIFI 2010*, Istambul, Turkey, 2010, Lecture Notes in Computer Science, vol. 6087, pp.180-195.

[23] ZedBoard Hardware User's Guide, Version 2.2, Jan. 27, 2014. [Online]. Available: http://www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[24] A. Weimerskirch, and C. Paar,"Generalizations of the Karatsuba Algorithm for Efficient Implementations," *IACR Cryptology ePrint Archive*, 2006:224, 2006.

[25] C. H. Lim and H. S. Hwang, "Fast Implementation of Elliptic Curve Arithmetic in $GF(p^n)$," in *Third International Workshop on Practice and Theory in Public Key Cryptography - PKC 2000*, Lecture Notes in Computer Science, vol. 1751, pp.405-421.

[26] A. J. Devegili1, C. Ó hÉigeartaigh, M. Scott, and R. Dahab, "Multiplication and Squaring on Pairing-Friendly Fields," *IACR Cryptology ePrint Archive*, 2006:471, 2006.

[27] A. J. Menezes, S. A. Vanstone, and P. C. van Oorschot, "Efficient Algorithms," in *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press Inc., 1996, ch. 14, sec. 2, pp. 592-599.

[28] P. L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp.519-521, Apr.,1985.

[29] Ç. K. Koç, T. Acar, and B. S. Kaliski, "Analyzing and Comparing Montgomery Multiplication Algorithms," *IEEE Micro*, vol. 16, no. 3, pp.26-33, 1996.

[30] C. Costello, T. Lange, and M. Naehrig. "Faster Pairing Computations on Curves with High-Degree Twists," in *In Public Key Cryptography - PKC 2010*, Lecture Notes in Computer Science, vol. 6056, pp.224-242.

[31] *Zynq-7000 SoC Technical Reference Manual*, Xilinx, San Jose, CA, USA, 2018, pp.742-752.

[32] J. McDougall, "Simple AMP: Bare-Metal System Running on Both Cortex-A9 Processors," Xilinx, San Jose, CA, USA, XAPP1079 (v1.0.1), Jan. 24, 2014.

[33] M. Scott and P. S. L. M Barreto, "Compressed Pairings," in *Advances in Cryptology - CRYPTO 2004*, Lecture Notes in Computer Science, vol. 3152, pp.140-156.

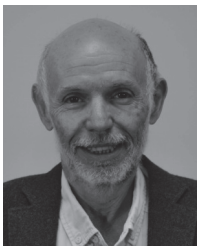
[34] K. Karabina, "Squaring in Cyclotomic Subgroups," *Mathematics of Computation*, vol. 82, no. 281, pp.555-579, 2013.

[35] R. Granger and M. Scott, "Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions," in *Public Key Cryptography - PKC 2010*, Lecture Notes in Computer Science, vol. 6056, pp.209-223.

[36] M. Scott, "Pairing Implementation Revisited," *IACR Cryptology ePrint Archive*, 2019:077, 2019.

**Raudel Cuiman Márquez,** Ingeniero en Automática en 2009, Máster en Sistemas Digitales en 2014 por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Auxiliar del Departamento de Automática y Computación de la CUJAE, Calle 114 Nº 11901 e/ Ciclo vía y Rotonda, CUJAE, Marianao, La Habana, Cuba. Teléf.: +53 7 266 3498. Sus intereses de investigación están relacionados con la implementación eficiente de algoritmos criptográficos sobre sistemas empotrados. Actualmente realiza su investigación doctoral sobre la implementación hardware de protocolos criptográficos basados en emparejamientos bilineales. Email: raudel@automatica.cujae.edu.cu

**Alejandro J. Cabrera Sarmiento,** Ingeniero Electricista, Doctor en Ciencias Técnicas por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Titular del Departamento de Automática y Computación de la CUJAE, Calle 114 Nº 11901 e/ Ciclo vía y Rotonda, CUJAE, Marianao, La Habana, Cuba. Teléf.: +53 7 266 3301. Sus líneas de investigación principales están relacionadas con el desarrollo de sistemas empotrados basados en FPGA y la aceleración de algoritmos sobre hardware reconfigurable, con aplicaciones en procesado de imágenes, control inteligente y criptografía. Email: alex@automatica.cujae.edu.cu



**Santiago Sánchez-Solano,** Doctor en Ciencias Físicas por la Universidad de Sevilla en 1990, Investigador Científico del CSIC adscrito al Instituto de Microelectrónica de Sevilla, IMSE-CNM, (CSIC/Universidad de Sevilla), Sevilla, España. Sus líneas de investigación se centran en el desarrollo de sistemas empotrados con componentes hardware-software sobre FPGA y la realización microelectrónica de sistemas neurodifusos, así como sus aplicaciones en robótica, procesamiento de imágenes, seguridad y redes de sensores inteligentes. Email: santiago@imse-cnm.csic.es