Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Diseño e implementación eficiente del emparejamiento etaT en dispositivos móviles y arquitecturas multinúcleo

Tesis que presenta

Luis Martínez Ramos

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Francisco Rodríguez Henríquez

México, D.F. **Diciembre 2008**

El éxito comienza con la voluntad

Si piensas que estás vencido, lo estás.
Si piensas que no te atreves, no lo harás.
Si piensas que te gustaría ganar pero no puedes, no lo lograrás.

Si piensas que perderás, ya has perdido,
porque en el mundo encontrarás
que el éxito comienza con la voluntad del hombre.

Todo está en el estado mental.
Porque muchas carreras se han perdido antes de haberse corrido,
y muchos cobardes han fracasado,
antes de haber su trabajo empezado.

Piensa en grande y tus hechos crecerán.
Piensa en pequeño y quedarás atrás.
Piensa que puedes y podrás.
Todo está en el estado mental.

Si piensas que estás aventajado, lo estás.
Tienes que pensar bien para elevarte.
Tienes que pensar seguro de ti mismo,
antes de intentar ganar un premio.

La batalla de la vida no siempre la gana
el hombre más fuerte, o el más ligero,
porque tarde o temprano, el hombre que gana,
es aquél que cree poder hacerlo.

Rudyard Kipling (1865 – 1936)

Agradecimientos

Gracias a mis Padres y hermanos quienes con su amor, apoyo y atenciones han hecho de mi una mejor persona.

A mi director de tesis el Dr. Francisco Rodríguez Henríquez le agradezco la confianza y el apoyo brindado durante el desarrollo de esta tesis.

Gracias a los Doctores Arturo Díaz Pérez y Guillermo Morales Luna por sus valiosos comentarios en la revisión y corrección para el enriquecimiento de este documento.

Un especial agradecimiento a la Sra. Sofía Reza por su apoyo y cariño incondicional, pero sobretodo, por su infinita paciencia.

A mis compañeros de maestría les deseo éxito y les doy gracias por la convivencia a lo largo de nuestra estancia en esta institución.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico que me fue otorgado a partir de su programa de becas y del proyecto número 60240 para llevar a buen término mis estudios de maestría.

Al Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV-IPN) por permitirme ser parte de esta institución y por el apoyo económico otorgado en la finalización de mis estudios de maestría.

Resumen

Los emparejamientos bilineales se han utilizado recientemente para la construcción de esquemas criptográficos con nuevas y novedosas propiedades. Desde su introducción en una aplicación criptográfica constructiva en el año 2000 por Joux, un número creciente de protocolos basados en los emparejamientos de Weil y Tate ha aparecido en la literatura, el ejemplo más significativo es el esquema de Criptografía Basada en la Identidad propuesto por Boneh y Franklin. Sin embargo, el cálculo de emparejamientos es computacionalmente muy costoso para ser utilizado en cualquier criptosistema basado en emparejamientos, por esta razón, numerosas aportaciones han aparecido, donde varias mejoras son propuestas para el cálculo de emparejamientos. Barreto et al. propusieron una variante del emparejamiento de Tate llamada η_T que hasta hoy es considerado el emparejamiento más eficiente. En esta tesis, presentamos una implementación eficiente en software para el cálculo del emparejamiento η_T sobre campos finitos binarios y ternarios. La implementación presentada en este trabajo logra una eficiencia competitiva con las implementaciones reportadas en la literatura abierta. Nosotros realizamos esta implementación sobre dos arquitecturas diferentes, la primera de ellas es un dispositivo móvil con recursos limitados, la segunda es una arquitectura multinúcleo que nos permite calcular el emparejamiento η_T en paralelo. También se presenta una aplicación criptográfica basada en emparejamientos que hace uso de otras primitivas, como el cifrador por bloques AES, la función picadillo SHA-256 y funciones picadillo llamadas map-to-points.

Abstract

Bilinear pairings have been recently used to construct cryptographic schemes with new and novel properties. Since its introduction in a constructive cryptographic application in 2000 by Joux, an increasing number of protocols based on Weil or Tate pairings have appeared in the literature, the most significant example is the Identity Based Encryption scheme proposed by Boneh and Franklin. However, the computing pairings is computationally very expensive to be used in any pairing-based cryptosystem, for this reason, many contributions have appeared, where several improvements are proposed to compute pairings. Barreto et al. proposed a variant of the Tate pairing called η_T which until today is considered the most efficient pairing. In this thesis, we present an efficient software implementation to compute the η_T pairing over binary and ternary finite fields. The implementation presented in this work achieves a competitive performance with implementations reported in the open literature. We made the implementation on two different architectures, the first one of them it is a mobile device with limited resources, the second one is a multicore architecture that allows us to compute the η_T pairing in parallel. Also one presents a cryptographic application based on pairings which it uses of others cryptographic primitives as the AES block cipher, SHA-256 and map-to-points hash functions.

ÍNDICE GENERAL

Índice de figuras VII

Índice de tablas IX

Índice de algoritmos XI

Introducción 1

1. Conceptos básicos 3

1.1. Seguridad y Criptografía . . . 3
1.1.1. Criptografía de clave privada . . . 4
1.1.2. Criptografía de clave pública . . . 5
1.2. Aritmética modular . . . 7
1.2.1. Grupo . . . 7
1.2.2. Anillo . . . 7
1.2.3. Campo . . . 8
1.2.4. Campos finitos . . . 8
1.2.4.1. Campos finitos de característica dos . . . 9
1.2.4.2. Campos finitos de característica tres . . . 10
1.3. Curvas elípticas sobre campos finitos . . . 11
1.3.1. Orden de una curva elíptica . . . 13
1.3.2. Orden de un punto . . . 13
1.3.3. Grado de seguridad (Embedding degree) . . . 13
1.3.4. Operaciones en curvas elípticas . . . 14
1.3.4.1. Suma . . . 14
1.3.4.2. Doblado . . . 15
1.3.4.3. Multiplicación escalar . . . 17
1.3.5. Problema del logaritmo discreto en curvas elípticas . . . 17

1.3.6. Curvas elípticas supersingulares . . . 18
1.3.6.1. Curvas elípticas supersingulares en $\mathbb{F}_{2^m}$ . . . 18
1.3.6.2. Curvas elípticas supersingulares en $\mathbb{F}_{3^m}$ . . . 19
1.4. Funciones picadillo . . . 20
1.4.1. Función picadillo map-to-point . . . 20

2. Emparejamientos bilineales **23**
2.1. Emparejamiento bilineal . . . 23
2.1.1. Definición de emparejamiento bilineal . . . 24
2.1.2. Propiedades . . . 24
2.2. Algoritmos de emparejamiento . . . 25
2.2.1. Emparejamiento de Tate modificado . . . 27
2.3. Aplicaciones de los emparejamientos . . . 29
2.3.1. Acuerdo de clave entre tres entidades en una sola ronda . . . 29
2.3.2. Criptografía Basada en la Identidad (IBE) . . . 30
2.3.2.1. Criptosistema basado en la identidad . . . 31
2.3.2.2. Firma digital basada en la identidad . . . 31
2.3.2.3. Seguridad del esquema IBE . . . 34
2.3.3. Firmas a ciegas . . . 34
2.4. Estado del arte . . . 35

3. Aritmética del emparejamiento η_T **41**
3.1. Infraestructura . . . 42
3.1.1. Infraestructura para la implementación en PDA . . . 42
3.1.2. Infraestructura para la implementación en computadora multinúcleo . . . 43
3.2. Consideraciones para característica dos . . . 43
3.2.1. Restricciones en PDA . . . 44
3.2.2. Restricciones en multinúcleo . . . 44
3.3. Implementación en característica dos . . . 44
3.3.1. Operaciones en $\mathbb{F}_{2^m}$ . . . 44
3.3.1.1. Suma . . . 45
3.3.1.2. Cuadrado . . . 45
3.3.1.3. Multiplicación . . . 46
3.3.1.4. Reducción modular . . . 47
3.3.1.5. Raíz cuadrada . . . 48
3.3.1.6. Inversión . . . 50
3.3.2. Operaciones en $\mathbb{F}_{2^{2m}}$ . . . 51
3.3.2.1. Suma . . . 52
3.3.2.2. Cuadrado . . . 52

3.3.2.3. Multiplicación . . . 53
3.3.2.4. Inversión . . . 53
3.3.3. Operaciones en $\mathbb{F}_{2^{4m}}$ . . . 54
3.3.3.1. Suma . . . 55
3.3.3.2. Cuadrado . . . 55
3.3.3.3. Multiplicación . . . 56
3.3.4. Resultados de la aritmética en $\mathbb{F}_{2^m}$ . . . 58
3.4. Consideraciones para característica tres . . . 59
3.4.1. Restricciones en PDA . . . 59
3.4.2. Restricciones en multinúcleo . . . 59
3.5. Implementación en característica tres . . . 59
3.5.1. Operaciones en $\mathbb{F}_{3^m}$ . . . 60
3.5.1.1. Suma y resta . . . 60
3.5.1.2. Cubo . . . 61
3.5.1.3. Multiplicación . . . 62
3.5.1.4. Reducción modular . . . 63
3.5.1.5. Raíz cúbica . . . 63
3.5.1.6. Inversión . . . 65
3.5.2. Operaciones en $\mathbb{F}_{3^{3m}}$ . . . 66
3.5.2.1. Suma y resta . . . 66
3.5.2.2. Multiplicación . . . 67
3.5.2.3. Inversión . . . 67
3.5.2.4. Cuadrado . . . 68
3.5.3. Operaciones en $\mathbb{F}_{3^{6m}}$ . . . 69
3.5.3.1. Suma y resta . . . 70
3.5.3.2. Cubo . . . 70
3.5.3.3. Multiplicación . . . 71
3.5.4. Resultados de la aritmética en $\mathbb{F}_{3^m}$ . . . 72

4. Implementación del emparejamiento η_T 77

4.1. ¿Por qué paralelizar el emparejamiento η_T? . . . 78
4.1.1. ¿Qué puede ser paralelizable en el emparejamiento η_T? . . . 78
4.2. Implementación en característica dos . . . 80
4.2.1. Emparejamiento η_T en característica dos . . . 80
4.2.2. Primera multiplicación en característica dos . . . 85
4.2.3. Exponenciación final en característica dos . . . 86
4.3. Resultados del emparejamiento η_T en característica dos . . . 88
4.4. Implementación en característica tres . . . 90
4.4.1. Emparejamiento η_T en característica tres . . . 90
4.4.2. Primera multiplicación en característica tres . . . 94

4.4.3. Exponenciación final en característica tres . . . 95
4.5. Resultados del emparejamiento η_T en característica tres . . . 98

5. Una aplicación **101**
5.1. Descripción . . . 101
5.1.1. Cifrado y descifrado . . . 105
5.1.2. Firma y verificación . . . 106
5.2. Análisis de tiempo . . . 108

6. Conclusiones **111**
6.1. Resumen de resultados . . . 111
6.2. Conclusiones . . . 112
6.3. Trabajo a futuro . . . 112

A. Cifrador AES **115**
A.1. Expansión de clave . . . 115
A.2. Cifrado . . . 116
A.3. Descifrado . . . 118

B. Tools **121**
B.1. Intel® C++ Compiler 10.1 . . . 121
B.1.1. Using the Compiler . . . 121
B.2. Intel® VTune Performance Analyzer . . . 129
B.2.1. Sampling Mode . . . 129
B.2.2. Call Graph Profiling . . . 131
B.2.3. Counter Monitor . . . 131
B.3. Intel® Thread Checker . . . 133
B.3.1. Severity Distribution . . . 134
B.3.2. Diagnostics List . . . 134
B.3.3. Viewing the Source . . . 135
B.4. Intel® Thread Profiler . . . 136
B.4.1. Profile view and Timeline view . . . 136
B.5. Intel® Threading Building Blocks . . . 139
B.5.1. Advantages . . . 140
B.5.2. Using TBB . . . 140
B.5.3. Initializing and Terminating the Library . . . 140
B.5.3.1. Scheduling Algorithm . . . 142
B.5.3.2. TBB in the Project . . . 142

Bibliografía **147**

ÍNDICE DE FIGURAS

1.1. Esquema de cifrado de clave privada. 4
1.2. Esquema de cifrado de clave pública. 6
1.3. Ejemplos de Curvas Elípticas: a) $y^2 = x^3 + ax + b$, b) $y^2 = x^3 + b$. . . . 12
1.4. Suma de dos puntos, $R = P + Q$. 15
1.5. Suma del punto, $P + (-P) = \mathcal{O}$. 15
1.6. Doblado del punto, $2P = P + P = R$. 16
1.7. Doblado del punto, $2P = \mathcal{O}$ si $P(x, 0)$. 16

2.1. Emparejamiento η_T. 24
2.2. Diffie-Hellman entre tres entidades en una sola ronda. 30
2.3. Esquema de cifrado basado en la identidad 32
2.4. Esquema de firmas basado en la identidad 32
2.5. Arquitectura de aplicaciones utilizando el emparejamiento η_T. 34

3.1. Suma en las arquitecturas PDA y multinúcleo 45
3.2. Reducción modular para un trinomio $x^m + x^n + 1$. 48
3.3. Raíz cuadrada: $c_0 + \sqrt{x}c_1$. 50

4.1. Paralelización del emparejamiento η_T. 84
4.2. Relación tiempo-número de procesadores del emparejamiento η_T en característica dos (multinúcleo). 89
4.3. Relación tiempo-número de procesadores del emparejamiento η_T en característica tres (multinúcleo). 100

5.1. Cifrado y descifrado utilizando IBE. 105
5.2. Firma y verificación utilizando IBE. 107

A.1. AES: ShiftRows 117

B.1. Single-file compilation. 123
B.2. VTune: Number of calls by function 129
B.3. VTune: Viewing the source code. 130
B.4. VTune: Call graph 131
B.5. VTune: Counter monitor 132
B.6. VTune: Threading view 132
B.7. Thread Checker: The calculation of the η_T pairing with 8 threads. . . . 133
B.8. Severity distribution. 134
B.9. Diagnostics list 134
B.10. Source and Assembler code. 135
B.11. Thread profiler: Serial implementation of the η_T pairing. 136
B.12. Thread profiler: Parallel implementation of the η_T pairing (2 cores). . . 137
B.13. Parallel implementation inefficient for 8 cores (unbalanced load). 138
B.14. Parallel implementation efficient for 8 cores. 139

ÍNDICE DE TABLAS

1.1. Suma y resta en característica dos. 9
1.2. Multiplicación en característica dos. 10
1.3. Suma en característica tres. 10
1.4. Resta en característica tres. 10
1.5. Multiplicación en característica tres. 11
1.6. Operaciones en curvas elípticas sobre $\mathbb{F}_q$ y $\mathbb{F}_{2^m}$ 17

2.1. Tamaños de clave recomendados. 28
2.2. Tiempos de cálculo del emparejamiento de Tate en diferentes campos finitos (mseg), obtenidos en [47]. 35
2.3. Comparación de tiempos con otros criptosistemas (mseg), obtenidos en [47]. 36
2.4. Tiempo (μs) con un procesador Pentium 4 a 2.4 GHz, resultados en [4]. ^{d} Cadena de adición con 12 multiplicaciones, ^{g} Cadena de adición con 14 multiplicaciones. 36
2.5. Costo del emparejamiento de Tate en MICAz, resultados en [57]. 37
2.6. Tiempo en μs de las operaciones en el campo, obtenidos en [42]. 37
2.7. Tiempo en μs de las operaciones en el campo, resultados en [42]. 38
2.8. Tiempo en mili-segundos del cálculo del emparejamiento de Tate, resultados en [42]. 38
2.9. Tiempo en mili-segundos del cálculo del emparejamiento de Tate por 2 algoritmos, resultados en [63]. 38
2.10. Emparejamiento η_T en característica tres 39
2.11. Número de ciclos para el cálculo del emparejamientos η_T. 39

3.1. Tiempos de la aritmética sobre $\mathbb{F}_{2^{233}}$. 58
3.2. Tiempos de la aritmética sobre $\mathbb{F}_{2^{503}}$. 58
3.3. Ejemplo: tabla Comb en característica tres con $w = 2$ para $a \in \mathbb{F}_{3^m}$. . . 62

3.4. Costo para construir la tabla Comb con $w = 4$, $a \in \mathbb{F}_{3^m}$. 63
3.5. Tiempos de la aritmética sobre $\mathbb{F}_{3^{97}}$. 72
3.6. Tiempos de la aritmética sobre $\mathbb{F}_{3^{503}}$. 72

4.1. Tiempo del emparejamiento η_T en característica dos (PDA). 88
4.2. Tiempos del emparejamiento η_T en característica dos (IA-32). 88
4.3. Tiempos del emparejamiento η_T en característica dos (x64). 88
4.4. Tiempo del emparejamiento η_T en característica tres (PDA). 99
4.5. Tiempos del emparejamiento η_T en característica tres (IA-32) 99
4.6. Tiempos del emparejamiento η_T en característica tres (x64) 99

5.1. Tiempos obtenidos de la aplicación en la PDA. 108

B.1. Data types: MMX and SSE. 126
B.2. η_T Pairing in characteristic two. 144
B.3. η_T Pairing in characteristic three. 145

ÍNDICE DE ALGORITMOS

1.1. Map-to-Point de Boneh et. al. [16] 21
1.2. Map-to-Point de Barreto y Kim [7] 21
2.1. Algoritmo de Miller 26
3.1. Multiplicación polinomial en $\mathbb{F}_{2^m}$ 47
3.2. Raíz cuadrada en $\mathbb{F}_{2^m}$ 50
3.3. Inversión en $\mathbb{F}_{2^m}$ 51
3.4. Suma en $\mathbb{F}_{2^{2m}}$ 52
3.5. Cuadrado en $\mathbb{F}_{2^{2m}}$ 53
3.6. Multiplicación en $\mathbb{F}_{2^{2m}}$ 53
3.7. Inversión en $\mathbb{F}_{2^{2m}}$ 54
3.8. Suma en $\mathbb{F}_{2^{4m}}$ 55
3.9. Cuadrado en $\mathbb{F}_{2^{4m}}$ 56
3.10. Multiplicación en $\mathbb{F}_{2^{4m}}$ 57
3.11. Multiplicación en $\mathbb{F}_{2^{4m}}$ Caso 1 57
3.12. Multiplicación en $\mathbb{F}_{2^{4m}}$ Caso 2 58
3.13. Raíz cúbica en $\mathbb{F}_{3^m}$ 64
3.14. Inversión en $\mathbb{F}_{3^m}$ - Algoritmo Ternario de Euclides 65
3.15. Suma/Resta en $\mathbb{F}_{3^{3m}}$ 66
3.16. Multiplicación en $\mathbb{F}_{3^{3m}}$ 67
3.17. Inversión en $\mathbb{F}_{3^{3m}}$ 68
3.18. Cuadrado en $\mathbb{F}_{3^{3m}}$ 69
3.19. Suma/Resta en $\mathbb{F}_{3^{3m}}$ 70
3.20. Cubo en $\mathbb{F}_{3^{6m}}$ 71
3.21. Multiplicación en $\mathbb{F}_{3^{6m}}$ 73
3.22. Multiplicación en $\mathbb{F}_{3^{6m}}$ Caso 1 74
3.23. Multiplicación en $\mathbb{F}_{3^{6m}}$ Caso 2 75
4.1. Emparejamiento η_T, sin raíces cuadradas (PDA) 81
4.2. Emparejamiento η_T, con raíces cuadradas (multinúcleo) 82

4.3. Emparejamiento η_T, con raíces cuadradas (versión paralela). 83
4.4. Primera multiplicación en característica dos [12]. 85
4.5. Exponenciación final en característica dos [12]. 86
4.6. Cálculo de U^{2^m+1} en $\mathbb{F}_{2^{4m}}$ [12]. 87
4.7. Emparejamiento η_T unrolled loop, sin raíz cúbica (característica tres) [11]. 91
4.8. Cubo: $(-t^2 + u\sigma - t\rho - \rho^2)^3$ 92
4.9. Emparejamiento η_T reversed loop, con raíz cúbica (característica tres) [11]. 92
4.10. Emparejamiento η_T reversed loop, con raíz cúbica (característica tres, paralelizado en 2 tareas). 93
4.11. Primera multiplicación en característica tres [11]. 95
4.12. Exponenciación final en característica tres [11]. 96
4.13. Cálculo de $U^{3^{3m}-1}$ en $\mathbb{F}^*_{3^{6m}}$ 97
4.14. Cálculo de U^{3^m+1} en $\mathbb{F}^*_{3^{6m}}$ 98
5.1. Cifrador utilizando IBE 106
5.2. Descifrador utilizando IBE 106
5.3. Firma utilizando IBE 107
5.4. Verificación utilizando IBE 108
A.1. AES: Expansión de clave 116
A.2. AES: Cifrador 117
A.3. AES: Descifrador 118

INTRODUCCIÓN

Con el avance de la tecnología se ha podido observar un incremento tanto en el uso de dispositivos móviles como de arquitecturas multinúcleo. Los dispositivos móviles ofrecen una gran flexibilidad en su uso y adquisición, sin embargo, cuenta con recursos limitados que restringen su poder de procesamiento. Al contrario, las arquitecturas multinúcleo ofrecen un gran poder de procesamiento y desempeño, no obstante, el desarrollo de aplicaciones orientadas a este tipo de arquitecturas es más complejo.

En ambas arquitecturas, la comunicación se realiza a través de un canal inseguro. De tal manera que es de vital importancia contar con herramientas que nos permitan brindar la seguridad requerida. Actualmente existen protocolos criptográficos que nos ayudan a cubrir esta necesidad.

La criptografía basada en emparejamientos bilineales es un área de investigación muy activa que ha permitido el diseño de nuevas y elegantes soluciones a problemas criptográficos. Un criptosistema que está compuesto de emparejamientos se fundamenta principalmente en el cálculo de operaciones aritméticas básicas sobre campos finitos.

En esta tesis nuestro principal objetivo es implementar una herramienta criptográfica que pueda ser utilizada eficientemente en las arquitecturas anteriormente mencionadas. Para lograr nuestro objetivo desarrollamos una biblioteca en software que utiliza un emparejamiento bilineal como función principal.

En la literatura abierta se pueden encontrar diferentes algoritmos para realizar el cálculo de un emparejamiento, en particular, utilizamos el emparejamiento η_T, en los campos finitos binarios y ternarios.

En esta tesis presentamos la implementación eficiente para el cálculo del emparejamiento η_T sobre un dispositivo móvil (PDA) así como para una arquitectura multinúcleo.

Construimos además, una aplicación que realiza las funciones criptográficas básicas (cifrado/descrifrado, firma/verificación) utilizando nuestra biblioteca basada en emparejamientos.

El resto de la tesis se encuentra organizada de la siguiente manera: en el capítulo 1 se describen los conceptos básicos de la criptografía y se presenta una descripción general de la teoría de grupos. En el capítulo 2 se presenta una introducción a los emparejamientos bilineales, sus aplicaciones y algunos de los trabajos reportados en la literatura abierta sobre este tema. En el capítulo 3, explicamos la construcción y aspectos de implementación de las operaciones aritméticas sobre campos finitos binarios y ternarios, además es incluida la implementación de las torres de campos necesarias para el cálculo del emparejamiento η_T en cada característica. En el capítulo 4 detallamos la implementación del emparejamiento η_T y reportamos los resultados obtenidos. En el capítulo 5 se describen el diseño, la arquitectura y características de una aplicación criptográfica utilizando la biblioteca desarrollada. Finalmente en el capítulo 6 presentamos las conclusiones de nuestro trabajo.

1

CONCEPTOS BÁSICOS

Para todo problema complejo hay una solución clara, sencilla y errónea.

Henry Louis Mencken (1880 – 1956)

1.1. Seguridad y Criptografía

La criptografía es el estudio de técnicas matemáticas relacionadas con aspectos de seguridad de la información, como son, la confidencialidad, la integridad de datos, la autenticación de entidad y la autenticación del origen de datos [53], entre las disciplinas que engloba cabe destacar la Teoría de la Información, la Complejidad Algorítmica y la Teoría de Números.

El objetivo básico de la criptografía es permitir a dos entidades comunicarse de forma segura a través de un canal de comunicación, para lograrlo la criptografía provee de servicios o funciones de seguridad [53]. A continuación se listan sólo algunos de estos servicios:

Confidencialidad: este servicio garantiza que la información sólo podrá ser consultada y/o manipulada por usuarios o entidades autorizadas.

Integridad de datos: garantiza que la información original no ha sido alterada ni accidentalmente ni de manera intencional por entidades no autorizadas.

Autenticación o identificación: permite certificar la identidad de una entidad.

Firmas: mecanismo para ligar la información con una entidad.

Sellos de tiempo: registro de la fecha de creación o de la existencia de la información.

No repudio: prevención de la negación de una entidad sobre sus acciones realizadas.

Las técnicas criptográficas se suelen dividir en dos categorías: criptografía de clave privada y criptografía de clave pública, la primera de ellas utiliza una única clave que es usada para cifrado y descifrado, la segunda utiliza un par de claves complementarias entre sí denominadas clave pública y clave privada respectivamente. A continuación se describen brevemente las ventajas y desventajas de estos dos esquemas criptográficos.

1.1.1. Criptografía de clave privada

En el esquema de clave privada se supone que las partes involucradas son las únicas entidades que poseen la clave secreta (emisor y receptor), la fortaleza de los algoritmos de clave privada radica principalmente en que la clave sólo es conocida por las entidades que desean transmitir información de manera confidencial. Cabe destacar que este tipo de criptografía es simple y eficiente de implementar, pero uno de los principales problemas con este esquema es la distribución de la clave [53].

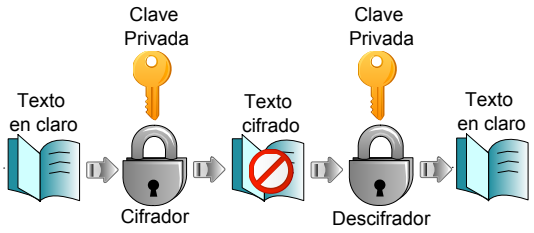


Figura 1.1: Esquema de cifrado de clave privada.

Ventajas

• Debido a la gran eficiencia de estos esquemas criptográficos, pueden procesar grandes cantidades de información.

• Las claves usadas son más pequeñas con respecto a las utilizadas en los cifradores de clave pública.

• Estos cifradores pueden emplearse como primitivas para la construcción de varios mecanismos criptográficos incluyendo generadores de números pseudoaleatorios, funciones picadillo y esquemas eficientes de firma digital.

• Pueden ser combinados con otros bloques criptográficos en diversos modos de operación que permitan ofrecer conceptos de seguridad más amplios y robustos.

Desventajas

• En un esquema de comunicación entre dos entidades, la clave debe de permanecer secreta, lo cual puede representar un costo computacional muy alto.

• En una red grande, se necesita administrar y manejar muchas claves. Por lo tanto, es necesario tener un buen esquema de administración de claves. Se necesitan $\frac{n(n-1)}{2}$ claves secretas en un sistema con n usuarios.

• En una comunicación entre dos entidades: A y B, una práctica recomendada es cambiar frecuentemente la clave usada, donde lo ideal es que exista una clave para cada sesión que sostengan las entidades A y B. Este procedimiento de cambio de llaves se vuelve muy complicado en el paradigma de clave privada.

• Los mecanismos de firma digital que se presentan en el cifrado de clave privada requieren que las claves sean grandes para la primitiva pública de verificación o incluso el uso de una tercera entidad en la cual confíen las partes involucradas en el esquema de firma digital.

Los esquemas de criptografía de clave privada se suelen dividir en cifradores por flujo y cifradores por bloque. Entre los cifradores por flujo más utilizados encontramos A5, RC4, WEP, y actualmente en un proyecto llamado eSTREAM organizado por ECRYPT anunció una iniciativa para presentar algoritmos de cifradores por bloque, abierta a toda la comunidad criptográfica, con la finalidad de obtener cifradores por bloque nuevos y seguros. Algunos de los trabajos encontrados en eSTREAM son HC-128, Rabbit y Trivium entre otros más.

Los cifradores por bloque más conocidos son DES, Hill y AES. El cifrador DES (*Data Encryption Standar*) era el cifrador por bloques estándar utilizado, sin embargo, en este cifrador se encontraron debilidades, colocando al cifrador AES (*Advanced Encryption Standard*) como el nuevo estándar. Por otro lado, el cifrador Hill es uno de los primeros cifradores por bloque conocidos, pero a este cifrador continuamente se le han realizado ataques exponiendo sus debilidades. En el 2006 en [2] presentaron una nueva modificación del cifrador Hill, donde se aseguraba que esta modificación cubría las deventajas de sus versiones anteriores. Como parte del trabajo realizado en esta tesis fue el criptoanálisis de esta nueva versión de Hill, como resultado se escribió un artículo de comentario donde se presentan diversos ataques realizados, para más detalles ver [67].

1.1.2. Criptografía de clave pública

En la Figura 1.2 se muestra el esquema de criptografía de clave pública en donde cada una de las entidades posee un par de claves, una denominada clave pública y otra

llamada clave privada. La clave pública, como su nombre lo sugiere, se puede distribuir a cualquier entidad que desee tenerla, mientras que la clave privada sólo será conocida por el dueño de la misma, ambas claves están relacionadas matemáticamente y deberá ser computacionalmente imposible conocer la clave privada a partir de la pública. La clave pública es utilizada para cifrar la información y la clave privada para descifrar la misma, o bien, para el esquema de firma digital, la clave privada se usa para firmar y la clave pública para la operación de verificación [53].

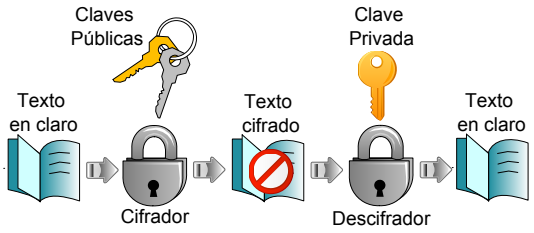


Figura 1.2: Esquema de cifrado de clave pública.

Ventajas

- Únicamente la clave privada debe estar guardada en secreto.

- Dependiendo del modo de uso, las claves privada y pública pueden tener una vida útil muy amplia, por ejemplo, varias sesiones o incluso varios años.

- Muchos esquemas de clave pública ofrecen mecanismos eficientes de firma digital.

- En una red grande, el número de claves necesarias es considerablemente más pequeño que un esquema de clave privada.

Desventajas

- El tiempo de ejecución de los métodos de cifrado de clave pública son considerablemente más lentos en comparación con los esquemas de clave privada.

- El tamaño de las claves son mucho más grandes que lo requerido por los cifradores de clave privada.

- El tamaño de la firma digital usando un esquema de clave pública es grande comparado con las etiquetas de autenticación de datos que se proveen con las técnicas de clave privada.

Los esquemas de criptografía de clave pública basan su seguridad en el problema de factorización de números grandes y el problema del logaritmo discreto. Entre los cifradores más utilizados y conocidos podemos encontrar a RSA que su seguridad está

basada en el problema de factorización de números, DSA es un esquema que su seguridad se concentra en el problema del logaritmo discreto. Otro esquema de criptografía de curvas elípticas es ECDSA que es considerado una variante de DSA, la seguridad de este esquema está dada por el problema del logaritmo discreto en curvas elípticas.

Un esquema considerado de clave pública son los emparejamientos bilineales, siendo éste el tema principal de esta tesis. Los emparejamientos bilineales son explicados a lo largo del capítulo 2.

1.2. Aritmética modular

En esta sección se presenta una explicación sobre campos finitos, con el objetivo de comprender algunas operaciones realizadas durante el desarrollo de esta tesis.

1.2.1. Grupo

Un grupo $(G, *)$ consiste de un conjunto G con una operación binaria $*$ en G que satisface los siguientes axiomas:

- La operación del grupo es cerrada, es decir: $\forall a, b \in G : a * b \in G$.
- La operación del grupo es asociativa, es decir $a * (b * c) = (a * b) * c$ para todos $a, b, c \in G$.
- Existe un elemento $1 \in G$, llamado elemento identidad, tal que $a * 1 = 1 * a = a$ para todo $a \in G$.
- Para cada $a \in G$ existe un elemento $a^{-1} \in G$ llamado el inverso de a, tal que $a * a^{-1} = a^{-1} * a = 1$.

Se dice que G es un grupo *abeliano* (o conmutativo), si $a * b = b * a$ para todo $a, b \in G$.

1.2.2. Anillo

Un anillo $(R, +, \times)$ consiste de un conjunto R con dos operaciones binarias denotadas por $+$ (suma) y $\times$ (multiplicación) en R y que satisface los siguientes axiomas:

- $(R, +)$ es un grupo abeliano aditivo con elemento identidad 0.
- La operación $\times$ es asociativa. Es decir, $a \times (b \times c) = (a \times b) \times c$ para todos $a, b, c \in R$.
- Existe un elemento identidad multiplicativa denotado por 1, tal que $1 \times a = a \times 1 = a$ para todo $a \in R$.

- La operación $\times$, es distributiva sobre la operación $+$:

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(b + c) \times a = (b \times a) + (c \times a)$$

para todos $a, b, c \in R$.

1.2.3. Campo

Un campo es un anillo en el cual la multiplicación es conmutativa y todo elemento diferente de 0 tiene un inverso multiplicativo.

- La característica de un campo es 0 si $\overbrace{1 + 1 + \cdots + 1}^{m \ veces}$ nunca es igual a 0 para cualquier $m \geq 1$, de lo contrario, la característica del campo es el mínimo entero positivo m, tal que $\sum_{i=1}^{m} 1 = 0$.
- Si la característica m de un campo no es 0, entonces m necesariamente es un número primo [53].

1.2.4. Campos finitos

Un campo finito o campo de Galois denotado por $GF(q = p^n)$ o $\mathbb{F}_{p^n}$, es un campo de característica p, que cuenta con un número q de elementos. También puede ser denotado como $\mathbb{F}_q$.

- Un campo finito $\mathbb{F}_{p^n}$ existe para cada número primo p y cada entero n positivo, y éste contiene un subcampo con p elementos. Tal subcampo es conocido como "campo base".
- Los elementos diferentes de 0 en $\mathbb{F}_q$ forman un grupo multiplicativo y es denotado por $\mathbb{F}_q^*$.
- Para cada elemento $\alpha \in \mathbb{F}_q$ no cero, la identidad $\alpha^{q-1} = 1$ se mantiene.
- $\mathbb{F}_q$ es un grupo cíclico de orden $q - 1$, por lo tanto $\alpha^q = \alpha$ para todo $\alpha \in \mathbb{F}_q$.

Cabe mencionar que los campos finitos cuentan con varias operaciones aritméticas como son las suma, resta, exponenciación, multiplicación, inversión para elementos diferentes de cero, por mencionar algunas. Las operaciones se realizan de forma modular bajo un primo grande o un polinomio irreducible, dependiendo del dominio del campo.

Los campos finitos son muy utilizados en la criptografía ya que las operaciones de diversos algoritmos criptográficos se construyen con operaciones aritméticas. Para el resto de esta tesis, sólo se consideran dos tipos de campos: los campos $GF(2^m)$ que son conocidos como campos finitos binario o de característica dos y los campos $GF(3^m)$ también conocidos como campos finitos ternarios o de característica tres, el orden mínimo de cada campo es $q = 2^m$ y $q = 3^m$ respectivamente.

1.2.4.1. Campos finitos de característica dos

Un campo finito binario $\mathbb{F}_{2^m}$ es una extensión del campo base $\mathbb{F}_2 = \{0, 1\}$, los elementos que pertenecen a este campo se pueden representar como polinomios de grado a lo más $m - 1$ y con coeficientes en $\{0, 1\}$, es decir, en el campo base.

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0 = \sum_{i=0}^{m-1} a_i x^i$$

donde

$$a_i \in \{0, 1\} \quad \forall \quad i = 0, 1, 2, \ldots, m-1$$

Las operaciones en un campo finito binario, son operaciones modulares bajo un polinomio irreducible binario, llamado también polinomio generador, de grado m y es utilizado para generar el campo, este polinomio es de la forma siguiente:

$$P(x) = x^m + \sum_{i=1}^{m-1} p_i x^i + 1$$

Al ser irreducible significa que no puede ser factorizado como un producto de polinomios binarios de grado menor que m.

Como ya se mencionó, los campos finitos cuentan con operaciones aritméticas básicas tales como la suma, resta, multiplicación, inversión entre otras más. En la Tabla 1.1 se presenta la operación de suma y resta para un elemento de grado cero, es decir el primer coeficiente del polinomio.

+/−	0	1
0	0	1
1	1	0

Tabla 1.1: Suma y resta en característica dos.

Para sumar dos elementos en $\mathbb{F}_{2^m}$, la operación es realizada coeficiente a coeficiente (mod 2). Otra operación importante es la multiplicación, en la Tabla 1.2 se representa la multiplicación entre dos coeficientes de un polinomio.

$*$	0	1
0	0	0
1	0	1

Tabla 1.2: Multiplicación en característica dos.

La multiplicación de dos elementos en $\mathbb{F}_{2^m}$ es más compleja, se realiza mediante una multiplicación polinomial seguida por una reducción modular utilizando el polinomio irreducible es decir:

$$C(x) = A(x)B(x) \bmod P(x)$$

donde $A(x), B(x), C(x) \in \mathbb{F}_{2^m}$.

Los campos finitos binarios tienen la propiedad de que operaciones como la elevación al cuadrado y la raíz cuadrada son lineales, por tanto su costo computacional es muy barato, éstas y otras operaciones serán explicadas con más detalle en el capítulo 3.

1.2.4.2. Campos finitos de característica tres

Un campo finito de característica tres o también llamado campo ternario, $\mathbb{F}_{3^m}$, es una extensión de $\mathbb{F}_3$ y sus elementos se pueden representar como polinomios de grado a lo más $m-1$, con coeficientes a_i en $\mathbb{F}_3$:

$$A(x) = \sum_{i=0}^{m-1} a_i x^i$$

donde

$$a_i \in \{0, 1, 2\} \quad \forall \quad i = 0, 1, 2, \ldots, m-1$$

En las Tablas 1.3 y 1.4 se muestra la definición para las operaciones de suma y resta respectivamente sobre el campo $\mathbb{F}_3 = \{0, 1, 2\}$.

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Tabla 1.3: Suma en característica tres.

$-$	0	1	2
0	0	2	1
1	1	0	2
2	2	1	0

Tabla 1.4: Resta en característica tres.

Al igual que en característica dos, para calcular la suma o resta de dos elementos en $\mathbb{F}_{3^m}$ se realiza coeficiente a coeficiente. Otra operación muy importante es la multiplicación la cual se define en la Tabla 1.5 para el campo $\mathbb{F}_3$, es decir entre dos coeficientes.

$*$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Tabla 1.5: Multiplicación en característica tres.

La multiplicación de dos elementos en $\mathbb{F}_{3^m}$ se realiza mediante una multiplicación polinomial seguida por una reducción modular utilizando el polinomio irreducible:

$$C(x) = A(x)B(x) \bmod P(x)$$

donde $A(x), B(x), C(x) \in \mathbb{F}_{3^m}$.

En los campos de característica tres se incluyen todas las operaciones aritméticas, pero al igual que los campos binarios, éstos presentan la característica de que las operaciones como la elevación al cubo y la raíz cúbica son operaciones lineales y son muy eficientes sobre campos ternarios.

1.3. Curvas elípticas sobre campos finitos

La teoría de las curvas elípticas es una de las creaciones más interesantes de la matemática del siglo XX, si bien sus antecedentes se remontan hasta la matemática griega. Pero fue a mediados de los años 80 cuando Neal Koblitz [48] y Victor Miller [55] propusieron utilizar curvas elípticas para diseñar sistemas criptográficos de clave pública.

La criptografía de curvas elípticas ha tenido un gran crecimiento, a tal grado que es hoy uno de los esquemas de criptografía de clave pública más usados, junto con RSA [58].

Una de las ventajas de utilizar curvas elípticas es que ofrecen un nivel de seguridad comparable con criptosistemas clásicos que utilizan tamaños de claves muy grandes. Por ejemplo, se estima que algunos criptosistemas convencionales con claves de 4096 bits pueden ser remplazados con sistemas de curvas elípticas de 313 bits, además otra ventaja es la eficiencia en su implementación [65].

Una curva elíptica denotada por E sobre un campo K, es definida por un tipo de ecuación canónica llamada *forma de Weierstrass* [40], que no es sino una ecuación de la forma:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1.1}$$

$a_1, a_2, a_3, a_4, a_6 \in K$ y $\Delta \neq 0$, donde Δ es el discriminante de E y es definido como sigue:

$$\left.\begin{array}{rcl} \Delta & = & -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6 \\ d_2 & = & a_1^2 + 4a_2 \\ d_4 & = & 2a_4 + a_1 a_3 \\ d_6 & = & a_3^2 + 4a_6 \\ d_8 & = & a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2 \end{array}\right\}$$

La condición $\Delta \neq 0$ asegura que la curva elíptica es "suave", es decir que no hay puntos con dos o más líneas tangentes diferentes.

Como primer acercamiento a una curva elíptica en la Figura 1.3 se presentan dos ejemplos de curvas elípticas sobre el campo de los números reales $\mathbb{R}$, la primera de la forma $y^2 = x^3 + ax + b$ y la otra $y^2 = x^3 + b$, que no son más que una forma simplificada de la ecuación de Weierstrass.

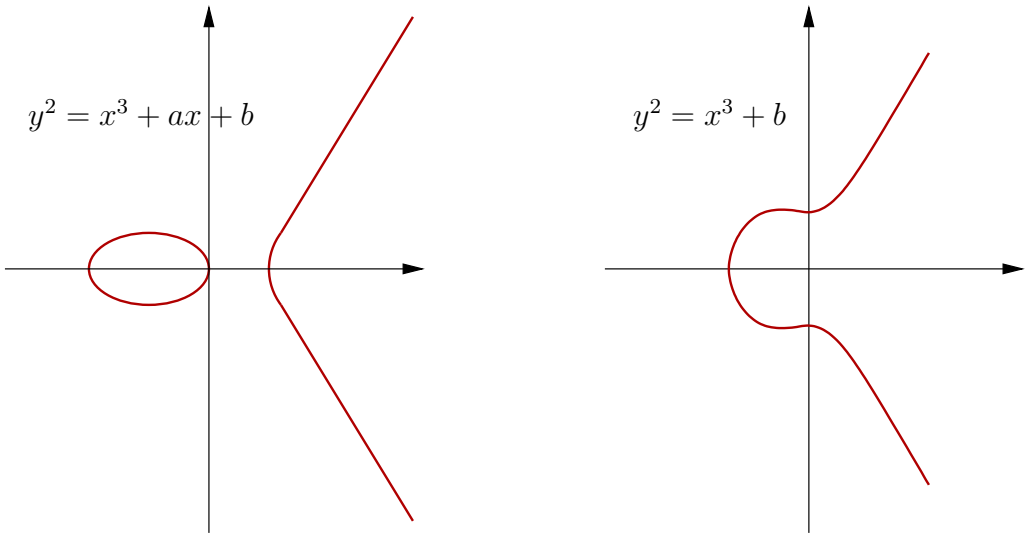


Figura 1.3: Ejemplos de Curvas Elípticas: a) $y^2 = x^3 + ax + b$, b) $y^2 = x^3 + b$.

La gráfica se obtiene mediante la evaluación de x y resolviendo la ecuación cuadrática para y, una curva elíptica puede consistir de dos partes distintas o sólo de una parte como se muestra en la Figura 1.3. Las curvas pueden estar definidas sobre diversos campos, entre los cuales podemos mencionar a los números reales $\mathbb{R}$, los números racionales $\mathbb{Q}$, los números complejos $\mathbb{C}$, sobre un campo primo $\mathbb{F}_q$ o sobre campos binarios $\mathbb{F}_{2^m}$ y campos ternarios $\mathbb{F}_{3^m}$, siendo estos dos últimos los campos de interés en este trabajo de tesis.

1.3.1. Orden de una curva elíptica

Sea E una curva elíptica definida sobre $\mathbb{F}_q$, el número de puntos en $E(\mathbb{F}_q)$, denotado por $\#E(\mathbb{F}_q)$, es llamado el *orden* de E sobre $\mathbb{F}_q$, este grupo consiste de todos los puntos en el campo que satisfacen la ecuación de la curva elíptica más el punto al infinito $\mathcal{O}$ [40].

$$\#E(\mathbb{F}_q) = \left|\left\{(x,y) \in y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\right\} \cup \{\mathcal{O}\}\right|$$

donde $\mathcal{O}$ es el punto al infinito o también llamado elemento neutro aditivo y toda curva elíptica cuenta con este elemento.

Los puntos que satisfacen la ecuación y que son diferentes de $\mathcal{O}$, son llamados puntos racionales en E, los puntos cuentan con coordenadas (x, y) y se cumple que $x, y \in \mathbb{F}_q$. Un teorema que nos ayuda a calcular el orden de una curva sobre un campo finito es llamado el teorema de *Hasse*, el cual se enuncia a continuación.

Teorema 1.3.1. (Hasse)[20] *Sea E una curva elíptica definida sobre $\mathbb{F}_q$, tenemos que $\#E(\mathbb{F}_q)$ esta dado por:*

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}$$

El intervalo $\left[q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}\right]$ es conocido como el *intervalo de Hasse.*

Una formulación alternativa con ayuda del teorema es $\#E(\mathbb{F}_q) = q + 1 - t$ con $|t| \leq 2\sqrt{q}$, donde t es llamado *traza* de E o *traza de Frobenius*. Como $2\sqrt{q}$ es pequeño en relación a q, entonces un estimativo es $\#E(\mathbb{F}_q) \approx q$.

1.3.2. Orden de un punto

De forma análoga, también existe el concepto de orden de un punto, que es el número de elementos que puede generar un punto P sobre la curva E, más formalmente, el orden de un punto P en $E(\mathbb{F}_q)$ es el mínimo entero k tal que $kP = \mathcal{O}$. El orden de cualquier punto siempre divide el orden de la curva $\#E(\mathbb{F}_q)$, por lo cual si r y l son enteros, entonces $rP = lP$ si y sólo si $r \equiv l \bmod q$.

1.3.3. Grado de seguridad (Embedding degree)

En un subgrupo $G = \langle P \rangle$ de una curva $E(\mathbb{F}_q)$, se dice que tiene un grado de seguridad k con respecto a l, si k es el entero más pequeño tal que $l|q^k - 1$ donde l es el orden de G.

Cuando se selecciona una curva para ser utilizada en un criptosistema de curvas elípticas se busca contar con un grado de seguridad k tan grande como sea posible para evitar el ataque MOV [52]. Para la mayoría de las curvas elípticas esta condición se cumple de manera automática. Sin embargo, para algunas clases de curvas llamadas curvas supersingulares, el grado de seguridad es relativamente pequeño, provocando que éstas no se utilicen en la criptografía de curvas elípticas.

El Instituto Nacional de Estándares y Tecnología del gobierno de Estados Unidos (*NIST* por sus siglas en inglés) recomienda curvas elípticas seguras para ser utilizadas en la implementación de criptosistemas de curvas elípticas [56] .

1.3.4. Operaciones en curvas elípticas

El grupo de puntos que satisfacen una curva elíptica generado a partir de un punto generador P es un grupo aditivo, es decir, la operación básica que existe en este grupo es la suma. Esta operación puede ser representada de forma gráfica en la curva elíptica.

Es necesario mencionar que la negación de un punto $P = (x, y)$ que satisface una curva elíptica sobre los números reales $\mathcal{R}$, es la reflexión del punto en el eje x, es decir, $-P = (x, -y)$, con esto se puede garantizar que si P satisface la curva elíptica, entonces $-P$ también satisface la curva.

1.3.4.1. Suma

Si P y Q son dos puntos distintos en una curva elíptica podemos calcular las suma de estos puntos obteniendo como resultado un tercer punto, la operación es denotada como $R = P + Q$.

Para sumar los puntos P y Q, una línea es trazada a través de los puntos como se muestra en la Figura 1.4, esta línea corta a la curva en un tercer punto llamado $-R$, el punto $-R$ es reflejado con respecto al eje x para obtener el punto R que es el punto resultante de la suma. En la Figura 1.4 podemos observar una representación geométrica de la suma de dos diferentes puntos en una curva elíptica.

Existe un caso especial en la suma cuando queremos sumar P con $-P$, debido a que la línea que forman estos puntos nunca corta a la curva elíptica en un tercer punto. Es por esta razón que el grupo de la curva incluye el elemento al infinito $\mathcal{O}$. por definición $P + (-P) = \mathcal{O}$ y como resultado de esta ecuación tenemos que $P + \mathcal{O} = P$ en el grupo, este caso es mostrado en la Figura 1.5.

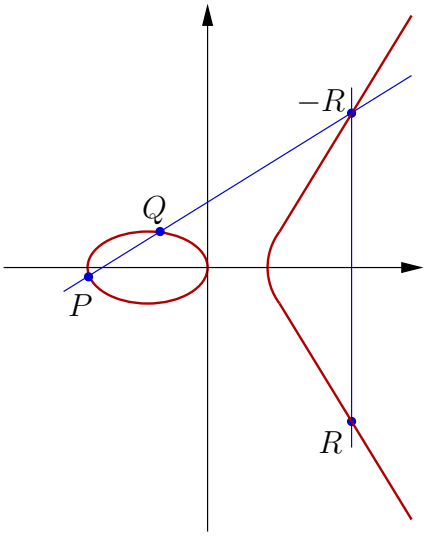


Figura 1.4: Suma de dos puntos, $R = P + Q$.

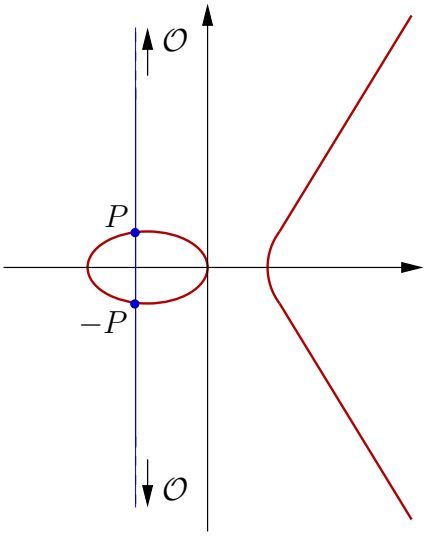


Figura 1.5: Suma del punto, $P + (-P) = \mathcal{O}$.

1.3.4.2. Doblado

La operación de doblado de un punto es denotado por $2P = P + P = R$ es mostrada en la Figura 1.6, esta operación consiste en sumar el punto P consigo mismo. De manera gráfica, una línea tangente a la curva que toca el punto P es dibujada, si se cumple que la coordenada y del punto es diferente de 0, entonces la tangente corta a la curva elíptica

en un otro punto $-R$, el punto $-R$ es reflejado con respecto al eje x y se obtiene el punto R a esta operación se le llama doblado del punto P.

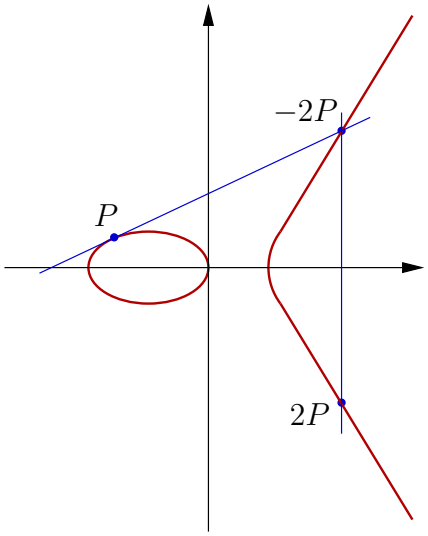


Figura 1.6: Doblado del punto, $2P = P + P = R$.

Cuando doblamos un punto P y la coordenada $y = 0$, entonces la tangente a la curva que toca a este punto nunca corta a la curva en algún otro punto por esta razón el resultado es $\mathcal{O}$, en la Figura 1.7 se puede observar este caso.

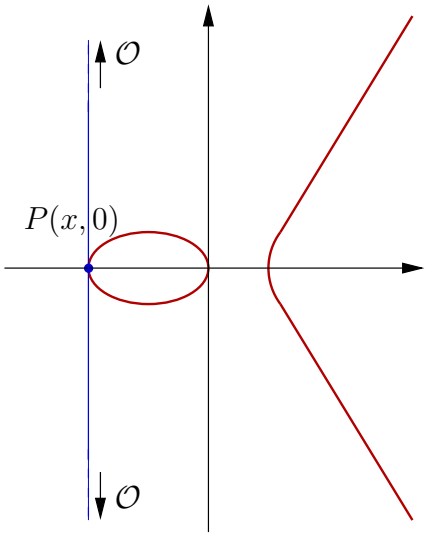


Figura 1.7: Doblado del punto, $2P = \mathcal{O}$ si $P(x, 0)$.

1.3.4.3. Multiplicación escalar

La operación de multiplicación de dos puntos en un grupo aditivo de curvas elípticas no existe, sin embargo, la multiplicación escalar rP puede ser obtenida mediante la suma de r veces el mismo punto P, esto es $rP = P + P + \ldots + P$ para realizar esta operación son necesarias las operaciones de suma y doblado de puntos.

Dos casos especiales son populares para el uso de curvas elípticas en el criptografía: uso de campos de la forma $\mathbb{F}_q$ y $\mathbb{F}_{2^m}$, las operaciones antes descritas se resumen en las siguientes ecuaciones en sus correspondientes campos:

Campo: $\mathbb{F}_q$	Campo: $\mathbb{F}_{2^m}$
$E(\mathbb{F}_q) : y^2 = x^3 + ax + b$	$E(\mathbb{F}_{2^m}) : y^2 + xy = x^3 + ax^2 + b$
$x_3 = \lambda^2 - x_1 - x_2$	$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$
$y_3 = \lambda(x_1 - x_3) - y_1$	$y_3 = \lambda(x_1 - x_3) + x_3 + y_1$
$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & if \quad P_1 \neq P_2 \\ \dfrac{3x_1^2 + a}{2y_1} & if \quad P_1 = P_2 \end{cases}$	$\lambda = \begin{cases} \dfrac{y_2 + y_1}{x_2 + x_1} & if \quad P_1 \neq P_2 \\ x_1 + \dfrac{y_1}{x_1} & if \quad P_1 = P_2 \end{cases}$

Tabla 1.6: Operaciones en curvas elípticas sobre $\mathbb{F}_q$ y $\mathbb{F}_{2^m}$

1.3.5. Problema del logaritmo discreto en curvas elípticas

Cada criptosistema de clave pública está basado en un problema matemático muy difícil, como es el problema del logaritmo discreto. Este problema es la base de la seguridad en varios criptosistemas incluyendo la criptografía de curvas elípticas.

Más específicamente, la seguridad de un criptosistema de curvas elípticas depende del problema del logaritmo discreto en curvas elípticas (ECDLP por sus siglas en inglés). Una de las ventajas de utilizar curvas elípticas es que proporcionan niveles de seguridad altos con longitudes de clave pequeñas es decir las operaciones se realizan con campos más pequeños.

Supóngase que se tiene dos puntos P y Q que pertenecen a una curva elíptica E con:

$$Q = kP = P + P + \ldots + P$$

para algún entero k. El problema radica en averiguar el valor de k si sólo conocemos P y Q.

Este problema puede no lucir como un problema de logaritmos, sin embargo se trata de una clara analogía del clásico problema del logaritmo discreto en campos finitos multiplicativos y por esto es llamado problema de logaritmo discreto en curvas elípticas.

1.3.6. Curvas elípticas supersingulares

Sea una curva E definida sobre $\mathbb{F}_{q=p^n}$ donde p es la característica del campo, consideremos que $t = q + 1 - \#E(\mathbb{F}_q)$. La curva E es llamada supersingular si y sólo si t es divisible por la característica del campo, es decir, $t \equiv 0 \bmod p$, que se cumple sólo si $\#E(\mathbb{F}_q) \equiv 1 \bmod p$. En otro caso la curva es ordinaria [66].

En el caso de que $p \geq 5$, E es supersigular sobre $\mathbb{F}_p$ sólo si la traza de Frobenius $t = 0$, es el caso en que $\#E(\mathbb{F}_q) = p + 1$

Por un lado, las curvas supersingulares se consideran débiles, ya que el problema del logaritmo discreto en este tipo de curvas puede ser reducido al problema del logaritmo discreto sobre un grupo multiplicativo de extensión $k-$ésima, $\mathbb{F}_{q^k}$, del campo $\mathbb{F}_q$ [52, 31]. Sin embargo, las curvas supersingulares tienen la característica de que las operaciones entre puntos son más eficientes que las operaciones en curvas ordinarias.

1.3.6.1. Curvas elípticas supersingulares en $\mathbb{F}_{2^m}$

Para propósitos criptográficos, es natural pensar en curvas elípticas definidas sobre $\mathbb{F}_{2^m}$ con m impar o más fuertemente un primo. Existen varias clases de curvas pero en este trabajo estamos interesados principalmente en la siguiente curva:

$$E(\mathbb{F}_{2^m}) : \; y^2 + y = x^3 + x + b, \quad b \in \{0, 1\}$$

que tiene un grado de seguridad $k = 4$ y donde el orden de la curva es: $\ell = 2^m + 1 \pm 2^{(m+1)/2}$ [8].

Para describir las operaciones sobre esta curva elíptica consideremos un punto arbitrario $P = (\alpha, \beta)$ en la curva donde $\alpha, \beta \in \mathbb{F}_{2^m}$.

Negación de un punto

La negación de un punto es la operación más sencilla, considerando el punto $P(\alpha, \beta)$ entonces $-P = (\alpha, \beta + 1)$.

Doblado de un punto

Sea $P = (\alpha, \beta)$, tenemos que $2P = (\alpha^4 + 1, \alpha^4 + \beta^4)$, esta operación es mucho más eficiente que la operación de doblado en curvas no supersingulares, dado que la elevación al cuadrado es muy eficiente en un campo binario.

Suma de puntos

Sea $P_1 = (\alpha_1, \beta_1)$, $P_2 = (\alpha_2, \beta_2)$, para calcular $P_3 = P_1 + P_2 = (\alpha_3, \beta_3)$:

$$\begin{aligned} \lambda &= \frac{\beta_1 + \beta_2}{\alpha_1 - \alpha_2} \\ \alpha_3 &= \lambda^2 + \alpha_1 + \alpha_2 \\ \beta_3 &= \lambda(\alpha_1 + \alpha_3) + \beta_1 + 1 \end{aligned}$$

1.3.6.2. Curvas elípticas supersingulares en $\mathbb{F}_{3^m}$

En campos finitos ternarios también existen varias clases de curvas elípticas supersingulares, pero nuevamente nos enfocaremos en una clase en especial, donde el multiplicador de seguridad es $k = 6$, y la ecuación de la curva es la siguiente:

$$E(\mathbb{F}_{3^m}) : \; y^2 = x^3 - x + b, \quad b \in \{-1, 1\}$$

donde el orden de la curva es: $\ell = 3^m + 1 \pm 3^{(m+1)/2}$ [8].

Negación de un punto

Si consideramos el punto $P(\alpha, \beta)$ entonces $-P = (\alpha, -\beta)$. Esta operación es muy sencilla pues sólo involucra una negación de un elemento $\beta \in \mathbb{F}_{3^m}$.

Doblado de un punto

Sea $P_1 = (\alpha_1, \beta_1)$, para calcular el valor de $2P = (\alpha_2, \beta_2)$ necesitamos:

$$\begin{aligned} \lambda &= 1/\alpha_1 \\ \alpha_2 &= \alpha_1 + \lambda^2 \\ \beta_2 &= -(\beta_1 + \lambda^3) \end{aligned}$$

Triplicado de un punto

Sea $P = (\alpha, \beta)$, tenemos que $3P = (\alpha^9 - b, -\beta^9)$, esta operación es muy eficiente dado que la elevación al cubo es una operación considerada lineal en un campo ternario.

Suma de dos puntos

Sea $P_1 = (\alpha_1, \beta_1)$, $P_2 = (\alpha_2, \beta_2)$, para calcular $P_3 = P_1 + P_2 = (\alpha_3, \beta_3)$:

$$\begin{aligned} \lambda &= \frac{\beta_2 - \beta_1}{\alpha_2 - \alpha_1} \\ \alpha_3 &= \lambda^2 - (\alpha_1 + \alpha_2) \\ \beta_3 &= \beta_1 + \beta_2 - \lambda^3) \end{aligned}$$

1.4. Funciones picadillo

Una función picadillo toma un mensaje M de longitud arbitraria y devuelve como resultado una secuencia de caracteres de longitud fija [53]. Una función picadillo debe de cumplir con las siguientes propiedades:

- Dado un M cualquiera, la función picadillo $H(M)$ debe ser fácil de calcular.
- Dado un M cualquiera, obtener M a partir de $H(M)$ debe ser comutacionalmente difícil.
- La función picadillo debe ser resistente a colisiones, es decir, computacionalmente no debe ser posible encontrar M y M' tales que $H(M) = H(M')$.

La funciones picadillo son aplicadas principalmente para resolver problemas relacionados con la integridad de mensajes. Los algoritmos de funciones picadillo más conocidos y usados en la actualidad son: SHA-1 y MD5. El primero fue desarrollado por la NSA, para ser incluido en el estandar DSS (Digital Signature Standard). El segundo es el resultado de una serie de mejoras que su diseñador Ron Rivest llevó a cabo a partir del algoritmo MD2 pasando por MD4 hasta llegar a lo que hoy conocemos como MD5. Tanto SHA-1 y MD5 procesan tramas de longitud arbitraria y producen una salida de 128 bits.

Otras aplicaciones que las funciones picadillo tienen son aquellas relacionadas con curvas elípticas y que cuentan con características especiales, a este tipo se les conoce como funciones picadillo *map-to-point*.

1.4.1. Función picadillo map-to-point

Una función picadillo map-to-point mapea una cadena de longitud arbitraria directamente a un punto que satisface una curva elíptica definida sobre un campo finito.

$$H_1 : \{0,1\}^* \rightarrow G_1^*$$

donde $\{0,1\}^*$ es cualquier cadena de bits y G_1^* es un grupo aditivo de una ecuación $E(\mathbb{F}_q)$ construido por un elemento generador P.

En el trabajo de Boneh et. al. [16] en el 2001 se presenta una aplicación de los emparejamientos bilineales, donde el emparejamiento de Weil es utilizado en un esquema de firmas cortas en el cual es necesaria una función picadillo map-to-point con curvas elípticas sobre campos finitos ternarios, $E(\mathbb{F}_{3^m}) : y^2 = x^3 - x + b$ donde $b = \pm 1$, el algoritmo hace uso de una función picadillo estándar (h'), su propuesta consiste en generar un valor de x y resolver la ecuación de la curva dando como resultado el algoritmo 1.1.

Algoritmo 1.1: Map-to-Point de Boneh et. al. [16]

Entrada: $M = \{0,1\}^*$
Salida: $P \in E(\mathbb{F}_{3^m})$

1 $i \leftarrow 0$;
2 Obtener $(x,t) = h'(i||M)$, donde $x \in \mathbb{F}_{3^m}$ y $t \in \{0,1\}$;
3 Calcular $u = x^3 - x + b$;
4 Resolver la ecuación cuadrática $y^2 = u$ sobre $\mathbb{F}_{3^m}$;
5 **if** *no se encuentra la solución* **then**
6 $i \leftarrow i + 1$;
7 regresar al paso 2;
8 **end**
9 utilizamos t para elegir una de las posibles soluciones y_0 o y_1;
10 **return** $P = (x, y_t)$

El algoritmo anterior es eficaz, pero resulta costoso en el paso 4 al calcular la raíz cuadrática sobre un campo ternario. En ese mismo año en un trabajo posterior de Barreto y Kim en [7] proponen una mejora al algoritmo para eliminar la raíz cuadrada como se muestra en el algoritmo 1.2.

Algoritmo 1.2: Map-to-Point de Barreto y Kim [7]

Entrada: $M = \{0,1\}^*$
Salida: $P \in E(\mathbb{F}_{3^m})$

1 $i \leftarrow 0$;
2 Obtener $(y,t) = h'(i||M)$, donde $x \in \mathbb{F}_{3^m}$ y $t \in \{0,1\}$;
3 Calcular $u = y^2 - b$;
4 Resolver la ecuación $C(x) = u$;
5 **if** *no se encuentra la solución* **then**
6 $i \leftarrow i + 1$;
7 regresar al paso 2;
8 **end**
9 utilizamos t para elegir una de las posibles soluciones y_0 o y_1;
10 **return** $P = (x, y_t)$

La idea es ahora proponer un valor para la variable y y resolver la ecuación del paso 4 donde $C(x) = x^3 - x$, es decir, se trata de resolver una ecuación cúbica.

2

EMPAREJAMIENTOS BILINEALES

Todas las teorías son legítimas y ninguna tiene importancia. Lo que importa es lo que se hace con ellas.

Jorge Luis Borges (1899 – 1986)

2.1. Emparejamiento bilineal

Los emparejamientos bilineales fueron incialmente introducidos en la criptografía por A. Menezes, E. Okamoto y S. Vanstone (MOV) en 1993 [52] como un ataque a los esquemas de criptografía de curvas elípticas. La seguridad de un criptosistema de curvas elípticas se basa en la dificultad de calcular logaritmos en un grupo que satisface una curva elíptica. Este ataque consiste en reducir el problema de logaritmo discreto en una curva $E(\mathbb{F}_q)$ al problema del logaritmo discreto en una extensión del campo $\mathbb{F}_q$ denotado por $\mathbb{F}_{q^k}$, y esto es logrado mediante el uso del emparejamiento de Weil.

En 1994 G. Frey y H. Rück (FR) [31], realizaron un trabajo donde muestran que el cálculo del logaritmo discreto puede ser reducido en algunas clases de curvas elípticas definidas sobre campos finitos y ellos propusieron utilizar el emparejamiento de Tate.

Una comparación entre las reducciones de MOV y FR fue publicada en Eurocrypt'99 [41] llegando a la conclusión que la reducción de FR se puede aplicar a más curvas y además puede ser calculada de manera más rápida que la reducción de MOV, ello a causa de que el emparejamiento de Tate es más eficiente que el emparejamiento de Weil.

Ambos ataques son muy eficientes en la clase de curvas llamadas supersingulares por tener un grado de seguridad pequeño como se mencionó en el capítulo anterior. Sin embargo, es precisamente esta propiedad la que provoca que las curvas supersingulares sean excelentes candidatas para ser utilizadas en la criptografía basada en emparejamientos.

En el año 2000 los primeros autores en utilizar los emparejamientos bilineales para dar soluciones a problemas criptográficos y no como herramientas para ataques fueron Joux [45] y Sakai, quienes aprovecharon las propiedades de los emparejamientos. A raíz de esto, el estudio de emparejamientos en la critpgrafía ha crecido a un ritmo acelerado, dando paso a la solución y creación de nuevos y elegantes protocolos criptográficos como la criptografía basada en la identidad [15], firmas cortas [16], esquemas para acuerdo de claves [45, 9] sólo por mencionar algunos.

2.1.1. Definición de emparejamiento bilineal

Sean G_1, G_2 dos grupos, ambos con el mismo orden primo q, donde $G_1 = \langle P \rangle$ es un grupo aditivo con elemento neutro aditivo $\mathcal{O}$ y G_2 un grupo multiplicativo con elemento neutro multiplicativo 1, y sea P un elemento generador arbitrario de G_1.

Suponemos que el problema del logaritmo discreto (DLP) es difícil en ambos grupos G_1 y G_2.

Un emparejamiento bilineal es una función que mapea dos elementos del grupo aditivo G_1 a un elemento del grupo multiplicativo G_2:

$$\hat{e} : G_1 \times G_1 \rightarrow G_2$$

En la Figura 2.1 se muestra de manera gráfica su funcionamiento. Los elementos de entrada son $P, Q \in G_1$, el elemento de salida $g \in G_2$ y el emparejamiento η_T.

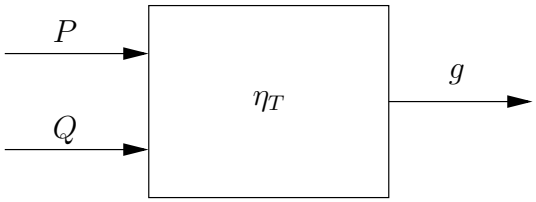


Figura 2.1: Emparejamiento η_T.

2.1.2. Propiedades

Un emparejamiento bilineal satisface loas siguientes propiedades:

1. Bilinealidad: Para todos $R, S, T \in G_1$,

$$\hat{e}(R+S,T) = \hat{e}(R,T)\hat{e}(S,T)$$

$$\hat{e}(R,S+T) = \hat{e}(R,S)\hat{e}(R,T)$$

2. No degeneración: Si P es un elemento generador en G_1, entonces $\hat{e}(P,P)$ es un generador de G_2, en otras palabras, $\hat{e}(P,P) \neq 1$, donde $P \neq \mathcal{O}$.

3. Computabilidad: Existe un algoritmo eficiente para calcular $\hat{e}(P,Q)$ para todo $P, Q \in G_1$.

Los emparejamientos bilineales presentan las siguientes propiedades:

1. $\hat{e}(S,\mathcal{O}) = 1$ y $\hat{e}(\mathcal{O},S) = 1$, $\forall\ S \in G_1$

2. $\hat{e}(S,-T) = \hat{e}(-S,T) = \hat{e}(S,T)^{-1}$, $\forall\ S,T \in G_1$

3. Bilinealidad: $\hat{e}(aS,bT) = \hat{e}(S,T)^{ab}$, $\quad \forall\ S,T \in G_1$ y $a,b \in \mathbb{Z}$

4. Simetría: $\hat{e}(S,T) = \hat{e}(T,S)$, $\forall\ S,T \in G_1$

5. Si se cumple que $\hat{e}(S,R) = 1,\ \forall\ R \in G_1$, entonces $S = \mathcal{O}$

Es la propiedad de bilinealidad la que permite el desarrollo de nuevos y emocionantes protocolos criptográficos.

2.2. Algoritmos de emparejamiento

El primero en proponer un algoritmo para calcular el emparejamiento de Weil fue Victor Miller en 1986 [54], este algoritmo es conocido como el Algoritmo de Miller, que consiste básicamente en el método de "doblar y sumar" que se utiliza en la multiplicación de puntos en curvas elípticas combinado con la evaluación de determinadas funciones que son las líneas rectas utilizadas en el proceso de suma de puntos. Estas líneas son obtenidas de la siguiente manera: sean P y Q puntos en la curva elíptica E, al sumar dos puntos una línea es trazada entre los dos puntos, esta línea es llamada l_1 y corta a la curva en un tercer punto $-R$, después es necesario trazar una línea vertical llamada l_2 para encontrar el punto $R = P+Q$, como se muestra en la Figura 1.4 de la sección 1.3.4.

Las líneas l_1 y l_2 son consideradas funciones que pertenecen a la curva y se puede obtener sus correspondientes divisores y obtener la siguiente igualdad de divisores:

$$(l_1/l_2) - (\mathcal{O}) + (R) = -(\mathcal{O}) + (Q) - (\mathcal{O}) + (P)$$

En [64] se explica la teoría sobre grupos divisores en una curva.
El algoritmo de Miller puede ser utilizado para calcular el emparejamiento de Tate.

Algoritmo 2.1: Algoritmo de Miller

Entrada: P, R

Salida: $f = \langle P, Q \rangle_r$ y la expansión binaria de r

```
1  T ← P; f ← 1;
2  for i ← ⌊log_2(r)⌋ − 1 downto 0 do
3      Calcular las ecuaciones de las lineas l_1 y l_2 resultantes del doblado
       de T;
4      T ← 2T;
5      f ← f^2 · l_1(Q + R)l_2(R)/(l_2(Q + R)l_1(R));
6      if r_i = 1 then
7          Calcular las ecuaciones de las lineas l_1 y l_2 resultantes de la
           suma de T con P;
8          T ← T + P;
9          f ← f · l_1(Q + R)l_2(R)/(l_2(Q + R)l_1(R));
10     end
11 end
12 return f
```

En el 2002 Barreto et al. [8] proponen realizar unas mejoras significativas para el algoritmo de Miller al ser utilizado en el cálculo el emparejamiento de Tate en curvas sobre campos de característica tres, como resultado obtiene un algoritmo conocido como BKLS para calcular el emparejamiento de Tate, ver [8].

Entre las principales diferencias entre el algoritmo de Miller y el BKLS, destacan que en este último se deja de utilizar la expansión de r para ser sustituido por un ciclo de longitud $2m-1$. También introducen el concepto de exponenciación final (*final exponentiation*), siendo esta operación muy importante en el emparejamiento de Tate para conservar la propiedad de no degeneración [8]. Barreto definió este exponente de la siguiente manera para campos binarios $q = 2^m$, el exponente $z = (q^4-1)/n = (q+1\pm\sqrt{2q})(q^2-1)$, y en un campo ternario $q = 3^m$ entonces $z = (q^6-1)/n = (q+1\pm\sqrt{3q})(q^3-1)(q+1)$ y para un campo de característica $p > 3$, $z = (p^2-1)/n = p-1$. Para más detalles del algoritmo BKLS y la exponenciación final revisar [8].

En el 2002 Galbraith et. al. [32], propusieron otra variante del algoritmo de Miller para el cálculo rápido del emparejamiento de Tate conocido como GHS por los nombres de los autores. Posteriormente, en el 2003 Iwan Duursma y Hyang-Sook Lee [28], propusieron una mejora más para el emparejamiento de Tate basándose en los trabajos [8, 32], su método es conocido como Duursma-Lee. Los autores proponen utilizar curvas de la forma $y^2 = x^p - x + a$ sobre $\mathbb{F}_{p^m}$, donde $p \geq 3$ y $(m, 2p) = 1$ (en este caso las curvas tiene un grado de seguridad $2p$), la complejidad estimada para el algoritmo de

Duursma-Lee indica una aceleración con un factor de 2 sobre los algoritmos descritos en [8, 32].

Para el 2007 Barreto et al. [6] introdujeron una novedosa variante del emparejamiento de Tate llamada η. Los autores muestran como η es aproximadamente dos veces más rápido que el método de Duursma-Lee, y la notación η_T es reservada para una versión mejorada donde se utiliza una exponenciación final, ver [6]. También se presentan los algoritmos correspondientes para el cálculo del emparejamiento η_T en las características dos y tres.

El emparejamiento η_T es el más eficiente hasta el día de hoy y es éste el emparejamiento de interés en esta tesis.

2.2.1. Emparejamiento de Tate modificado

Beuchat et al. [13] presentan una definición del emparejamiento de Tate modificado en característica tres Sea $N = \#E(\mathbb{F}_{3^m}) = 3^m + 1 + \mu b 3^{\frac{m+1}{2}}$ con:

$$\mu = \begin{cases} +1 & \text{si } m \equiv 1, 11 \mod 12 \text{ o} \\ -1 & \text{si } m \equiv 5, 7 \mod 12 \end{cases}$$

Sea ℓ un factor primo de N y donde $E(\mathbb{F}_{3^m})[\ell]$ denota el subgrupo de las ℓ-torsiones de $E(\mathbb{F}_{3^m})$, es decir, el conjunto de puntos $P \in E(\mathbb{F}_{3^m})$, tal que $[\ell]P = \mathcal{O}$. El emparejamiento de Tate modificado es una función que tiene como entrada dos puntos de $E(\mathbb{F}_{3^m})[\ell]$ y su salida es un elemento del grupo de las ℓ-ésimas raíces de la unidad que es denotado por μ_ℓ. El grado de seguridad k es el entero más pequeño para que μ_ℓ sea contenido en el grupo multiplicativo $\mathbb{F}^*_{3^{km}}$, es decir, k es el entero más pequeño que divide a $3^{km}-1$ y es definido como $\mu_l = \left\{R \in \mathbb{F}^*_{3^{km}} \text{ tal que } R^\ell = 1\right\}$, en característica tres.

El emparejamiento de Tate modificado de orden ℓ es entonces una función:

$$\hat{e}(\cdot,\cdot) : E(\mathbb{F}_{3^m})[\ell] \times E(\mathbb{F}_{3^m})[\ell] \to \mathbb{F}^*_{3^{km}}$$

dado por

$$\hat{e}(P,Q) = f_{\ell,P}(\psi(Q))^{(3^{km}-1)/\ell}$$

donde:

•ψ es un mapa de distorsión de $E(\mathbb{F}_{3^m})[\ell]$ a $E(\mathbb{F}_{3^{6m}})[\ell] \setminus E(\mathbb{F}_{3^m})[\ell]$ definido como $(\psi((x_q, y_q))) = (\rho - x_q, y_q)$ para todo $Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})[\ell]$, donde ρ y σ son elementos de $\mathbb{F}_{3^{6m}}$ y que satisfacen satisfacen las ecuaciones $\rho^3 - \rho - b = 0$ y $\sigma^2 + 1 = 0$. Notar que $\{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$ es una base de $\mathbb{F}_{3^{km}}$ sobre $\mathbb{F}_{3^m}$. Los elementos en $\mathbb{F}_{3^{6m}}$ son representados como $R = r_0 + r_1\sigma + r_2\rho + r_3\sigma\rho + r_4\rho^2 + r_5\sigma\rho^2$ donde los elementos

$r_i's$ pertenecen al campo $\mathbb{F}_{3^m}$.

• $f_{n,P}$ con $n \in \mathbb{N}$ y $P \in E(\mathbb{F}_{3^m})[\ell]$ es una función racional definida sobre $E(\mathbb{F}_{3^{6m}})[\ell]$ con divisor $(f_{n,P}) = n(P) - ([n]P) - (n-1)(\mathcal{O})$. Barreto et al. en [6] proponen evaluar $f_{n,P}$ sobre puntos.

• $f_{\ell,P}(\psi(Q))$ se encuentra definido hasta las ℓ-ésimas potencias, lo cual es indeseable en la mayoría de las aplicaciones criptográficas. El exponenciación a $(3^{6m}-1)$, es conocida como "exponenciación final", la cual permite obtener un valor único que pertenece a un subgrupo multiplicativo de $\mathbb{F}^*_{3^{6m}}$.

Galbraith et al. [33] observaron que el emparejamiento de Tate modificado puede ser calculado con respecto al grupo de orden N:

$$\begin{aligned} M &= \frac{3^{6m}-1}{N} \\ &= (3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}}) \end{aligned}$$

Cabe mencionar que el emparejamiento de Tate modificado puede ser calculado con el emparejamiento η_T como se menciona en [12].

Hasta ahora se han descrito los emparejamientos bilineales, pero hay que tener presente que debemos de prestar atención a los niveles de seguridad, para esto existe una relación directa entre el tamaño del campo utilizado, $\mathbb{F}_q$, y el grado de seguridad de la curva elíptica supersingular utilizada k, como se muestra en [34]. 2^k es el número de operaciones necesarias para romper un cifrador por bloque con una clave de k-bits. En la Tabla 2.1 se presentan los niveles de seguridad para los esquema de cifrado de curvas elípticas (ECC) y el criptosistema RSA presentados por NIST, Lenstra y el proyecto ECRYPT, ver [34] para más detalles.

Autor	k	ECC	RSA
NIST	80	160	1024
	128	256	3072
	256	512	15360
Lenstra	80	160	1329
	128	256	4440
	256	512	26268
ECRYPT	80	160	1248
	128	256	3248
	256	512	15424

Tabla 2.1: Tamaños de clave recomendados.

Un punto importante para los esquema basados en emparejamientos es conseguir un nivel de seguridad adecuado para ser utilizados de forma sergura en aplicaciones. La Tabla 2.1 puede ser utilizada para establecer un adecuado nivel de seguridad en los emparejamientos, por ejemplo si deseamos un nivel de seguridad de 128 bits y utilizando los tamaños de clave recomendados por NIST, necesitamos seleccionar un subgrupo de $E(\mathbb{F}_q)$ que tenga al menos 3072 bits, es decir, q deberá tener un tamaño de al menos 256 bits. También, debemos asegurar que $\mathbb{F}_{q^k}$ sea de un tamaño de al menos 3072 bits.

En la Tabla 2.1, podemos observar que Lenstra presenta los tamaños de clave más grandes. Ahora, si queremos lograr un nivel de seguridad de 128 bits en con una curva definida sobre un campo ternario, $\mathbb{F}_{3^m}$, y sabemos que el característica tres el grado de seguridad es $k = 6$, y debemos lograr 4440 bits de longitud en $\mathbb{F}_{3^{km}}$, y hay que tomar en cuenta que en característica tres se utilizan trits y la complejidad aritmética es mayor ($\log_2(3)$), con esto podemos obtener el tamaño del campo base ($\mathbb{F}_{3^m}$):

$$m \cdot \log_2(3) \cdot 6 = 4440$$

donde la incógnita es m. Al resolver esta ecuación obtenemos que $m \approx 466.88$, sin embargo, es necesario trabajar sobre un campo primo para evitar ataques, el primo más cercano a 466 resulta ser 467, con esto podemos decir que se puede construir un emparejamiento bilineal sobre el campo $\mathbb{F}_{3^{467}}$ y que cuenta con un nivel de seguridad de 128 bits.

2.3. Aplicaciones de los emparejamientos

Como se mencionó en las anteriores secciones los emparejamientos bilineales inicialmente fueron utilizados para criptoanálisis [52, 31], pero años despues se utilizaron de una manera constructiva con gran atención para la implementación de novedosos sistemas criptográficos. A continuación se presenta una breve descripción de dos importantes protocolos basados en emparejamientos: un protocolo de acuerdo de clave por Joux y un esquema de criptografía basada en la indentidad por Boneh y Franklin.

2.3.1. Acuerdo de clave entre tres entidades en una sola ronda

Joux sorprendentemente en el año 2000 [45] presentó una solución para lograr un acuerdo de clave entre tres entidades en una sola ronda utilizando emparejamientos bilineales.

El protocolo asume la existencia de tres participantes (entidades) $i \in [0, 1, 2]$, cada parte en forma secreta genera un valor entero x_i y calcula $[x_i]\,P$ y este resultado es

transmitido a las demás partes, entonces cada entidad recibe los elementos $[x_{i-1}]\,P$ y $[x_{i+1}]\,P$ (donde el subíndice de x se considera modulo 3).

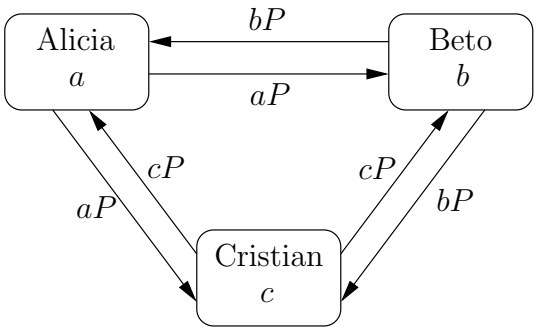


Figura 2.2: Diffie-Hellman entre tres entidades en una sola ronda.

Note que todos los mensajes son independientes y por lo tanto, se puede decir que todas las comunicaciones entre los participantes se producen en una sola ronda. Con esto, cada una de las partes puede establecer una clave secreta compartida al calcular:

$$\begin{aligned} K &= \hat{e}\left([x_{i-1}]\,P, [x_{i+1}]\,P\right)^{x_i} \\ &= \hat{e}\left(P, P\right)^{(x_{i-1})(x_i)(x_{i+1})} \end{aligned}$$

El protocolo de Joux puede ser extendido para n participantes mediante el uso de un mapeo eficiente *multilineal* de la forma $G_1^{n-1} \to G_2$, sin embargo la construcción de un mapeo multilineal que pueda ser calculado de manera eficiente es todavía un problema abierto, de hecho, Boneh y Silverberg en [17] presentan pruebas de que talvez no sea posible construir tales mapeos multilineales.

Este protocolo es útil como un ejemplo de cómo los emparejamientos bilineales se pueden utilizar para proporcionar una solución elegante a protocolos criptográficos que se pensaba eran imposibles.

2.3.2. Criptografía Basada en la Identidad (IBE)

El concepto de criptografía basada en identidad (*IBE* por sus siglas en inglés) fue presentado por Shamir en 1985 en [61]. Se considera que este esquema pertenece a la clase de criptografía asimétrica, y la principal característica de este esquema criptográfico es que permite comunicar a cualquier par de usuarios de forma segura y que entre ellos puedan verificar las firmas digitales generadas sin intercambios de claves públicas o privadas.

El sistema supone la existencia de una entidad de confianza llamada *generador de clave privada* (*PKG* por sus siglas en inglés), cuyo único propósito es proporcionar a cada usuario en el sistema, su clave privada, permitiendo a un usuario firmar y cifrar mensajes que envía a través de un canal de comunicación inseguro y también descifrar y verificar los mensajes que recibe en una forma totalmente independiente.

Este esquema tiene la propiedad de que la clave pública de un usuario es fácilmente derivada en función de su identidad (nombre, correo electrónico, dirección de red, número de seguro social, etc), mientras que la clave privada es calculada por la autoridad de confianza PKG.

Conociendo la identidad de un usuario es posible enviar mensajes que sólo este usuario pueda leer y también podemos verificar las firmas que sólo él puede producir.

2.3.2.1. Criptosistema basado en la identidad

En todos los esquemas de cifrado, un mensaje inteligible m es cifrado con una clave k_e, trasmitido como un mensaje ininteligible c a través de un canal, y es descifrado con una clave k_d. Típicamente la elección de las claves utiliza un algoritmo pseudo-aleatorio para la generación de una semilla K.

En un criptosistema de clave privada, $k_e = k_d = K$, y se tiene un canal por separado que se mantiene en secreto y garantiza la autenticidad de la clave.

En el esquema de clave pública, las claves son derivadas de K por medio de dos diferentes funciones $k_e = f_e(K)$ y $k_d = f_d(K)$, y el canal separado debe de garantizar la autenticidad de la clave.

En el esquema basado en la identidad, la clave de cifrado es la identidad del usuario $k_e = i$, y la clave de descifrado es derivada de i y K ($k_d = f(i, K)$), el canal por separado es completamente eliminado y es remplazado por una simple iteración con el PKG, este esquema se muestra en la Figura 2.3.

2.3.2.2. Firma digital basada en la identidad

Un esquema de firmas digitales basado en la identidad es como un espejo con respecto al criptosistema basado en la identidad.

Para realizar las operaciones de firma y verificación: el mensaje m es firmado con la clave generada k_g (clave privada), el mensaje m es transmitido junto con la firma generada s y ésta es verificada con la clave de verificación k_v (clave pública), este esquema se muestra en la Figura 2.4.

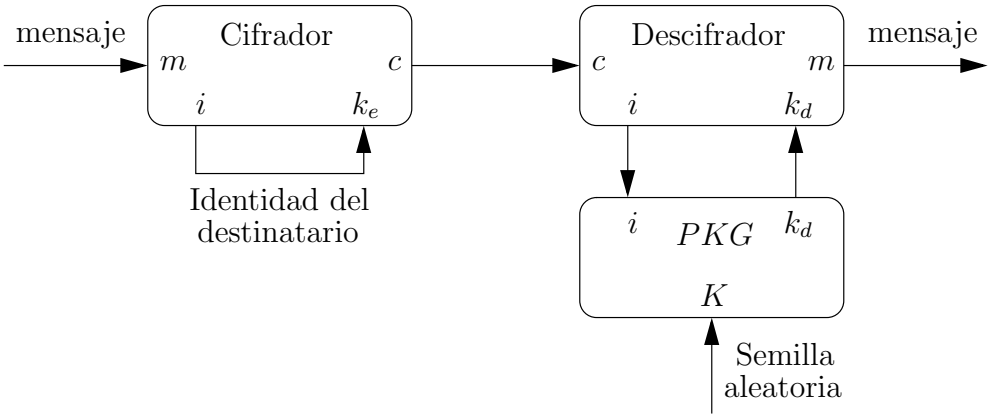


Figura 2.3: Esquema de cifrado basado en la identidad

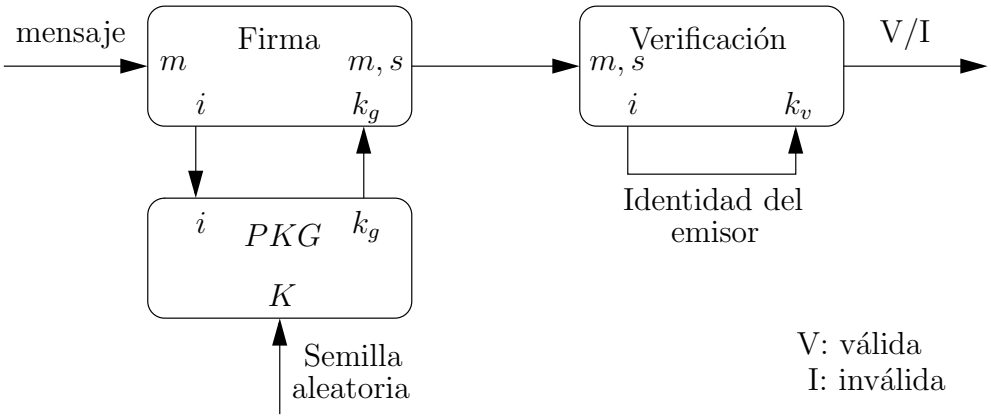


Figura 2.4: Esquema de firmas basado en la identidad

La criptografía basada en la identidad propuesta por Shamir quedó como un problema abierto. Una primer solución basada en el cálculo de residuos cuadráticos fue propuesta en el 2001 por Cocks [19], ese mismo año Boneh y Franklin en [15] propusieron resolver este problema utilizando emparejamientos bilineales, donde los autores describen que la criptografía basada en la identidad es especificada por cuatro algoritmos: *Setup, Extract, Encrypt, Decrypt.*

Setup Se elige un s y se calcula $P_{pub} = sP$. Se eligen dos funciones picadillo, la primera $H_1 : \{0,1\}^* \rightarrow G_1^*$ llamada *map-to-point* y $H_2 : G_2 \rightarrow \{0,1\}^n$, donde n es la longitud del mensaje, la clave maestra es s y la clave pública global es P_{pub}.

Extract Dado un identificador público $ID \in \{0,1\}^*$, calcular la clave pública $Q_{ID} = H_1(ID) \in G_1$ y la clave privada $S_{ID} = sQ_{ID}$.

Encrypt Escoger un r, calcular el mensaje cifrado del mensaje M como:

$$C = \langle rP, M \oplus H_2(g_{ID}^r) \rangle$$

donde: $g_{ID} = \hat{e}\,(Q_{ID}, P_{pub})$

Decrypt Dado $C = \langle U, V \rangle$, calcular

$$M = V \oplus H_2\,(\hat{e}\,(S_{ID}, U))$$

Un año más tarde Hess en [44], presentó un esquema de firma y verificación basado en la identidad retomando el trabajo de Boneh y Franklin. Este esquema nuevamente cuenta con cuatro algoritmos principales pero con pequeñas modificaciones para satisfacer las operaciones de firma y verificación:

Setup Se elige un s y se calcula $P_{pub} = sP$. Se eligen dos funciones picadillo $H_1 : \{0,1\}^* \rightarrow G_1^*$ (*map-to-point*) y $H : \{0,1\}^* \times G_2 \rightarrow Z_q^*$, la clave maestra es s y la clave pública global es P_{pub}.

Extract Dado un identificador público $ID \in \{0,1\}^*$, calcular la clave pública $Q_{ID} = H_1(ID) \in G_1$ y la clave privada $S_{ID} = sQ_{ID}$.

Sign Dada la clave secreta S_{ID} y un mensaje $M \in \{0,1\}^*$, el signatario escoge un arbitrario $P_1 \in G_1^*$ y un aleatorio $k \in Z_q^*$ y se calcula:

$$\begin{aligned} r &= \hat{e}(P_1, P)^k \\ v &= H(M, r) \\ u &= vS_{ID} + kP_1 \end{aligned}$$

la firma está dada por el par (u, v).

Verify Dado Q_{ID}, un mensaje M y la firma (u, v), la verificación se realiza de la siguiente manera:

$$r = \hat{e}(u, P)\hat{e}(Q_{ID}, -P_{pub})^v$$

la firma es válida si y sólo si $v = H(M, r)$.

En los trabajos [15, 44] antes mencionados se utiliza una función hash (H_1) llamada *map-to-point* la cual será descrita en la sección 1.4.1.

2.3.2.3. Seguridad del esquema IBE

La seguridad total de un criptosistema basado en la identidad, depende de los siguientes puntos:

- La seguridad de las funciones criptográficas.
- La información secreta almacenada en los centros generadores de claves.
- La minuciosidad de los controles de identidad realizados por el PKG antes de emitir las claves privadas a los usuarios.
- Las precauciones que toman los usuarios para prevenir la pérdida, duplicación o uso no autorizado de su identidad.

2.3.3. Firmas a ciegas

Una aplicación más de los emparejamientos bilineales se presenta en la construcción de esquemas de firmas a ciegas, en la literatura han aparecido diversos esquemas de firmas a ciegas que hacen uso de los emparejamientos como en [68, 35, 49].

Una de las actividades realizadas en conjunto con este trabajo de tesis, fue el análisis de desempeño de nueve esquemas de firmas a ciegas, donde tres esquemas se basan en la operación de exponenciación modular, tres en la multiplicación escalar en curvas elípticas y los últimos tres en emparejamientos bilineales, como resultado se logró una publicación donde este análisis es presentado, ver [51].

En la Figura 2.5 se presenta un modelo de capas de la arquitectura necesaria para la construcción de aplicaciones criptográficas utilizando emparejamientos. Aquí se hace referencia al emparejamiento η_T sobre campos finitos binarios y ternarios.

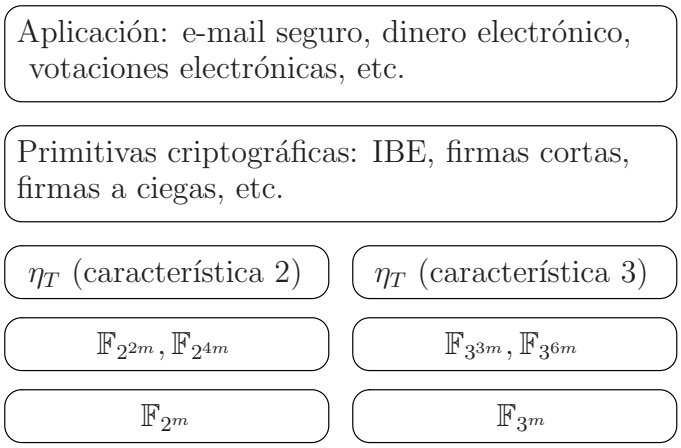


Figura 2.5: Arquitectura de aplicaciones utilizando el emparejamiento η_T.

En la primer capa de la Figura 2.5 se representa la aritmética nencesaria en los campos $\mathbb{F}_{2^m}$ y $\mathbb{F}_{3^m}$. La segunda capa representa la aritmética necesaria hasta construir las extensiones $\mathbb{F}_{2^{4m}}$ y $\mathbb{F}_{3^{6m}}$; la tercer capa representa el emparejamiento η_T en ambas características. En la cuarta capa se encuentran las primitivas criptográficas y finalmente la capa superior representa las aplicaciones.

Como se ha presentado a lo largo de este capítulo, gracias a los emparejamientos bilineales se han generado nuevas y elegantes soluciones criptográficas a problemas difíciles, por lo tanto, es importante investigar la forma de acelerar el cálculo del emparejamiento para poder realizar protocolos y aplicaciones criptográficas más eficientes. En los últimos años, una gran cantidad de trabajos han aparecido en la literatura para hacer frente a este problema, y como resultado se ha avanzado mucho en la implementación de emparejamientos de manera eficiente.

En la siguiente sección titulada "Estado del arte" se revisan algunos trabajos existentes en la literatura con avances en el cálculo eficiente de emparejamientos.

2.4. Estado del arte

En [47], se realizó una implementación del emparejamiento de Tate en un dispositivo móvil (celular FOMA SH90liS), sobre campos finitos de característica tres de grados $m = \{97, 167, 193, 239\}$, utilizando Java como lenguaje de programación. Los autores reportan que su implementación optimizada para $m = 97$ alcanzó cerca de 0.5 segundos para el cálculo del emparejamiento de Tate, además también muestran una comparación de tiempos contra otros criptosistemas como se muestra en las siguiente tablas:

Operación	$\mathbb{F}_{3^{97}}$ *optimizado*	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{167}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$
Suma	0.0173	0.0171	0.0202	0.0203	0.0198
Resta	0.0196	0.0193	0.0225	0.0232	0.0210
Multiplicación	0.2400	0.2897	0.6651	0.8638	1.1891
Cubo	0.0473	0.0886	0.1149	0.1254	0.1362
Inversión	1.5288	1.5411	3.7500	5.0203	6.6621
Raíz cúbica	0.2982	0.3701	0.5112	0.6094	0.7941
Emparejamiento de Tate	509.22	627.65	1724.93	2368.58	3557.42

Tabla 2.2: Tiempos de cálculo del emparejamiento de Tate en diferentes campos finitos (mseg), obtenidos en [47].

La columna llamada "$\mathbb{F}_{3^{97}}$ *optimizado*" de la Figura 2.2 se refiere a una implementación en la que los autores realizaron una optimización para las operaciones aritméticas y por consecuencia en el cálculo del emparejamiento.

Operación	FOMA SH90liS	Pentium M
Emparejamiento η_T en $\mathbb{F}_{3^{97}}$	509.22	10.15
Exponenciación Modular de 1024-bit RSA	4238.40	75.07
Multiplicación Escalar de ECC sobre $\mathbb{F}_{2^{163}}$	13777.50	116.83

Tabla 2.3: Comparación de tiempos con otros criptosistemas (mseg), obtenidos en [47].

Los autores señalan que el teléfono móvil que utilizaron no es el modelo más avanzado, por lo que indican que la velocidad de procesamiento de los siguientes modelos debe de ser mayor. La conclusión general es que los criptosistemas basados en emparejamientos pueden ser eficientemente implementados en teléfonos móviles usando Java.

En el artículo de Beuchat et al. [10], se describe una arquitectura para un coprocesador de emparejamiento η_T, que está en su mayoría basada en un acelerador hardware. Éste consiste de un único elemento de procesamiento, de varios registros y una unidad de control que contiene una máquina de estados finitos (FSM) y una memoria de instrucciones (ROM).

En el trabajo [4], se presenta una implementación de aritmética $\mathbb{F}_{3^m}$ en software. La implementación fue realizada en un máquina con un procesador Pentium 4 de 32 bits a 2.4 GHz, se usó como lenguaje de programación C, en particular se utilizó el compilador GNU C Linux/x86.

	Suma	Mult	a^3	$a^{1/3}$	Inv(Exp)	Inv.(EEA)
$\mathbb{F}_{3^{239}} = \mathbb{F}_3[x]/(x^{239} + x^{24} - 1)$						
gcc	0.05	5.0	0.32	1.6	137^d	55
$\mathbb{F}_{3^{509}} = \mathbb{F}_3[x]/(x^{509} - x^{477} + x^{445} + x^{32} - 1)$						
gcc	0.09	15.5	0.70	2.5	575^g	213
$\mathbb{F}_{3^{1223}} = \mathbb{F}_3[x]/(x^{1223} + x^{255} + 1)$						
gcc	0.06	17.9	-	-	-	-

Tabla 2.4: Tiempo (μs) con un procesador Pentium 4 a 2.4 GHz, resultados en [4]. ^{d} Cadena de adición con 12 multiplicaciones, ^{g} Cadena de adición con 14 multiplicaciones.

En el trabajo [57], se argumenta que la criptografía basada en identidad es ideal para las redes de sensores inalámbricas y viceversa, los autores presentan resultados del cálculo del emparejamiento de Tate sobre nodos con recursos limitados. Su implementación está realizada sobre un nodo MICAz que cuenta con un micro-controlador ATmega128 (8-bit/7.38 MHz, 4KB SRAM, 128KB de memoria flash). Este trabajo utiliza el algoritmo de Miller para el cálculo del emparejamiento, donde los autores consideraron los siguientes parámetros:

1. El emparejamiento de Tate en curvas elípticas sobre campos con un primo grande.

2. El grado de seguridad (*embedding degree*) $k = 2$, q es un primo de $256 - bits$, y l es de $128 - bits$.

3. Utilizar coordenadas proyectivas sobre la curva $E/\mathbb{F}_q : y^2 = x^3 + x$.

El resultado obtenido de esta implementación es el siguiente:

Tiempo(seg)	RAM (bytes)	ROM (bytes)
30.21	1.831	18.384

Tabla 2.5: Costo del emparejamiento de Tate en MICAz, resultados en [57].

En el trabajo de Harrison et al. [42], se examinan tres formas de implementar operaciones básicas en característica tres $\mathbb{F}_{3^{97}}$, para ser usadas en criptosistemas basados en el emparejamiento de Tate. La implementación se realizó en una computadora Sparc Ultra 10. Los tiempos que calcularon se muestran en la Tabla 2.6:

	Suma	Multiplicación	a^3	$a^{1/3}$	Inversión
$\mathbb{F}_{3^{97}}$-N	2.7	830	37	419	1200
$\mathbb{F}_{3^{97}}$-I-N	1.10	84	4	49	680
$\mathbb{F}_{3^{97}}$-II-N	0.20	23	10 o 1.5	16	240

Tabla 2.6: Tiempo en μs de las operaciones en el campo, obtenidos en [42].

donde:

- $\mathbb{F}_{3^{97}}$-N: corresponde a una implementación utilizando la técnica estándar para representar cada elemento de 3^{97} con un arreglo de 97 enteros.

- $\mathbb{F}_{3^{97}}$-I-N: corresponde a una implementación utilizando una multiplicación sencilla (*naive multiplication*).

- $\mathbb{F}_{3^{97}}$-II-N: corresponde a una implementación donde la suma se realiza palabra por palabra y la multiplicación en forma serial.

Para realizar cálculos en criptosistemas basados en el emparejamiento de Tate, no sólo se necesita realizar operaciones en el campo $\mathbb{F}_{3^p}$ si no que también en su extensión $\mathbb{F}_{3^{6p}}$, donde los autores obtuvieron los siguientes resultados:

	Suma	Multiplicación	a^3	$a^{1/3}$	Inversión
$\mathbb{F}_{3^{97}}$-N	17	7772	281	5674	117000
$\mathbb{F}_{3^{97}}$-I-N	7	902	49	868	18000
$\mathbb{F}_{3^{97}}$-II-N	1	319	24	323	6000

Tabla 2.7: Tiempo en μs de las operaciones en el campo, resultados en [42].

Finalmente para el cálculo del emparejamiento de Tate obtuvieron los siguientes resultados:

Operación	$\mathbb{F}_{3^{97}}$
Emparejamiento de Tate	63

Tabla 2.8: Tiempo en mili-segundos del cálculo del emparejamiento de Tate, resultados en [42].

En [63], los autores describen que existen tres esquemas para el cálculo del emparejamiento de Tate, con curvas elípticas cúbicas, curvas elípticas binarias y curvas hiperelípticas binarias. Ellos implementaron el emparejamiento de Tate con curvas elípticas binarias porque los algoritmos son más sencillos y la aritmética binaria es más eficiente, su propuesta es una arquitectura en FPGA sobre los campos binarios $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{283}}$. Los autores indican que la latencia para el emparejamiento de Tate ha sido reducida y que su implementación se ejecuta entre 10 a 20 veces más rápido que las implementaciones equivalentes de otros esquemas basados en emparejamientos con el mismo nivel de seguridad.

En este mismo trabajo los autores implementaron dos algoritmos en software para generar vectores de prueba para el cálculo de emparejamientos con la finalidad de ser usados en su implementación en FPGA, usaron una biblioteca llamada LiDIA escrita en C++ para la aritmética y ellos implementaron las operaciones de alto nivel, los códigos fueron compilados con g++ 3.0.4 en una estación de trabajo Xeon a 2.8 Ghz, en la siguiente Tabla 2.9 se muestran los resultados obtenidos:

$\mathbb{F}_{2^m}$	Latencia (Alg. 1)	Latencia (Alg. 2)	Alg. 2 vs. Alg. 1
$\mathbb{F}_{2^{239}}$	10.8	7.5	1.44
$\mathbb{F}_{2^{283}}$	18.1	12.2	1.48

Tabla 2.9: Tiempo en mili-segundos del cálculo del emparejamiento de Tate por 2 algoritmos, resultados en [63].

La columna de más a la derecha de la Tabla 2.9, representa el factor de aceleración existente entre los tiempos del algoritmo 1 (Alg. 1) y el algoritmo 2 (Alg. 2), donde Alg. 1 es una implementación completamente secuencial para el cálculo del emparejamiento de Tate, mientras que Alg. 2 representa una implementación paralela para realizar este

cálculo.

En un trabajo realizado por Mitsunari Shigeo, se presenta el cálculo del emparejamiento η_T en característica tres, $\mathbb{F}_{3^{97}}$, el autor realizó una implementación eficiente en software para realizar el cálculo del emparejamiento en forma paralela en un procesador con 2 núcleos (procesador Intel Core 2 Duo 2.6 GHz), además de utilizar la tecnología SSE3 (para más detalles sobre SSE ver el apéndice B). El lenguaje de programación utilizado es C++. En la Tabla 2.10 se presentan los tiempos para el cálculo del emparejamiento que el autor reporta en su forma serial y paralela.

Operación	Serial	Paralela
Emparejamiento η_T	149 $\mu s.$	92 $\mu s.$

Tabla 2.10: Emparejamiento η_T en característica tres

Este trabajo no se encuentra públicado, pero se encuentra disponible en `http://homepage1.nifty.com/herumi/crypt/pairing.html`.

Darrel Hankerson, Alfred Menezes y Michael Scott, presentan un borrador de un capítulo de un libro que actualmente los autores tienen en desarrollo. El capítulo de este libro es titulado "Software Implementation of Pairings" y se encuentra disponible en `http://www.math.uwaterloo.ca/~ajmeneze/publications/pairings_software.pdf`. Los autores reportan el número de ciclos para el cálculo del emparejamiento sobre arquitecturas, la primera con un procesador Pentium 4 de 32 bits y la segunda cuenta con un procesador Opteron de 64 bits, la tercera es un procesador Core 2 Duo, y finalmente una UltraSPARC III. En la Tabla 2.11 se presentan algunos resultados interesantes, para el cálculo del emparejamiento η_T.

Campos	Pentium 4		Core 2 Duo	
	PG	SSE	PG	SSE
$\mathbb{F}_{2^{1223}}$	201×10^6	81×10^6	48×10^6	39×10^6
$\mathbb{F}_{3^{509}}$	139×10^6	86×10^6	46×10^6	33×10^6

Tabla 2.11: Número de ciclos para el cálculo del emparejamientos η_T.

Es necesario mencionar que en la Tabla 2.11 la columna PG corresponde a una implementación utilizando los registros de propósito general del procesador, mientras que la columna SSE corresponde a una implementación utilizando los registros de SSE que permiten realizar operaciones de forma vectorizada (128 bits). También hay que mencionar que los autores presentan otros resultados que pueden ser revisados en `http://`

www.math.uwaterloo.ca/~ajmeneze/publications/pairings_software.pdf, si embargo, los resultados aquí presentados resultan de mayor interés en este trabajo de tesis.

3

ARITMÉTICA DEL EMPAREJAMIENTO η_T

Si he conseguido ver más lejos, es porque me he apoyado en hombros de gigantes.

Isaac Newton (1642 – 1727)

Para el cálculo del emparejamiento η_T que estamos estudiando se realizan una gran cantidad de operaciones aritméticas como son sumas, multiplicaciones, exponenciación, inversión, entre otras, sin embargo algunas operaciones como la suma se realizan sin costo considerable de cómputo, ya que la suma puede ser reemplazada por operaciones estándar en cualquier lenguaje de programación (como la operación xor). La operación, de mayor costo y a la que consecuentemente se le dedica más atención es la multiplicación, ya que de ésta dependen en su mayoría las operaciones que se realizan para el cálculo del emparejamiento, por lo tanto resulta crucial buscar la mejor forma de implementarla.

Diversos trabajos en la literatura demuestran que los campos con los que se realiza una mejor implementación es sobre campos $\mathbb{Z}_p$ y campos finitos de característica dos, $\mathbb{F}_{2^m}$. En los últimos años los campos finitos de característica tres, $\mathbb{F}_{3^m}$, han sido de gran interés para la implementación de emparejamientos y se han presentado numerosas aportaciones en la literatura abierta utilizando este tipo de campos como se muestra en el capítulo anterior.

Una de las partes más importantes para la implementación de operaciones aritméticas eficientes es la forma de diseñar los algoritmos, más concretamente en la manera en que se realicen las operaciones sobre el campo. Se deben buscar los métodos más efectivos y para esto hay que tomar en cuenta la representación de los elementos del campo.

Para la representación de elementos en un campo finito podemos encontrar dos tipos de base que son popularmente conocidas, las bases polinomiales y las bases normales. Por definición una base polinomial es un conjunto de la forma $\{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{m-1}$ donde $\alpha \in \mathbb{F}_{p^m}$, todo elemento de $\mathbb{F}_{p^m}$ es combinación lineal de la base y los elementos de la base son linealmente independientes. El segundo tipo de base son las llamadas bases normales, donde una base B es normal de $\mathbb{F}_{p^m}$, si la base es un conjunto de la forma $B = \{\beta, \beta^p, \beta^{p^2}, \ldots, \beta^{p^{m-1}}\}$, en esta representación el proceso de la multiplicación es sensiblemente más complicado.

En este trabajo la base utilizada para representar los elementos es la base polinomial tanto en característica dos, $\mathbb{F}_{2^m}$, como en característica tres, $\mathbb{F}_{3^m}$. En las secciones siguientes se describen aspectos de la infraestructura utilizada y cuestiones de implementación de la aritmética necesaria para la contrucción eficiente del emparejamiento η_T.

3.1. Infraestructura

Este trabajo tiene como objetivo principal el desarrollo de una implementación en software eficiente para el cálculo del emparejamiento η_T sobre dos arquitecturas diferentes: la primera de ellas es un dispositivo móvil (PDA, del inglés *Personal digital assistant (Asistente digital personal)*) con recursos limitados y la segunda una computadora con arquitectura multinúcleo. Para lograrlo se diseño, desarrolló e implementó una biblioteca para realizar el cálculo de manera eficiente, a continuación se da una descripción las arquitecturas utilizadas.

3.1.1. Infraestructura para la implementación en PDA

La implementación sobre la PDA, también se ejecuto sobre una computadora Laptop. Tanto las características de la Laptop y la PDA se listan a continuación.

1. Hardware
   - PDA sharp Zaurus-5600
     - Procesador Intel XScale 400 MHz PXA250
     - Memoria ROM de 32 Mb
     - Memoria RAM de 64 Mb
     - Pantalla TFT LCD de 3.5"(240x320 píxeles)
     - Sistema operativo Linux2 (Embedix3)
   - Computadora Laptop:

- Procesador Intel Dual Core 2.0 Ghz
- Memoria RAM 2 Gb.
- Sistema operativo Gentoo Linux (32 bits)

2. Software

   - Lenguaje de programación C/C++
   - Compilador GNU/Linux
   - Lenguaje Maple (para la generación de vectores de prueba)

3.1.2. Infraestructura para la implementación en computadora multinúcleo

1. Hardware

   - Servidor
     - Procesador Intel®Xeon®CPU 2.5 Ghz (HTN - 8 procesadores).
     - Memoria RAM de 4 Gb
     - Sistema operativo Windows XP SP. 3 (32 bits)
     - Sistema operativo Windows XP SP. 2 (64 bits)
     - Sistema operativo Red Hat Enterprise Linux 5.1 (64 bits)

2. Software

   - Lenguaje de programación C/C++ y Ensamblador (ASM)
   - Compilador Intel® C++ 10.1.021 (IA-32, Intel(R) 64)
   - Streaming SIMD Extensions (SSE)
   - Intel® Vtune™ Performance Analyzer
   - Intel® Thread Profiler 3.1
   - Intel® Thread Checker 3.1
   - Intel® Threading Building Blocks 2.0 (TBB)
   - Lenguaje Maple (para la generación de vectores de prueba)

3.2. Consideraciones para característica dos

Debido a que se trata de dos arquitecturas con recursos y capacidades de cómputo diferentes, la implementación de las operaciones básicas y por consecuencia del emparejamiento η_T se realiza sobre diferentes tamaños de campos finitos. Es importante señalar que la curva supersingular utilizada es la misma:

Curva elíptica supersingular: $E(\mathbb{F}_{2^m}): \ y^2 + y = x^3 + x + b \quad (b \in \{0, 1\})$
Grado de seguridad (*embedding degree*): $k = 4$

Los campos utilizados para cada arquitectura tienen las siguientes restricciones.

3.2.1. Restricciones en PDA

El campo finito utilizado y el polinomio irreducible para la implementación en la PDA es:

Campo finito: $GF(2^{233})$
Polinomio irreducible: $P(x) = x^{233} + x^{74} + 1$

3.2.2. Restricciones en multinúcleo

El campo finito utilizado para la implementación en la computadora multinúcleo es:

Campo finito: $GF(2^{503})$
Polinomio irreducible: $P(x) = x^{503} + x^3 + 1$

Como se muestra anteriormente en ambos campos finitos las operaciones se realizan bajo un polinomio irreducible que resulta ser un trinomio, debido a que este nos ofrece un mejor desempeño en la operación de reducción modular como se explicara más adelante.

3.3. Implementación en característica dos

Por razones de implementación tanto en hardware como en software, los campos preferidos son extensiones de $\mathbb{F}_2$, ya que los elementos del campo se pueden ver como cadenas de bits.

3.3.1. Operaciones en $\mathbb{F}_{2^m}$

Los elementos que pertenecen al campo finito de característica dos $\mathbb{F}_{2^m}$ son polinomios de grado a lo más $m - 1$, con coeficientes a_i en $\mathbb{F}_2$:

$$A(x) = \sum_{i=0}^{m-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}$$

y las operaciones que se realizan en el campo son operaciones modulares bajo un polinomio irreducible de grado m.

3.3.1.1. Suma

La operación de suma y resta en campos finitos de característica dos son operaciones idénticas, dado que el inverso aditivo de un elemento en el campo, es el mismo elemento, y la suma y resta se realiza bit a bit con una operacion xor ($\oplus$), ver la Tabla 1.1.

Así pues, la suma de dos elementos $A, B \in \mathbb{F}_{2^m}$ es realizada sumando los coeficientes polinomiales en $\mathbb{F}_2$ como sigue:

$$A \pm B = \sum_{i=0}^{m-1} a_i x^i \pm \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{m-1} (a_i \pm b_i) x^i$$

Esta operación es por naturaleza muy sencilla y es implementada en software a nivel palabra. En la PDA se utilizó una palabra de 32 bits y la suma se realiza palabra por palabra. En la arquitectura multinúcleo el tamaño de palabra básico nuevamente es de 32 bits, pero aquí se utilizó la tecnología SSE (*Streaming SIMD Extensions*) para poder empaquetar 4 palabras de 32 bits y asi formar un vector de 128 bits y procesar esta cantidad de información al mismo tiempo como se muestra en la Figura 3.1 (para más detalles sobre SSE ver el apéndice B).

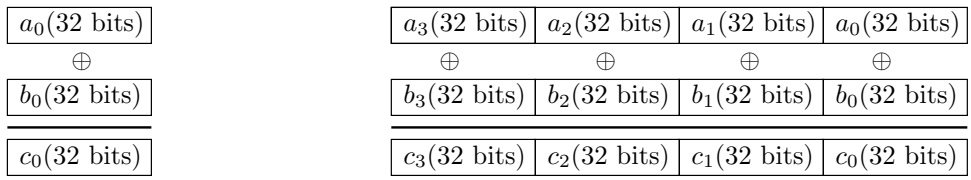


Figura 3.1: Suma en las arquitecturas PDA y multinúcleo

3.3.1.2. Cuadrado

La operación de elevar al cuadrado un elemento que pertenece al campo $\mathbb{F}_{2^m}$ es considerada en característica 2 una operación lineal, pues sólo consiste en intercalar un cero entre cada bit del elemento original como se describe por la siguiente ecuación.

$$A(x) = \sum_{j=0}^{m-1} a_j x^j \Longrightarrow (A(x))^2 = \sum_{j=0}^{m-1} a_j \alpha^{2j}$$

La mejor implementación para realizar esta operación de manera eficiente es utilizando una tabla de consulta para intercalar los ceros necesarios, la tabla recibe como entrada una cadena de bits y regresa la expansión de la cadena, al finalizar esta operación obtenemos un polinomio fuera del campo de grado $2m - 2$, por esta razón, es

necesario realizar una reducción modular, ver sección 3.3.1.4, para obtener nuevamente un elemento en el campo.

La principal diferencia entre la implementación en la arquitectura de la PDA y la arquitectura multinúcleo es el tamaño de la tabla de consulta, en la PDA se construyó una tabla de consulta con 16 entradas permitiéndonos manejar 4 bits por cada consulta, mientras que en la arquitectura multinúcleo la tabla cuenta con 256 entradas para poder manipular hasta 8 bits por consulta.

3.3.1.3. Multiplicación

La operación de multiplicación sobre campos $\mathbb{F}_{2^m}$ es implementada de manera eficiente utilizando una combinación entre el algoritmo propuesto por Karatsuba y Ofman [46], y el método Comb [50].

Karatsuba y Ofman propusieron reducir la complejidad para realizar una multiplicación de números grandes, hasta el día de hoy este algoritmo sigue siendo muy eficiente y su complejidad es subcuadrática $O(m^{log_2 3})$, el análisis realizado por los autores es el siguiente, donde los elementos pueden ser representados como sigue:

$$\begin{aligned} A(x) &= \sum_{i=0}^{m-1} a_i x^i = \sum_{i=m/2}^{m-1} a_i x^i + \sum_{i=0}^{\lfloor m/2 \rfloor - 1} a_i x^i \\ A(x) &= x^{\lfloor m/2 \rfloor} \sum_{i=0}^{(m/2)-1} a_{i+(m/2)} x^i + \sum_{i=0}^{\lfloor m/2 \rfloor - 1} a_i x^i \end{aligned}$$

Ahora en forma compacta podemos representar dos elementos:

$$\begin{aligned} A(x) &= x^{\lfloor m/2 \rfloor} A^H + A^L \\ B(x) &= x^{\lfloor m/2 \rfloor} B^H + B^L \end{aligned}$$

y el producto polinomial está dado por 4 multiplicaciones como se muestra a continuación:

$$C(x) = A^H B^H x^m + \left(A^H B^L + A^L B^H\right) x^{m/2} + A^L B^L$$

la idea fundamental del algoritmo Karatsuba-Ofman se basa en reescribir el producto polinomial de la siguiente forma:

$$C(x) = A^H B^H x^m + \left(\left(A^H + A^L\right)\left(B^H + B^L\right) - A^H B^H - A^L B^L\right) x^{m/2} + A^L B^L$$

como se puede ver esta fórmula sólo necesita el cómputo de 3 multiplicaciones en el campo entre polinomios de grado $(m/2) - 1$, es decir, de aproximadamente $m/2$ bits, el algoritmo 3.1 corresponde al método de Karatsuba-Ofman que fue programado.

Algoritmo 3.1: Multiplicación polinomial en $\mathbb{F}_{2^m}$.

Entrada: $A, B \in \mathbb{F}_{2^m}$
Salida: $C = AB$

```
1  if r==1 then
2      C = mul_n(A, B);
3      return;
4  end
5  for i ← 0 to r/2 − 1 do
6      M_{A_i} = A_i^L + A_i^H;
7      M_{B_i} = B_i^L + B_i^H;
8  end
9  mul2^k(C^L, A^L, B^L);
10 mul2^k(M, M_A, M_B);
11 mul2^k(C^H, A^H, B^H);
12 for i ← 0 to r − 1 do
13     M_i = M_i + C_i^L + C_i^H;
14 end
15 for i ← 0 to r − 1 do
16     C_{r/2+i} = C_{r/2+i} + M_i;
17 end
```

3.3.1.4. Reducción modular

La reducción modular es la operación que se encarga de regresar nuevamente al campo a los elementos resultantes de otra operación como la multiplicación o la elevación al cuadrado.

Normalmente se elige que el polinomo irreducible sea un trinomio de la forma $x^m + x^k + 1$ pues implica que se realicen menos sumas y la reducción sea mas rápida. La reducción de $c(x)$ módulo $p(x)$ puede ser realizada eliminando el bit más significativo y asi sucesivamente hasta obtener un elemento nuevamente dentro del campo.

Esta operación puede ser realizada de manera más eficiente reduciendo una palabra o más al mismo tiempo. Por ejemplo para el campo que se esta utilizando en la PDA que es $\mathbb{F}_{2^{233}}$, y su polinomio irreducible es:

$$p(x) = x^{233} + x^{74} + 1$$

tenemos que:

$$
\begin{aligned}
x^{233} &\equiv x^{74} + 1 \bmod p(x) \\
x^{234} &\equiv x^{75} + x \bmod p(x) \\
&\vdots \\
x^{392} &\equiv x^{233} + x^{159} \equiv x^{159} + x^{74} + 1 \bmod p(x) \\
&\vdots \\
x^{463} &\equiv x^{304} + x^{230} \equiv x^{230} + x^{145} + x^{71} \bmod p(x) \\
x^{464} &\equiv x^{305} + x^{231} \equiv x^{231} + x^{146} + x^{72} \bmod p(x)
\end{aligned}
$$

con esto logramos una reducción más eficiente, en la Figura 3.2 se presenta un modelo de cajas que representa el comportamiento en general para la reducción utilizando un trinomio como los que utilizamos en este trabajo.

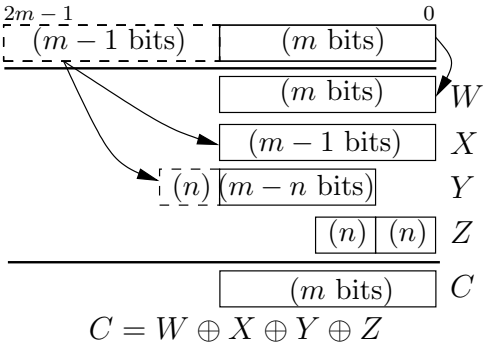


Figura 3.2: Reducción modular para un trinomio $x^m + x^n + 1$.

3.3.1.5. Raíz cuadrada

La raíz cuadrada es una operación que sólo se implementó en la arquitectura multinúcleo pues se pretende explotar sus capacidades de cómputo como la paralelización de tareas, para lograrlo se programó una versión del emparejamiento η_T que utiliza esta operación pero que tiene la característica de poder ser paralelizable, esta versión de emparejamiento es presentada en el siguiente capítulo.

La raíz cuadrada es una operación compleja pero recientemente se publicó un método para calcular la raíz cuadrada de manera eficiente con sólo realizar sumas como se presenta en los trabajos [29, 60]. Este método está basado en el teorema Petit de Fermat que establece que $a^{p-1} \equiv 1 \bmod p$. Entonces la raíz cuadrada puede ser calculada como $\sqrt{a} = a^{2^{m-1}}$, que requiere de $m-1$ operaciones de elevar al cuadrado, pero el método eficiente consiste en expresar la raíz en términos de la raíz cuadrada de x, es decir:

Sea $a = \sum a_i x^{i}_{i=0}{}^{m-1} \in \mathbb{F}_{2^m}, a_i \in \{0,1\}$, podemos expresar la raíz cuadrada como

sigue:

$$\sqrt{a} = \left(\sum_{i=0}^{m-1} a_i x^i\right)^{2^{m-1}} = \sum_{i=0}^{m-1} a_i (x^{2^{m-1}})^i$$

y dividiendo a en potencias pares e impares tenemos:

$$\begin{aligned}
\sqrt{a} &= \sum_{i=0}^{(m-1)/2} a_{2i}(x^{2^{m-1}})^{2i} + \sum_{i=0}^{(m-3)/2} a_{2i+1}(x^{2^{m-1}})^{2i+1} \\
&= \sum_{i=0}^{(m-1)/2} a_{2i}x^i + \sum_{i=0}^{(m-3)/2} a_{2i+1}x^{2^{m-1}}x^i \\
&= \sum_{i \text{ par}} a_{2i}x^{\frac{i}{2}} + \sqrt{x}\sum_{i \text{ impar}} a_{2i+1}x^{\frac{i-1}{2}}
\end{aligned}$$

esta tarea consiste en construir un vector de los coeficiente pares $a_{par} = (a_{m-1}, \ldots, a_4, a_2, a_0)$ y el vector de impares $a_{impar} = (a_{m-2}, \ldots, a_5, a_3, a_1)$ de aproximadamente $m/2$ bits cada uno y asumiendo que m es impar, entonces para obtener la raíz cuadrada sólo se tiene que precalcular el valor de $\sqrt{x}$ para ser multiplicado por a_{impar} y sumar a_{par} seguido de una reducción por si el elemento queda fuera del campo.

Para calcular el valor de $\sqrt{x}$ es necesario utilizar el polinomio irreducible por ejemplo un trinomio como los que utilizamos en este trabajo $p(x) = x^m + x^k + 1$ y considerando que k es impar observemos que $1 \equiv x^m + x^k \bmod p(x)$ y multiplicando por x y obteniendo la raíz tenemos que:

$$\sqrt{x} \equiv x^{\frac{m+1}{2}} + x^{\frac{k+1}{2}} \bmod p(x)$$

observemos que el producto $\sqrt{x} \cdot a_{impar}$ requiere sólo dos corrimientos a la izquierda, es por esta razón que la raíz cuadrada es considerada una operación lineal.

En nuestra implementación utilizamos el trinomio irreducible $p(x) = x^{503} + x^3 + 1$ entonces obtenemos que $\sqrt{x} \equiv x^{252} + x^2$ y para obtener los vectores par e impar se utilizan dos tablas de consulta con 8 bits de entrada para extraer los coeficiente correpondientes para cada vector, la extracción se realiza palabra por palabra. El algoritmo 3.2 es utilizado para el cálculo de la raíz cuadrada.

Algoritmo 3.2: Raíz cuadrada en $\mathbb{F}_{2^m}$

Entrada: $A(x) = a_{m-1}x^{m-1} + \ldots + a_2x^2 + a_1x + a_0 \in \mathbb{F}_{2^m}$ y el polinomio irreducible $P(x)$

Salida: $R(x) = \sqrt{A(x)} \in \mathbb{F}_{2^m}$

1 $a_{par} \leftarrow (a_{m-1}x^{\frac{m-1}{2}} + \ldots + a_4x^2 + a_2x + a_0)$;
2 $a_{impar} \leftarrow (a_{m-2}x^{\frac{m-3}{2}} + \ldots + a_5x^2 + a_3x + a_1)$;
3 $R \leftarrow a_{par} + \sqrt{x} \cdot a_{impar}$;
4 $R \leftarrow R \bmod P(x)$;
5 **return** R

En la Figura 3.3 se presenta de forma gráfica el cálculo de raíz cuadrada utilizada en nuestra implementación, considerando que el campo es $\mathbb{F}_{2^{503}}$ con el polinomio irreducible es $p(x) = x^{503} + +x^3 + 1$ y el valor precalculado de $\sqrt{x} = x^{252} + x^2$:

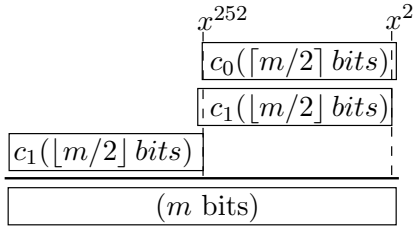


Figura 3.3: Raíz cuadrada: $c_0 + \sqrt{x}c_1$.

Como se puede obervar el paso 4 (la reducción) del algoritmo 3.2 no es necesario pues el elemento resultante de sumar los vectores se encuentra dentro del campo, es decir, es un polinomio de 503 bits y es de grado a lo más 502.

3.3.1.6. Inversión

Para obtener el inverso multiplicativo de un elemento en el campo $\mathbb{F}_{2^m}$ se utilizó el algoritmo de casi inverso (*AIA - Almost Iverse Algorithm*) que es una variante del algoritmo extendido de Euclides (*EEA* por sus siglas en inglés) como se muestra en [59, 39].

El algoritmo 3.3 se presenta la inversión completa, pero hay que mencionar que hasta finalizar el paso 11 se tiene el casi inverso, es decir, $B(x)A(x) \equiv x^k \bmod P(x)$ y para obtener el inverso tenemos que aplicar, $(A(x))^{-1} \equiv B(x)x^{-k} \bmod P(x)$ que se obtiene en el paso 12.

Algoritmo 3.3: Inversión en $\mathbb{F}_{2^m}$

Entrada: $A(x) \in \mathbb{F}_{2^m}$ y el polinomio irreducible $P(x)$
Salida: $B = (A(x))^{-1} \in \mathbb{F}_{2^m}$

1 $b \leftarrow 1$; $c \leftarrow 0$; $u \leftarrow A(x)$; $v \leftarrow P(x)$; $k \leftarrow 0$;
2 **repeat**
3 **while** x *divide* u **do**
4 $u \leftarrow u/x$; $c \leftarrow cx$; $k \leftarrow k + 1$;
5 **end**
6 **if** $deg(u) < deg(v)$ **then**
7 $u \leftrightarrow v$; $b \leftrightarrow c$;
8 **end**
9 **if** $u = 1$ **then** break;
10 $u \leftarrow u + v$; $b \leftarrow b + c$;
11 **until** $u = 1$;
12 $B \leftarrow bx^{-k} \bmod P(x)$;
13 **return** B

Este algoritmo elimina los bits más significativos de u y es por eso que la condición $deg(u) < deg(v)$ se satisface rápidamente, por esta razón el algoritmo toma un menor número de iteraciones que el EEA.

La operación en el paso 12 es una división que es realizada sólo con sumar el polinomio irreducible $P(x)$ con b si es que el bit menos significativo de b está encendido, en caso contrario la suma no es necesaria, después se procede solamente con un corrimiento a la derecha y con esto obtenemos b/x, este proceso es repetido hasta obtener $b/x^k = bx^{-k} \bmod P(x)$ que es el inverso multiplicativo buscado, para más detalles de este algoritmo revisar [59].

3.3.2. Operaciones en $\mathbb{F}_{2^{2m}}$

Es necesario la construcción de una torre de campos finitos (*tower fields*) para el cálculo del emparejamiento η_T, en esta sección se describen las operaciones implementadas como parte de la biblioteca para lograrlo.

La primera torre de campo necesaria en característica dos es $\mathbb{F}_{2^{2m}}$, para su construcción se necesita una base para representar los elementos en ese campo, la cual es: $\{1, s\}$, es decir, aquí los elementos son una combinación lineal de la base con coeficientes en el campo $\mathbb{F}_{2^m}$.

$$U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}, \text{ donde } u_0, u_1 \in \mathbb{F}_{2^m}$$

La torre de campo finito puede ser expresado como:

$$\mathbb{F}_{2^{2m}} \cong \mathbb{F}_{2^m}[s]/(s^2 + s + 1)$$

también hay que mencionar una propiedad de la base: $s^2 = s+1$, la cual es muy utilizada para desarrollar operaciones como el cuadrado y la multiplicación.

Las operaciones necesarias en esta torre de campo son la suma, cuadrado, multiplicación e inversión, y son descritas a continuación.

3.3.2.1. Suma

La suma es nuevamente la operación más sencilla, la forma de realizarla es sumar coeficiente a coeficiente, utilizando las operaciones implementadas en el campo $\mathbb{F}_{2^m}$. Sea $U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}$ y $V = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$, tenemos que:

$$\begin{aligned} U + V &= (u_0 + u_1 s) + (v_0 + v_1 s) \\ &= (u_0 + v_0) + (u_1 + v_1)s \end{aligned}$$

Algoritmo 3.4: Suma en $\mathbb{F}_{2^{2m}}$

Entrada: $U = u_0 + u_1 s, V = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$
Salida: $W = U + V \in \mathbb{F}_{2^{2m}}$

1 $w_0 \leftarrow u_0 + v_0$;
2 $w_1 \leftarrow u_1 + v_1$;
3 **return** $w_0 + w_1 s$

3.3.2.2. Cuadrado

Para calcular el cuadrado de un elemento en $\mathbb{F}_{2^{2m}}$ es necesario desarrollar el cuadado de la siguiente forma:

$$(u_0 + u_1 s)^2 = u_0^2 + u_1^2 s^2$$

pero podemos observar que el elemento resultante no pertenece a $\mathbb{F}_{2^{2m}}$ desde que uno de los coeficientes multiplica a s^2 y éste no es un elemento de la base, para esto es necesario aplicar la propiedad $s^2 = s + 1$ y entonces obtenemos:

$$(u_0 + u_1 s)^2 = (u_0^2 + u_1^2) + u_1^2 s$$

El algoritmo 3.5 forma parte de la biblioteca y es el correspondiente para calcular el cuadrado de un elemento en $\mathbb{F}_{2^{2m}}$ de manera eficiente.

Algoritmo 3.5: Cuadrado en $\mathbb{F}_{2^{2m}}$

Entrada: $U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}$
Salida: $V = U^2 \in \mathbb{F}_{2^{2m}}$

1 $v_0 \leftarrow u_0^2$;
2 $v_1 \leftarrow u_1^2$;
3 $v_0 \leftarrow v_0 + v_1$;
4 **return** $v_0 + v_1 s$

El costo de este algoritmo es de 2 elevaciones al cuadrado y una suma en el campo $\mathbb{F}_{2^m}$.

3.3.2.3. Multiplicación

La multiplicación en $\mathbb{F}_{2^{2m}}$ puede ser calculada utilizando la técnica de Karatsuba-Ofman [46] que consiste en la reescritura del producto y utilizando la propiedad de la base. Como resultado son necesarias 3 multiplicaciones en el campo base $\mathbb{F}_{2^m}$ como se muestra acontinuación.

$$\begin{aligned}(u_0 + u_1 s)(v_0 + v_1 s) &= u_0 v_0 + (u_0 v_1 + u_1 v_0)s + u_1 v_1 s^2 \\ &= u_0 v_0 + u_1 v_1 + (u_0 v_1 + u_1 v_0 + u_1 v_1)s \\ &= u_0 v_0 + u_1 v_1 + ((u_0 + u_1)(v_0 + v_1) + u_0 v_0)s\end{aligned}$$

El algoritmo 3.6 fue programado como parte de la biblioteca para realizar esta operación.

Algoritmo 3.6: Multiplicación en $\mathbb{F}_{2^{2m}}$

Entrada: $U = u_0 + u_1 s, V = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$
Salida: $W = UV \in \mathbb{F}_{2^{2m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow v_0 + v_1$;
2 $m_0 \leftarrow u_0 v_0$; $m_1 \leftarrow u_1 v_1$; $m_2 \leftarrow a_0 a_1$;
3 $w_0 \leftarrow m_0 + m_1$;
4 $w_1 \leftarrow m_0 + m_2$;
5 **return** $w_0 + w_1 s$

El costo del algoritmo son 3 multiplicaciones y 4 sumas en el campos base $\mathbb{F}_{2^m}$.

3.3.2.4. Inversión

La inversión es por naturaleza la operación aritmética más costosa, sin embargo es posible obtener el inverso de un elemento en $\mathbb{F}_{2^{2m}}$ de manera eficiente de la siguiente forma. Utilizando el desarrollo de la multiplicación de dos elementos $U, V \in \mathbb{F}_{2^{2m}}$, y

suponemos que V es el inverso multiplicativo de U, con $U \neq 0$ y que $UV = 1$, obtenemos el siguiente sistema de ecuaciones:

$$\begin{cases} u_0v_0 + u_1v_1 &= 1 \\ u_0v_1 + u_1v_0 + u_1v_1 &= 0 \end{cases}$$

La solución al sistema de ecuaciones es:

$$v_0 = w^{-1}(u_0 + u_1) \text{ , y } v_1 = w^{-1}u_1$$

donde $w = u_0^2 + (u_0 + u_1)u_1 \in \mathbb{F}_{2^m}$.

El algoritmo 3.7 se programó para realizar esta operación.

Algoritmo 3.7: Inversión en $\mathbb{F}_{2^{2m}}$

Entrada: $U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}$
Salida: $V = U^{-1} = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$

1 $a_0 \leftarrow u_0 + u_1$;
2 $m_0 \leftarrow u_0^2$; $m_1 \leftarrow a_0 u_1$;
3 $a_1 \leftarrow m_0 + m_1$;
4 $i \leftarrow a_1^{-1}$;
5 $v_0 \leftarrow a_0 i$;
6 $v_1 \leftarrow u_1 i$;
7 **return** $v_0 + v_1 s$

Como se puede observar en el algoritmo anterior sólo es necesaria una inversión, además también son necesarias 3 multiplicaciones, 1 elevación al cuadrado y 2 sumas en el campo $\mathbb{F}_{2^m}$.

3.3.3. Operaciones en $\mathbb{F}_{2^{4m}}$

La última torre de campo finito necesaria en característica dos es $\mathbb{F}_{2^{4m}}$, esto es consecuencia del grado de seguridad que tiene la curva elíptica supersingular que se está utilizando, que es $k = 4$, ver la sección 3.2.

De igual forma es necesaria una base la cual es: $\{1, s, t, st\}$ donde los elementos se puede ver como una combinación de la base con coeficientes en $\mathbb{F}_{2^m}$, este campo puede ser representado por:

$$\mathbb{F}_{2^{4m}} \cong \mathbb{F}_{2^m}[s,t]/(s^2 + s + 1)/(t^2 + t + s)$$

como se puede observar se conserva la propiedad $s^2 = s + 1$ y además se agrega una nueva que es $t^2 = t + s$.

Aquí las únicas operaciones necesarias son la suma, elevación al cuadrado y la multiplicación que se describen en las siguiente secciones.

3.3.3.1. Suma

La suma en $\mathbb{F}_{2^{4m}}$ es realizada coeficiente a coeficiente como se muestra a continuación:

$$\begin{aligned} U+V &= (u_0+u_1s+u_2t+u_3st)+(v_0+v_1s+v_2t+v_3st) \\ &= (u_0+v_0)+(u_1+v_1)s+(u_2+v_2)t+(u_3+v_3)st \end{aligned}$$

El siguiente algoritmo fue implementado para realizar la suma de dos elementos en $\mathbb{F}_{2^{4m}}$, su complejidad es lineal, debido a que sólo involucra operaciones de suma.

Algoritmo 3.8: Suma en $\mathbb{F}_{2^{4m}}$

Entrada: $U = u_0 + u_1 s, V = v_0 + v_1 s \in \mathbb{F}_{2^{4m}}$
Salida: $W = U + V \in \mathbb{F}_{2^{4m}}$

1 $w_0 \leftarrow u_0 + v_0$; $w_1 \leftarrow u_1 + v_1$;
2 $w_2 \leftarrow u_2 + v_2$; $w_3 \leftarrow u_3 + v_3$;
3 **return** $w_0 + w_1 s + w_2 t + w_3 st$

El costo del algoritmo 3.8 son solamente 4 sumas en el campo base $\mathbb{F}_{2^m}$.

3.3.3.2. Cuadrado

Nuevamente para obtener el cuadrado de un elemento en $\mathbb{F}_{2^{4m}}$ es necesario desarrollar el cuadrado de la siguiente manera: Sea $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$, entonces U^2 es:

$$V = U^2 = u_0^2 + u_1^2 s^2 + u_2^2 t^2 + u_3^2 s^2 t^2$$

y recordemos que $s^2 = s + 1$, $t^2 = t + s$, entonces $s^2 t^2 = 1 + t + st$ y con esto los coeficientes para $V = v_0 + v_1 s + v_2 t + v_3 st \in \mathbb{F}_{2^{4m}}$ son los siguientes:

$$\begin{aligned} v_0 &= u_0^2 + u_1^2 + u_3^2 & v_2 &= u_2^2 + u_3^2 \\ v_1 &= u_1^2 + u_2^2 & v_3 &= u_3^2 \end{aligned}$$

con el resultado obtenido anteriormente podemos generar el algoritmo 3.9, el cual fue implementado en software para ser integrado a la biblioteca.

Algoritmo 3.9: Cuadrado en $\mathbb{F}_{2^{4m}}$

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$
Salida: $V = U^2 \in \mathbb{F}_{2^{4m}}$

1 $s_0 \leftarrow u_0^2$; $s_1 \leftarrow u_1^2$;
2 $s_2 \leftarrow u_2^2$; $v_3 \leftarrow u_3^2$;
3 $v_0 \leftarrow s_0 + s_1 + v_3$;
4 $v_1 \leftarrow s_1 + s_2$;
5 $v_2 \leftarrow s_2 + v_3$;
6 **return** $v_0 + v_1 s + v_2 t + v_3 st$

El costo del algoritmo anterior son 4 elevadas al cuadrado y 4 sumas en el campo $\mathbb{F}_{2^m}$.

3.3.3.3. Multiplicación

La multiplicación en $\mathbb{F}_{2^{4m}}$ es obtenida al desarrollar la multiplicaciones de dos elementos como sigue: sea $U, V \in \mathbb{F}_{2^{4m}}$.

$$W = UV = (u_0 + u_1 s + u_2 t + u_3 st)(v_0 + v_1 s + v_2 t + v_3 st)$$

Después de desarrollar la multiplicación y utilizar las propiedades de la base, los coeficiente para W quedan de la siguiente forma:

$$\begin{aligned} W \;=\; & (u_0v_0 + u_1v_1 + u_3v_2 + u_2v_3 + u_3v_3) + \\ & (u_1v_0 + u_0v_1 + u_1v_1 + u_2v_2 + u_3v_2 + u_2v_3)s + \\ & (u_2v_0 + u_0v_2 + u_3v_1 + u_2v_2 + u_1v_3 + u_3v_3)t + \\ & (u_3v_0 + u_2v_1 + u_1v_2 + u_0v_3 + u_3v_1 + u_1v_3 + u_3v_2 + u_2v_3 + u_3v_3)st \end{aligned}$$

Hasta este punto son necesarias 16 multiplicaciones y resulta ser un método ineficiente, pero utilizando la técnica de Karatsuba podemos reescribir el resultado de la siguiente forma:

$$\begin{aligned} W \;=\; & [u_0v_0 + u_1v_1 + u_2v_2 + (u_2 + u_3)(v_2 + v_3)] + \\ & [u_0v_0 + (u_0 + u_1)(v_0 + v_1) + u_3v_3 + (u_2 + u_3)(v_2 + v_3)]\, s + \\ & [u_0v_0 + u_1v_1 + (u_0 + u_2)(v_0 + v_2) + (u_1 + u_3)(v_1 + v_3)]\, t + \\ & [u_0v_0 + (u_0 + u_1)(v_0 + v_1) + (u_0 + u_2)(v_0 + v_2) + \\ & (u_0 + u_1 + u_2 + u_3)(v_0 + v_1 + v_2 + v_3)]\, st \end{aligned}$$

Como podemos observar el número de sumas fue aumentado pero en este caso sólo son necesarias 9 multiplicaciones que resulta ser mucho más eficiente. El algoritmo 3.10 es utilizado para realizar multiplicación de elementos en $\mathbb{F}_{2^{4m}}$. El costo del algoritmo son 9 multiplicaciones y 20 sumas en $\mathbb{F}_{2^m}$.

Algoritmo 3.10: Multiplicación en $\mathbb{F}_{2^{4m}}$

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st, V = v_0 + v_1 s + v_2 t + v_3 st \in \mathbb{F}_{2^{4m}}$
Salida: $W = UV \in \mathbb{F}_{2^{4m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow v_0 + v_1$;
2 $a_2 \leftarrow u_0 + u_2$; $a_3 \leftarrow v_0 + v_2$;
3 $a_4 \leftarrow u_1 + u_3$; $a_5 \leftarrow v_1 + v_3$;
4 $a_6 \leftarrow u_2 + u_3$; $a_7 \leftarrow v_2 + v_3$;
5 $a_8 \leftarrow a_0 + a_6$; $a_9 \leftarrow a_1 + a_7$;
6 $m_0 \leftarrow u_0 v_0$; $m_1 \leftarrow u_1 v_1$; $m_2 \leftarrow u_2 v_2$; $m_3 \leftarrow u_3 v_3$;
7 $m_4 \leftarrow a_0 a_1$; $m_5 \leftarrow a_2 a_3$; $m_6 \leftarrow a_4 a_5$; $m_7 \leftarrow a_6 a_7$; $m_8 \leftarrow a_8 a_9$;
8 $a_{10} \leftarrow m_0 + m_1$; $a_{11} \leftarrow m_0 + m_4$;
9 $w_0 \leftarrow a_{10} + m_2 + m_7$;
10 $w_1 \leftarrow a_{11} + m_3 + m_7$;
11 $w_2 \leftarrow a_{10} + m_5 + m_6$;
12 $w_3 \leftarrow a_{11} + m_5 + m_8$;
13 **return** $w_0 + w_1 s + w_2 t + w_3 st$

Existen dos casos especiales en la multiplicación en $\mathbb{F}_{2^{4m}}$ debido a que en el cálculo del emparejamiento η_T algunos elementos cuentan con coeficientes en 1 o en 0. El primer caso es cuando uno de los elementos tiene su último coeficiente (st) igual a 0 y su tercer coeficiente (t) igual a 1 mientras que el otro elemento tiene todos sus coeficientes diferentes de 0 y 1. El procedimiento para obtener la multiplicación es similar al ya mencionado y el algoritmo 3.11 corresponde a esta multiplicación.

Algoritmo 3.11: Multiplicación en $\mathbb{F}_{2^{4m}}$ Caso 1

Entrada: $U = u_0 + u_1 s + t, V = v_0 + v_1 s + v_2 t + v_3 st \in \mathbb{F}_{2^{4m}}$
Salida: $W = UV \in \mathbb{F}_{2^{4m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow v_0 + v_1$; $a_2 \leftarrow v_2 + v_3$;
2 $m_0 \leftarrow u_0 v_0$; $m_1 \leftarrow u_1 v_1$; $m_2 \leftarrow u_0 v_2$; $m_3 \leftarrow u_1 v_3$;
3 $m_4 \leftarrow a_0 a_1$; $m_5 \leftarrow a_0 a_2$;
4 $w_0 \leftarrow m_0 + m_1 + v_3$;
5 $w_1 \leftarrow m_0 + m_4 + v_2 + v_3$;
6 $w_2 \leftarrow m_2 + m_3 + v_0 + v_2$;
7 $w_3 \leftarrow m_2 + m_5 + v_1 + v_3$;
8 **return** $w_0 + w_1 s + w_2 t + w_3 st$

El costo del algoritmo anterior son 6 multiplicaciones y 14 sumas en $\mathbb{F}_{2^m}$. Otro caso especial es cuando los dos elementos tiene su tercer elemento (t) igual a 1 y su último coeficiente (st) en 0. El algoritmo 3.12 es utilizado para realizar esta multiplicación.

Algoritmo 3.12: Multiplicación en $\mathbb{F}_{2^{4m}}$ Caso 2

Entrada: $U = u_0 + u_1 s + t, V = v_0 + v_1 s + t \in \mathbb{F}_{2^{4m}}$
Salida: $W = UV \in \mathbb{F}_{2^{4m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow v_0 + v_1$;
2 $m_0 \leftarrow u_0 v_0$; $m_1 \leftarrow u_1 v_1$; $m_2 \leftarrow a_0 a_1$;
3 $w_0 \leftarrow m_0 + m_1$;
4 $w_1 \leftarrow m_0 + m_2 + 1$;
5 $w_2 \leftarrow u_0 + v_0 + 1$;
6 $w_3 \leftarrow u_1 + v_1$;
7 **return** $w_0 + w_1 s + w_2 t + w_3 st$

El costo del algoritmo son 3 multiplicaciones 6 sumas y dos instrucciones lógicas XOR utilizadas en los pasos 4 y 5 respectivamente.

3.3.4. Resultados de la aritmética en $\mathbb{F}_{2^m}$

En la Tabla 3.1 se presentan los tiempos obtenidos para las operaciones aritméticas sobre el campos $\mathbb{F}_{2^{233}}$, estos tiempos corresponde a la implementación dedicada a la PDA. Tambien se presentan los tiempos obtenidos de la ejecución de estas operaciones sobre una computadora Laptop.

Operación	PDA Zaurus SL-5600	Laptop
Suma	0.239 μs.	2.5 ns.
Multiplicación	29.55 μs.	0.89 μs.
Cuadrado	9.57 μs.	0.1 μs.
Inversión	242 μs.	10.95 μs.

Tabla 3.1: Tiempos de la aritmética sobre $\mathbb{F}_{2^{233}}$.

En la Tabla 3.2 se muestran los tiempos obtenidos de la aritmética sobre $\mathbb{F}_{2^{503}}$ utilizando un sistema operativo de 32 bits (IA-32) y 64 bits (x64).

Operación	x64	IA-32
Suma	0.0046 μs.	0.0048 μs.
Multiplicación	1.52 μs.	1.9296 μs.
Cuadrado	0.090 μs.	0.096 μs.
Raíz cuadrada	0.112 μs.	0.120 μs.
Inversión	54.12 μs.	58.12 μs.

Tabla 3.2: Tiempos de la aritmética sobre $\mathbb{F}_{2^{503}}$.

Las características de las arquitecturas donde se realizaron las pruebas se presentan en la sección 3.1.

3.4. Consideraciones para característica tres

Al igual que en característica dos, aquí también la implementación de las operaciones básicas y del emparejamiento η_T se realiza sobre diferentes tamaños de campos finitos para cada arquitectura, sin embargo la curva elíptica supersingular utilizada es la misma:

$$\begin{aligned} \text{Curva elíptica supersingular:} \quad & E(\mathbb{F}_{3^m}) : \ y^2 = x^3 + x + b \quad (b \in \{-1, 1\}) \\ \text{Grado de seguridad (}\textit{embedding degree}\text{):} \quad & k = 6 \end{aligned}$$

Los campos utilizados para cada arquitectura tienen las siguientes restricciones.

3.4.1. Restricciones en PDA

El campo finito utilizado y el polinomio irreducible para la implementación en la PDA es:

$$\begin{aligned} \text{Campo finito:} \quad & GF(3^{97}) \\ \text{Polinomio irreducible:} \quad & P(x) = x^{97} - x^{12} + 1 \end{aligned}$$

3.4.2. Restricciones en multinúcleo

El campo finito utilizado para la implementación en la computadora multinúcleo es:

$$\begin{aligned} \text{Campo finito:} \quad & GF(3^{503}) \\ \text{Polinomio irreducible:} \quad & P(x) = x^{503} - x^{35} + 1 \end{aligned}$$

3.5. Implementación en característica tres

Un campo ternario es una extensión del campo $\mathbb{F}_3$, donde cada elemento de $\mathbb{F}_3$ es representado utilizando dos bits, a esta representación se le llama trit y la codificación utilizada para su implementación en sofware es la siguiente:

$$\mathbb{F}_3 = \{(0,0) \mapsto 0, \ (0,1) \mapsto 1, \ (1,0) \mapsto 2\}$$

En otras palabras, cada elemento de $\mathbb{F}_3$ cuenta con una parte alta y una parte baja, en general todo elemento $a \in \mathbb{F}_3$ es de la forma $a = (a_h, a_l)$, con la restricción de que a_l

y a_h no pueden estar encendidos al mismo tiempo, esta codificiación para la representación es recomendada en [42].

Con la finalidad de obtener una buena implementación se realizaron pruebas para decidir cual es la mejor forma de representar los elementos en software. Es claro que son necesarios 2 bits para representar un dígito en $\mathbb{F}_3$, pero también hay que tomar en cuenta que si estos 2 bits deben de estar en una misma cadena de forma continua o si es mejor tener la parte baja separada de la parte alta.

El tipo de dato básico utilizado es al igual que en característica dos una palabra de 32 bits. Como primera aproximación se pensaría poder representar los dígitos como una sola cadena de bits, es decir, $(a_{ih}a_{il}\ldots a_{2h}a_{2l}a_{1h}a_{1l}a_{0h}a_{0l})$, entonces podríamos representar 16 dígitos con una palabra.

Sin embargo, en la PDA se observó un comportamiento en el que las operaciones aritméticas tienen un mejor desempeño cuando se utilizan 4 bits para representar un dígito, aunque al precio de utilizar el doble de espacio para almacenar un elemento en $\mathbb{F}_{3^m}$. Esto es, la representación de un dígito es de la forma $00a_ha_l$, permitiéndonos almacenar únicamente 8 dígitos en una palabra de 32 bits, pero esta representación deja de ser eficiente para una arquitectura que cuente con un procesador de propósito general como se mostrará más adelante con los resultados.

Por otro lado en la computadora multinúcleo, el mejor desempeño en la aritmética es logrado al representar los elementos con dos vectores, uno que contiene todas las partes bajas y otro con las partes altas de cada dígito que componen un elemento en $\mathbb{F}_{3^m}$, estos vectores son encapsulados en tipo de dato abstracto[1] para obtener un mejor control sobre ellos.

3.5.1. Operaciones en $\mathbb{F}_{3^m}$

3.5.1.1. Suma y resta

En 2002 Galbraith et al. [33] demostraron cómo realizar la suma en característica tres utilizando 12 instrucciones lógicas que consistian en AND, OR, XOR y NOT, pero en ese mismo año Harrison et. al. [42] propusieron una mejora utilizando sólo 7 instrucciones lógicas que consisten de OR y XOR. Hasta ahora, este es el mínimo número de instrucciones para calcular la suma en $\mathbb{F}_3$.

$$\begin{aligned} t &= (a_l|b_h) \oplus (a_h|b_l) \\ c_l &= t \oplus (a_h|b_h) \\ c_h &= t \oplus (a_l|b_l) \end{aligned}$$

[1] En programación un tipo de dato abstracto (ADT, Abstrat Data Type) es un conjunto de datos u objetos al cual se le asocian operaciones.

Sin embargo, el orden en que son procesadas las 7 instrucciones causan un gran impacto en la eficiencia para su implementación en software. Estas instrucciones se pueden reescribir de la siguiente manera para calcular $c = a + b$:

$$\begin{aligned} t &= (a_l|a_h)\&(b_l|b_h) \\ c_l &= t \oplus (a_l|b_l) \\ c_h &= t \oplus (a_h|b_h) \end{aligned}$$

La resta en $\mathbb{F}_3$ es muy similar a la suma, de igual forma son necesarias 7 instrucciones, sea $a, b \in \mathbb{F}_3$, la operación $c = a - b$ es definida como sigue:

$$\begin{aligned} t &= (a_l|a_h)\&(b_l|b_h) \\ c_l &= t \oplus (a_l|b_h) \\ c_h &= t \oplus (a_h|b_l) \end{aligned}$$

La suma y la resta en $\mathbb{F}_3$ se describen en las Tablas 1.3 y 1.4 respectivamente de la sección 1.2.4.2.

La implementación en la PDA se realiza a nivel palabra (32 bits), es decir, se procesan 8 dígitos al mismo tiempo como ya se mencionó anteiormente, la gran diferencia con la implementación en la arquitectura multinúcleo es que se manipulan dos vectores uno con las partes bajas y el otro con las partes altas, con esto podemos procesar un total de 32 dígitos, pero además se utiliza SSE para empaquetar las palabras en un vector y así podemos procesar 128 dígitos al mismo tiempo.

3.5.1.2. Cubo

Como la operación de elevar al cubo en característica tres es considerada lineal, pues sólo consiste en intercalar dos dígitos en cero entre cada dígito del elemento original, esta operación está definida por la siguiente ecuación:

$$A(x) = \sum_{j=0}^{m-1} a_j x^j \Longrightarrow (A(x))^3 = \sum_{j=0}^{m-1} a_j x^{3j}$$

La implementación más eficiente se logra utilizando una tabla de consulta para realizar la expansión del elemento, como resultado se obtiene un elemento fuera del campo de aproximadamente 3 veces la longitud original, es decir, un elemento con longitud de m trits es expandido a $3m-2$ trits, para terminar de realizar la operación se procede con una reducción modular que es similar a la reducción en característica dos, esta reducción es explicada en la sección 3.5.1.4.

3.5.1.3. Multiplicación

La multiplicación en $\mathbb{F}_{3^m}$ es una operación que fue implementada para cada arquitectura de manera distinta. En la PDA se implementó el método de Karatsuba-Ofman [46] para realizar la multiplicación de los vectores más grandes y como este método consiste en la división de los elementos para obtener operandos más pequeños hasta llegar a multiplicar a nivel palabra que es realizando utilizando el método Comb [50] modificado para característica tres.

En la implementación para la arquitectura multinúcleo se apreció un mejor desempeño al utilizar únicamente el método Comb [50] con una pequeña modificación para característica tres con la restricción de que aquí se utiliza una ventana $w = 4$. Entonces se tiene que precalcular una tabla que contiene $3^4 = 81$ entradas.

El pre-cálculo de la tabla es relativamente sencillo y muy barato desde que prácticamente la mitad de los elementos en la tabla son obtenidos por medio de una simple negación. Un ejemplo pequeño es si quisiéramos contruir la tabla para un elemento $a \in \mathbb{F}_{3^m}$ y utilizando una ventana $w = 2$ tendríamos entonces una tabla con $3^2 = 9$ entradas, obtenidas de la siguiente manera:

Entrada		Valor
[00]	$\leftarrow$	0
[01]	$\leftarrow$	a
[02]	$\leftarrow$	$\sim$ [01]
[10]	$\leftarrow$	[01] $\ll 1$
[11]	$\leftarrow$	[10] + [01]
[12]	$\leftarrow$	[10] + [02]
[20]	$\leftarrow$	$\sim$ [10]
[21]	$\leftarrow$	$\sim$ [12]
[22]	$\leftarrow$	$\sim$ [11]

Tabla 3.3: Ejemplo: tabla Comb en característica tres con $w = 2$ para $a \in \mathbb{F}_{3^m}$.

En la tabla anterior $\sim$ representa la negación lógica y $\ll$ un corrimiento a la derecha, como se puede observar 4 de las 9 entradas son calculadas con una simple negación y son necesarias también dos sumas, una inicialización en 0 y la asignación del elemento a. De esta forma se construyó la tabla para nuestra implementación en la arquitectura multinúcleo utilizando $w = 4$ dándonos un total de $3^4 = 81$ entradas, donde el costo de las operaciones se presenta en la Tabla 3.4:

Operación	#
Asignación de 0:	1
Asignación de a:	1
Negación:	40
Suma:	36
Corrimientos:	3

Tabla 3.4: Costo para construir la tabla Comb con $w = 4$, $a \in \mathbb{F}_{3^m}$.

Tanto en la implementacion en la PDA y la arquitectura multinúcleo al finalizar la multiplicación se realiza una reducción modular para obtener nuevamente un elemento en el campo $\mathbb{F}_{3^m}$.

3.5.1.4. Reducción modular

La reducción modular es una operación cuyo costo depende del polinomio irreducible utilizado. En nuestra biblioteca se utilizaron simpre trinomios es decir polinomios de la forma $x^m - x^k + 1$, para característica tres, en la PDA se utilizó $P(x) = x^{97} - x^{12} + 1$ y en la arquitectura multinúcleo $P(x) = x^{503} - x^{35} + 1$. La reducción es similar que en característica dos.

3.5.1.5. Raíz cúbica

La raíz cúbica fue implementada sólo para la aquitectura multinúcleo, pues la versión del emparejamiento η_T que utiliza raices cúbicas ofrece la característica de ser paralelizable que es el objetivo de la implementación en una arquitectura multinúcleo.

El cálculo de la raíz cúbica depende del polinomio irreducible utilizado como lo describe Barreto en [5], en nuestro caso el polinomio es $P(x) = x^{503} - x^{35} + 1$, la forma de calcular la raíz es la siguiente:

Sea $a \in \mathbb{F}_{3^m}$ y $m \equiv 2 \bmod 3$, en nuestro caso $m = 503$, entonces:

$$\begin{aligned} a &= \sum_{i=0}^{m-1} a_i x^i \\ &= \sum_{i=0}^{u} a_{3i} x^{3i} + x \sum_{i=0}^{u} a_{3i+1} x^{3i} + x^2 \sum_{i=0}^{u} a_{3i+2} x^{3i} \\ \therefore \sqrt[3]{a} &= \sum_{i=0}^{u} a_{3i} x^i + x^{1/3} \sum_{i=0}^{u} a_{3i+1} x^i + x^{2/3} \sum_{i=0}^{u} a_{3i+2} x^i \end{aligned}$$

ahora se definen $c_0 = \sum_{i=0}^{u} a_{3i}x^i, c_1 = \sum_{i=0}^{u} a_{3i+1}x^i, c_2 = \sum_{i=0}^{u} a_{3i+2}x^i$, obteniendo una forma simplificada:

$$\sqrt[3]{a} = c_0 + x^{1/3}c_1 + x^{2/3}c_2$$

Como podemos observar el valor de u aún no ha sido definido y es obtenido al representar a m de la siguiente forma $m = 3u + r$ y además también podemos representar a k como $k = 3v + r$, donde $r = m \equiv k \equiv 2 \bmod 3$. En nuestro caso $r = 2$,$u = 167$ y $v = 11$ y los polinomios c_0, c_1 y c_2 son fácilmente calculados, puesto que sólo es necesario reacomodar el polinomio original en 3 nuevos polinomios de aproximadamente 1/3 de la longitud del polinomio inicial.

Ahora sólo falta definir los valores para $x^{1/3}$ y $x^{2/3}$. Barreto describe que para el caso en que $m \equiv \ k \equiv \ 2 \bmod 3$, podemos representar de forma general el polinomio irreducible de la siguiente forma $x^{3u+2} + ax^{3v+2} + b = 0$, dándonos como resultado:

$$\begin{aligned} x^{1/3} &= -bx^{u+1} - abx^{v+1}, \\ x^{2/3} &= x^{2u+2} - ax^{u+v+2} + x^{2v+2} \end{aligned}$$

Cabe mencionar que este método es eficiente debido a que el peso de Hamming[2] de $x^{1/3}$ y $x^{2/3}$ es pequeño como en las ecuaciones anteriores, pero si el peso de Hamming es muy grande el método deja de ser eficiente. En el trabajo de Ahmadi et al. [3] ayudan a determinar el peso de Hamming para $x^{1/3}$ y $x^{2/3}$ para casos más complejos.

Con todo lo anterior, los parámetros para nuestra implementación son: polinomio irreducible $P(x) = x^{503} - x^{35} + 1$ donde $u = 167$, $v = 11$, $r = 2$ y

$$\begin{aligned} x^{1/3} &= -x^{168} + x^{12}, \\ x^{2/3} &= x^{336} + x^{180} + x^{24} \end{aligned}$$

En el algoritmo 3.13 se representa el proceso necesario para el cálculo de la raíz cúbica de un elemento en $\mathbb{F}_{3^m}$.

Algoritmo 3.13: Raíz cúbica en $\mathbb{F}_{3^m}$

Entrada: $A(x) = a_{m-1}x^{m-1} + \ldots + a_2x^2 + a_1x + a_0 \in \mathbb{F}_{3^m}$ y el polinomio irreducible $P(x)$
Salida: $R(x) = \sqrt[3]{A(x)} \in \mathbb{F}_{3^m}$

1 $c_0 \leftarrow (a_{3u}x^u + \ldots + a_6x^2 + a_3x + a_0)$;
2 $c_1 \leftarrow (a_{3u+1}x^u + \ldots + a_7x^2 + a_4x + a_1)$;
3 $c_2 \leftarrow (a_{3u+2}x^u + \ldots + a_8x^2 + a_5x + a_2)$;
4 $R \leftarrow c_0 + x^{1/3}c_1 + x^{2/3}c_2$;
5 $R \leftarrow R \bmod P(x)$;
6 **return** R

[2]Peso de Hamming (Hamming weight): número de coeficientes diferentes de cero en una representación polinomial.

En nuestra implementación los polinomios c_0, c_1 y c_2 son obtenidos con ayuda de una tabla de consulta para extraer los coeficientes correspondientes a cada uno.

3.5.1.6. Inversión

Para calcular el inverso multiplicativo de un elemento en característica tres se programó una variante del algoritmo binario extendido de Euclides que fue propuesta en [42], este método recibe el nombre de Algoritmo Ternario de Euclides. El algoritmo 3.14 fue programado de manera eficiente para obtener $D = B^{-1} \bmod P$, donde $D, B \in \mathbb{F}_{3^m}$ y P es el polinomio irreducible del campo.

Algoritmo 3.14: Inversión en $\mathbb{F}_{3^m}$ - Algoritmo Ternario de Euclides

Entrada: $B \in \mathbb{F}_{3^m}$ y el polinomio irreducible P
Salida: $D = B^{-1} \bmod P$

```
1  a ← F; B ← 0; D ← 1;
2  while a! = 0 do
3      while tc(a) = 0 do
4          if tc(B)! = 0 then
5              if tc(B) = tc(F) then B ← B − F;
6              else B ← B + F;
7          end
8          a ← a/x; B ← B/x;
9      end
10     while tc(b) = 0 do
11         if tc(D)! = 0 then
12             if tc(D) = tc(F) then D ← D − F;
13             else D ← D + F;
14         end
15         b ← b/x; D ← D/x;
16     end
17     if deg(a) ≥ deg(b) then
18         if tc(a) = tc(b) then a ← a − b; B ← B − D ;
19         else a ← a + b; B ← B + D;
20     else
21         if tc(a) = tc(b) then b ← b − a; D ← D − B ;
22         else b ← b + a; D ← D + B;
23     end
24 end
25 if b! = 1 then D ← −D;
26 return D
```

3.5.2. Operaciones en $\mathbb{F}_{3^{3m}}$

Para el cálculo del emparejamiento η_T sobre una curva elíptica supersingular en un campo ternario es necesario la construcción de dos torres de campos finitos, en esta sección se describe la construcción de la torre de campo $\mathbb{F}_{3^{3m}}$.

Para la construcción de $\mathbb{F}_{3^{3m}}$ es necesaria una base para representar los elementos en este campo, la base utilizada es $\{1, \rho, \rho^2\}$, utilizando esta base la torre de campo finito puede ser expresado como:

$$\mathbb{F}_{3^{3m}} \cong \mathbb{F}_{3^m}[\rho]/(\rho^3 - \rho - b)$$

donde b es el coeficiente constante de la curva elíptica utilizada. Es necesario mencionar que una propiedad en la base es $\rho^3 = \rho + b$.

Aquí los elementos son una combinación lineal de la base con coeficientes en $\mathbb{F}_{3^m}$, es decir:

$$U = u_0 + u_1\rho + u_2\rho^2$$

Las operaciones implementadas en esta torre de campo son la suma, resta, multiplicación, inversión y además también es necesaria la operación de elevar un elemento al cuadrado, aquí no es necesario implementar la elevación al cubo pues no es utilizada en el emparejamiento η_T, a continuación se describen a detalle todas las operaciones aritméticas.

3.5.2.1. Suma y resta

La suma y la resta son realizadas coeficiente a coeficiente de cada elemento. Sea $U, V \in \mathbb{F}_{3^{3m}}$. Se tiene:

$$\begin{aligned} U \pm V &= (u_0 + u_1\rho + u_2\rho^2) \pm (v_0 + v_1\rho + v_2\rho^2) \\ &= (u_0 \pm v_0) + (u_1 \pm v_1)\rho + (u_2 \pm v_2)\rho^2 \end{aligned}$$

donde $\pm$ puede ser la suma (+) o la resta (-). El algoritmo 3.15 es utilizado para realizar la suma o resta, el costo de este algoritmo son 3 sumas/restas en el campo $\mathbb{F}_{3^m}$, esta operación es muy sencilla y su complejidad es lineal con el tamaño de los operandos.

Algoritmo 3.15: Suma/Resta en $\mathbb{F}_{3^{3m}}$

Entrada: $U = u_0 + u_1\rho + u_2\rho^2, V = v_0 + v_1\rho + v_2\rho^2 \in \mathbb{F}_{3^{3m}}$
Salida: $W = U \pm V \in \mathbb{F}_{3^{3m}}$

1 $w_0 \leftarrow u_0 \pm v_0$;
2 $w_1 \leftarrow u_1 \pm v_1$;
3 $w_2 \leftarrow u_2 \pm v_2$;
4 **return** $w_0 + w_1\rho + w_2\rho^2$

3.5.2.2. Multiplicación

La mutliplicación es una operación importante, pues hay que buscar la manera más eficiente de implementarla porque de esta operación depende en gran medida el cálculo del emparejamiento η_T. Supongamos que tenemos dos elementos $U = u_0+u_1\rho+u_2\rho^2, V = v_0 + v_1\rho + v_2\rho^2 \in \mathbb{F}_{3^{3m}}$, donde $u_i, v_i \in \mathbb{F}_{3^m}, 0 \leq i \leq 2$:

$$W = UV = (u_0 + u_1\rho + u_2\rho^2)(v_0 + v_1\rho + v_2\rho^2)$$

al desarrollar el producto con ayuda de la propiedad de la base $\rho^3 = \rho + b$, y utilizando la técnica de Karatsuba, los coeficientes para W quedan de las siguiente manera:

$$\begin{aligned} w_0 &= u_0v_0 - u_1v_1 - u_2v_2 + b(bu_1 + u_2)(v_1 + bv_2) \\ w_1 &= -u_0v_0 - (b+1)u_1v_1 + (u_0 + u_1)(v_0 + v_1) + (bu_1 + u_2)(v_1 + bv_2) \\ w_2 &= -u_0v_0 + u_1v_1 + (u_0 + u_2)(v_0 + v_2) \end{aligned}$$

Con esto para realizar la multiplicación de dos elementos en $\mathbb{F}_{3^{3m}}$ se implementó el algoritmo 3.16.

Algoritmo 3.16: Multiplicación en $\mathbb{F}_{3^{3m}}$

Entrada: $U = u_0 + u_1\rho + u_2\rho^2, V = v_0 + v_1\rho + v_2\rho^2 \in \mathbb{F}_{3^{3m}}$
Salida: $W = UV \in \mathbb{F}_{3^{3m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_0 + v_2$; $a_2 \leftarrow bu_1 + u_2$;
2 $a_3 \leftarrow v_0 + v_1$; $a_4 \leftarrow v_0 + v_2$; $a_5 \leftarrow v_1 + bv_2$;
3 $m_0 \leftarrow u_0v_0$; $m_1 \leftarrow u_1v_1$; $m_2 \leftarrow u_2v_2$;
4 $m_3 \leftarrow a_0a_3$; $m_4 \leftarrow a_1a_4$; $m_5 \leftarrow a_2a_5$;
5 $a_6 \leftarrow m_0 - m_1$;
6 $w_0 \leftarrow a_6 - m_2 + bm_5$;
7 **if** *b=1* **then**
8 $\quad w_1 \leftarrow -a_6 + m_3 + m_5$;
9 **else**
10 $\quad w_1 \leftarrow -m_0 + m_3 + m_5$;
11 **end**
12 $w_2 \leftarrow -a_6 + m_4$;
13 **return** $w_0 + w_1\rho + w_2\rho^2$

El costo del algoritmo anterior son 6 multiplicaciones y 14 sumas en el campo $\mathbb{F}_{3^m}$.

3.5.2.3. Inversión

Para la implementación de la inversión en $\mathbb{F}_{3^{3m}}$ suponemos que $U = u_0 + u_1\rho + u_2\rho^2, V = v_0 + v_1\rho + v_2\rho^2 \in \mathbb{F}_{3^{3m}}$ tal que $UV \equiv 1$, y conocemos que $U \neq 0$, entonces al

desarrollar el producto obtenemos el siguiente sistema de ecuaciones:

$$\begin{cases} u_0v_0 + bu_2v_1 + bu_1v_2 &= 1 \\ (u_1v_0 + (u_0+u_2)v_1 + (u_1+bu_2)v_2)\rho &= 0 \\ (u_2v_0 + u_1v_1 + (u_0+u_2)v_2)\rho^2 &= 0 \end{cases}$$

donde la solución a este sistema de ecuaciones para V es:

$$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = w^{-1} \begin{pmatrix} u_0^2 - (u_1^2 - u_2^2) - u_2(u_0 + bu_1) \\ bu_2^2 - u_0u_1 \\ u_1^2 - u_2^2 - u_0u_2 \end{pmatrix}$$

con $w = u_0^2(u_0 - u_2) + u_1^2(-u_0 + bu_1) + u_2^2(-(-u_0 + bu_1) + u_2) \in \mathbb{F}_{3^m}$.

El algoritmo 3.17 es el resultante para calcular el inverso multiplicativo de un elemento en $\mathbb{F}_{3^{3m}}$.

Algoritmo 3.17: Inversión en $\mathbb{F}_{3^{3m}}$

Entrada: $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$
Salida: $V = U^{-1} \in \mathbb{F}_{3^{3m}}$

1 $a_0 \leftarrow u_0 + bu_1$; $a_1 \leftarrow u_0 - u_2$;
2 $a_2 \leftarrow -u_0 + u_1$; $a_3 \leftarrow -a_2 + u_2$;
3 $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_1^2$; $m_2 \leftarrow u_2^2$;
4 $m_3 \leftarrow u_0u_1$; $m_4 \leftarrow u_0u_2$; $m_5 \leftarrow u_2a_0$;
5 $m_6 \leftarrow m_0a_1$; $m_7 \leftarrow m_1a_2$; $m_8 \leftarrow m_2a_3$;
6 $w \leftarrow m_6 + m_7 + m_8$;
7 $i \leftarrow w^{-1}$;
8 $a_4 \leftarrow m_1 - m_2$; $a_5 \leftarrow -a_4 + m_0 - m_5$;
9 $a_6 \leftarrow bm_2 - m_3$; $a_7 \leftarrow a_4 - m_4$;
10 $v_0 \leftarrow ia_5$; $v_1 \leftarrow ia_6$; $v_2 \leftarrow ia_7$;
11 **return** $v_0 + v_1\rho + v_2\rho^2$

Como se puede observar se necesita elevar al cuadrado elementos en $\mathbb{F}_{3^m}$, pero elevar al cuadrado en característica tres es muy costoso y no existe un método eficiente para obtener cuadrados, por esta razón es utilizada la multiplicación en su lugar y con esto el costo del algoritmo 3.17 son 12 multiplicaciones, 11 sumas y una inversión en el campo base $\mathbb{F}_{3^m}$.

3.5.2.4. Cuadrado

Para el cálculo del emparejamiento es necesario elevar al cuadrado elementos en $\mathbb{F}_{3^{3m}}$, hay que mencionar que aquí no es necesario elevar al cubo dado que en el emparejamiento no es utilizado.

Para elevar un elemento en $\mathbb{F}_{3^{3m}}$ al cuadrado suponemos que $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$, y deseamos obtener $V = U^2 \in \mathbb{F}_{3^{3m}}$ y esta dado de la siguiente manera:

$$V = U^2 = (u_0 + u_1\rho + u_2\rho^2)^2$$

al desarrollar el cuadrado utilizando la propiedad de la base que se mencionó anteriormente, obtenemos los coeficientes para V de la siguiente manera:

$$\begin{aligned} v_0 &= u_0^2 - bu_1u_2 \\ v_1 &= bu_2^2 - u_0u_1 - u_1u_2 \\ v_2 &= (u_0 + u_1 + u_2)(u_0 + u_1 + u_2) - u_0^2 + u_0u_1 + u_1u_2 \end{aligned}$$

Y el algoritmo 3.18 fue implementado como parte de la biblioteca para realizar esta operación y es resultante del análisis anterior.

Algoritmo 3.18: Cuadrado en $\mathbb{F}_{3^{3m}}$

Entrada: $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$
Salida: $V = U^2 \in \mathbb{F}_{3^{3m}}$

1 $a_0 \leftarrow u_0 + u_1$;
2 $a_1 \leftarrow a_0 + u_2$;
3 $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_0u_1$; $m_2 \leftarrow u_1u_2$;
4 $m_3 \leftarrow u_2^2$; $m_4 \leftarrow a_1^2$;
5 $a_2 \leftarrow m_1 + m_2$;
6 $v_0 \leftarrow m_0 - bm_2$;
7 $v_1 \leftarrow bm_3 - a_2$;
8 $v_2 \leftarrow m_4 - m_0 + a_2$;
9 **return** $v_0 + v_1\rho + v_2\rho^2$

El costo de este algoritmo son 5 multiplicaciones 7 sumas en $\mathbb{F}_3$. Podemos observar que elevar al cuadrado un elemento en $\mathbb{F}_{3^{3m}}$ es más eficiente que utilizar el algoritmo 3.16 para realizar la multiplicación de un elemento consigo mismo.

3.5.3. Operaciones en $\mathbb{F}_{3^{6m}}$

En característica tres la última torre de campo necesaria es $\mathbb{F}_{3^{6m}}$, debido a que el grado de seguridad de la curva utilizada es $k = 6$.

La base utilizada para la construcción del campo $\mathbb{F}_{3^{6m}}$ es $\{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$. Aquí los elementos son una combinación lineal de la base con coeficientes en $\mathbb{F}_{3^m}$ y el tower field es representado de la siguiente forma:

$$\mathbb{F}_{3^{6m}} \cong \mathbb{F}_{3^m}[\sigma\rho]/(\sigma^2 + 1)/(\rho^3 - \rho - b)$$

Esta base tiene como propiedades $\sigma^2 = -1$ y $\rho^3 = \rho + b$. Los elementos en este campo tienen la siguiente forma general:

$$U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2$$

Las operaciones que se implementaron en este campo son la suma, resta, elvación al cubo y la multiplicación; en las secciones siguientes se describen estas operaciones.

3.5.3.1. Suma y resta

Nuevamente la suma y resta son las operaciones más baratas y su implementación es muy sencilla debido a que la suma de dos elementos en $\mathbb{F}_{3^{6m}}$ es realizada coeficiente a coeficiente, Sea $U, V \in \mathbb{F}_{3^{6m}}$, entonces definimos $U \pm V$ como sigue:

$$\begin{aligned} U \pm V &= [u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2] \pm \\ &\quad [v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2] \\ &= (u_0 \pm v_0) + (u_1 \pm v_1)\sigma + (u_2 \pm v_2)\rho + \\ &\quad (u_3 \pm v_3)\sigma\rho + (u_4 \pm v_4)\rho^2 + (u_5 \pm v_5)\sigma\rho^2 \end{aligned}$$

El algoritmo 3.19 es utilizado para sumar/restar dos elementos en $\mathbb{F}_{3^{6m}}$.

Algoritmo 3.19: Suma/Resta en $\mathbb{F}_{3^{3m}}$

Entrada: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2, V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$

Salida: $W = U \pm V \in \mathbb{F}_{3^{6m}}$

1 $w_0 \leftarrow u_0 \pm v_0$; $w_1 \leftarrow u_1 \pm v_1$;
2 $w_2 \leftarrow u_2 \pm v_2$; $w_3 \leftarrow u_3 \pm v_3$;
3 $w_4 \leftarrow u_4 \pm v_4$; $w_5 \leftarrow u_5 \pm v_5$;
4 **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2$

El algoritmo anterior es muy sencillo y solamente utiliza 6 sumas en $\mathbb{F}_{3^m}$, fue implementado en software para ser integrada en la biblioteca que se desarrolló en este trabajo de tesis.

3.5.3.2. Cubo

Una operación necesaria para el emparejamiento en característica tres es la elevación al cubo en $\mathbb{F}_{3^{6m}}$, para lograrlo se tiene el siguiente análisis: Sea $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$, entonces U^3 es:

$$V = U^3 = u_0^3 + u_1^3\sigma^3 + u_2^3\rho^3 + u_3^3(\sigma\rho)^3 + u_4^3(\rho^2)^3 + u_5^3(\sigma\rho^2)^3$$

y recordemos que $\rho^3 = \rho + b$, $\sigma^2 = -1$, entonces los coeficientes para $V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$ son los siguientes:

$$\begin{array}{ll} v_0 = u_0^3 + bu_2^3 + u_4^3 & v_3 = -u_3^3 + bu_5^3 \\ v_1 = -u_1^3 - bu_3^3 - u_5^3 & v_4 = u_4^3 \\ v_2 = u_2^3 - bu_4^3 & v_5 = -u_5^3 \end{array}$$

Para realizar esta operación el algoritmo 3.20 fue implementado en software para poder ser utilizado en el cálculo del emparejamiento η_T.

Algoritmo 3.20: Cubo en $\mathbb{F}_{3^{6m}}$

Entrada: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$
Salida: $V = U^3 \in \mathbb{F}_{3^{6m}}$

1 $c_0 \leftarrow u_0^3$; $c_1 \leftarrow u_1^3$; $c_2 \leftarrow u_2^3$;
2 $c_3 \leftarrow u_3^3$; $v_4 \leftarrow u_4^3$; $v_5 \leftarrow u_5^3$;
3 $v_0 \leftarrow c_0 + bc_2 + v_4$;
4 $v_1 \leftarrow c_1 + bc_3 + v_5$;
5 $v_2 \leftarrow c_2 - bv_4$;
6 $v_3 \leftarrow c_3 - bv_5$;
7 **return** $(v_0 - v_1\sigma + v_2\rho - v_3\sigma\rho + v_4\rho^2 - v_5\sigma\rho^2)$

El costo del algoritmo anterior son 6 elevadas al cuadrado y 6 sumas/restas en $\mathbb{F}_{3^{6m}}$, hay que poner atención en el paso 7 del algoritmo donde es obtenido el resultado, porque aquí los polinomios v_1, v_3 y v_5 son negativos, es decir, tenemos que obtener el inverso aditivo de estos valores para satisfacer las ecuaciones resultantes del análisis como se mostro al inicio de esta sección.

3.5.3.3. Multiplicación

La última operación necesaria en esta torre de campo para el cálculo del emparejamiento η_T es la multiplicación. En el 2007 Gorla et. al. [36] propusieron un método para realizar una multiplicación de manera eficiente en $\mathbb{F}_{3^{6m}}$ con sólo utilizar 15 multiplicaciones en $\mathbb{F}_{3^m}$, este método es eficiente y es utilizado para el cálculo del emparejamiento, el algoritmo 3.21 fue implementado en C de manera eficiciente para ser utilizado como parte de la biblioteca que se realizó en este trabajo.

Para el cálculo del emparejamiento η_T en característica tres es necesaria una multiplicación especial sobre $\mathbb{F}_{3^{6m}}$ cuando uno de los elementos no cuenta con todos sus coeficientes diferentes de 0 y ± 1, es caso es el siguiente:

$$\begin{array}{rcl} W & = & UV \text{ donde } W, U, V \in \mathbb{F}_{3^{6m}}, \text{ y con} \\ U & = & u_0 + u_1\sigma + u_2\rho - \rho^2 \\ V & = & v_0 + w_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2 \end{array}$$

Esta multiplicación es muy importante pues es la más utilizada para el cálculo del emparejamiento η_T además de que es más eficiciente pues sólo necesita 12 multiplicaciones sobre $\mathbb{F}_{3^m}$, el algoritmo 3.22 es el correspondiente para realizar esta multiplicación.

Otro caso especial en la multiplicación cuando los oeprandos tiene la siguiente forma $U = u_0 + u_1\sigma + u_2\rho - \rho^2$ y $V = v_0 + v_1\sigma + v_2\rho - \rho^2$. Este cálculo es obtenido con el algoritmo 3.23. El costo del algoritmo 3.23 son 6 multiplicaciones y 21 sumas/restas sobre el campo base $\mathbb{F}_{3^m}$.

3.5.4. Resultados de la aritmética en $\mathbb{F}_{3^m}$

Los resultados de tiempos obtenidos para la aritmética en $\mathbb{F}_{3^{97}}$ sobre la PDA se muestran en la Tabla 3.5, esta implementación se ejecutó sobre una Laptop. Las características de las arquitecturas se presentaron en la sección 3.1.

Operación	PDA Zaurus SL-5600	Laptop
Suma	0.5298 μs.	0.025 μs.
Resta	0.6357 μs.	0.03 μs.
Multiplicación	67.7 μs.	2.6 μs.
Cubo	16.3 μs.	0.38 μs.
Inversión	243.4 μs.	11.8 μs.

Tabla 3.5: Tiempos de la aritmética sobre $\mathbb{F}_{3^{97}}$.

De igual forma que en característica dos, se realizó una implementación para una arquitectura multinúcleo utilizando el campo $\mathbb{F}_{3^{503}}$. La Tabla 3.6 muestra los tiempos obtenidos de esta implementación funcionando en un sistema operativo de 32 bits (IA-32) y 64 bits (x64).

Operación	x64	IA-32
Suma & Resta	0.00498 μs.	0.00578 μs.
Multiplicación	10.12 μs.	11.6 μs.
Cub0	0.309 μs.	0.336 μs.
Raíz cúbica	0.331 μs.	0.398 μs.
Inversión	118 μs.	120 μs.

Tabla 3.6: Tiempos de la aritmética sobre $\mathbb{F}_{3^{503}}$.

Las características de estas plataformas fueron mencionadas en la sección 3.1.

Algoritmo 3.21: Multiplicación en $\mathbb{F}_{3^{6m}}$

Entrada: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2, V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$

Salida: $W = UV \in \mathbb{F}_{3^{6m}}$

$r_0 \leftarrow u_0 + u_4$; $a_0 \leftarrow r_0 + u_2$; $a_{12} \leftarrow r_0 - u_2$;
$r_0 \leftarrow v_0 + v_4$; $a_3 \leftarrow r_0 + v_2$; $a_{15} \leftarrow r_0 - v_2$;
$r_0 \leftarrow u_0 - u_4$; $a_6 \leftarrow r_0 - u_3$; $a_{18} \leftarrow r_0 + u_3$;
$r_0 \leftarrow v_0 - v_4$; $a_9 \leftarrow r_0 - v_3$; $a_{21} \leftarrow r_0 + v_3$;
$r_0 \leftarrow u_1 + u_5$; $a_1 \leftarrow r_0 + u_3$; $a_{13} \leftarrow r_0 - u_3$;
$r_0 \leftarrow v_1 + v_5$; $a_4 \leftarrow r_0 + v_3$; $a_{16} \leftarrow r_0 - v_3$;
$r_0 \leftarrow u_1 - u_5$; $a_7 \leftarrow r_0 + u_2$; $a_{19} \leftarrow r_0 - u_2$;
$r_0 \leftarrow v_1 - v_5$; $a_{10} \leftarrow r_0 + v_2$; $a_{22} \leftarrow r_0 - v_2$;
$a_2 \leftarrow a_0 + a_1$; $a_5 \leftarrow a_3 + a_4$; $a_8 \leftarrow a_6 + a_7$; $a_{11} \leftarrow a_9 + a_{10}$;
$a_{14} \leftarrow a_{12} + a_{13}$; $a_{17} \leftarrow a_{15} + a_{16}$; $a_{20} \leftarrow a_{18} + a_{19}$; $a_{23} \leftarrow a_{21} + a_{22}$;
$a_{24} \leftarrow u_4 + u_5$; $a_{25} \leftarrow v_4 + v_5$;
$m_0 \leftarrow a_0a_3$; $m_1 \leftarrow a_2a_5$; $m_2 \leftarrow a_1a_4$; $m_3 \leftarrow a_6a_9$;
$m_4 \leftarrow a_8a_{11}$; $m_5 \leftarrow a_7a_{10}$; $m_6 \leftarrow a_{12}a_{15}$; $m_7 \leftarrow a_{14}a_{17}$;
$m_8 \leftarrow a_{13}a_{16}$; $m_9 \leftarrow a_{18}a_{21}$; $m_{10} \leftarrow a_{20}a_{23}$; $m_{11} \leftarrow a_{19}a_{22}$;
$m_{12} \leftarrow u_4v_4$; $m_{13} \leftarrow a_{24}a_{25}$; $m_{14} \leftarrow u_5v_5$;
if $b = 1$ **then**
    $t_0 \leftarrow m_0 + m_4 + m_{12}$; $t_1 \leftarrow m_2 + m_{10} + m_{14}$;
    $t_2 \leftarrow m_6 + m_{12}$; $t_3 \leftarrow -m_8 - m_{14}$;
    $t_4 \leftarrow m_7 + m_{13}$; $t_5 \leftarrow t_3 + m_2$; $t_6 \leftarrow t_2 - m_0$;
    $t_7 \leftarrow t_3 - m_2 + m_5 + m_{11}$; $t_8 \leftarrow t_2 + m_0 - m_3 - m_9$;
    $w_0 \leftarrow -t_0 + t_1 - m_3 + m_{11}$;
    $w_1 \leftarrow t_0 + t_1 - m_1 + m_5 + m_9 - m_{13}$;
    $w_2 \leftarrow t_5 + t_6$; $w_3 \leftarrow t_5 - t_6 + t_4 - m_1$;
    $w_4 \leftarrow t_7 + t_8$; $w_5 \leftarrow t_7 - t_8 + t_4 + m_1 - m_4 - m_{10}$;
else
    $t_0 \leftarrow m_4 + m_8 + m_{14}$; $t_1 \leftarrow m_6 + m_{12}$;
    $t_2 \leftarrow t_1 + m_{10}$; $t_3 \leftarrow m_2 + m_{14}$;
    $t_4 \leftarrow t_3 - m_8$; $t_5 \leftarrow -m_0 + m_6 - m_{12}$;
    $t_6 \leftarrow -t_3 + m_5 - m_8 + m_{11}$;
    $t_7 \leftarrow t_1 + m_0 - m_3 - m_9$; $t_8 \leftarrow m_1 + m_{13}$;
    $w_0 \leftarrow t_0 - t_2 + m_5 - m_9$;
    $w_1 \leftarrow t_0 + t_2 + m_3 - m_7 + m_{11} - m_{13}$;
    $w_2 \leftarrow t_4 + t_5$; $w_3 \leftarrow t_4 - t_5 - t_8 + m_7$;
    $w_4 \leftarrow t_6 + t_7$; $w_5 \leftarrow t_6 - t_7 + t_8 - m_4 + m_7 - m_{10}$;
end
return $(w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2)$

Algoritmo 3.22: Multiplicación en $\mathbb{F}_{3^{6m}}$ Caso 1

Entrada: $U = u_0 + u_1\sigma + u_2\rho - \rho^2, V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$

Salida: $W = UV \in \mathbb{F}_{3^{6m}}$

$r_0 \leftarrow u_0 - 1$; $a_0 \leftarrow r_0 + u_2$; $a_{12} \leftarrow r_0 - u_2$;
$r_0 \leftarrow v_0 + v_4$; $a_3 \leftarrow r_0 + v_2$; $a_{15} \leftarrow r_0 - v_2$; $a_6, a_{18} \leftarrow u_0 + 1$;
$r_0 \leftarrow v_0 - v_4$; $a_9 \leftarrow r_0 - v_3$; $a_{21} \leftarrow r_0 + v_3$; $a_1, a_{13} \leftarrow u_1$;
$r_0 \leftarrow v_1 + v_5$; $a_4 \leftarrow r_0 + v_3$; $a_{16} \leftarrow r_0 - v_3$;
$a_7 \leftarrow u_1 + u_2$; $a_{19} \leftarrow u_1 - u_2$;
$r_0 \leftarrow v_1 - v_5$; $a_{10} \leftarrow r_0 + v_2$; $a_{22} \leftarrow r_0 - v_2$;
$a_2 \leftarrow a_0 + a_1$; $a_5 \leftarrow a_3 + a_4$; $a_8 \leftarrow a_6 + a_7$; $a_{11} \leftarrow a_9 + a_{10}$;
$a_{14} \leftarrow a_{12} + a_{13}$; $a_{17} \leftarrow a_{15} + a_{16}$; $a_{20} \leftarrow a_{18} + a_{19}$; $a_{23} \leftarrow a_{21} + a_{22}$;
$a_{24} \leftarrow v_4 + v_5$;
$m_0 \leftarrow a_0 a_3$; $m_1 \leftarrow a_2 a_5$; $m_2 \leftarrow a_1 a_4$; $m_3 \leftarrow a_6 a_9$;
$m_4 \leftarrow a_8 a_{11}$; $m_5 \leftarrow a_7 a_{10}$; $m_6 \leftarrow a_{12} a_{15}$; $m_7 \leftarrow a_{14} a_{17}$;
$m_8 \leftarrow a_{13} a_{16}$; $m_9 \leftarrow a_{18} a_{21}$; $m_{10} \leftarrow a_{20} a_{23}$; $m_{11} \leftarrow a_{19} a_{22}$;
if $b = 1$ **then**
    $t_0 \leftarrow m_0 + m_4 - v_4$; $t_1 \leftarrow m_2 + m_{10}$;
    $t_2 \leftarrow m_6 - v_4$; $t_3 \leftarrow -m_8$;
    $t_4 \leftarrow m_7 - a_{24}$; $t_5 \leftarrow t_3 + m_2$; $t_6 \leftarrow t_2 - m_0$;
    $t_7 \leftarrow t_3 - m_2 + m_5 + m_{11}$; $t_8 \leftarrow t_2 + m_0 - m_3 - m_9$;
    $w_0 \leftarrow -t_0 + t_1 - m_3 + m_{11}$;
    $w_1 \leftarrow t_0 + t_1 - m_1 + m_5 + m_9 + a_{24}$;
    $w_2 \leftarrow t_5 + t_6$; $w_3 \leftarrow t_5 - t_6 + t_4 - m_1$;
    $w_4 \leftarrow t_7 + t_8$; $w_5 \leftarrow t_7 - t_8 + t_4 + m_1 - m_4 - m_{10}$;
else
    $t_0 \leftarrow m_4 + m_8$; $t_1 \leftarrow m_6 - v_4$;
    $t_2 \leftarrow t_1 + m_{10}$; $t_3 \leftarrow m_2$;
    $t_4 \leftarrow t_3 - m_8$; $t_5 \leftarrow -m_0 + m_6 + v_4$;
    $t_6 \leftarrow -t_3 + m_5 - m_8 + m_{11}$;
    $t_7 \leftarrow t_1 + m_0 - m_3 - m_9$; $t_8 \leftarrow m_1 - a_{24}$;
    $w_0 \leftarrow t_0 - t_2 + m_5 - m_9$;
    $w_1 \leftarrow t_0 + t_2 + m_3 - m_7 + m_{11} + a_{24}$;
    $w_2 \leftarrow t_4 + t_5$; $w_3 \leftarrow t_4 - t_5 - t_8 + m_7$;
    $w_4 \leftarrow t_6 + t_7$; $w_5 \leftarrow t_6 - t_7 + t_8 - m_4 + m_7 - m_{10}$;
end
return $(w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2)$

Algoritmo 3.23: Multiplicación en $\mathbb{F}_{3^{6m}}$ Caso 2

Entrada: $U = u_0 + u_1\sigma + u_2\rho - \rho^2, V = v_0 + v_1\sigma + v_2\rho - \rho^2 \in \mathbb{F}_{3^{6m}}$
Salida: $W = UV \in \mathbb{F}_{3^{6m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_0 + u_2$; $a_2 \leftarrow u_1 + u_2$;
2 $a_3 \leftarrow v_0 + v_1$; $a_4 \leftarrow v_0 + v_2$; $a_5 \leftarrow v_1 + v_2$;
3 $a_6 \leftarrow u_2 + v_2$;
4 $m_0 \leftarrow u_0v_0$; $m_1 \leftarrow u_1v_1$; $m_2 \leftarrow u_2v_2$;
5 $m_3 \leftarrow a_0a_3$; $m_4 \leftarrow a_1a_4$; $m_5 \leftarrow a_2a_5$;
6 $w_0 \leftarrow m_0 - m_1 - ba_6$;
7 $w_1 \leftarrow m_3 - m_0 - m_1$;
8 $w_2 \leftarrow m_4 - m_0 - m_2 - a_6 + b$;
9 $w_3 \leftarrow m_5 - m_1 - m_2$;
10 $w_4 \leftarrow 1 + m_2 - u_0 - v_0$;
11 $w_5 \leftarrow -u_1 - v_1$;
12 **return** $(w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2)$

4

IMPLEMENTACIÓN DEL EMPAREJAMIENTO η_T

El trabajo ayuda siempre, pués trabajar
no es realizar lo que uno imaginaba, sino
descubrir lo que uno tiene dentro.

Boris Pasternak (1890 – 1960)

Como se mencionó en el capítulo 2, el emparejamiento más eficiente conocido es η_T que fue propuesto por Barreto et al. [6], el cual, de manera general, consiste de un ciclo principal y una operación adicional llamada exponenciación final. Desde entonces se han reportado variantes de este emparejamiento para característica dos y tres. En este capítulo se presentan los algoritmos que fueron implementados en software para cada característica y para cada arquitectura.

La implementación del emparejamiento η_T fue realizada en dos arquitecturas, la primera de ellas es un dispositivo móvil (PDA) que cuenta con recursos limitados, y la segunda es una arquitectura multicore, las características de estas arquitecturas se sumarizan en la sección 3.1. En la PDA la implementación del emparejamiento es completamente secuencial, por otro lado para la arquitectura multinúcleo se realizó una implementación paralela que es capaz de explotar el paralelismo inmerso en los algoritmos para el cálculo del emparejamiento η_T, estos algoritmos son descritos a lo largo de este capítulo.

4.1. ¿Por qué paralelizar el emparejamiento η_T?

Uno de los objetivos en esta tesis fue realizar una implementación para el cálculo del emparejamiento η_T sobre una arquitectura multinúcleo. Hoy en día las necesidades de cómputo de numerosas aplicaciones, más concretamente en esta tesis, el cálculo del emparejamiento η_T, obligan a desarrollar software eficiente y seguro sobre aquitecturas multinúcleo. Además, el auge de los procesadores multinúcleo ha aumentado la difusión del procesamiento paralelo que cada vez está más al alcance del público en general.

En [11, 12] se presentan algoritmos que por su naturaleza ofrecen la ventaja de poder realizar el cálculo del emparejamiento η_T en paralelo. Con el fin obtener una implementación eficiente sobre una arquitectura multinúcleo, durante el periodo julio-septiembre del año 2008 se realizó una estancia en Intel Tecnología de México S.A. de C.V Guadalajara Design Center, donde se utilizaron herramientas proporcionadas por Intel, las cuales permiten mejorar el desempeño de las aplicaciones desarrolladas para arquitecturas multinúcleo. Las herramientas utilizadas para la implementación muticore son:

- Intel® Thread Checker
- Intel® Thread Profiler
- Intel® VTune Performance Analyzer
- Intel® Theadring Building Blocks
- Intel® C++ Compiler

Todas estas herramientas fueron utilizadas, Thread Checkery Thread Profiler son herramientas que nos ayudan a verificar el adecuado funcionamiento de una implementación multihilos, VTune Performance Analyzer es utilizado para análizar el comportamiento de un programa, es decir, la secuencia de llamadas a funciones entre otras cosas, C++ Compiler es utilizado para compilar el código fuente y generar un archivo ejecutable y Theadring Building Block es una biblioteca escrita en lenguaje C++ para construir aplicaciones multihilos. Para más detalles sobre el uso de estas herramientas ver el apéndice B.

4.1.1. ¿Qué puede ser paralelizable en el emparejamiento η_T?

El emparejamiento η_T como fue descrito por Barreto et al. [6] consiste de un ciclo principal de $(m-1)/2$ iteraciones seguido de un exponenciación final, es decir, gran parte del cálculo del emparejamiento η_T para ambas características es realizado de forma iterativa; como ya se mencionó en [11, 12] podemos encontrar algoritmos para realizar

este cálculo donde las iteraciones del emparejamiento presentan una dependencia entre los datos.

Existen dos paradigmas que pueden ser utilizados en la programación paralela que son: flujo de control (*Control-Flow*) y paralelismo en datos (*Data-parallel*) [37]. En este trabajo se utilizó el paradigma de paralelización de datos, que consiste en subdividir el conjunto de datos de entrada en un algoritmo, de tal forma que cada procesador realiza la misma secuencia de operaciones que los otros procesadores sobre un subconjunto de datos asignado.

Para llevar a cabo la paralelización de un algoritmo se deben de seguir los siguientes pasos, descomposición, asignación, comunicación y mapeo [30]. La descomposición consiste en dividir el trabajo en tareas más pequeñas, se continua con la asignación de las tareas en procesos y se revisan aspectos de la comunicación entre estos procesos, y finalmente el mapeo consiste en la asignación física de los procesos en cada procesador con los que cuenta la arquitectura.

La comunicación y sincronización entre las diferentes subtareas son típicamente una limitante para conseguir un buen rendimiendo de los programas en paralelo, para hacer frente a estos problemas, las herramientas Checker, Profile y VTune son utilizadas como apoyo en la implementación del emparejamiento η_T, ver apéndice B.

Como ya se mencionó la paralelización se realizó sobre el ciclo principal del emparejamiento no sobre la exponenciación final, ni tampoco sobre la aritmética en los campos $\mathbb{F}_{2^m}$ y $\mathbb{F}_{3^m}$, ni en la construcción de las torres de campos $\mathbb{F}_{2^{km}}$ y $\mathbb{F}_{3^{km}}$, debido a que se realizaron pruebas sobre la paralelización en la aritmética dando como resultado una implementación ineficiente, pues las instrucciones de creación y sincronización de tareas toma un tiempo aproximado entre 60 y 75 micro-segundos (dependiendo del sistema operativo) y la mayoría de las operaciones en la aritmética son mucho más rápidas, ver Tablas 3.2 y 3.6.

La paralelización realizada en el cálculo del emparejamiento η_T es una paralelización implícita, es decir, la paralelización es realizada por el programador, utilizando dos técnicas, la primera es utilizando la creación y manipulación de hilos y la segunda es utilizando un tecnología de Intel llamada Threading Building Blocks que se encarga de la sincronización de tareas.

A continuación se describen los algoritmos que fueron implementados en software para el cálculo eficiente del emparejamiento η_T en cararacterísticas dos y tres sobre la PDA y la arquitectura multinúcleo.

4.2. Implementación en característica dos

Por razones de eficiencia, se programó una versión diferente tanto para la PDA como para la arquitectura multinúcleo.

Como ya se mencionó en los capítulos anteriores, la curva elíptica supersingular utilizada en característica dos es:

$$y^2 + y = x^3 + x + b, \text{ con } b \in \{0, 1\}$$

con grado de seguridad $k = 4$.

4.2.1. Emparejamiento η_T en característica dos

Para el cálculo del emparejamiento es necesario definir algunos parametros iniciales como se muestra en [12].

Los parametros a definir son $\delta = 1$ cuando $m \equiv 1, 7 \bmod 8$; en otro caso $\delta = 1 - b$ y $\overline{\delta} = 1 - \delta$. Hay que mencionar que el número de puntos racionales en la curva esta dado por: $N = \#E(\mathbb{F}_{2^m}) = 2^m + 1 + \nu 2^{(m+1)/2}$, con $\nu = (-1)^\delta$ [8] y el grado de seguridad para la curva utilizada es $k = 4$, tal que $N|2^{km} - 1$. Otros parametros necesarios en los algoritmos 4.1 y 4.2 son α, β y γ, que son definidos de la siguiente manera [12]:

$$\alpha = \begin{cases} 0 & \text{si } m \equiv 3 \bmod 4 \\ 1 & \text{si } m \equiv 1 \bmod 4 \end{cases} \quad \beta = \begin{cases} b & \text{si } m \equiv 1, 3 \bmod 8 \\ 1 - b & \text{si } m \equiv 5, 7 \bmod 8 \end{cases}$$

$$\gamma = \begin{cases} 0 & \text{si } m \equiv 1, 7 \bmod 8 \\ 1 & \text{si } m \equiv 3, 5 \bmod 8 \end{cases}$$

hay que notar que $\gamma + \delta = b$.

El emparejamiento implementado en la PDA, es una versión del emparejamiento η_T que no utiliza raíces cuadradas reportada en [12], este algoritmo es completamente secuencial debido a que existe una fuerte dependencia entre los cálculos intermedios y la mayoría de las operaciones utilizadas son multiplicaciones y elevadas al cuadrado. El campo base utilizado para la PDA es $\mathbb{F}_{2^{233}}$, donde el polinomio irreducible es $P(x) = x^{233} + x^{74} + 1$.

El algoritmo 4.1 fue implementado en software como parte de la biblioteca para el cálculo eficiente del emparejamiento η_T sobre la PDA. Hay que mencionar que dentro de este algoritmo se pueden distinguir algunas partes principales como es el paso 9, que recibe el nombre de primera multiplicación (*first multiplication*), como se presenta en [12], se continua con un ciclo principal de $(m - 1)/2$ iteraciones y finalmente en el paso

19 se realiza la exponenciación final.

Este algoritmo no puede ser paralelizado debido a que la parte deseable a paralelizar es el ciclo principal, observemos que el paso 17 realiza el producto $F \leftarrow FG$ en cada iteración, y la siguiente iteración necesita calcular en el paso 11 $F \leftarrow F^2$, aquí se presenta una dependencia de datos que nos impide la paralelización de este algoritmo.

El algoritmo 4.1 tiene la ventaja de no utilizar la raíz cuadrada pero su principal desventaja es que no puede ser paralelizado de manera significativa, debido a esto el algoritmo 4.2 fue programado en la arquitectura multinúcleo, en este algoritmo no existe una dependencia de los resultados intermedios. El algoritmo se describe a continuación.

Algoritmo 4.1: Emparejamiento η_T, sin raíces cuadradas (PDA)

Entrada: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{2^m}) : \ y^2 + y = x^3 + x + b \quad (b \in \{0, 1\})$

Salida: $g = \eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$

1 $y_p \leftarrow y_p + \overline{\delta}$;
2 $x_p \leftarrow x_p^2$; $y_p \leftarrow y_p^2$;
3 $y_p \leftarrow y_p + b$; $u \leftarrow x_p + 1$;
4 $g_1 \leftarrow u + x_q$; $g_0 \leftarrow x_p x_q + y_p + y_q + g_1$;
5 $x_q \leftarrow x_q + 1$;
6 $g_2 \leftarrow x_p^2 + x_q$;
7 $G \leftarrow g_0 + g_1 s + t$;
8 $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$;
9 $F \leftarrow LG$; // Alg. 4.4
10 **for** $i \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
11 $\quad F \leftarrow F^2$;
12 $\quad x_q \leftarrow x_q^4$; $y_q \leftarrow y_q^4$;
13 $\quad x_q \leftarrow x_q + 1$; $y_q \leftarrow y_q + x_q$;
14 $\quad g_0 \leftarrow u x_q + y_p + y_q$;
15 $\quad g_1 \leftarrow x_p + x_q$;
16 $\quad G \leftarrow g_0 + g_1 s + t$;
17 $\quad F \leftarrow FG$; // Alg. 3.11
18 **end**
19 **return** $F^{(2^{2m}-1)(2^m+1-\nu 2^{\frac{m+1}{2}})}$; // Alg. 4.5

En el algoritmo 4.2 se puede observar que la región a paralelizar es el ciclo principal dividiendo las $(m-1)/2$ iteraciones en 2 ciclos más pequeños que realicen la mitad de iteraciones respectivamente. El algoritmo 4.3 es un versión paralelizada para el cálculo

del emparejamiento η_T en característica dos, éste algoritmo puede ser implementado en un arquitectura que cuenta con dos procesadores.

Algoritmo 4.2: Emparejamiento η_T, con raíces cuadradas (multinúcleo)

Entrada: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{2^m}) : \; y^2 + y = x^3 + x + b \quad (b \in \{0, 1\})$

Salida: $g = \eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$

1 $y_p \leftarrow y_p + \overline{\delta}$;
2 $u \leftarrow x_p + \alpha$; $v \leftarrow x_q + \alpha$;
3 $g_0 \leftarrow uv + y_p + y_q + \beta$;
4 $g_1 \leftarrow u + x_q$; $g_2 \leftarrow v + x_p^2$;
5 $G \leftarrow g_0 + g_1 s + t$;
6 $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$;
7 $F \leftarrow LG$; // Alg. 4.4
8 **for** $i \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
9 $\quad x_p \leftarrow \sqrt{x_p}$; $y_p \leftarrow \sqrt{y_p}$;
10 $\quad x_q \leftarrow x_q^2$; $y_q \leftarrow y_q^2$;
11 $\quad u \leftarrow x_p + \alpha$; $v \leftarrow x_q + \alpha$;
12 $\quad g_0 \leftarrow uv + y_p + y_q + \beta$;
13 $\quad g_1 \leftarrow u + x_q$;
14 $\quad G \leftarrow g_0 + g_1 s + t$;
15 $\quad F \leftarrow FG$; // Alg. 3.11
16 **end**
17 **return** $F^{(2^{2m}-1)(2^m+1-\nu 2^{\frac{m+1}{2}})}$; // Alg. 4.5

En el algoritmo 4.3 se agregaron variables ($T_{x_p}, T_{y_p}, T_{x_q}$ y T_{y_q}) que sirven para representar las tablas que guardan los valores de las $(m-1)/2$ raíces del punto P y las $(m-1)/2$ elevadas al cuadrado del punto Q que son necesarias para el cálculo del emparejamiento η_T. También hay que notar que se agregó una multiplicación (paso 20) que se encarga de multiplicar los resultados obtenidos por cada ciclo, la razón es porque si observamos el algoritmo 4.2 en el paso 15 se van almacenando los productos en la variable $F = FG_1G_2 \ldots G_{\frac{m-1}{2}}$ y al paralelizar el cálculo en dos ciclos como se muestra en el algoritmo 4.3, el ciclo de la izquierda obtiene el valor de $F_1 = F_1G_1G_2 \ldots G_{\frac{m-1}{4}}$ mientras que el ciclo de la derecha obtiene $F_2 = G_{\frac{m-1}{4}+1} \ldots G_{\frac{m-1}{2}}$.

Algoritmo 4.3: Emparejamiento η_T, con raíces cuadradas (versión paralela).

Entrada: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{2^m}) : \; y^2 + y = x^3 + x + b \quad (b \in \{0, 1\})$

Salida: $g = \eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$

1 $y_p \leftarrow y_p + \overline{\delta}$;
2 $u \leftarrow x_p + \alpha$; $v \leftarrow x_q + \alpha$;
3 $g_0 \leftarrow uv + y_p + y_q + \beta$;
4 $g_1 \leftarrow u + x_q$; $g_2 \leftarrow v + x_p^2$;
5 $G \leftarrow g_0 + g_1 s + t$;
6 $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$;
7 $F_1 \leftarrow LG$; // Alg. 4.4
8 $T_{x_p}[0] \leftarrow x_p$; $T_{y_p}[0] \leftarrow y_p$; $T_{x_q}[0] \leftarrow x_q$; $T_{y_q}[0] \leftarrow y_q$; $F_2 \leftarrow 1$;
/* Los valores de x_p, y_p, x_q y y_q son pre-calculados y almacenados en su tabla correspondiente. */
9 **for** $i \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
10 $T_{x_p}[i] \leftarrow \sqrt{T_{x_p}[i-1]}$; $T_{y_p}[i] \leftarrow \sqrt{T_{y_p}[i-1]}$;
11 $T_{x_q}[i] \leftarrow (T_{x_q}[i-1])^2$; $T_{y_q}[i] \leftarrow (T_{y_q}[i-1])^2$;
12 **end**
/* El ciclo principal es dividido y paralelizado en 2 ciclos más pequeños. */

13 **for** $i \leftarrow 1$ **to** $\frac{m-1}{4}$ **do**	**for** $j \leftarrow \frac{m-1}{4} + 1$ **to** $\frac{m-1}{2}$ **do**
14 $u \leftarrow T_{x_p}[i] + \alpha$; $v \leftarrow T_{x_q}[j] + \alpha$;	$u' \leftarrow T_{x_p}[j] + \alpha$; $v' \leftarrow T_{x_q}[i] + \alpha$;
15 $g_0 \leftarrow uv + T_{y_p}[i] + T_{y_q}[i] + \beta$;	$g'_0 \leftarrow u'v' + T_{y_p}[j] + T_{y_q}[j] + \beta$;
16 $g_1 \leftarrow u + T_{x_q}[i]$;	$g'_1 \leftarrow u' + T_{x_q}[j]$;
17 $G \leftarrow g_0 + g_1 s + t$;	$G' \leftarrow g'_0 + g'_1 s + t$;
18 $F_1 \leftarrow F_1 G$;	$F_2 \leftarrow F_2 G'$; // Alg. 3.11
19 **end**	**end**

/* Se agrega una multiplicación entre los resultados obtenidos por cada ciclo. */
20 $F \leftarrow F_1 F_2$; // Alg. 3.10
21 **return** $F^{(2^{2m}-1)(2^m+1-\nu 2^{\frac{m+1}{2}})}$; // Alg. 4.5

Debemos de prestar atención en el ciclo de la derecha del algoritmo 4.3 pues en su primera iteración en el paso 18 se cálcula el valor $F_2 = F_2 G_{\frac{m-1}{4}+1}$, donde en F_2 comienza inicializada en 1, aquí no es necesario realizar la multiplicación por completo dado que $F_2 = F_2 G_{\frac{m-1}{4}+1} = (1) G_{\frac{m-1}{4}+1}$, para las iteraciones siguientes se realiza la multiplicación con el algoritmo especificado.

Idealmente, la ejecución en paralelo de las operaciones, resulta en una aceleración neta global que se aproxima idealmente a un factor de 2, que se ve reflejada en el cálculo del emparejamiento η_T, ver las Tablas de resultados 4.2 y 4.3.

En esta tesis se desarrollaron implementaciones paralelas para el cálculo del emparejamiento η_T que hacen uso desde 2 hasta 8 procesadores. Para realizar el cálculo del emparejamiento en paralelo con más procesadores, el ciclo principal es dividido. En la Figura 4.1 se muestra de manera gráfica cómo se realiza el cálculo del emparejamiento η_T desde su forma serial hasta la paralelización en 8 núcleos, en la figura se indíca el número de iteraciones que realiza cada ciclo. Hay que mencionar que los ciclos son distribuidos en diferentes hilos con el fin de que se ejecuten en forma paralela.

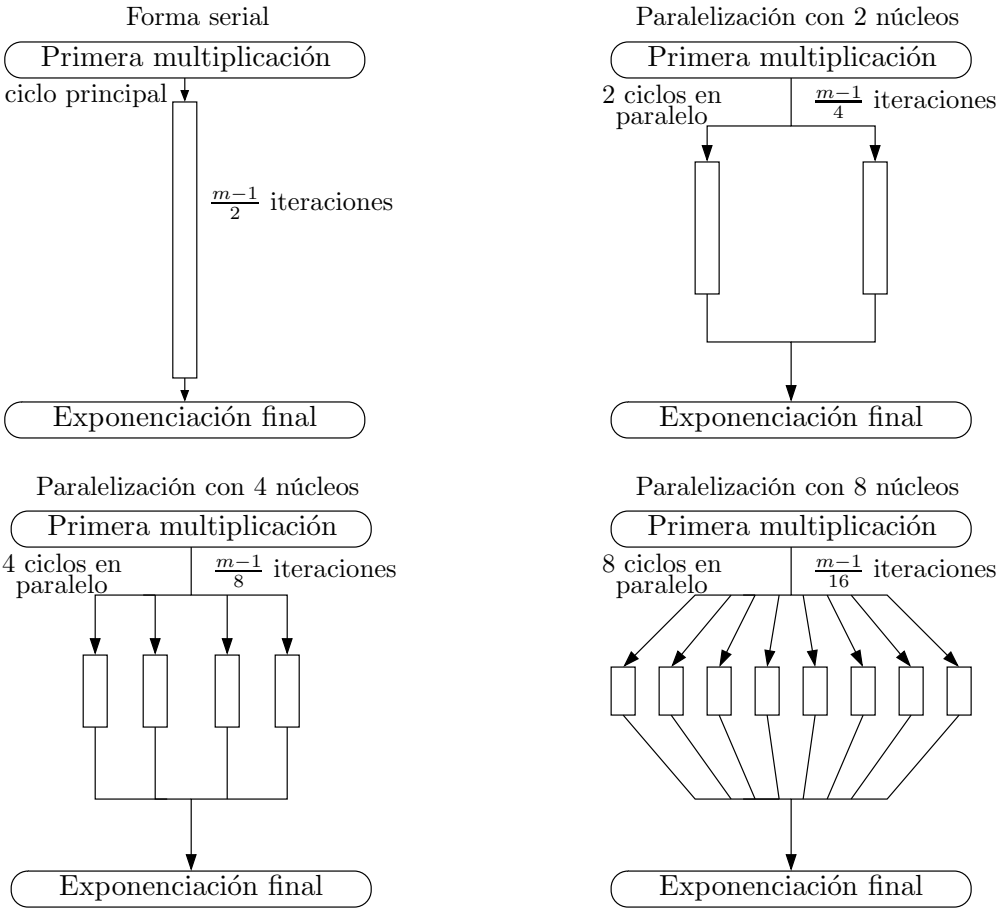


Figura 4.1: Paralelización del emparejamiento η_T.

De la implementación realizada para el cálculo en forma paralela puede obtenerse

una granularidad[1] fina (*Fine-grain*) o granularidad gruesa (*Coarse-grain*), pues el grano fino es obtenido cuando se tiene un número de tareas grande involucrando una mayor sincronización y comunicación entre las tareas, como es el caso de la implementación dedicada para 8 procesadores. El grano grueso se refiere a un número de tareas pequeño y cuenta con poca comunicación y sincronización, como cuando se realiza el cálculo del emparejamiento η_T con sólo dos procesadores.

Tanto para el algoritmo implementado sobre la PDA y el algoritmo sobre la arquitectura multinúcleo, son necesarias dos operaciones más, la primera multiplicación (*first multiplication*) y la exponenciación final, estas operaciones son explicadas en las secciones siguientes.

4.2.2. Primera multiplicación en característica dos

La primera multiplicación puede ser calculada de la siguiente manera:

$$\begin{aligned}(g_0 + g_1 s + t)((g_0 + g_2) + (g_1 + 1)s + t) \;=\; & g_0(g_0 + g_2) + g_1^2 + g_1 + \\ & (g_0 + g_1 g_2 + g_1^2 + g_1 + 1)s + \\ & (g_2 + 1)t + st\end{aligned}$$

El algoritmo 4.4 fue implementado para realizar la primera multiplicación necesaria en los algoritmos de emparejamiento η_T que se mencionaron anteriormente.

Algoritmo 4.4: Primera multiplicación en característica dos [12].

Entrada: $U = (g_0 + g_1 s + t), V = ((g_0 + g_2) + (g_1 + 1)s + t) \in \mathbb{F}_{2^{4m}}$
Salida: $W = UV \in \mathbb{F}_{2^{4m}}$

1 $s_0 \leftarrow g_1^2$;
2 $a_0 \leftarrow g_0 + g_2$; $a_1 \leftarrow g_1 + s_0$;
3 $m_0 \leftarrow g_0 a_0$; $m_1 \leftarrow g_1 g_2$;
4 $w_0 \leftarrow m_0 + a_1$; $w_1 \leftarrow m_1 + g_0 + a_1 + 1$;
5 $w_2 \leftarrow g_2 + 1$;
6 $w_3 \leftarrow 1$;
7 **return** $w_0 + w_1 s + w_2 t + w_3 st$

El costo de este algoritmo son 2 multiplicaciones, 1 elevación al cuadrado y 5 sumas en el campo $\mathbb{F}_{2^m}$.

[1] Granularidad (*Granularity*): en la programación paralela, de manera general se divide en paralelismo de grano fino y grano grueso y es una medida cualitativa de la relación entre el cómputo y la comunicación. [30]

4.2.3. Exponenciación final en característica dos

La exponenciación final es utilizada al final del cálculo del emparejamiento η_T como se muestra en [6]. Esta operación consiste en obtener la potencia del emparejamiento $\eta_T(P,Q)^M$, donde M en característica dos es definido como sigue:

$$M = \frac{q^k - 1}{N}$$

donde $q = 2^m$ es el campo, k el grado de seguridad de la curva supersingular utilizada, y $N = \#E(F_q)$ el el número de puntos que satisfacen la curva elíptica, entonces:

$$M = \frac{2^{4m} - 1}{2^m + 1 + \nu 2^{(m+1)/2}} = (2^{2m} - 1)(2^m + 1 - \nu 2^{\frac{m+1}{2}})$$

El algoritmo 4.5 es propuesto en [12] y es utilizado para realizar esta operación.

Algoritmo 4.5: Exponenciación final en característica dos [12].

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}^*_{2^{4m}}$
Salida: $V = U^M \in \mathbb{F}^*_{2^{4m}}$ con $M = (2^{2m} - 1)(2^m + 1 - \nu 2^{\frac{m+1}{2}})$

1 $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_1^2$; $m_2 \leftarrow u_2^2$; $m_3 \leftarrow u_3^2$;
2 $T_0 \leftarrow (m_0 + m_1) + m_1 s$; $T_1 \leftarrow (m_2 + m_3) + m_3 s$;
3 $T_2 \leftarrow m_3 + m_2 s$; $T_3 \leftarrow (u_0 + u_1 s)(u_2 + u_3 s)$;
4 $T_4 \leftarrow (T_0 + T_2)$; $D \leftarrow T_3 + T_4$;
5 $D \leftarrow D^{-1}$; // Alg. 3.7
6 $T_5 \leftarrow T_1 D$; $T_6 \leftarrow T_4 D$;
7 $V_0 \leftarrow T_5 + T_6$;
8 $V_1, W_1 \leftarrow T_5$;
9 **if** $\nu = -1$ **then**
10 $W_0 \leftarrow V_0$;
11 **else**
12 $W_0 \leftarrow T_6$;
13 **end**
14 $V \leftarrow V_0 + V_1 t$; $W \leftarrow W_0 + W_1 t$;
15 $V \leftarrow V^{2^m+1}$;
16 **for** $i \leftarrow 1$ **to** $\frac{m+1}{2}$ **do**
17 $W \leftarrow W^2$;
18 **end**
19 **return** VW

Observemos que en el algoritmo 4.5 las variables $T_i's, V_i's, W_i's$ y D pertenecen al campo $\mathbb{F}_{2^{2m}}$ mientras que V y W pertenecen a $\mathbb{F}_{2^{4m}}$. También hay que notar que en el paso

15 es necesaria una exponenciación de la forma V^{2^m+1} que es realizada como sigue. Sea $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$ y utilizando las propiedades de la base de la siguiente forma: $s^{2^m} = s + 1$ y $t^{2^m} = t + s + \alpha + 1$ obtenemos que:

$$U^{2^m} = \begin{cases} (u_0 + u_1 + u_3) + (u_1 + u_2)s + (u_2 + u_3)t + u_3 st & \text{si } \alpha = 1 \\ (u_0 + u_1 + u_2) + (u_1 + u_2 + u_3)s + (u_2 + u_3)t + u_3 st & \text{si } \alpha = 0 \end{cases}$$

y como se propone en [12] podemos obtener $U^{2^m+1} = U^{2^m} U$ obteniendo el siguiente resultado:

$$U^{2^m+1} = \begin{cases} (u_0u_1 + u_0u_3 + u_1u_2 + (u_0 + u_1)^2) + \\ ((u_0 + u_1)(u_2 + u_3) + u_0u_3 + u_2u_3 + (u_2 + u_3)^2)s & \text{si } \alpha = 1 \\ (u_0u_3 + u_1u_2 + u_2u_3 + (u_2 + u_3)^2)t + (u_2u_3 + (u_2 + u_3)^2)st \\ \hline ((u_0 + u_1)(u_2 + u_3) + u_0u_1 + u_0u_3 + (u_0 + u_1)^2) + \\ ((u_0 + u_1)(u_2 + u_3) + u_1u_2 + u_2u_3 + (u_2 + u_3)^2)s + & \text{si } \alpha = 0 \\ (u_0u_3 + u_1u_2)t + (u_2u_3 + (u_2 + u_3)^2)st \end{cases}$$

Con el resultado anterior obtenemos el algoritmo 4.6.

Algoritmo 4.6: Cálculo de U^{2^m+1} en $\mathbb{F}_{2^{4m}}$ [12].

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}^*_{2^{4m}}$
Salida: $V = U^{2^m+1} \in \mathbb{F}_{2^{4m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_2 + u_3$;
2 $m_0 \leftarrow a_0 a_1$; $m_1 \leftarrow u_0 u_1$; $m_2 \leftarrow u_0 u_3$;
3 $m_3 \leftarrow u_1 + u_2$; $m_4 \leftarrow u_2 + u_3$;
4 $s_0 \leftarrow a_0^2$; $s_1 \leftarrow a_1^2$;
5 $v_2 \leftarrow m_2 + m_3$;
6 $v_3 \leftarrow m_4 + s_1$;
7 **if** $\alpha = 1$ **then**
8 $\quad v_1 \leftarrow v_3 + m_0 + m_2$;
9 $\quad v_0 \leftarrow v_2 + m_1 + s_0$;
10 $\quad v_2 \leftarrow v_2 + v_3$;
11 **else**
12 $\quad v_1 \leftarrow v_3 + m_0 + m_3$;
13 $\quad v_0 \leftarrow m_0 + m_1 + m_2 + s_0$;
14 **end**
15 **return** $v_0 + v_1 s + v_2 t + v_3 st$

El costo de algoritmo 4.6 son 5 multiplicaciones, 2 elevaciones al cuadrado y 14 sumas sobre el campo $\mathbb{F}_{2^m}$.

4.3. Resultados del emparejamiento η_T en característica dos

En esta sección se presentan los costos en tiempo asociados al cálculo del emparejamiento η_T incluyendo la exponenciación final, tanto en la PDA como la arquitectura multinúcleo. Las características de las arquitecturas se mencionan en la sección 3.1.

En la Tabla 4.1 se presentan los tiempos obtenidos para el cálculo del emparejamiento η_T correspondiente al algoritmo 4.1, donde el campo base es $\mathbb{F}_{2^{233}}$. Esta misma implementación se ejecutó en una computadora laptop.

Operación	PDA	Laptop
Emparejamiento η_T	29.5 $ms.$	1.075 $ms.$

Tabla 4.1: Tiempo del emparejamiento η_T en característica dos (PDA).

En las Tablas 4.2 y 4.3 se sumarizan los resultados obtenidos para la implementación en la arquitectura multinúcleo sobre el campo $\mathbb{F}_{2^{503}}$. La Tabla 4.2 presenta los resultados obtenidos con un sistema operativo de 32 bits (IA-32) mientras que en la Tabla 4.3 los resultados son obtenidos bajo un sistema operativo de 64 bits (x64).

# Procesadores	Hilos	Hilos+Eventos	TBB
1-Serial	3.891 $ms.$		
2	2.453 $ms.$	2.265 $ms.$	2.243 ms
4	1.709 $ms.$	1.4156 $ms.$	1.406 ms
8	1.565 $ms.$	1.056 $ms.$	1.109 ms

Tabla 4.2: Tiempos del emparejamiento η_T en característica dos (IA-32).

# Procesadores	Hilos	Hilos+Eventos	TBB
1-Serial	3.80 $ms.$		
2	2.433 ms	2.19ms	2.173ms
4	1.68 ms	1.396 ms	1.387ms
8	1.51 ms	1.043 ms	1.096ms

Tabla 4.3: Tiempos del emparejamiento η_T en característica dos (x64).

En las tablas anteriores se observan 3 columnas, la primera titulada "Hilos" se refiere a una implementación donde la paralelización es realizada mediante la creación y terminación de los hilos, en la segunda columna "Hilos+Eventos" hace referencia a una

implementación donde sólo es necesaria la creación de hilos y la sincronización es realizada mendiante eventos y la tercer columna "TBB" es una implementación utilizando la tecnología Threading Building Blocks de Intel, ver apéndice B.

Como se puede observar en las Tablas 4.2 y 4.3, las técnicas Hilos+Eventos y TBB ofrecen resultados más eficientes comparadas con la técnica que sólo utiliza Threads.

En la Figura 4.2 se representa de manera gráfica el comportamiento del tiempo para el cálculo del emparejamiento η_T en relación con el número de procesadores.

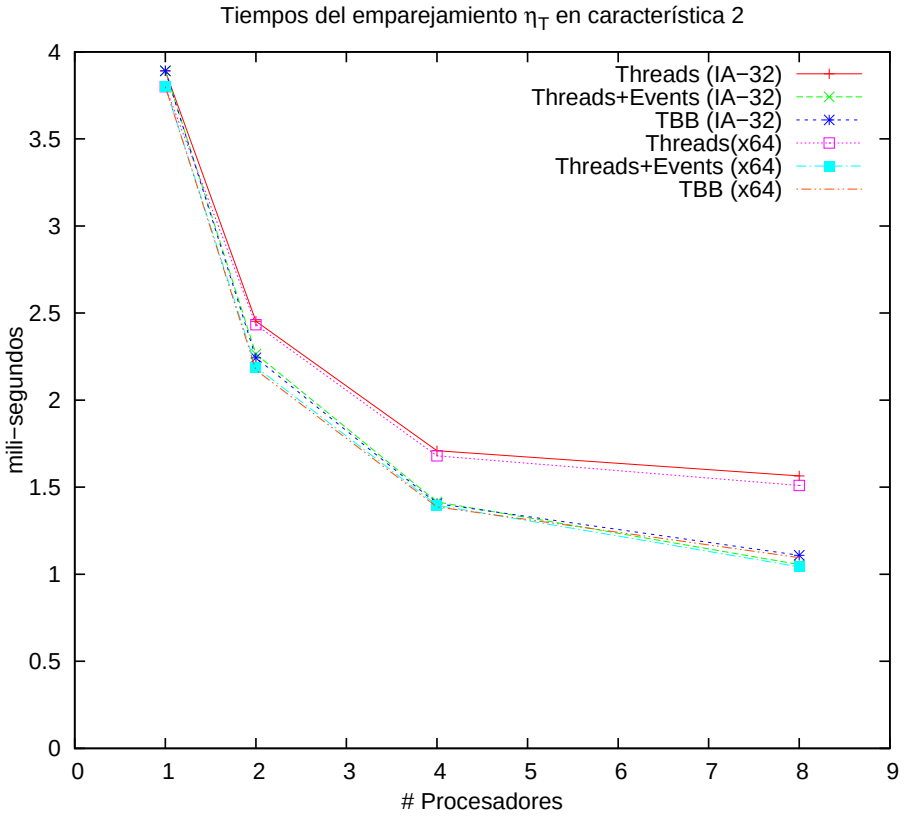


Figura 4.2: Relación tiempo-número de procesadores del emparejamiento η_T en característica dos (multinúcleo).

En la Figura 4.2, observamos una reducción del tiempo para el cálculo del emparejamiento η_T en característica dos conforme el número de procesadores se incrementa, consiguiendo aproximadamente un factor de aceleración dos. Observamos que la eficiencia obtenida con 8 procesadores es ligeramente mayor a la eficiencia resultante con 4 procesadores, sin embargo, actualemente el costo de adquirir una arquitectura con 8 procesadores es altamente elevado. Con ésto puntualizamos que la mejor relación cos-

to/beneficio es lograda al utilizar una arquitectura con 4 procesadores.

4.4. Implementación en característica tres

En esta sección se describen los algoritmos para el cálculo del emparejamiento η_T en característica tres y los detalles de su implementación en software.

4.4.1. Emparejamiento η_T en característica tres

En el 2007 Beuchat et al. [14] propusieron un algoritmo eficiente para el cálculo del emparejamiento η_T en $\mathbb{F}_{3^m}$ sin utilizar la raíz cúbica, su algoritmo es resultado de una mejora del algoritmo presentado por Barreto et al. [6]. Beuchat et al. en [10] proponen un algoritmo que consiste de la combinación del emparejamiento η_T y la exponenciación final. Despues Beuchat et al. en [11] presentan una mejora a los algoritmos presentados en [14, 10], siendo estos últimos los algoritmos de mayor interés en esta tesis.

Primero hay que definir algunos parametros necesarios en los algoritmos 4.7 y 4.9 para el cálculo del emparejamiento η_T que podemos encontrar en [14]. La curva elíptica supersingular utilizada en característica tres es $E(\mathbb{F}_{3^m}) : y^2 = x^3 - x + b$ donde $b \in \{-1, 1\}$ y el número de puntos que satiface la curva esta dado por $N = \#E(\mathbb{F}_{3^m}) = 3^m + 1 + \mu b 3^{\frac{m+1}{2}}$ [8] con:

$$\mu = \begin{cases} +1 & \text{si } m \equiv 1, 11 \bmod 12 \\ -1 & \text{si } m \equiv 5, 7 \bmod 12 \end{cases}$$

y el grado de seguridad es $k = 6$, otros parámetros utilizados son λ y ν que de acuerdo a [11] de la siguiente forma:

$$\lambda = (-1)^{\frac{m+1}{2}} = \begin{cases} +1 & \text{si } m \equiv 7, 11 \bmod 12 \\ -1 & \text{si } m \equiv 1, 5 \bmod 12 \end{cases}$$

$$\nu = \mu\lambda = \begin{cases} +1 & \text{si } m \equiv 5, 11 \bmod 12 \\ -1 & \text{si } m \equiv 1, 7 \bmod 12 \end{cases}$$

Con estos parametros definidos podemos continuar con la implementación del emparejamiento η_T en característica tres.

En la PDA se implementó un algoritmo que no utiliza la raíz cúbica para el cálculo del emparejamiento η_T en característica tres. El algoritmo 4.7 fue implementado en software como parte de la biblioteca para la PDA. Se puede observar que este algoritmo consta de un ciclo principal y de la exponenciación final, cabe señalar que el paso 4 recibe el nombre de "primera multiplicación" (*first multiplication*) [11] y puede ser calculado de manera eficiente como se muestra en seguida.

Algoritmo 4.7: Emparejamiento η_T unrolled loop, sin raíz cúbica (característica tres) [11].

Entrada: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})$
Salida: $\eta_T(P, Q) \in \mathbb{F}^*_{3^{6m}}$

1 $y_p \leftarrow -\mu b y_p$;
2 $x_q \leftarrow x_q^3$; $y_q \leftarrow y_q^3$;
3 $t \leftarrow x_p + x_q + b$;
4 $R \leftarrow (\lambda y_p t - \lambda y_q \sigma - \lambda y_p \rho)(-t^2 + y_p y_q \sigma - t\rho - \rho^2)$; // Alg. 4.11
5 $d \leftarrow b$;
6 **for** $i \leftarrow 1$ **to** $\frac{m-1}{4}$ **do**
7 $R \leftarrow R^9$;
8 $d \leftarrow d - b \bmod 3$;
9 $x_q \leftarrow x_q^9$; $y_q \leftarrow y_q^9$;
10 $t \leftarrow x_p + x_q + d$; $u \leftarrow y_p y_q$;
11 $S \leftarrow (-t^2 - u\sigma - t\rho - \rho^2)^3$; // Alg. 4.8
12 $d \leftarrow d - b \bmod 3$;
13 $x_q \leftarrow x_q^9$; $y_q \leftarrow y_q^9$;
14 $t \leftarrow x_p + x_q + d$; $u \leftarrow y_p y_q$;
15 $S' \leftarrow -t^2 + u\sigma - t\rho - \rho^2$;
16 $S \leftarrow SS'$; // Alg. 3.23
17 $R \leftarrow RS$; // Alg. 3.21
18 **end**
19 **return** $R^{(3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}})}$; // Alg. 4.12

El algoritmo 4.7 requiere sólo de $\frac{m-1}{4}$ iteraciones en el loop principal, ver [11], cabe mencionar que la primera multiplicación del paso 4 pueder ser obtenida de forma eficiente, utilizando el algoritmo 4.11, por otro lado el cubo del paso 11 es especial debido a que se trata de un elemento en $\mathbb{F}_{3^{6m}}$ con algunos coeficientes en 0, este cálculo es realizado de la siguiente forma: Sea $U = (-t^2 + u\sigma - t\rho - \rho^2)^3$ y de acuerdo al análisis realizado en la sección 3.5.3.2 obtenemos que:

$$V = U^3 = v_0 + v_1\sigma + v_2\rho - \rho^2$$

donde:

$$\begin{aligned} v_0 &= -t^6 - bt^3 - 1 \\ v_1 &= -u^3 \\ v_2 &= -t^3 + b \end{aligned}$$

El algoritmo 4.8 fue implementado en software para realizar el cálculo de $(-t^2 + u\sigma - t\rho - \rho^2)^3$.

Algoritmo 4.8: Cubo: $(-t^2 + u\sigma - t\rho - \rho^2)^3$

Entrada: $t, u \in \mathbb{F}_{3^m}$
Salida: $V = (-t^2 + u\sigma - t\rho - \rho^2)^3 \in \mathbb{F}^*_{3^{6m}}$

1 $c_0 \leftarrow t^3$; $c_1 \leftarrow -u^3$;
2 $m_0 \leftarrow c_0^2$;
3 $v_0 \leftarrow -m_0 - bc_0 - 1$;
4 $v_1 \leftarrow c_1$;
5 $v_2 \leftarrow b - c_0$;
6 **return** $v_0 + v_1\sigma + v_2\rho - \rho^2$

Para la arquitectura multinúcleo se implementó el algoritmo 4.9, el cual utiliza la raíz cúbica, pero a diferencia del algoritmo utilizado en la PDA, éste tiene la característica de que el loop principal puede ser paralelizado, debido a que no existe una fuerte dependencia de datos en el algoritmo.

Si analizamos el algoritmo 4.9, observamos que los primeros 3 pasos se deben de realizar de forma secuencial, mientras que el loop principal puede ser paralelizado, dado que en el paso 9 en cada iteración es obtenido como el producto $R \leftarrow RS$, donde S depende de los pasos 5,6,7 y 8 y podemos ver que todos estos pasos no dependen de las iteraciones anteriores, más precisamente S depende directamente de los valores que pueden tomar las variable x_p, y_p, x_q y y_q en cada iteración.

Algoritmo 4.9: Emparejamiento η_T reversed loop, con raíz cúbica (característica tres) [11].

Entrada: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})$
Salida: $\eta_T(P, Q) \in \mathbb{F}^*_{3^{6m}}$

1 $y_p \leftarrow -\mu b y_p$;
2 $t \leftarrow x_p + x_q - \nu b$;
3 $R \leftarrow (\lambda y_p t + y_q\sigma - \lambda y_p\rho)(-t^2 - \lambda y_p y_q\sigma - t\rho - \rho^2)$; // Alg. 4.11
4 **for** $i \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
5 $x_p \leftarrow \sqrt[3]{x_p}$; $y_p \leftarrow \sqrt[3]{y_p}$;
6 $x_q \leftarrow x_q^3$; $y_q \leftarrow y_q^3$;
7 $t \leftarrow x_p + x_q - \nu b$; $u \leftarrow y_p y_q$;
8 $S \leftarrow -t^2 - \lambda u\sigma - t\rho - \rho^2$;
9 $R \leftarrow RS$; // Alg. 3.22
10 **end**
11 **return** $R^{(3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}})}$; // Alg. 4.12

Observemos que al terminar el ciclo principal R es generado de la siguiente forma $R = RS_1S_2S_2 \ldots S_{\frac{m-1}{2}}$, donde cada valor de S_i es obtenido en cada iteración dentro del

ciclo con $1 \leq i \leq \frac{m-1}{2}$.

Si se pre-calculan todos los valores posibles que pueden tomar x_p, y_p, x_q y y_q en cada iteración, entonces podemos paralelizar el cálculo del emparejamiento. Una aproximación de esta idea se presenta en el algoritmo 4.10.

Algoritmo 4.10: Emparejamiento η_T reversed loop, con raíz cúbica (característica tres, paralelizado en 2 tareas).

Entrada: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})$
Salida: $\eta_T(P, Q) \in \mathbb{F}^*_{3^{6m}}$

1 $y_p \leftarrow -\mu b y_p$;
2 $t \leftarrow x_p + x_q - \nu b$;
3 $R_1 \leftarrow (\lambda y_p t + y_q \sigma - \lambda y_p \rho)(-t^2 - \lambda y_p y_q \sigma - t\rho - \rho^2)$; // Alg. 4.11
4 $T_{x_p}[0] \leftarrow x_p$; $T_{y_p}[0] \leftarrow y_p$; $T_{x_q}[0] \leftarrow x_q$; $T_{y_q}[0] \leftarrow y_q$; $R_2 \leftarrow 1$;
/* Los valores de x_p, y_p, x_q y y_q son pre-calculados y almacenados en su tabla correspondiente. */
5 **for** $i \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
6 $T_{x_p}[i] \leftarrow \sqrt[3]{T_{x_p}[i-1]}$; $T_{y_p}[i] \leftarrow \sqrt[3]{T_{y_p}[i-1]}$;
7 $T_{x_q}[i] \leftarrow (T_{x_q}[i-1])^3$; $T_{y_q}[i] \leftarrow (T_{y_q}[i-1])^3$;
8 **end**
/* El ciclo principal es dividido y paralelizado en 2 ciclos más pequeños. */

9	**for** $i \leftarrow 1$ **to** $\frac{m-1}{4}$ **do**	**for** $j \leftarrow \frac{m-1}{4} + 1$ **to** $\frac{m-1}{2}$ **do**
10	$t \leftarrow T_{x_p}[i] + T_{x_q}[i] - \nu b$;	$t' \leftarrow T_{x_p}[j] + T_{x_q}[j] - \nu b$;
11	$u \leftarrow T_{y_p}[i] T_{y_q}[i]$;	$u' \leftarrow T_{y_p}[j] T_{y_q}[j]$;
12	$S \leftarrow -t^2 - \lambda u \sigma - t\rho - \rho^2$;	$S' \leftarrow -t'^2 - \lambda u' \sigma - t'\rho - \rho^2$;
13	$R_1 \leftarrow R_1 S$;	$R_2 \leftarrow R_2 S'$; // Alg. 3.22
14	**end**	**end**

/* Se agrega una multiplicación entre los resultados obtenidos por cada ciclo. */
15 $R = R_1 R_2$; // Alg. 3.21
16 **return** $R^{(3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}})}$; // Alg. 4.12

En el algoritmo 4.10 se introdujeron las variables $T_{x_p}, T_{y_p}, T_{x_q}$ y T_{y_q} que corresponden a las tablas de consulta para todos los valores que son necesarios de las coordenadas de los puntos de entrada P, Q. Un detalle que debemos tomar en cuenta se encuentra en el segundo ciclo en su primera iteración, ésta realiza el cálculo $R_2 = R_2 S'$, y en un inicio $R_2 = 1$, es por esto que en la primer iteración en el paso 13 ahorramos multiplicaciones pues: $R_2 = R_2 S' = (1) S' = S'$.

También hay que notar que se agregó una multiplicación en el paso 15, esto es debido a que el primer ciclo calcula el producto $R_1 = R_1 S_1 S_2 S_2 \dots S_{\frac{m-1}{4}}$ y el segundo ciclo obtiene $R_2 = S_{\frac{m-1}{4}+1} S_{\frac{m-1}{4}+2} \dots S_{\frac{m-1}{2}}$ y recordemos que en el algoritmo original 4.9 $R = R S_1 S_2 S_2 \dots S_{\frac{m-1}{2}}$.

El algoritmo 4.10 representa de forma general la paralelización del cálculo del emparejamiento η_T en dos ciclos que fue implementado en la arquitectura multinúcleo, además también este algoritmo puede ser paralalelizado en más de dos ciclos, para poder acelerar el cálculo del emparejamiento η_T debido a que la característica principal del algoritmo 4.9 es que ofrece una paralelización de forma natural por eso fue elegido para ser implementado en la aquitectura multinúcleo.

En esta tesis se realizaron implementaciones para el cálculo del emparejamiento η_T en su forma secuencial que sólo utiliza 1 procesador y versiones paralelas hasta utilizar los 8 procesadores que la arquitectura multinúcleo nos ofrece.

4.4.2. Primera multiplicación en característica tres

La primera multiplicación (*first multiplication*) necesaria en el paso 4 del algoritmo 4.7 y también en el paso 3 del algoritmo 4.9 puede ser obtenida de la siguiente manera:

$$\begin{aligned} W &= w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2 \\ &= (\lambda y_p t - \lambda y_q \sigma - \lambda y_p \rho)(-t^2 + y_p y_q \sigma - t\rho - \rho^2) \end{aligned}$$

Donde los valores para los coeficientes son definidos como sigue [11]:

$$\begin{aligned} w_0 &= -\lambda y_p t^3 + \lambda y_p y_q^2 + b\lambda y_p \\ w_1 &= \lambda y_p^2 y_q t + \lambda y_q t^2 \\ w_2 &= \lambda y_p \\ w_3 &= \lambda y_q t - \lambda y_p^2 y_q \\ w_4 &= 0 \\ w_5 &= \lambda y_q \end{aligned}$$

Es preciso aclarar que la primera multiplicación del algoritmo 4.9 tiene una pequeña diferencia con la del algoritmo 4.7, debido a que sólo aparece el término y_q en lugar de $-\lambda y_q$, sin embargo, esta diferencia no genera costo adicional para su cálculo; el algoritmo 4.11 es utilizado para realizar la primera multiplicación de manera eficiente.

Algoritmo 4.11: Primera multiplicación en característica tres [11].

Entrada: $U = (\lambda y_p t - \lambda y_q \sigma - \lambda y_p \rho), V = (-t^2 + y_p y_q \sigma - t\rho - \rho^2) \in \mathbb{F}_{3^{6m}}$
Salida: $W = UV \in \mathbb{F}_{3^{6m}}$

1 $m_0 \leftarrow y_q t$; $m_1 \leftarrow y_p y_q$; $m_2 \leftarrow y_p m_1$;
2 $a_0 \leftarrow \lambda m_0 + \lambda m_2$;
3 $c_0 \leftarrow t^3$;
4 $m_3 \leftarrow a_0 t$; $m_4 \leftarrow y_p c_0$; $m_5 \leftarrow y_q m_1$;
5 $w_0 \leftarrow -\lambda m_4 + \lambda m_5 + b\lambda y_p$;
6 $w_1 \leftarrow m_3$;
7 $w_2 \leftarrow \lambda y_p$;
8 $w_3 \leftarrow \lambda m_0 - \lambda m_2$;
9 $w_5 \leftarrow \lambda y_q$;
10 **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_5\sigma\rho^2$;

El algoritmo 4.11 fue implementado de forma completamente secuencial y su costo son 6 multiplicaciones, 1 elevación al cubo y 6 sumas sobre el campo $\mathbb{F}_{3^m}$.

4.4.3. Exponenciación final en característica tres

Como se ha presentado, el emparejamiento η_T consiste de un ciclo principal y una exponenciación final, ésta última es necesaria para generar una valor único del emparejamiento bilineal. En el 2007 Masaaki et al. [62] propusieron un algoritmo para acelerar el cálculo de la exponenciación final del emparejamiento η_T en el campo $\mathbb{F}_{3^{6m}}$.

La exponenciación final necesaria para el emparejamiento η_T, es de la forma U^M donde $U \in \mathbb{F}_{3^{6m}}$, y $M = (p^k - 1)/\#E(\mathbb{F}_{3^m})$, es decir:

$$M = \frac{3^{6m} - 1}{\#E(\mathbb{F}_{3^m})} = (3^{3m} - 1)(3^m + 1)(3^m + 1 - \mu b 3^{\frac{m+1}{2}})$$

y la función de elevar a la potencia M es la exponenciación final en el emparejamiento η_T.

Para la exponenciación final se utiliza el torus $T_2(\mathbb{F}_{3^{3m}})$ para comprimir los valores de $\mathbb{F}_{3^{6m}}$ [38] donde cada elemento en $\mathbb{F}^*_{3^{6m}}$ es representado como $U = U_0 + U_1\sigma$ con $U_0, U_1 \in \mathbb{F}_{3^{3m}}$ y su elemento conjugado es $\overline{U} = U_0 - U_1\sigma$, entonces $U\overline{U} = U_0^2 + U_1^2$, por lo tanto $T_2(\mathbb{F}_{3^{3m}})$ puede ser representado como sigue [38]:

$$T_2(\mathbb{F}_{3^{3m}}) = \left\{U_0 + U_1\sigma \in \mathbb{F}^*_{3^{6m}} : U_0^2 + U_1^2 = 1\right\}$$

Con lo anterior, la exponenciación final $U^M = U^{(3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}})}$, puede ser calculada como sigue:

$$U^M = \left\{ \begin{array}{ll} D \cdot E^{-1} & \text{si } b = 1 \\ D \cdot E & \text{si } b = -1 \end{array} \right\}$$

donde $D = C^{3^m+1}$ y $E = C^{3^{(m+1)/2}}$, con $C = B^{3^m+1}$ y finalmente $B = U^{3^{3m}-1}$. El algoritmo 4.12 es resultante del análisis anterior y es utilizado para realizar la exponenciación final, como se muestra en [38].

Algoritmo 4.12: Exponenciación final en característica tres [11].

Entrada: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$
Salida: $V = U^M \in \mathbb{F}^*_{3^{6m}}$ con $M = (3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}})$

1 $V \leftarrow U^{3^{3m}-1}$; // Algoritmo 4.13
2 $V \leftarrow V^{3^m+1}$; // Algoritmo 4.14
3 $W \leftarrow V$;
4 **for** $i \leftarrow 1$ **to** $\frac{m+1}{2}$ **do**
5 $W \leftarrow W^3$;
6 **end**
7 $V \leftarrow V^{3^m+1}$; // Algoritmo 4.14
8 **if** $b = 1$ **then**
9 **return** VW^{-1};
10 **else**
11 **return** VW;
12 **end**

El paso 9 del algoritmo 4.12 es necesaria una inversión del elemento W, pero recordemos que utilizando el torus tenemos que $W = W_0 + W_1\sigma \in \mathbb{F}_{3^{6m}}$, donde $W_0, W_1 \in \mathbb{F}_{3^{3m}}$, y su inverso es definido como $W^{-1} = W_0 - W_1\sigma$, es decir:

$$\begin{aligned} W^{-1} &= W_0 - W_1\sigma \\ &= (w_0 + w_2\rho + w_4\rho^2) + (w_1 + w_3\rho + w_5\rho^2)\sigma \\ &= (w_0 - w_1\sigma + w_2\rho - w_3\sigma\rho + w_4\rho^2 - w_5\sigma\rho^2) \end{aligned}$$

Gracias a ésto el paso 9 no se considera una operación costosa debido a que sólo es necesario la negación de 3 coeficientes que pertenecen al campo $\mathbb{F}_{3^m}$. Un paso interesante en el algoritmo 4.12 es el 1 que corresponde a una exponenciacion de la forma $U^{3^{3m}-1}$ donde $U \in \mathbb{F}_{3^{6m}}$, recordemos que la base en $\mathbb{F}_{3^{6m}}$ es $\{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$ y que podemos representar cada elemento $U \in \mathbb{F}_{3^{6m}}$ como sigue:

$$U = U_0 + U_1\sigma = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2$$

donde $U_0 = u_0 + u_2\rho + u_4\rho^2, U_1 = u_1 + u_3\rho + u_5\rho^2 \in \mathbb{F}_{3^{3m}}$, ésto significa que $\mathbb{F}_{3^{6m}}$ es una extensión cuadrática de $\mathbb{F}_{3^{3m}}$ con la base $\{1, \sigma\}$ [62].

Recordemos que $\sigma^2 + 1 = 0$, entonces $\sigma^{3^{3m}} = -\sigma$. Podemos establecer la siguiente relación:

$$U^{3^{3m}} = (U_0 + U_1\sigma)^{3^{3m}} = U_0^{3^{3m}} + U_1^{3^{3m}}\sigma^{3^{3m}} = U_0 - U_1\sigma$$

para todo $U = U_0 + U_1\sigma \in \mathbb{F}^*_{3^{6m}}$, entonces:

$$U^{3^{3m}-1} = \frac{U^{3^{3m}}}{U} = \frac{U_0 - U_1\sigma}{U_0 + U_1\sigma}$$

y tenemos que $(U_0 + U_1\sigma)(U_0 - U_1\sigma) = U_0^2 + U_1^2 \in \mathbb{F}^*_{3^{3m}}$, finalmente obtenemos que:

$$U^{3^{3m}-1} = \frac{(U_0 - U_1\sigma)^2}{U_0^2 + U_1^2} = \frac{(U_0^2 - U_1^2) - 2U_0U_1\sigma}{U_0^2 + U_1^2}$$

El algoritmo 4.13 fue implementado para realizar el cálculo de $U^{3^{3m}-1}$ como en [62, 11].

Algoritmo 4.13: Cálculo de $U^{3^{3m}-1}$ en $\mathbb{F}^*_{3^{6m}}$

Entrada: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$
Salida: $V = U^{3^{3m}-1} \in T_2(\mathbb{F}_{3^{3m}})$

1 $m_0 \leftarrow (u_0 + u_2\rho + u_4\rho^2)^2$; // Algoritmo 3.18
2 $m_1 \leftarrow (u_1 + u_3\rho + u_5\rho^2)^2$; // Algoritmo 3.18
3 $m_2 \leftarrow (u_0 + u_2\rho + u_4\rho^2)(u_1 + u_3\rho + u_5\rho^2)$;
4 $a_0 \leftarrow m_0 - m_1$; $a_1 \leftarrow m_0 + m_1$;
5 $i \leftarrow a_1^{-1}$; // Algoritmo 3.17
6 $V_0 \leftarrow a_0 i$;
7 $V_1 \leftarrow m_2 i$;
8 **return** $V_0 + V_1\sigma$;

Otra operación más dentro de la exponenciación final es el cálculo de U^{3^m+1}, nuevamente en [62, 11] podemos encontrar una manera eficiente para lograrlo y su análisis correspondiente; el algoritmo 4.14 fue implementado para realizar esta operación.

Algoritmo 4.14: Cálculo de U^{3^m+1} en $\mathbb{F}^*_{3^{6m}}$

Entrada: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$
Salida: $V = U^{3^m+1} \in T_2(\mathbb{F}_{3^{3m}})$

$a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_2 + u_3$; $a_2 \leftarrow u_4 - u_5$;
$m_0 \leftarrow u_0u_4$; $m_1 \leftarrow u_1u_5$; $m_2 \leftarrow u_2u_4$;
$m_3 \leftarrow u_3u_5$; $m_4 \leftarrow a_0a_2$; $m_5 \leftarrow u_1u_2$;
$m_6 \leftarrow u_0u_3$; $m_7 \leftarrow a_0a_1$; $m_8 \leftarrow a_1a_2$;
$a_3 \leftarrow m_5 + m_6 - m_7$; $a_4 \leftarrow -m_2 - m_3$;
$a_5 \leftarrow -m_2 + m_3$; $m_6 \leftarrow -m_0 + m_1 + m_4$;
else if $m \equiv 1 \bmod 6$ **then**
    $v_0 \leftarrow 1 + m_0 + m_1 + ba_4$;
    $v_1 \leftarrow bm_5 - bm_6 + a_6$;
    $v_2 \leftarrow -a_3 + a_4$;
    $v_3 \leftarrow m_8 + a_5 - ba_6$;
    $v_4 \leftarrow -ba_3 - ba_4$;
    $v_5 \leftarrow bm_8 + ba_5$;
else if $m \equiv 5 \bmod 6$ **then**
    $v_0 \leftarrow 1 + m_0 + m_1 - ba_4$;
    $v_1 \leftarrow -bm_5 + bm_6 + a_6$;
    $v_2 \leftarrow a_3$;
    $v_3 \leftarrow m_8 + a_5 + ba_6$;
    $v_4 \leftarrow -ba_3 - ba_4$;
    $v_5 \leftarrow -bm_8 - ba_5$;
end
return $v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2$;

El algoritmo anterior cálcula U^{3^m+1} utilizando 9 multiplicaciones y entre 18 o 19 sumas sobre el campo $\mathbb{F}_{3^m}$ dependiendo del valor de m modulo 6.

4.5. Resultados del emparejamiento η_T en característica tres

En la Tabla 4.4 se presentan los tiempos obtenidos para el cálculo del emparejamiento η_T en característica tres, donde el campo base es $\mathbb{F}_{3^{97}}$. Estos resultados corresponden a la implementación del Algoritmo 4.7. Esta misma implementación se ejecutó en una computadora laptop. Las características de las plataformas en donde se obtuvieron estos resultados se presentan en la sección 3.1.

Operation	PDA	Laptop
η_T Pairing	66.9 $ms.$	2.683 $ms.$

Tabla 4.4: Tiempo del emparejamiento η_T en característica tres (PDA).

En las Tablas 4.5 y 4.6 se presentan los tiempos obtenidos para el cálculo del emparejamiento η_T en la arquitectura multinúcleo sobre un sistema operativo de 32 bits (IA-32) y 64 bits (x64) respectivamente.

La implementación serial pertenece al algoritmo 4.9, mientras que para la implementación en paralelo, es utilizado el algoritmo 4.10 paralelizando el ciclo principal desde 2 hasta 8 ciclos más pequeños.

# Procesadores	Threads+Events	TBB
1-Serial	37.34 $ms.$	
2	19.96$ms.$	19.81 $ms.$
4	11.45 $ms.$	11.406 $ms.$
8	7.56 $ms.$	7.64 $ms.$

Tabla 4.5: Tiempos del emparejamiento η_T en característica tres (IA-32)

# Procesadores	Threads+Events	TBB
1-Serial	36.48 $ms.$	
2	18.39$ms.$	18.34 $ms.$
4	10.88 $ms.$	10.96 $ms.$
8	7.246 $ms.$	7.271 $ms.$

Tabla 4.6: Tiempos del emparejamiento η_T en característica tres (x64)

En la Figura 4.3 se muestra el comportamiento del tiempo necesario para el cálculo del emparejamiento η_T en característica tres, con respecto el número de procesadores a utilizar es aumentado.

Como en caraterística dos, podemos observar en la Figura 4.3 que el factor de aceleración conseguido es dos cuando se realiza el cálculo del emparejamiento η_T utilizando dos procesadores y la mejor relación costo/beneficio es el uso de una arquitectura con 4 procesadores, refiriendonos al costo económico de adquirir actualmente una arquitectura con varios procesadores y el benefio en el tiempo para el cálculo del emparejamiento.

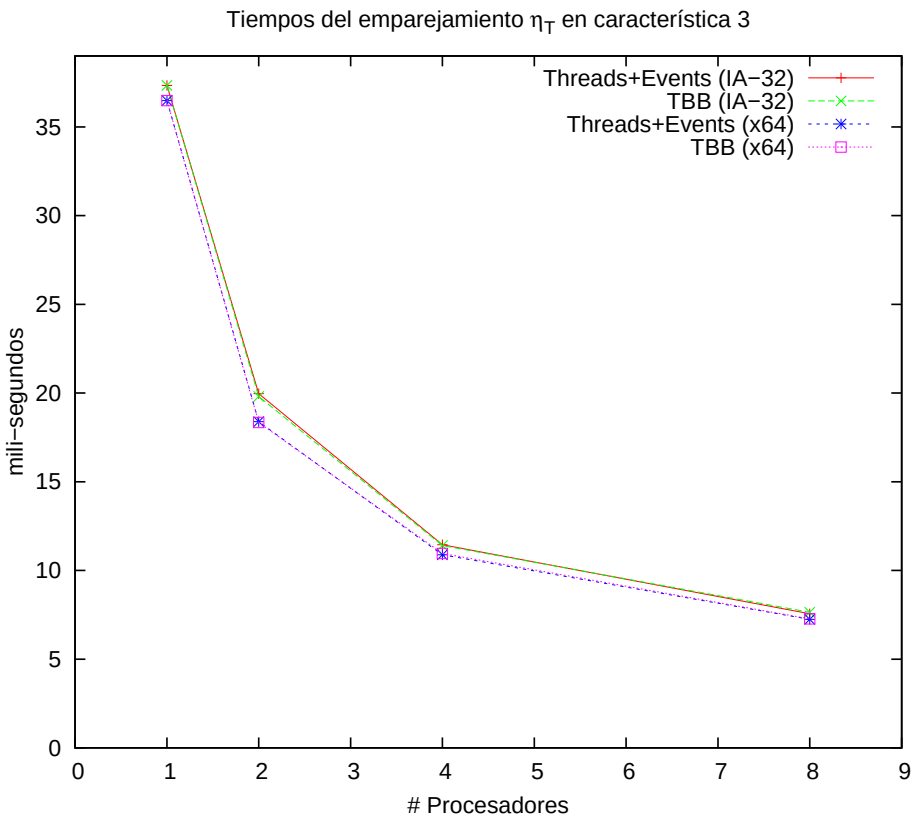


Figura 4.3: Relación tiempo-número de procesadores del emparejamiento η_T en característica tres (multinúcleo).

5 UNA APLICACIÓN

La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica.

Aristóteles (384 a. C. – 322 a. C.)

A lo largo de los capítulos anteriores se ha presentado la descripción e implementación de los diferentes algoritmos y procedimientos involucrado en la construcción de la criptografía basada en emparejamientos, desde la aritmética modular básica hasta el cálculo del emparejamiento η_T en campos finitos binarios y ternarios. Estas implementaciones fueron optimizadas consiguiendo buenos resultados sobre una PDA y una arquitectura multinúcleo. Como se ha mencionado los emparejamientos bilineales cuentan con buenas propiedades que pueden ser aprovechadas para la construcción de novedosos protocolos criptográficos. En este capítulo describiremos el diseño de una aplicación criptográfica desarrollada como parte de esta tesis, donde uno de los bloques básicos utilizados en el esquema consiste en el cálculo del emparejamiento η_T.

La aplicación fue realizada para la PDA utilizando las implementaciones del emparejamiento η_T que fueron dedicadas a este dispositivo en ambas características.

5.1. Descripción

La aplicación se encuentra motivada por el protocolo PGP (*Pretty Good Privacy*) que fue desarrollado por Phil Zimmermann. Este esquema tiene la finalidad de brindar seguridad y autenticación de la información que es compartida entre dos o más personas/entidades a través del Internet, haciendo uso de un esquema de criptografía de

clave pública, para más detalles sobre PGP revisar el RFC[1] [18] distribuido por el IETF[2].

La aplicación cuenta con dos funcionamientos primordiales, el primero es proteger la información "privilegiada", ésto es logrado mediante el uso de un esquema de cifrado y descifrado, el segundo es poder asegurar la integridad de los datos así como también la autenticidad del remitente, utilizando para ésto un esquema de firma y verificación. En esta aplicación es utilizado el esquema de criptografía basada en la identidad (*IBE* por sus siglas en inglés) [61], que fue descrito en la sección 2.3.2, y de igual forma un esquema de firma digital basado en la identidad.

Una ventaja importante y característica de utilizar criptografía basada en la identidad, es que entre las entidades participantes no es necesario ningún tipo de intercambio de clave. La clave pública de cada usuario es derivada directamente de la identidad del usuario (identificador), que puede ser su nombre, e-mail, número de seguro social, etc., mientras que la clave privada es proporcionada por una entidad de confianza llamada generador de clave privada (PKG).

La construcción de un criptosistema basado en la identidad fue propuesto por Boneh y Franklin [15], el cual consiste de 4 algoritmos principales que hacen uso de un emparejamiento bilineal, estos algoritmos se describen a continuación, pero antes recordemos que un emparejamiento bilineal es una función que mapea dos elementos del grupo aditivo G_1 a un elemento del grupo multiplicativo G_2, como es presentado en el capítulo 2:

$$\hat{e} : G_1 \times G_1 \rightarrow G_2$$

Pasos para la construcción de las operaciones de cifrado y descifrado basado en la identidad:

1. Setup: sea el punto P conocido, se elige un escalar s y se calcula $P_{pub} = sP$. Se eligen dos funciones picadillo, la primera $H_1 : \{0,1\}^* \rightarrow G_1^*$ llamada *map-to-point* y $H_2 : G_2 \rightarrow \{0,1\}^n$, donde n es la longitud del mensaje, la clave maestra es s y la clave pública global es P_{pub}.

   En este paso son necesarias dos funciones picadillo, la primera de ellas es llamada *map-to-point* H_1, que mapea una cadena de longitud arbitraria a un elemento en el grupo aditivo G_1. Ésta se implementó en software utilizando la técnica que es descrita en la sección 1.4.1 de esta tesis y, como se puede ver en el algoritmo 1.2, es necesaria una función picadillo estándar. Debido a ésto, se implementó la función picadillo SHA-256 y esta misma es utilizada para la función H_2 que translada un elemento del grupo multiplicativo G_2 a una cadena de longitud fija en este caso

[1] RFC: Request For Comments.
[2] IETF: The Internet Engineering Task Force. http://www.ietf.org/

un "digesto" de 256 bits.
También hay que indicar que la clave maestra s es conocida solamente por el centro generador de clave privada (PKG).

2. Extract: dado un identificador público $ID \in \{0,1\}^*$, calcular la clave pública $Q_{ID} = H_1(ID) \in G_1$ y la clave privada $S_{ID} = sQ_{ID}$.

   Aquí hay que aclarar que la clave pública $Q_{ID} = H_1(ID) \in G_1$, puede ser calculada por todas las entidades participantes, es decir, cualquier usuario es capaz de obtener la clave pública de los demás usuarios, por otro lado la clave privada $S_{ID} = sQ_{ID}$ sólo puede ser proporcionada por el PKG, dado que únicamente él conoce el secreto s.

3. Encrypt: escoger un aleatorio r y calcular el mensaje cifrado C del mensaje M como:

   $$C = \langle rP, M \oplus H_2(g_{ID}^r) \rangle$$

   donde: $g_{ID} = \hat{e}\,(Q_{ID}, P_{pub})$.

   Este paso es utilizado para cifrar un mensaje, consiste en realizar un emparejamiento (g_{ID}), una exponenciación g_{ID}^r, una xor entre el mensaje original y el digesto de $H_2(g_{ID}^r)$ y finalmente una multiplicación escalar rP.

4. Decrypt: dado $C = \langle U, V \rangle$, calcular

   $$M = V \oplus H_2\left(\hat{e}\,(S_{ID}, U)\right)$$

   Esta operación es utilizada para descifrar un mensaje. Si observamos el paso anterior podemos decir que $U = rP$, $V = M \oplus (H_2(\hat{e}(Q_{ID}, P_{pub})^r))$, y del paso 2 tenemos que $S_{ID} = sQ_{ID}$, después sustituyendo y aprovechando las propiedades que ofrece el emparejamiento podemos descifrar el mensaje como sigue:

   $$\begin{aligned} M &= V \oplus H_2\left(\hat{e}\,(S_{ID}, U)\right) \\ &= M \oplus (H_2(\hat{e}(Q_{ID}, P_{pub})^r)) \oplus H_2\left(\hat{e}\,(sQ_{ID}, rP)\right) \\ &= M \end{aligned}$$

El criptosistema basado en la identidad (IBE) es utilizado en conjunto con un cifrador de clave privada, como es común encontrarse en la práctica, pues recordemos que la criptografía de clave privada es simple y muy eficiente, pero presenta el problema de la distribución de la clave, mientras que la criptografía de clave pública se basa en operaciones más costosas pero no existen problemas en la distribución de la clave pública y mejor aún en IBE no es necesario ningún tipo de intercambio de claves.

El cifrador de clave privada que fue implementado en software para ser utilizado en esta aplicación es el cifrador por bloques llamado AES (*Advanced Encryption Standard*),

también conocido como Rijndael, ver el apéndice A para más detalles sobre AES. En esta aplicación se utilizó un tamaño de clave de 128 bits para AES.

Para realizar las operaciones de firma y verificación se utilizó un esquema de firmas propuesto en [44], que nuevamente consta de 4 pasos principales:

1. Setup: se elige un escalar s y se calcula $P_{pub} = sP$. Se eligen dos funciones picadillo $H_1 : \{0,1\}^* \rightarrow G_1^*$ (*map-to-point*) y $H : \{0,1\}^* \times G_2 \rightarrow Z_q^*$, la clave maestra es s y la clave pública global es P_{pub}.

   En este paso observamos que nos hace falta revisar la implementación de la función picadillo H. Ésta recibe un par de entradas formado por una cadena de longitud arbitraria (el mensaje) y un elemento en G_2 que es resultante del emparejamiento η_T. Para realizar esta función picadillo, el resultado del emparejamiento es concatenado con el mensaje y entonces es utilizada la función SHA-256.

2. Extract: dado un identificador público $ID \in \{0,1\}^*$, calcular la clave pública $Q_{ID} = H_1(ID) \in G_1$ y la clave privada $S_{ID} = sQ_{ID}$.

   Nuevamente aquí la clave pública puede ser generada por cualquier usuario y la clave privada sólo es generada por el *PKG*.

3. Sign: dada la clave privada S_{ID} y un mensaje $M \in \{0,1\}^*$, el signatario escoge de forma arbitraria a $P_1 \in G_1^*$ y un escalar k y se calcula:

$$\begin{aligned} r &= \hat{e}(P_1, P)^k \\ v &= H(M, r) \\ u &= vS_{ID} + kP_1 \end{aligned}$$

   la firma está dada por el par (u, v). Recordemos que H realiza la función picadillo de la concatenación del emparejamiento r con el mensaje M.

4. Verify: Dado Q_{ID}, el mensaje M y su firma (u, v), la verificación se realiza de la siguiente manera:

$$t = \hat{e}(u, P)\hat{e}(Q_{ID}, -P_{pub})^v$$

   la firma es válida si y sólo si $v = H(M, t)$. Para probar este paso necesitamos observar del paso anterior que $u = vS_{ID} + kP_1 = vsQ_{ID} + kP_1$, del paso 1 tenemos $P_{pub} = sP$ y aplicando las propiedades del emparejamiento y sustituyendo tenemos:

$$\begin{aligned} t &= \hat{e}(u, P)\hat{e}(Q_{ID}, -P_{pub})^v = \hat{e}(vsQ_{ID} + kP_1, P)\hat{e}(Q_{ID}, -P_{pub})^v \\ &= \hat{e}(vsQ_{ID}, P)\hat{e}(kP_1, P)\hat{e}(vQ_{ID}, sP) = \hat{e}(vsQ_{ID}, P - P)\hat{e}(kP_1, P) \\ &= \hat{e}(vsQ_{ID}, \mathcal{O})\hat{e}(kP_1, P) \\ &= \hat{e}(P_1, P)^k \end{aligned}$$

y podemos ver que t es exactamente igual a r del paso 3, haciendo la verificación válida.

A continuación se describe el modelo utilizado para la construcción de esta aplicación.

5.1.1. Cifrado y descifrado

Las operaciones de cifrado y descifrado se pueden observar en la Figura 5.1, para entender esta figura es necesario revisarla de izquierda a derecha. En la parte izquierda se genera una clave de sesión de forma aleatoria (128 bits) que es utilizada como clave privada para el cifrador AES con el cual es cifrado el texto en claro, esta clave de sesión es cifrada utilizando el bloque de IBE en modo cifrador, el bloque IBE recibe como entrada el identificador (ID) del destinatario. El mensaje cifrado por AES y la clave se sesión cifrada con IBE son transmitidas al destinatario.

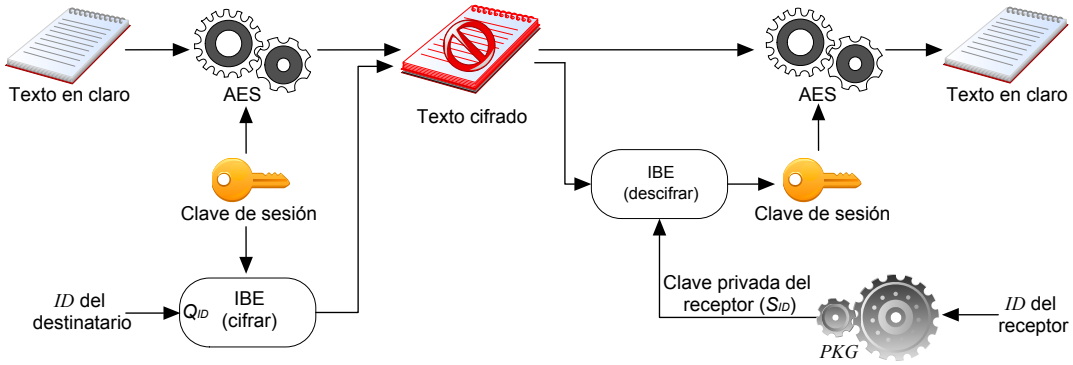


Figura 5.1: Cifrado y descifrado utilizando IBE.

En la parte derecha de la Figura 5.1, es realizada la operación de descifrado. Para descifrar la clave de sesión es necesario contar con la clave privada (S_{ID}) que es solicitada a la entidad PKG, con ayuda de la clave privada, IBE es utilizado en modo descifrar para recuperar la clave de sesión. Finalmente esta clave de sesión es utilizada nuevamente con AES para así recuperar el mensaje original.

Un punto que debemos aclarar, es la interacción entre las entidades participantes y el PKG al momento de obtener su clave privada. Esta actividad sólo es realizada en una sola ocasión (al inicio) y cuando el PKG ha terminado de proporcionar las claves privadas a las entidades, debe de desaparecer [61], para evitar que el el esquema criptográfico se vea comprometido, pues el PKG es capaz de generar cualquier clave privada .

Los algoritmos 5.1 y 5.2 presentan el pseudo-código utilizado para realizar las operaciones de cifrado y descifrado en la aplicación.

Algoritmo 5.1: Cifrador utilizando IBE

Entrada: Identidad del destinatario $ID = \{0,1\}^*$ y el mensaje $M = \{0,1\}^*$
Salida: $C = \langle U, V, W \rangle$

1 $Q_{ID} \leftarrow mapTopoint(ID)$;
2 $r \leftarrow rand()$;
3 $skey \leftarrow rand()$;
4 $U \leftarrow rP$; $g \leftarrow \eta_T(Q_{ID}, rP_{pub})$; $G \leftarrow hash(g)$;
5 $V \leftarrow skey \oplus G$;
6 $W \leftarrow cifradorAES(M, skey)$;
7 **return** $\langle U, V, W \rangle$

Nótese que *skey* es la clave de sesión que es utilizada con AES y su generación es aleatoria en cada ocasión que se desea cifrar un nuevo mensaje.

Algoritmo 5.2: Descifrador utilizando IBE

Entrada: $C = \langle U, V, W \rangle$ y la clave privada S_{ID}
Salida: M

1 $g \leftarrow \eta_T(S_{ID}, U)$;
2 $skey \leftarrow V \oplus hash(g)$;
3 $M \leftarrow descifradorAES(W, skey)$;
4 **return** M

En los algoritmos anteriores son utilizados tanto el emparejamiento η_T como el cifrador AES, además hay que aclarar que los los puntos P y P_{pub} son públicos y conocidos por todos los usuarios del sistema.

Hay que prestar especial atención al algoritmo 5.2, que recibe como una de sus entradas la clave privada S_{ID}. Esta clave es solicitada al PKG y el algoritmo de dicha solicitud es muy simple, pues recordemos que la clave privada es definida por una multiplicación escalar $S_{ID} = sQ_{ID}$, donde la única condición es que s es generado y mantenido en secreto por el PKG.

5.1.2. Firma y verificación

Las operaciones de firma y verificación son descritos de forma gráfica en la Figura 5.2. La operación de firma es realizada sobre el digesto del mensaje con ayuda del bloque IBE en modo de firma y utilizando la clave privada S_{ID} del emisor (signatario), la firma resultante es transmitida junto con el mensaje original, para posteriormente poder realizar la operación de verificación.

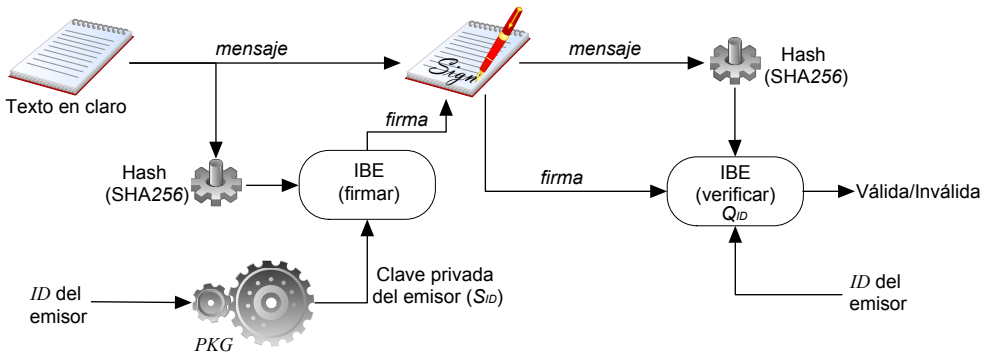


Figura 5.2: Firma y verificación utilizando IBE.

El destinatario recibe el documento junto con su respectiva firma, en este punto el destinatario tiene la opción de llevar a cabo la verificación de la firma o no, para realizar la verificación se calcula el digesto del mensaje y junto con la clave pública del emisor ($Q_{ID} = mapTopoint(ID)$) son utilizados en el bloque IBE en modo de verificación el cual nos indica si la firma es válida o no es válida.

El algoritmo 5.3 es un pseudo-código de la implementación en software para realizar la operación de firma digital, donde son utilizados el emparejamiento η_T y la función picadillo SHA-256 como primitivas criptográficas básicas.

Algoritmo 5.3: Firma utilizando IBE

Entrada: Clave privada S_{ID} y el mensaje $M = \{0,1\}^*$
Salida: $S = \langle M, u, v \rangle$

1 $k \leftarrow rand()$;
2 $P_1 \leftarrow$ elegir un punto aleatorio;
3 $r \leftarrow \eta_T(P, P_1)^k$;
4 $v \leftarrow hash(M||r)$;
5 $u \leftarrow vS_{ID} + kP_1$;
6 **return** $\langle M, u, v \rangle$

Un detalle interesante en el algoritmo 5.3 es el paso 2, pues consiste en la generación de un punto aleatorio $P_1 \in G_1^*$, este puede ser un multiplo de punto ya conocido P o puede ser también el mismo P como se propone en [44].

De la misma forma el pseudo-código del algoritmo 5.4 fue implementado para realizar la operación complementaria a la firma digital, es decir, la verificación.

Algoritmo 5.4: Verificación utilizando IBE

Entrada: $S = \langle M, u, v \rangle$ y el identificador del emisor ID
Salida: Válida/inválida

```
1 Q_ID ← mapTopoint(ID)// Clave pública del emisor
2 t ← η_T(u, P);
3 h ← η_T(Q_ID, −P_pub)^v;
4 r ← t · h;
5 if v = hash(M||r) then
6     return firma válida;
7 else
8     return firma inválida;
9 end
```

Hay que aclarar que esta aplicación fue dedicada para la PDA y está limitada a los recursos con los que cuenta este dispositivo, es decir, su capacidad de cómputo es reducida, sin embargo, esta implementación es portable y puede ser utilizada en cualquier otra computadora con capacidades y características diferentes.

5.2. Análisis de tiempo

Para esta aplicación realizamos un análisis de los tiempos de operación. En la Tabla 5.1 se muestran los tiempos necesarios para las operaciones indicadas sobre la arquitectura PDA.

Operación	Característica 2 ($\mathbb{F}_{2^{233}}$)	Característica 3 ($\mathbb{F}_{3^{97}}$)
Cifrado (IBE)	95.085 $ms.$	78.8 $ms.$
Descifrado (IBE)	30.4 $ms.$	67.4 $ms.$
Firma (IBE)	46.7 $ms.$	88.396 $ms.$
Verificación (IBE)	123.37 $ms.$	143.6 $ms.$
Map-To-Point	58.2 $ms.$	4 $ms.$
Cifrado AES (1 bloque)	220 $\mu s.$	
Expansion de clave AES	20.2 $\mu s.$	

Tabla 5.1: Tiempos obtenidos de la aplicación en la PDA.

La fila correspondiente a la operación de cifrado en la Tabla 5.1, indíca el tiempo necesario para realizar los pasos del 1 al 5 del algoritmo 5.1. La fila de la operación de descifrado indíca el tiempo de los pasos 1 y 2 del algoritmo 5.2.

Los tiempos correspondiente a las filas de firma corresponde al algoritmo 5.3, y la fila de verificación corresponde al algorimto 5.4. Para ambas operaciones se realizó la prueba con un mensaje, M, de longitud de 128 bits. Hay que aclarar qeu este tiempo se ve afectado dependiendo de la longitud del mensaje.

La diferencia de tiempos entre característica dos y tres se debe al cálculo de la función picadillo *map-to-poin*. Como es presentado por Barreto y Kim en [7] para la función map-to-point en característica tres es necesario resolver un sistema de ecuaciones, pero describen que se puede mejorar al obtener la matriz para elevar al cubo y realizar las operaciones como sigue:

$$\begin{aligned} y^2 &= x^3 - x + b \\ y^2 - b &= x^3 - x \\ y^2 - b &= M_3 x - Ix \\ y^2 - b &= (M_3 - I)x \end{aligned}$$

con $b \in \{1, -1\}$ y donde M_3 en nuestro caso es la matriz para elevar al cuadrado un elemento en $\mathbb{F}_{3^{97}}$ y I es la matriz identidad, hay que mencionar que ahora tenemos que ver los elementos x y $y^2 - b$ como vectores. Entonces sólo basta con encontrar la matriz inversa, $M_{inv3} = (M_3 - I)^{-1}$, y realizar el siguiente producto:

$$M_{inv3}(y^2 - b) = x$$

para encontrar el valor de x.

Para característica dos se utilizó un técnica similar con la diferencia de que ahora podemos obtener la matriz para elevar al cuadrado, y la ecuación de la curva elíptica es $y^2 + y = x^3 + x + b$ con $b \in \{0, 1\}$. Entonces podemos realizar las siguientes operaciones:

$$\begin{aligned} y^2 + y &= x^3 + x + b \\ M_2 y + Iy &= x^3 + x + b \\ (M_2 + I)y &= x^3 + x + b \end{aligned}$$

donde M_2 corresponde a la matriz para elevar al cuadrado un elemento en $\mathbb{F}_{2^{233}}$, ver [43] donde se describe como obtener esta matriz. En este caso la matriz resultante de la suma de $M_2 + I$ es una matriz singular, es decir, no tiene inversa, por esta razón se resulve el sistema de ecuaciones por medio de eliminación gaussiana, este sistema esta compuesto por 233 ecuaciones con 233 incognitas.

Esta aplicación se encuentra funcionando y es utilizada a través de un consola de instrucciones, indicando con esto que su interfaz con el usuario se encuentra en modo texto.

6

CONCLUSIONES

Conclusión es el lugar donde llegaste cansado de pensar.

Anónimo.

El presente trabajo de tesis consiste en el diseño e implementación eficiente en software de una biblioteca para cálculo del emparejamiento η_T en dos arquitecturas con recursos diferentes sobre campos finitos binarios y ternarios.

6.1. Resumen de resultados

Los logros obtenidos en esta tesis se enlistan a continuación.

- Una implementación eficiente en software de una biblioteca para la aritmética modular en los campos $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{503}}$, $\mathbb{F}_{3^{97}}$ y $\mathbb{F}_{3^{503}}$.
- Una implementación eficiente en software de la aritmética de las torres de de campos $\mathbb{F}_{2^{2\cdot 233}}$, $\mathbb{F}_{2^{4\cdot 233}}$, $\mathbb{F}_{2^{2\cdot 503}}$, $\mathbb{F}_{2^{4\cdot 503}}$, $\mathbb{F}_{3^{3\cdot 97}}$, $\mathbb{F}_{3^{6\cdot 97}}$, $\mathbb{F}_{3^{3\cdot 503}}$ y $\mathbb{F}_{3^{6\cdot 503}}$.
- Se obtuvo una implementación eficiente para el cálculo completo del emparajamiento η_T en los campos $\mathbb{F}_{2^{233}}$ y $\mathbb{F}_{3^{97}}$ sobre una arquitectura con recuros limitados, PDA.
- Se realizó un análisis de los algoritmos de emparejamiento con el fin de realizar una paralelización para el cálculo del emparejamiento η_T en los campos $\mathbb{F}_{2^{503}}$ y $\mathbb{F}_{3^{503}}$ sobre una arquitectura mutinúcleo. Esta implementación es escalada desde el uso de un sólo procesador hasta el uso de ocho procesadores, para el cálculo del emparejamiento η_T.

- En esta tesis se presentaron algunas posibles aplicaciones de los emparejamientos bilineales, de las cuales una de ellas, criptografía basada en la identidad (IBE), fue realizada sobre la PDA, comprendida por las primitivas de cifrado-descifrado y firma-verificación.
- Se realizó una implementación eficiente del cifrador por bloques AES, para ser utilizado en la aplicación desarrollada sobre la PDA.
- Se realizó una implementación de las funciones picadillo llamadas *map-to-points* para características dos y tres, además de la función picadillo SHA-256, que son utilizadas en la aplicación sobre la PDA.
- Durante el primer año de tesis se logro una publicación de un artículo de comentario donde se presentan diversos ataques a una modificación del cifrador Hill [67].
- Parte de este trabajo de tesis fue presentado en Computing Science and Automatic Control (CCE 2008), donde se analizaron nueve esquemas de firmas a ciegas [51].

6.2. Conclusiones

- Debido al gran interés del uso de emparejamientos bilineales en protocolos criptográficos, resulta crucial contar con implementaciones para el cálculo eficiente y seguro de estos.
- En esta tesis presentamos los resultados obtenidos por el diseño e implementación eficiente para el cálculo del emparejamiento η_T sobre campos finitos binarios y ternarios. Consideramos que los resultados presentados aquí cumplen con las expectativas planteadas al inicio de este trabajo.
- Con los resultados obtenidos en esta tesis podemos puntualizar que las arquitecturas multinúcleo pueden ser ampliamente utilizadas para el cálculo del emparejamiento η_T de forma paralela.
- La eficiencia obtenida en el cálculo del emparejamiento η_T en ambas características es debida en parte a la elección del lenguaje de programación y a la forma en que se realizaron las implementaciones.

6.3. Trabajo a futuro

- A pesar de que en esta tesis se obtuvieron resultados competitivos con respecto a los trabajos existentes en la literatura abierta, consideramos que nuestra implementación puede ser mejorada.

- Proveer de una implementación que cuente con un mayor nivel de seguridad para el emparejamiento η_T en característica dos, es decir, utilizar un campo binario de aproximadamente 1110 bits.
- En cuanto a la aplicación en la PDA, uno de los trabajos a futuro es realizar una interfaz gráfica para una mejor interacción con el usuario.

A

CIFRADOR AES

En este apéndice se dará una descripción del cifrador por bloques llamado AES (*Adavanced Encryption Standard*) también conocido como Rijndael. El nombre Rijndael es un acrónimo formado por los nombres de sus autores, Joan Daemen y Vincent Rijmen.

El Instituto Nacional de Estándares y Tecnología (*NIST* por sus siglas en inglés) adoptó oficialemente el algoritmo de Rijndael en octubre del año 2000, como nuevo Estándar Avanzado de Cifrado (AES) para aplicaciones criptogáficas remplazando al viejo DES (*Data Encryption Standard*) y a su variante el 3-DES.

EL gobierno Estadounidense en [1], especifica que AES procesa bloques de datos de longitud fija de 128 bits, mientras que la clave puede ser elegida entre 128, 192, y 256 bits. Como todos los esquemas criptográficos, AES consiste de 3 algoritmos principales que son conocidos como: expansión de clave, cifrado y descrifrado. A continuación se describen cada uno de éstos.

A.1. Expansión de clave

La expansión de clave es un proceso utilizado para generar las subclaves que son utilizadas a lo largo de los procesos de cifrado y descrifrado. El algoritmo A.1 es utilizado para la expansión de la clave, éste recibe como entrada una clave (key) generada de forma aleatoria y N_k que denota el número de columnas necesarias para representar la clave key en una matriz, en esta tesis se utilizó una clave de 128 bits, obteniendo una matriz

de 4×4 bytes, debido a ésto tenemos que $N_k = 4$.

Algoritmo A.1: AES: Expansión de clave

Entrada: Clave $Key\,[4 \times N_k]$, N_k
Salida: $w\,[N_b(N_r + 1)]$

```
1  i ← 0;
2  while i < N_k do
3      w[i] ← (key[4i], key[4i + 1], key[4i + 2], key[4i + 3]);
4      i ← i + 1;
5  end
6  i ← N_k;
7  while i < N_b(N_r + 1) do
8      temp ← w[i − 1];
9      if (i mód N_k ≡ 0) then
10         temp ← SubWord(RotWord(temp)) ⊕ Rcon [i/N_k];
11     else if N_k > 6 y (i mód N_k ≡ 4) then
12         temp ← SubWord(temp);
13     end
14     w [i] ← w [i − N_k] ⊕ temp;
15     i ← i + 1;
16 end
```

En el algoritmo anterior son utilizadas dos funciones especiales, la primera es llamada $SubWord$ la cual realiza una sustitución con ayuda de una caja-S, la segunda es $RotWord$ que realiza un corrimiento cíclico a la plabra que recibe, es decir, $[w_0, w_1, w_2, w_3]$ es transformado a $[w_1, w_2, w_3, w_0]$. Cabe mencionar que también es utilizada un tabla llamada $Rcon$ con la cual se realiza una operación XOR. La caja-S utilizada para $SubWord$ y la tabla $Rcon$ se pueden consultar en [1].

A.2. Cifrado

El algoritmo de cifrado consiste en aplicar un número determinado de rondas (N_r) a un valor intermedio llamado estado (matriz de estado de 4×4 bytes). En cada ronda son aplicadas cuatro transformaciones invertibles principales como se muestra en el algoritmo A.2, donde los datos de entrada son: el bloque a cifrar (texto en claro) y las claves a utilizar, mientras que la salida del algoritmo corresponde al cifrado del bloque (texto cifrado).

Algoritmo A.2: AES: Cifrador

Entrada: Bloque a cifrar $M\,[4 \times N_b]$ y clave $w\,[N_b(N_r + 1)]$
Salida: Bloque cifrado $C\,[4 \times N_b]$

```
1  estado ← M;
2  AddRoundKey(estado, w[0, N_b − 1]);
3  for ronda ← 1 to N_r − 1 do
4      SubBytes(estado);
5      ShiftRows(estado);
6      MixColumns(estado);
7      AddRoundKey(estado, w[ronda ∗ N_b, (ronda − 1)N_b − 1]);
8  end
9  SubBytes(estado);
10 ShiftRows(estado);
11 AddRoundKey(estado, w[rondaN_b, (N_r + 1)N_b − 1]);
12 C = estado;
13 return C
```

La transformación *SubBytes* (sustitución de bytes) consiste de una sustitución no lineal de cada byte de la matriz de estado, mediante una caja-S invertible, para más detalles sobre esta caja-S consultar [1].

La transformación *ShiftRows* (desplazar filas) consiste en desplazar a la izquierda de manera cíclica las filas de la matriz de estado. Aquí cada fila f_i se desplaza i posiciones a la izquierda, con $0 \leq i \leq 4$, donde la primer fila no es alterada y el resto sufren un desplazamiento como se muestra en la figura A.1.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \Longrightarrow \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{bmatrix}$$

Figura A.1: AES: ShiftRows

En la transformación *MixColumns* (mezclar columnas), cada columna es tomada como un polinomio donde sus coeficientes pertenecen al campo $\mathbb{F}_{2^8}$ y se realiza la multiplicación módulo $x^4 + 1$ con siguiente la matriz (valores en hexadecimal):

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

La última transformación es llamada *AddRoundKey* (suma de llave de ronda), ésta es realizada mediante una operación XOR entre la matriz estado y la subclave correspondiente a cada ronda, el orden de utilizado de las claves es ascendente.

A.3. Descifrado

Para la opeación de descifrado sólo basta con invertir el orden de las transformaciónes y adémas invertir el funcionamiento de cada transformación. El algoritmo A.3 corresponde al descifrador AES.

Algoritmo A.3: AES: Descifrador

Entrada: Bloque cifrado $C\,[4 \times N_b]$ y clave $w\,[N_b(N_r + 1)]$
Salida: Bloque descifrado $M\,[4 \times N_b]$

```
1  estado ← C;
2  AddRoundKey(estado, w[N_r N_b, (N_r + 1)N_b − 1]);
3  for ronda ← N_r − 1 downto 1 do
4      InvShiftRows(estado);
5      InvSubBytes(estado);
6      AddRoundKey(estado, w[ronda ∗ N_b, (ronda − 1)N_b − 1]);
7      InvMixColumns(estado);
8  end
9  InvShiftRows(estado);
10 InvSubBytes(estado);
11 AddRoundKey(estado, w[0, N_b − 1]);
12 M = estado;
13 return M
```

La transformación $InvShiftRows$ es la operación inversa a $ShiftRows$, aquí cada fila f_i se desplaza i posiciones a la derecha, con $0 \leq i \leq 4$.

La transformación $InvSubBytes$ opera sobre cada byte de la matriz estado, utilizando nuevamente una tabla llamada InvCaja-S, esta tabla de sustitución se puede consultar en [1].

$InvMixColumns$ es la transformación inversa correspondiente a $MixColumns$ utilizada en el cifrador, en esta se realiza una multiplicación de forma matricial, donde las columnas de matriz estado son tomadas como polinomios y multiplicados con las filas

de la matriz:

$$\begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix}$$

esta matriz corresponde a la matriz inversa utilizada en $MixColumns$ y sus valores estan dados en hexadecimal.

La última transformación es $AddRoundKey$, que realiza la suma (XOR) de la matriz estado con la subclave de ronda correspondiente. Aquí el orden de las claves a sumar es descendiente.

B

TOOLS

B.1. Intel® C++ Compiler 10.1

The Intel® C++ Compiler 10.1 lets you build and optimize C/C++ applications for the Windows and Linux operating systems [21]. This product provides tools for software developers to create applications that run at top speeds on IA-32, Intel® 64 and IA-64 architecture processors. The optimizations include support for Intel® Streaming SIMD Extensions (SSE, SSE2, SSE3 and SSE4) for accelerated integer and floating-point operations [26, 27].

B.1.1. Using the Compiler

C++ is supported by some very good compilers and optimized function libraries, in this case by the Intel® C++ Compiler 10.1.

This compiler does not have its own IDE. It is intended as a plug-in to Microsoft Visual Studio when compiling for Windows and to Eclipse when compiling for Linux. It can also be used as a stand alone compiler when called from a command line or a make utility. It supports 32-bit and 64-bit Windows and 32-bit and 64-bit Linux as well as Intel-based Mac OS and Itanium systems.

Optimizing the executable

The compiler is a very important tool, that helps us make the optimization of our

code in a very quick and easy way, just with use of flags of optimization offered to their different environments (Linux*, Windows and MAC environments).

Automatic optimizations

The automatic optimizations allow us to take advantage of the architectural differences, new instruction sets, or advances in processor design, however, the resulting optimized code might contain unconditional use of features that are not supported on other, earlier processors.

The automatic optimizations offered by the compiler are as follows:

Windows*	Linux*	Description
/Od	-O0	Disables optimizations
/O1	-O1	Optimizes to favor smaller code size and code locality
/O2	-O2	Optimizes for code speed (default)
/O3	-O3	Optimize for Data Cache (loop optimizations)

one of the flags used in the project is O3, because it offers a more drastic optimization:

- Maximize Speed plus High Level Optimizations (/O3) [Intel C++]
  - /O3 in Windows*, -O3 in Linux*

Interprocedural Optimization

Interprocedural Optimization (IPO) allows the compiler to analyze your code to determine where you can benefit from specific optimizations. The optimizations that can be applied are related to the specific architectures.

IPO is an automatic, multi-step process: compilation and linking; however, IPO supports two compilation models: single-file compilation and multi-file compilation. See [21].

Single-file compilation, which uses the `-ip` (Linux*) or `/Qip` (Windows*) option, results in one, real object file for each source file being compiled. During single-file compilation the compiler performs inline function expansion for calls to procedures defined within the current source file. The compiler performs some single-file interprocedural optimization at the default optimization level: `-O2` or `/O2`, as shown in Figure B.1.

```
1>------ Build started: Project: Pair_EtaT, Configuration: Release Win32 ------
1>Compiling with Intel(R) C++ 10.1.021 [IA-32]... (Intel C++ Environment)
1>main.cpp
1>Linking... (Intel C++ Environment)
1>ipo: remark #11001: performing single-file optimizations
1>ipo: remark #11005: generating object file ipo_57886obj.obj
1>xilink: executing 'link'
      :
1>Pair_EtaT - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Figura B.1: Single-file compilation.

Multi-file compilation, which uses the `-ipo` (Linux*) or `/Qipo` (Windows) option, results in one or more mock object files rather than normal object files. Additionally, the compiler collects information from the individual source files that make up the program. Using this information, the compiler performs optimizations across functions and procedures in different source files. Inlining is the most powerful optimization supported by IPO.

Linux*	Windows	Description
`-ipo` or `-ipoN`	`/Qipo` or `-ipoN`	Enables interprocedural optimization for multi-file compilations.
`-ip`	`/Qip`	Enables interprocedural optimizations for single file compilations. Instructs the compiler to generate a separate, real object file for each source file.

Profile-Guided Optimization

Profile-guided Optimization (PGO) improves application performance by reorganizing code layout to reduce instruction-cache problems, shrinking code size, and reducing branch mispredictions. PGO provides information to the compiler about areas of an application that are most frequently executed. By knowing these areas, the compiler is able to be more selective and specific in optimizing the application.

PGO consists of three phases or steps.

- Step one is to instrument the program. In this phase, the compiler creates and links an instrumented program from your source code and special code from the compiler.
  command: `icl eta_T_Pairing.c /Qprof_gen /Qprof_dir`

- Step two is to run the instrumented executable. Each time you execute the instrumented code, the instrumented program generates a dynamic information file, which is used in the final compilation.
  command: `eta_T_Pairing.exe`

- Step three is a final compilation. When you compile a second time, the dynamic information files are merged into a summary file. Using the summary of the profile information in this file, the compiler attempts to optimize the execution of the most heavily traveled paths in the program.
  command: `icl eta_T_Pairing.c /Qprof_use /Qprof_dir`

PGO enables the compiler to take better advantage of the processor architecture, more effective use of instruction paging and cache memory, and make better branch predictions. PGO provides the following benefits:

- Use profile information for register allocation to optimize the location of spill code.
- Improve branch prediction for indirect function calls by identifying the most likely targets.
- Detect and do not vectorize loops that execute only a small number of iterations, reducing the run time overhead that vectorization might otherwise add.

PGO works best for code with many frequently executed branches that are difficult to predict at compile time. An example is the code with intensive error-checking in which the error conditions are false most of the time.

Interprocedural Optimization (IPO) and Profile-guided Optimization (PGO) can affect each other; using PGO can often enable the compiler to make better decisions about function inlining, which increases the effectiveness of interprocedural optimizations. Unlike other optimizations, such as those strictly for size or speed, the results of IPO and PGO vary. This variability is due to the unique characteristics of each program, which often include different profiles and different opportunities for optimizations.

Using Parallelism

The compiler provides a form of parallel implicitly, that is set the flag /Qparallel in Windows* pr -Qparallel in Linux*, this enable the auto-parallelizer to generate multithreaded for loops that can be safely executed un parallel. But this does not allow us to have full control over the parallelization, for this reason we decided to use an explicit parallelization. The explicit parallelization is more complex of implement, because the programmer is responsible for the parallelization.

To implement an explicit parallelization, we have to identify portions of code to thread, encapsulate the code in a function, is needed a driver function to coordinate work of multiple threads and make a proper synchronization between threads. For the parallelization can use the following tools:

- Windows Threads
- POSIX Threads
- Threading Building Blocks

C++ is an advanced high-level language with a wealth of advanced features rarely found in other languages. But the C++ language also includes the low-level C language as a subset, giving access to low-level optimizations.

The compiler has an excellent support for inline assembly on all platforms and the possibility of using the same inline assembly syntax in both Windows and Linux.

The compiler offers a way to write code ASM, very easy to use, that is to use so-called intrinsic functions.

Intrinsics for Intel® C++ Compilers

Intrinsics are assembly-coded functions that allow us to use C++ function calls and variables in place of assembly instructions. Intrinsics are expanded inline eliminating function call overhead. Providing the same benefit as using inline assembly, intrinsics improve code readability, assist instruction scheduling, and help reduce debugging.

The Intel® C++ Compiler enables easy implementation of assembly instructions through the use of intrinsics. Intrinsics are provided for Intel® Streaming SIMD Extensions 4 (SSE4), Supplemental Streaming SIMD Extensions 3 (SSSE3), Streaming SIMD Extensions 3 (SSE3), Streaming SIMD Extensions 2 (SSE2), and Streaming SIMD Extensions (SSE) instructions [26, 27].

The Intel® C++ Compiler provides intrinsics that work on specific architectures and intrinsics that work across IA-32, Intel® 64, and IA-64 architectures. Most intrinsics map directly to a corresponding assembly instruction, some map to several assembly instructions.

Details about Intrinsics

The MMX(TM) technology and Streaming SIMD Extension (SSE) instructions use the following features:

- Registers: enable packed data of up to 128 bits in length for optimal SIMD processing

• Data Types: enable packing of up to 16 elements (bytes) of data in one register

Registers

Intel processors provide special register sets.
The MMX instructions use eight 64-bit registers (mm0 to mm7) which are aliased on the floating-point stack registers.
The Streaming SIMD Extensions use eight 128-bit registers (xmm0 to xmm7).

Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single instruction multiple data processing (SIMD).

For each computational and data manipulation instruction in the new extension sets (SSE), there is a corresponding C intrinsic that implements that instruction directly, see [26, 27] to find more information of the intrinsic functions. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

Data Types

Intrinsic functions use four new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions. The following table details for which instructions each of the new data types are available.

New Data Type	MMX Technology	SSE	SSE2	SSE3
__m64	Available	Available	Available	Available
__m128	Not Available	Available	Available	Available
__m128d	Not Available	Not Available	Available	Available
__m128i	Not Available	Not Available	Available	Available

Tabla B.1: Data types: MMX and SSE.

The `__m64` data type is used to represent the contents of an MMX register, which is the register that is used by the MMX technology intrinsics. The `__m64` data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

The `__m128` data type is used to represent the contents of a Streaming SIMD Extension register used by the Streaming SIMD Extension intrinsics. The `__m128` data type can hold four 32-bit floating-point values.

The `__m128d` data type can hold two 64-bit floating-point values.

The __m128i data type can hold sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit integer values.

The compiler aligns __m128d and __m128i local and global data to 16-byte boundaries on the stack. To align integer, float, or double arrays, you can use the declspec align statement.

These data types are not basic ANSI C data types. You must observe the following usage restrictions:

- Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
- Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
- Use data types only with the respective intrinsics.

In this work are only used SSE instructions and their respective registers. Example of how to use intrinsic functions: The following code belongs to our project.

```
struct GF3_m {
  unsigned int L[MAX_SIZE];
  unisgned int H[MAX_SIZE];
} GF3m, *pGF3m;
__inline void f3m_add(  GF3m& x,  GF3m& y, GF3m& r) {
  __m128d aL, aH, bL, bH;
  aL = *(const __m128d *)&(x.L);
  aH = *(const __m128d *)&(x.H);
  bL = *(const __m128d *)&(y.L);
  bH = *(const __m128d *)&(y.H);
  __m128d T = _mm_and_pd(_mm_or_pd(aL, aH), _mm_or_pd(bL, bH));
  *(__m128d*)(r.L) = _mm_xor_pd(T, _mm_or_pd(aL, bL));
  *(__m128d*)(r.H) = _mm_xor_pd(T, _mm_or_pd(aH, bH));
  .
  .
}
```

logic functions used are: _mm_and_pd, _mm_or_pd and _mm_xor_pd.

Alignment Support

Aligning data improves the performance of intrinsics. When using the Streaming SIMD Extensions, you should align data to 16 bytes in memory operations. Specifically, we must align __m128 objects as addresses passed to the _mm_load and _mm_store

intrinsics.

Use `__declspec(align(n))` to direct the compiler to align data more strictly than it otherwise would. For example, a data object of type int is allocated at a byte address which is a multiple of 4 by default. However, by using `__declspec(align(n))`, you can direct the compiler to instead use an address which is a multiple of 8, 16, or 32.

You can use this data alignment support as an advantage in optimizing cache line usage. By clustering small objects that are commonly used together into a struct, and forcing the struct to be allocated at the beginning of a cache line, you can effectively guarantee that each object is loaded into the cache as soon as any one is accessed, resulting in a significant performance benefit.

The syntax of this extended-attribute is as follows:
align(n)
where n is an integral power of 2. If a value is specified that is less than the alignment of the affected data type, it has no effect. In other words, data is aligned to the maximum of its own alignment or the alignment specified with `__declspec(align(n))`.

Example:

```
#define uint  unsigned int
#define MyAlign(x) __declspec( align(x) )
typedef struct GF3_m {
   MyAlign(16) uint L[MAX_SIZE];
   MyAlign(16) uint H[MAX_SIZE];
} GF3m, *pGF3m;
```

In this example, the memory addresses are alienated in multiples of 16 bytes.

Using the optimizations with which we can improve the performance of our application, the command line to compile our application is as follows:

Windows*

```
#>icl eta_T_Pairing /O3 /Qip /Qprof_gen /Qprof_dir
#>eta_T_Pairing.exe
```

Linux*

```
#>icl eta_T_Pairing -O3 -Qip -Qprof_gen -Qprof_dir
#>./a.out
```

B.2. Intel® VTune Performance Analyzer

The VTune(TM) Performance Analyzer provides an integrated performance analysis and tuning environment with graphical user interface and command-line capabilities that help you analyze code performance on systems with IA-32 architecture, Intel(R) Itanium(R) architecture, and systems with Intel(R) EM64T technology [25].

VTune helps you identify and characterize performance issues by:

- Collecting performance data from the system running your application.
- Organizing and displaying the data in a variety of interactive views, from system-wide down to source code or processor instruction perspective.
- Identifying potential performance issues and suggesting improvements.

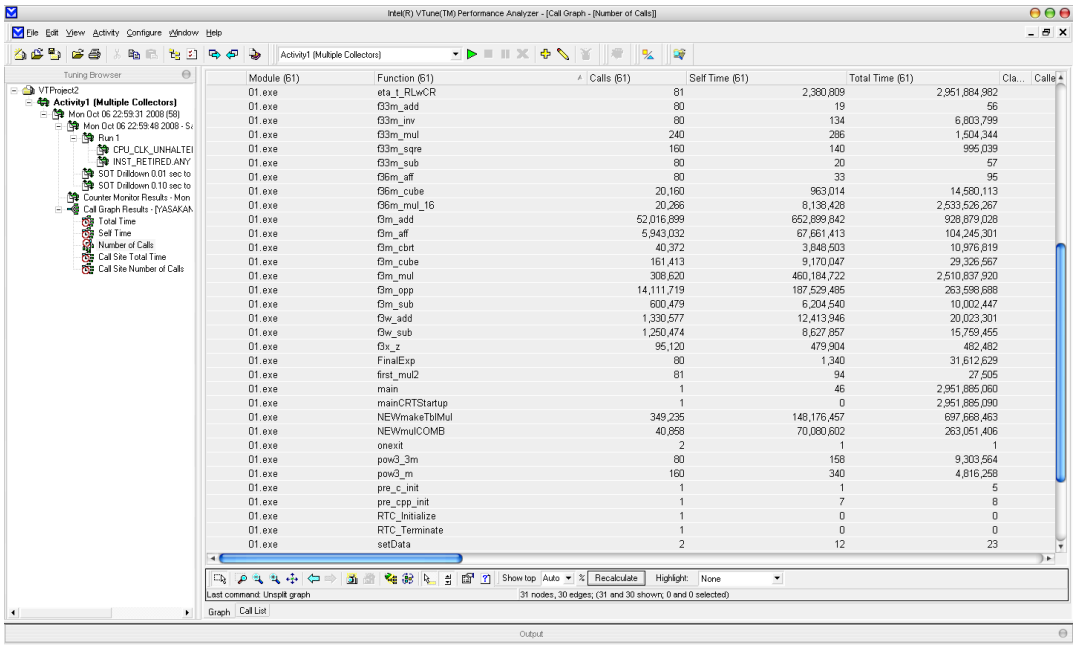


Figura B.2: VTune: Number of calls by function

The figure B.2 is a results screen that provides VTune which shows the functions declared in the application to compute the η_T Pairing and the number of calls made to each function.

B.2.1. Sampling Mode

During sampling, the VTune monitors all the software executing on your system including the operating system, applications, and device drivers. When you run the VTune analyzer does the following:

1. Waits for the delay sampling time (if specified) to elapse and then starts collecting samples.

2. Interrupts the processor at the specified sampling interval and collects samples of instruction addresses. For every interrupt, the VTune analyzer collects one sample.

3. Stores the execution context of the software currently executing on your system.

The analyzer analyzes the collected samples and displays the hotspots or bottlenecks in the Hotspot view.

We can see the details of the hotspots to the source or assembly code. As an example, to compute of the η_T pairing in characteristic two, the function most expensive is a function called `mul_8word32`, that is the multiplication of two large numbers, each represented with 8 words of 32 bits.

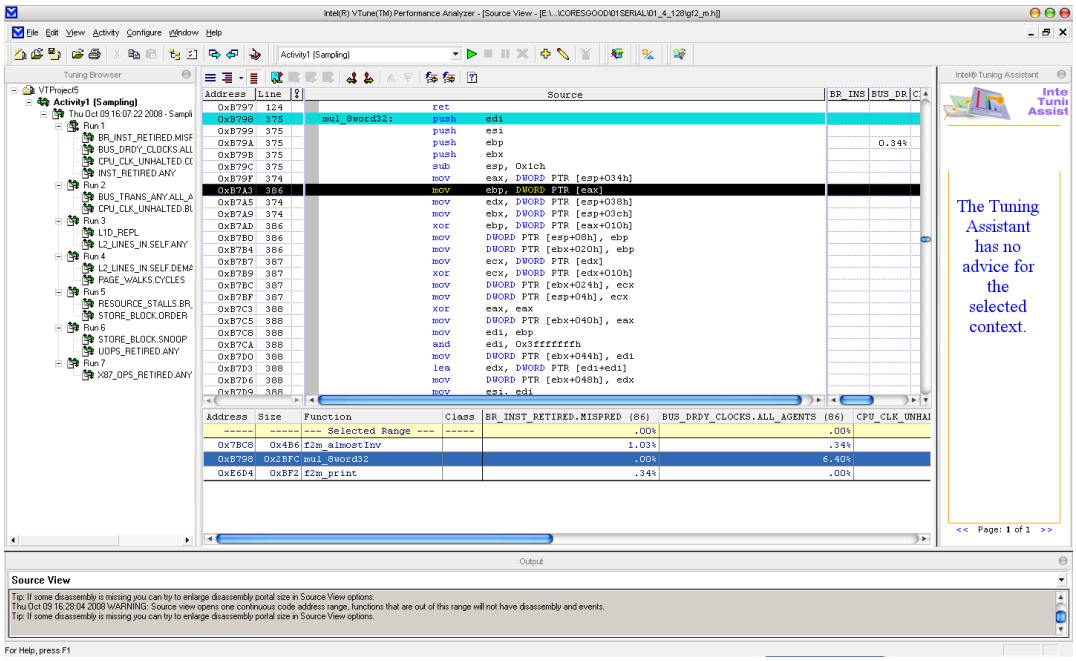


Figura B.3: VTune: Viewing the source code.

View the source causing the hotspots and get advice on how you can modify your code to remove the bottlenecks and improve the performance of your application, however, this operation is complex (`mul_8word32`) and for this reason turns out to be costly.

B.2.2. Call Graph Profiling

Call graph profiling provides developers with a pictorial view of program flow to quickly identify critical functions and call sequences. You can use the different profilers in the VTune analyzer to understand the different aspects of the performance of your application.

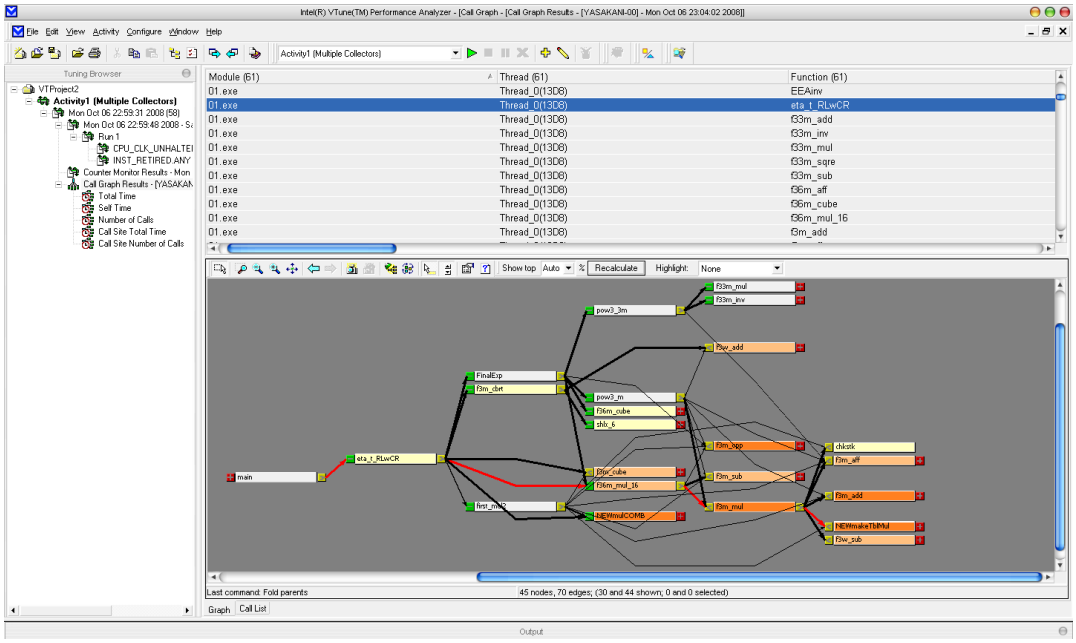


Figura B.4: VTune: Call graph

On having executed the program for computing the η_T Pairing we can see that the critical path of the program is defined by the multiplication function, see figure B.4. To use the call graph set the flag "/fixed:no" to the link command line or set the environment variable "LINK=/fixed:no".

B.2.3. Counter Monitor

Counter monitor allows developers to track system activity during runtime which helps identify system level performance issues. VTune can also be used for:

- Determine if a sample section of code can be threaded
- Determine if the implemented threads are balanced or not
    - On the thread view, each of the threads should be consuming the same amount of time, if not, you must shift the amount of work between the threads

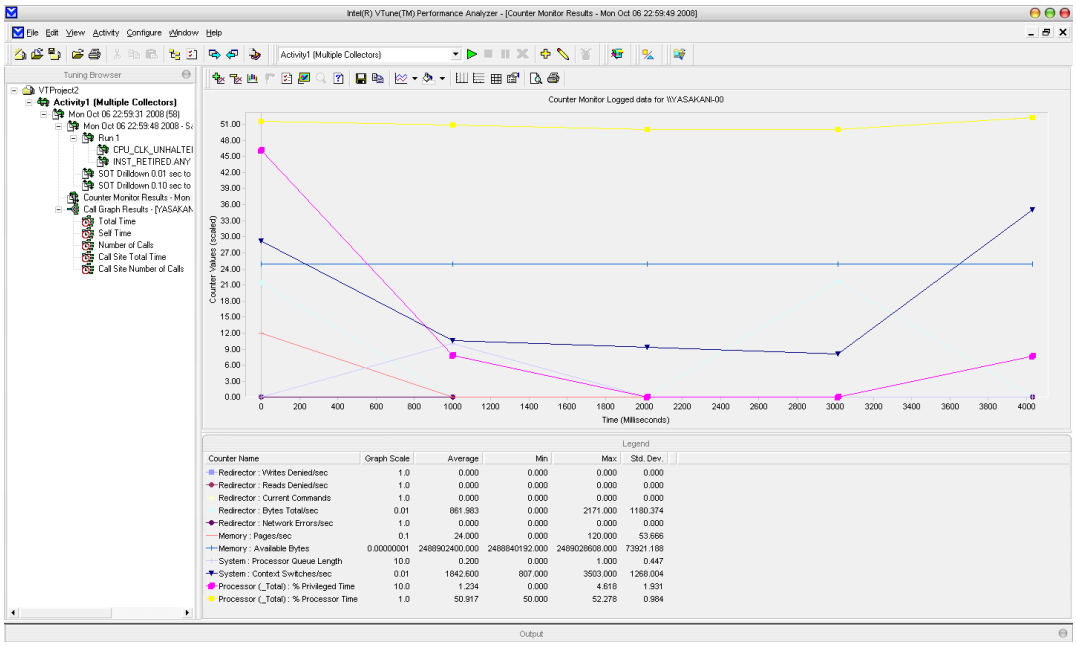


Figura B.5: VTune: Counter monitor

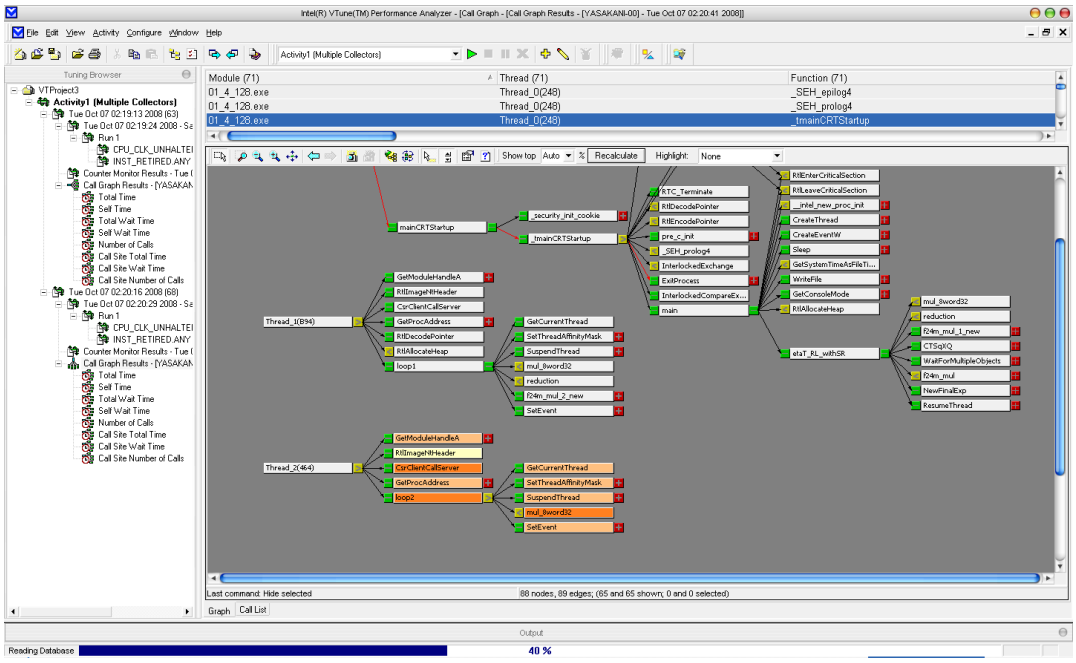


Figura B.6: VTune: Threading view

In the figure B.6 we can observe the behavior for the calculation of the η_T pairing using two processors, that is to say, 2 threads.

B.3. Intel® Thread Checker

Intel(R) Thread Checker is designed to help developers create threaded applications more quickly by detecting thread safety issues in programs that use Win32*, Win64* and OpenMP* threads. For a review of how to create an Activity using Thread Checker, see [22].

To build using the Intel® Compiler from the command line, issue the command: `icl /Zi /Qtcheck MyProgram.c /link /fixed:no`.

The picture below is a sampling of the results provided by the Intel® Thread Checker that shares the environment (IDE) with Intel® VTune.

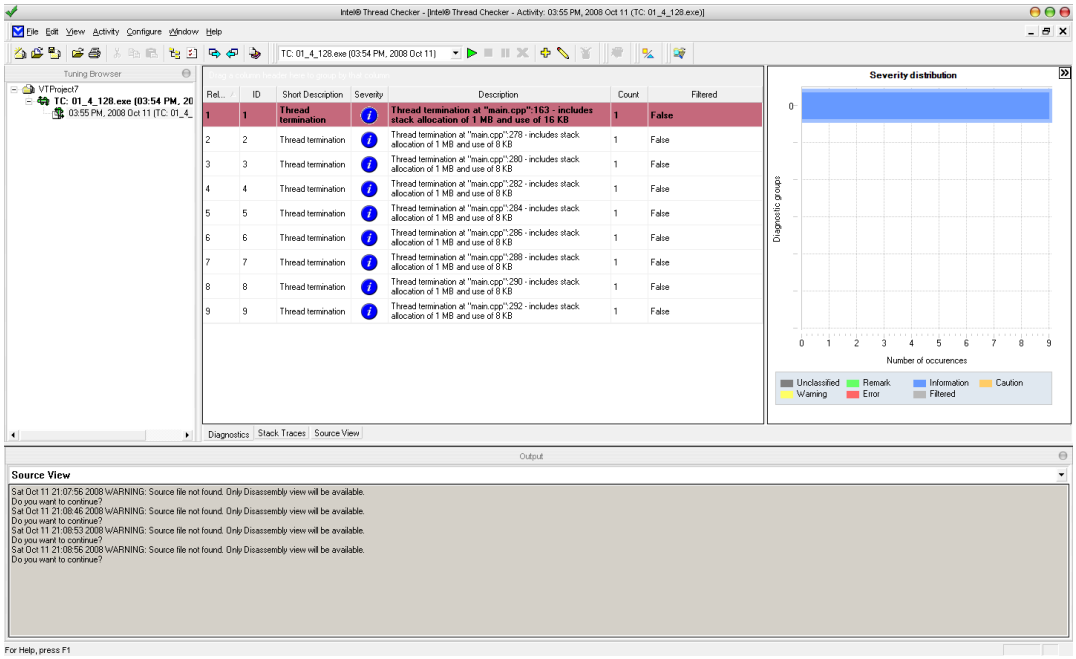


Figura B.7: Thread Checker: The calculation of the η_T pairing with 8 threads.

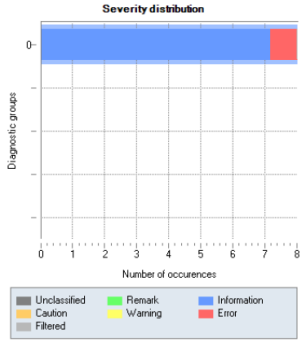
When your application uses more than one thread, you must set the `/MDd` compiler option under the `C/C++ >Code generation >Runtime` library property.

B.3.1. Severity Distribution

The Severity distribution or Graphical Summary View in the right-hand pane of Thread Checker gives you a "big picture" snapshot of the number and types of errors found. The bars are color-coded to indicate the severity of problems found during the Activity run: "red" indicates the most severe errors, "orange" indicates warnings, "yellow" indicates cautions, "blue" and "green" indicate informational remarks.

The figure B.8 shows a sample of the Severity distribution. Note that most of the diagnostics are "blue" indicating that they are informational remarks. The remaining diagnostics are "red" indicating severe errors.

Figura B.8: Severity distribution.



B.3.2. Diagnostics List

The main screen shows a list called "Diagnostics List" it identifies each diagnostic by a unique ID, provides a short description and its estimated Severity as well as additional details including information about where the diagnostic occurred, its frequency, and more. See the figure B.9.

Rel...	ID	Short Description	Severity	Description	Count	Filtered
1	1	Thread termination		Thread termination at "main.cpp":163 - includes stack allocation of 1 MB and use of 16 KB	1	False
2	2	Thread termination		Thread termination at "main.cpp":278 - includes stack allocation of 1 MB and use of 8 KB	1	False
3	3	Thread termination		Thread termination at "main.cpp":280 - includes stack allocation of 1 MB and use of 8 KB	1	False
4	4	Thread termination		Thread termination at "main.cpp":282 - includes stack allocation of 1 MB and use of 8 KB	1	False
5	5	Thread termination		Thread termination at "main.cpp":284 - includes stack allocation of 1 MB and use of 8 KB	1	False
6	6	Thread termination		Thread termination at "main.cpp":286 - includes stack allocation of 1 MB and use of 8 KB	1	False
7	7	Thread termination		Thread termination at "main.cpp":288 - includes stack allocation of 1 MB and use of 8 KB	1	False
8	8	Thread termination		Thread termination at "main.cpp":290 - includes stack allocation of 1 MB and use of 8 KB	1	False
9	9	Thread termination		Thread termination at "main.cpp":292 - includes stack allocation of 1 MB and use of 8 KB	1	False

Figura B.9: Diagnostics list

The `/Qtcheck` option is an Intel® Compiler option that enables compile-time source instrumentation of the binary program.

Source locations where shared memory or synchronization conflicts could cause non-deterministic results or deadlock are shown, including call stacks for each participating thread.

B.3.3. Viewing the Source

Source View enables you to locate where a diagnostic occurred in your source code.

You can see the source code of the application where the problems are detected: data races, storage conflicts, deadlock, and so on. You can choose which way to see the source code: assembly(ASM) or the programming language used to build the application (C, C++).

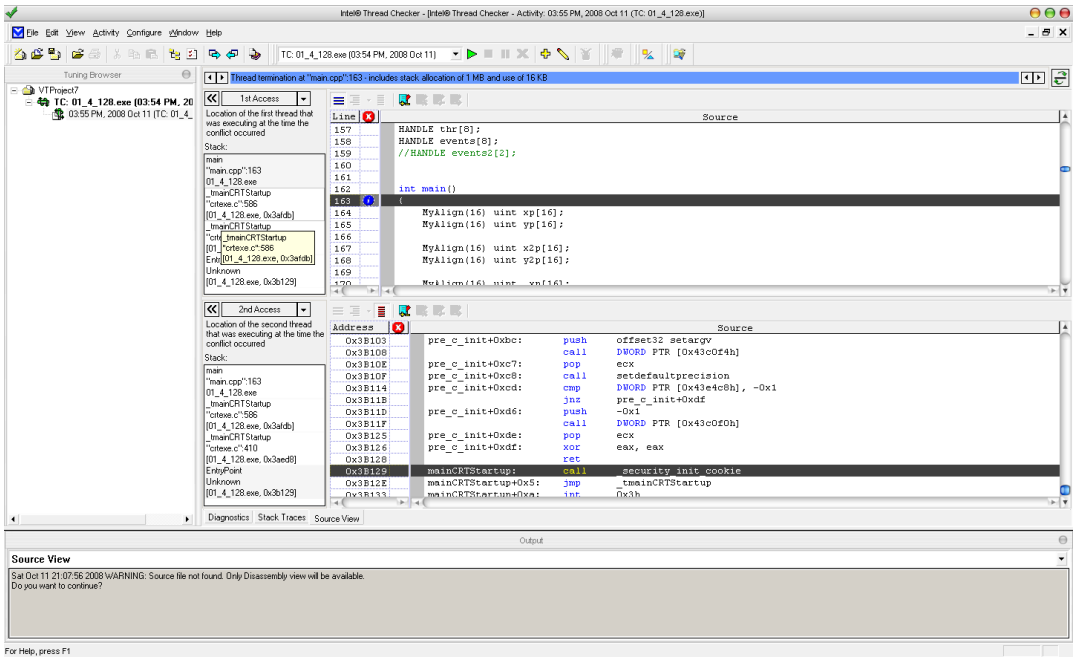


Figura B.10: Source and Assembler code.

If Thread Checker does not show results, select `Help >Search` and type "troubleshooting Thread Checker" to search for relevant help topics. The help topic Troubleshooting Intel® Thread Checker offers possible solutions.

B.4. Intel® Thread Profiler

Use Thread Profiler to identify bottlenecks that limit the parallel performance of your multi-threaded application, locate synchronization delays, stalled threads, excessive blocking time, and ineffective utilization of processors, find the best sections of code to optimize for sequential performance and for threaded performance and compare scalability across different numbers of processors or using different threading methods [23].

The following figure show the results of the execution of the project to compute the η_T Pairing in way secuential. The results vary depending on your system configuration, but you should see two color charts, one with horizontal bars called *Timeline view*, and one with vertical bars called *Profile view*.

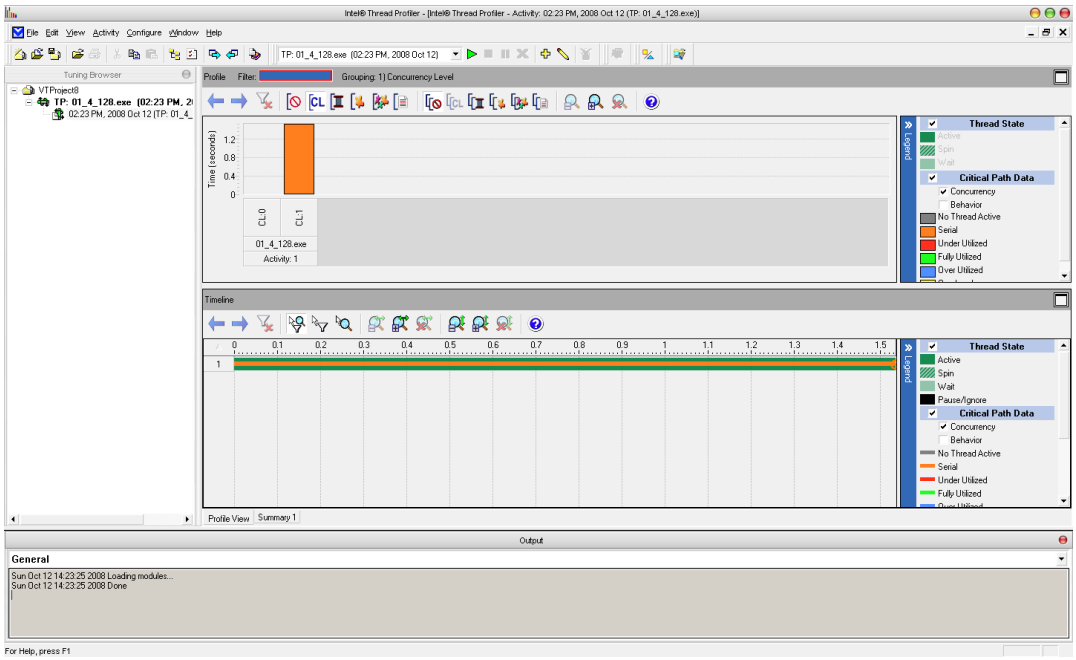


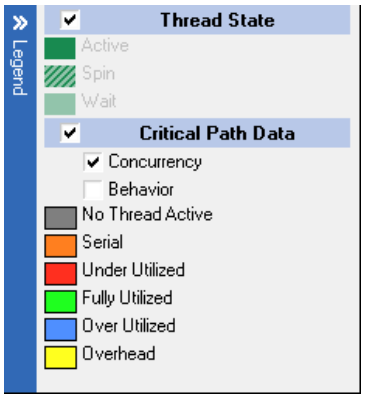
Figura B.11: Thread profiler: Serial implementation of the η_T pairing.

B.4.1. Profile view and Timeline view

The Profile view (top) displays a high-level summary of the time spent on the critical path, decomposed into time categories. The Timeline view (bottom) illustrates the behavior of your program over time.

In Profile view, you can also click any bar to display the detailed timing information in the Statistics table under the graph.

The color scheme is given in the Legend as shown in the following figure.



Timeline view shows the contribution of each thread to the total program, whether on the default critical path or not.

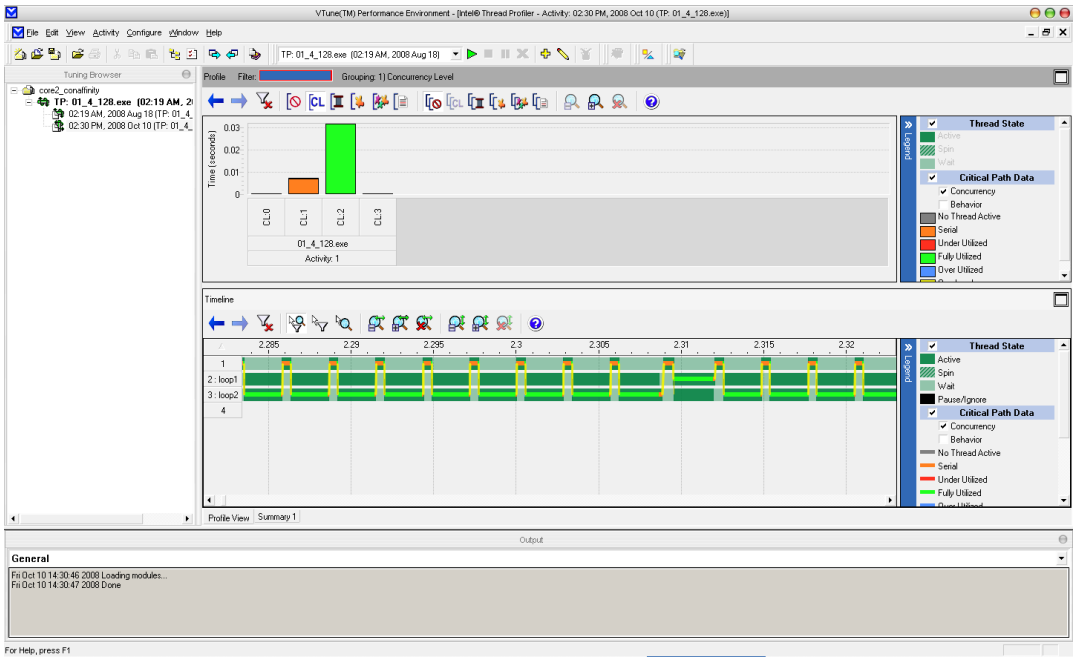


Figura B.12: Thread profiler: Parallel implementation of the η_T pairing (2 cores).

The Timeline view shows the behavior of the program over time and across threads. The figure above is an example of the execution of the project to compute the η_T Pairing using two threads.

Thread Profiler tracks the flow of all threads in the application. The critical path is the continuous path from the beginning of execution to the critical path target. By default, the critical path target is the end of program execution.

Thread Profiler breaks time into different Time Categories, represented in the graphs and Legends by different colors.

- Serial times (shades of orange) indicate serial portions of the code.
- Under Utilized times (shades of red) indicate that the code is not fully utilizing all processors.
- Fully Utilized times (shades of green) indicate that portions of the code demonstrate good processor utilization.
- Over Utilized times (shades of blue) indicate that portions of the code use more threads than there are processors.

The most common problem when you deploy an application multi-threading is the unbalanced load and the overhead caused by the transitions between threads ie. communication and synchronization between threads.

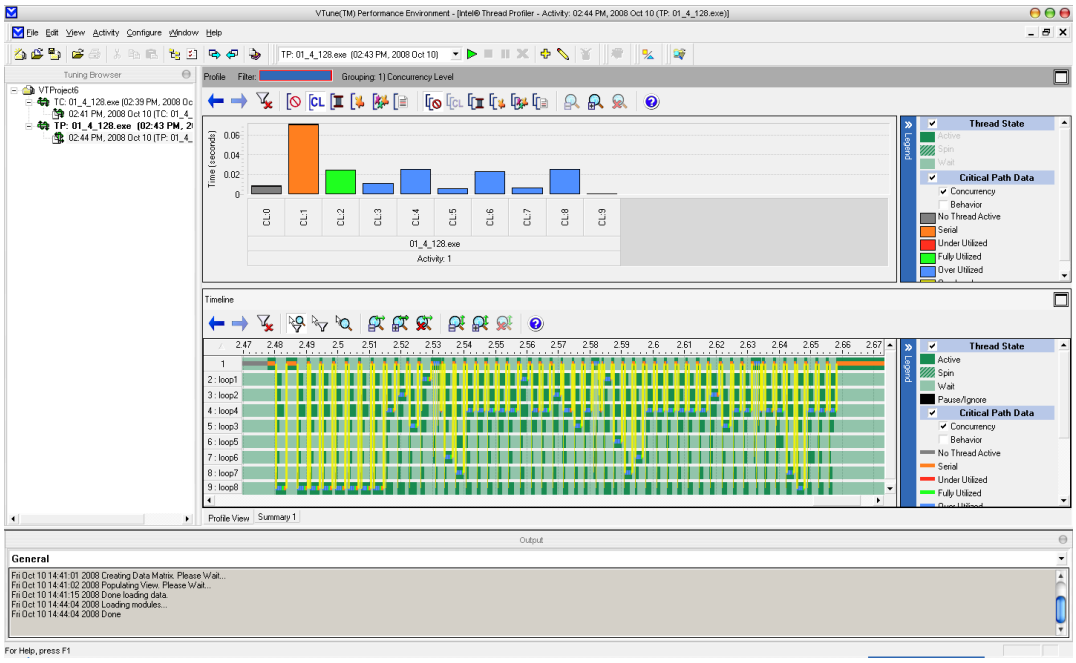


Figura B.13: Parallel implementation inefficient for 8 cores (unbalanced load).

The figure B.13 shows a clear example of unbalanced load though the program accomplishes its task correctly, the implementation is inefficient.

Thanks to profiler you can correct the unbalanced and reduce overhead.
In the next figure you should see improvements, in Timeline view, the threads now show improved workload balance.

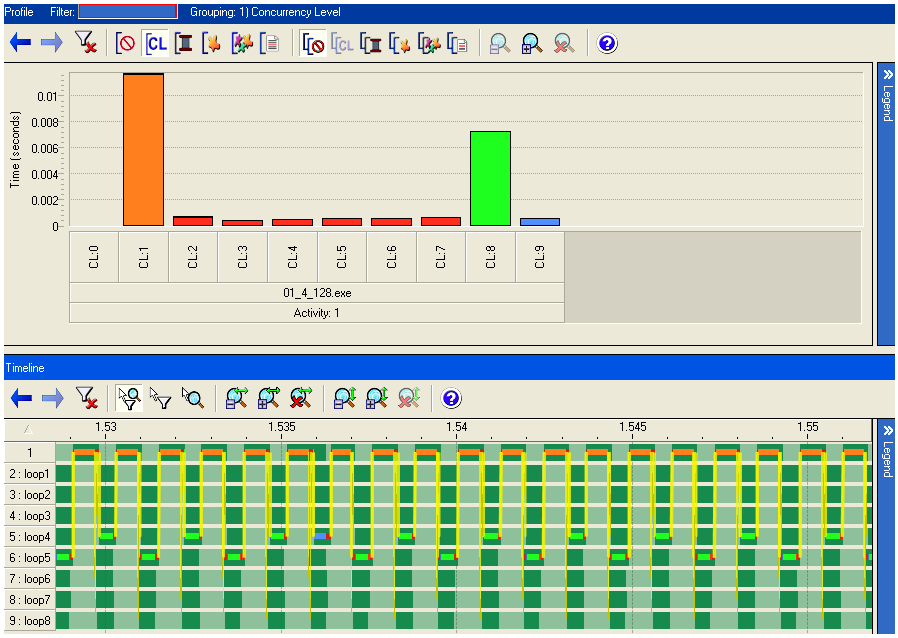


Figura B.14: Parallel implementation efficient for 8 cores.

The figure B.14 shows the compute of the η_T Pairing distributed in the 8 processors working in parallel. This latest plot belongs to a more efficient implementation that performs the compute of the η_T Pairing on a multicore architecture, and avoids the problem of load balancing and overhead.

B.5. Intel® Threading Building Blocks

Intel® Threading Building Blocks is a runtime-based parallel programming model for C++ code that uses threads [24]. It consists of a template-based runtime library to help you harness the latent performance of multicore processors. Use Intelő Threading Building Blocks to write scalable applications that:

- Specify tasks instead of threads.
- Emphasize data parallel programming.

- Take advantage of concurrent collections and parallel algorithms.

Intel® Threading Building Blocks is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. It is designed to promote scalable data parallel programming. Additionally, it fully supports nested parallelism, so you can build larger parallel components from smaller parallel components. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner.

B.5.1. Advantages

Improves developer productivity by using task-based abstractions that make it easier to get scalable and reliable parallel applications with less lines of code. Task-based algorithms, containers and synchronization primitives simplify parallel application development,

Application performance automatically improves as processor core count increases by using abstract tasks. Sophisticated task scheduler dynamically maps tasks to threads to balance the load among available cores, preserve locality and maximize parallel performance.

Intel® Threading Building Blocks is coded in C++ and available on a multitude of platforms to provide a cross-platform solution for parallelism. Intel TBB is available as a standalone product or with the Intel® Compiler Professional Editions for a more complete and cost-effective solution.

B.5.2. Using TBB

The simplest form of scalable parallelism is a loop of iterations that can each run simultaneously without interfering with each other.

B.5.3. Initializing and Terminating the Library

Any thread that uses an algorithm template from the library or the task scheduler must have an initialized `tbb::task_scheduler_init` object. The components are in defined in namespace `tbb`.

The task scheduler shuts down when all `task_scheduler_init` objects terminate. By default, the constructor for `task_scheduler_init` does the initialization and the destructor does termination.

```
#include "tbb/task_scheduler_init.h"
using namespace tbb;

int main( ) {
  task_scheduler_init init;
  ...
  return 0;
}
```

The constructor for `task_scheduler_init` takes an optional parameter that specifies the number of desired threads, including the calling thread. There is a method `task_scheduler_init::terminate` for terminating the library early before the `task_scheduler_init` is destroyed.

```
int main( ) {
  int nthread = 3;
  task_scheduler_init init(task_scheduler_init::deferred);
  init.initialize(nthread);
  ...
  init.terminate();
  return 0;
}
```

you can omit the call to `terminate()`, because the destructor for `task_scheduler_init` checks if the `task_scheduler_init` was initialized, and if so, performs the termination.

The task scheduler is somewhat expensive to start up and shut down, put the `task_scheduler_init` in `main`, and do not try to create a scheduler every time you use a parallel algorithm template.

Class task

Class `task` is the base class for tasks, and is an abstract base class. Programmers are expected to derive classes from `task`, and override the virtual method `task* task::execute()`.

Method `execute` should perform the necessary actions for running the task, and then return the next `task` to execute, or NULL if the scheduler should choose the next task to execute. Typically, if non-NULL, the returned task is one of the children of `this`.

Processing of execute

When the scheduler decides that a thread should begin executing a *task*, it performs the following steps:

1. Invoke `execute()` and wait for it to return.
2. If the task has not been marked by a method `recycle_*`:
   - If the task's *parent* is not null, then atomically decrement *parent->refcount*, and if becomes zero, put the *parent* into the ready pool.
   - Call the task's destructor.
   - Free the memory for task for reuse.
3. If the task has been marked for recycling:
   - If marked by `recycle_to_reexecute`, put the task back into the ready pool.
   - Otherwise it was marked by `recycle_as_child` or `recycle_as_continuation`.

B.5.3.1. Scheduling Algorithm

The scheduler employs a technique known as *work stealing*. Each thread keeps a "ready pool" of tasks that are ready to run.

The ready pool is structured as an array of lists of `task`, where the list for the *ith* element corresponds to tasks at level i in the tree. The lists are manipulated in last-in first-out order. A task at level i spawns child tasks at level $i+1$.

A thread pulls tasks from the deepest non-empty list in the array. If there are no non-empty lists, the thread randomly steals a task from the shallowest list of another thread. A thread also implicitly steals if it completes the last child, in which case it starts executing the task that was waiting on the children.

B.5.3.2. TBB in the Project

In this project class called `Loop1Class` and `Loop2Class` perform the same instructions with different data. Both classes are completely independent and can be run in parallel.

```
class Loop1Class:public task{
  ...
public:
  Loop1Class(){  }     //constructor.
//override the virtual method execute
```

```
  task * execute(){
    int i,j;
    GF3m tbl[16][16];
    GF3m S[6], t, u;
    ...
    S[4].L[0] =0x0;  S[4].H[0] =0x1;
    for( i=1; i<=125; i++ ){
      f3m_add( XPcbrt[i], XQcb[i], t );
      NEWmakeTblMul(YPcbrt[i], tbl);
      NEWmulCOMB( tbl, YQcb[i], u );
      NEWmakeTblMul(t, tbl);
      NEWmulCOMB( tbl, t, S[0] );
      f3m_opp( S[0], S[0] );
      f3m_opp( u, S[1] );
      f3m_opp( t, S[2] );
      f36m_mul_16_MOD( S, Rl01, Rl01 );
    }
    return NULL;
  }
};
```

In the class LoopMaster will spawn all the loop classes as children threads (*Loop1Class* and *Loop2Class*). Here is where it really is the parallelization for the tasks.

With the method *spawn* the threads do their work and with the instruction *spawn_ and_ wait_for_* the main thread waiting for the results of the threads children in this case for *loop1C* and *loop2c*.

```
class LoopMaster:public task{
public:
  LoopMaster(){}
  task* execute(){
    Loop1Class &loop1C = *new(allocate_child()) Loop1Class();
    Loop2Class &loop2C = *new(allocate_child()) Loop2Class();
    spawn(loop1C);
    spawn_and_wait_for_all(loop2C);
    return NULL;
  }
};
```

The following bit of code belongs to the main function called `eta_t_TBB` that performs the compute of the η_T Pairing. Here is an instance of class LoopMaster and then calls the function `spawn_root_and_wait`

```
__inline void eta_t_TBB( GF3m& xp, . . .)
{
    .
    .
    LoopMaster &master = *new(task::allocate_root()) LoopMaster();
    task::spawn_root_and_wait(master);
    .
    .
}
```

Segments of code presented here belong to an implementation for a dual core architecture to calculate efficiently the η_T Pairing, but also were built efficient implementations for a total of up to 8 cores.

To compile the program we have to add the following items:

- *Include files*: <InstalPath>/Intel/TBB/<version>/include/TBB/*.h
- *.lib files*: <InstalPath>/Intel/TBB/<version>/<architecture>/ <VCversion>/lib/<lib>.lib
- *.dll files*: <InstalPath>/Intel/TBB/<version>/<architecture>/ <VCversion>/bin/<lib><malloc>.dll

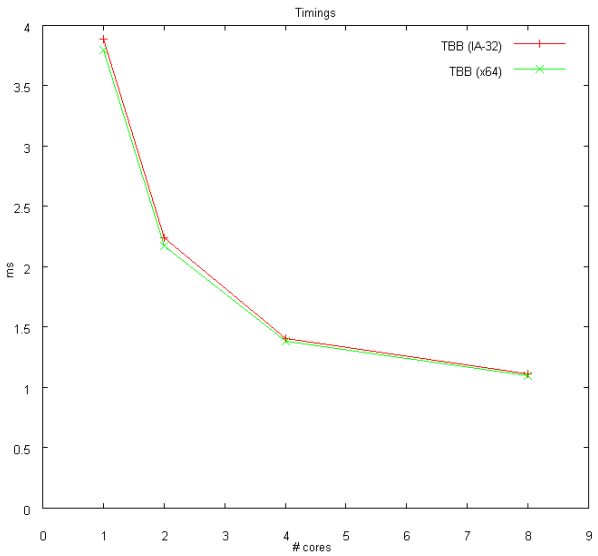
Results

The results are as follows:

Implemetation of the η_T Pairing in characteristic two

Cores	TBB(IA-32)	TBB(x64)
Serial	3.891 ms.	3.80 ms.
2	2.243 ms	2.173ms
4	1.406 ms	1.387ms
8	1.109 ms	1.096ms

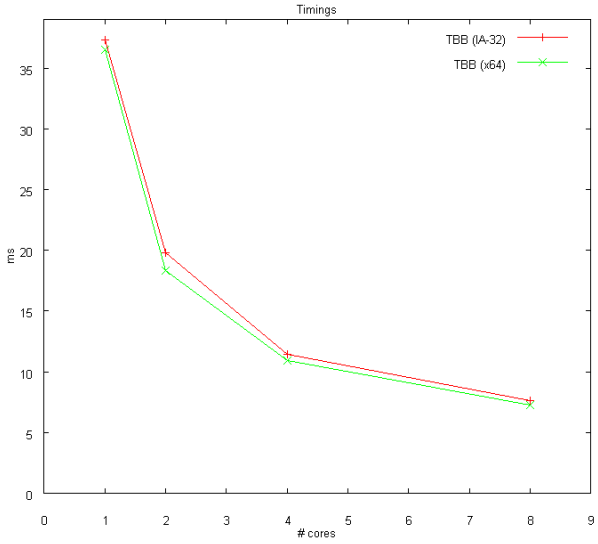
Tabla B.2: η_T Pairing in characteristic two.



Implemetation of the η_T Pairing in characteristic three

Cores	TBB(IA-32)	TBB(x64)
Serial	37.34 $ms.$	36.48 $ms.$
2	19.81 $ms.$	18.34 $ms.$
4	11.406 $ms.$	10.96 $ms.$
8	7.64 $ms.$	7.271 $ms.$

Tabla B.3: η_T Pairing in characteristic three.



BIBLIOGRAFÍA

[1] Federal Information Processing Standards Publication 197 (FIBS PUB 197). Advanced Encryption Standard (AES), November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[2] Ismail I. A., Amin Mohammed, and Diab Hossam. How to repair the hill cipher. *Journal of Zhejiang University - Science A*, 7:2022–2030, February 2006.

[3] Omran Ahmadi, Darrel Hankerson, and Alfred J. Menezes. Formulas for cube roots in $\mathbb{F}_{3^m}$. *Discrete Applied Mathematics*, 155(3):260–270, 2007.

[4] Omran Ahmadi, Darrel Hankerson, and Alfred J. Menezes. Software implementation of arithmetic in $\mathbb{F}_{3^m}$. In Claude Carlet and Berk Sunar, editors, *WAIFI*, volume 4547 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2007.

[5] Paulo S. L. M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004. http://eprint.iacr.org/.

[6] Paulo S. L. M. Barreto, Steven D. Galbraith, Colm Ó' Héigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, 2007. See also Cryptology ePrint Archive, Report 2004/375.

[7] Paulo S. L. M. Barreto and Hae Yong Kim. Fast hashing onto elliptic curves over fields of characteristic 3. Cryptology ePrint Archive, Report 2001/098, 2001. http://eprint.iacr.org/.

[8] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, volume 2442, pages 354–368, London, UK, 2002. Springer-Verlag.

[9] Rana Barua, Ratna Dutta, and Palash Sarkar. Extending joux's protocol to multi party key agreement. Cryptology ePrint Archive, Report 2003/062, 2003. http://eprint.iacr.org/.

[10] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, and Eiji Okamoto. Arithmetic operators for pairing-based cryptography. In P. Paillier and I. Verbauwhede, editors, *Proceedings of CHES 2007*, number 4727 in Lecture Notes in Computer Science, pages 239–255. Springer, 2007. Best paper award.

[11] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, Masaaki Shirase, and Tsuyoshi Takagi. Algorithms and arithmetic operators for computing the η_T pairing in characteristic three. *IEEE Trans. Comput.*, 57(11):1454–1468, November 2008. See also Cryptology ePrint Archive, Report 2007/417.

[12] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, and Francisco Rodríguez-Henríquez. A comparison between hardware accelerators for the modified tate pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. Cryptology ePrint Archive, Report 2008/115, 2008. http://eprint.iacr.org/.

[13] Jean-Luc Beuchat, Hiroshi Doi, Kaoru Fujita, Atsuo Inomata, Akira Kanaoka, Masayoshi Katouno, Masahiro Mambo, Eiji Okamoto, Takeshi Okamoto, Takaaki Shiga, Masaaki Shirase, Ryuji Soga, Tsuyoshi Takagi, Ananda Vithanage, and Hiroyasu Yamamoto. Fpga and asic implementations of the η_t pairing in characteristic three. Cryptology ePrint Archive, Report 2008/280, 2008. http://eprint.iacr.org/.

[14] Jean-Luc Beuchat, Masaaki Shirase, Tsuyoshi Takagi, and Eiji Okamoto. An algorithm for the nt pairing calculation in characteristic three and its hardware implementation. In *ARITH '07: Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 97–104, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229, London, UK, 2001. Springer-Verlag.

[16] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248, pages 514–532, London, UK, 2001. Springer-Verlag.

[17] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, pages 71–90, 2002.

[18] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. RFC-4880 OpenPGP Message Format, November 2007. (ObsoletesRFC1991, RFC2440)(Status: PROPOSED STANDARD).

[19] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363, London, UK, 2001. Springer-Verlag.

[20] Henri Cohen and Gerhard Frey. *Handbook of elliptic and hyperelliptic curve cryptography*. Discrete Mathematics and Its Applications 34, Chapman & Hall/CRC Press, July 2005.

[21] Intel Corporation. Intel® C++ Compiler Documentation. Document number: 304968-021US.

[22] Intel Corporation. Intel® Thread Checker Documentation.

[23] Intel Corporation. Intel® Thread Profiler Documentation.

[24] Intel Corporation. Intel® Threading Building Blocks. Refence Manual, Document number: 315415-001US.

[25] Intel Corporation. Intel® VTune Performance Analyzer Documentation.

[26] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A Instruction Set Reference, A-M, April 2008.

[27] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A Instruction Set Reference, N-Z, April 2008.

[28] Iwan Duursma and Hyang-Sook Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *ASIACRYPT '03: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2894, pages 111–123. Springer-Verlag, 2003.

[29] Kenny Fong, Darrel Hankerson, Julio Lopez, and Alfred J. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, August 2004.

[30] Ian Foster. *Designing and Building Parallel Programs*. Addison Wesley, 1995.

[31] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, 1994.

[32] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the tate pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, volume 2369, pages 324–337, London, UK, 2002. Springer-Verlag.

[33] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the tate pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer-Verlag.

[34] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156(16):3113–3121, 2008.

[35] W. Gao, X. Wang, G. Wang, and F. Li. One-round id-based blind signature scheme without ros assumption. Cryptology ePrint Archive, Report 2007/007, 2007. http://eprint.iacr.org/.

[36] Elisa Gorla, Christoph Puttmann, and Jamshid Shokrollahi. Explicit formulas for efficient multiplication in $\mathbb{F}_{3^{6m}}$. In *Lecture Notes in Computer Sciences; Selected Areas in Cryptography 2007*, volume 4876, pages 173–183, Ottawa, Canada, August 2007. Springer.

[37] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing (2nd Edition)*. Addison Wesley, January 2003.

[38] R. Granger and M. Stam. On small characteristic algebraic tori in pairing-based cryptography. *LMS Journal of Computation and Mathematics*, 9:64–85, 2004.

[39] Darrel Hankerson, Julio López Hernandez, and Alfred J. Menezes. Software implementation of elliptic curve cryptography over binary fields. In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–24, London, UK, 2000. Springer-Verlag.

[40] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, New Jersey, USA, 2003.

[41] Ryuichi Harasawa, Junji Shikata, Joe Suzuki, and Hideki Imai. Comparing the MOV and FR Reductions in Elliptic Curve Cryptography. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT' 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 190–205. Springer, 1999.

[42] Keith Harrison, Dan Page, and Nigel Smart. Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems. *London Mathematical Society J. Comput. Math.*, 5(-):181–193, November 2002.

[43] Francisco Rodriguez Henriquez, Guillermo Morales-Luna, and Julio López. Low-Complexity Bit-Parallel Square Root Computation over $GF(2^m)$ for All Trinomials. *IEEE Trans. Comput.*, 57(4):472–480, 2008.

[44] Florian Hess. Efficient identity based signature schemes based on pairings. In *SAC '02: Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, pages 310–324, London, UK, 2003. Springer-Verlag.

[45] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 2000.

[46] Anatoly A. Karatsuba and Yu Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady (English translation)*, 7:595–596, January 1963.

[47] Yuto Kawahara, Tsuyoshi Takagi, and Eiji Okamoto. Efficient implementation of tate pairing on a mobile phone using java. In Yuping Wang, Yiu ming Cheung, and Hailin Liu, editors, *CIS*, volume 4456 of *Lecture Notes in Computer Science*, pages 396–405. Springer, 2006.

[48] Neal Koblitz. Elliptic curve in cryptography. *American Mathematical Society J. Comput. Math.*, 48(177):207–209, January 1987.

[49] G. Kumar-Verma. New id-based fair blind signatures. Cryptology ePrint Archive, Report 2008/093, 2008. http://eprint.iacr.org/.

[50] Julio López and Ricardo Dahab. High-speed software multiplication in f2m. In *INDOCRYPT '00: Proceedings of the First International Conference on Progress in Cryptology*, pages 203–212, London, UK, 2000. Springer-Verlag.

[51] Lourdes López-García, Luis Martínez-Ramos and Francisco Rodríguez-Henríquez. A comparative performance analysis of nine blind signature schemes. In *The 5th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2008)*, Mexico City, Mexico., November 2008.

[52] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[53] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, October 1996.

[54] Victor S. Miller. Short programs for functions on curves, 1986. Unpublished manuscript, available online at http://crypto.stanford.edu/miller/miller.pdf.

[55] Victor S Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[56] National Institute of Standards and Technology. Recommended elliptic curves for federal government use, July 1999. Disponible en: http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf.

[57] Leonardo B. Oliveira, Diego Aranha, Eduardo Morais, Felipe Daguano, Julio López, and Ricardo Dahab. Tinytate: Identity-based encryption for sensor networks. Cryptology ePrint Archive, Report 2007/020, 2007. http://eprint.iacr.org/.

[58] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and publick-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[59] Richard Schroeppel, Hilarie Orman, Sean W. O'Malley, and Oliver Spatscheck. Fast key exchange with elliptic curve systems. In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 43–56, London, UK, 1995. Springer-Verlag.

[60] Michael Scott. Optimal irreducible polynomials for $\mathbb{F}_(2^m)$ arithmetic. Cryptology ePrint Archive, Report 2007/192, 2007. http://eprint.iacr.org/.

[61] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[62] Masaaki Shirase, Tsuyoshi Takagi, and Eiji Okamoto. Some efficient algorithms for the final exponentiation of η_T pairing. In *ISPEC '07: 3rd International Information Security Practice and Experience Conference*, volume 4464 of *Lecture Notes in Computer Science*, pages 254–268, Hong Kong, China, May 2001. Springer-Verlag.

[63] Chang Shu, Soonhak Kwon, and Kris Gaj. FPGA accelerated tate pairing based cryptosystems over binary fields. In *Field Programmable Technology, 2006. FPT'06. IEEE International Conference on*, pages 173–180. IEEE, December 2006.

[64] Joseph H. Silverman. The arithmetic of elliptic curves. Springer - Graduate Texts in Mathematics, Vol. 106, 1986. 1st ed. 1986. Corr. 3rd printing, 1994.

[65] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 2005.

[66] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman & Hall / CRC, Secaucus, New Jersey, USA, 2003.

[67] Y. Rangel-Romero, R. Vega-García, A. Menchaca-Méndez, D. Acoltzi-Cervantes, L. Martínez-Ramos, M. Mecate-Zambrano, F. Montalvo-Lezama, J. Barrón-Vidales, N. Cortez-Duarte and F. Rodríguez-Henríquez. Comments on "How to repair the Hill cipher". *Journal of Zhejiang University - Science A*, 9(2):211–214, February 2008.

[68] Fangguo Zhang and Kwangjo Kim. Id-based blind signature and ring signature from pairings. In *ASIACRYPT '02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–547, London, UK, 2002. Springer-Verlag.