Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Implementación de la criptografía basada en atributos en un dispositivo móvil

Tesis que presenta

Ana Helena Sánchez Ramírez

para obtener el Grado de

Maestra en Ciencias

en Computación

Asesor: Dr. Francisco Rodríguez Henríquez

México, D.F. **Octubre 2012**

Resumen

El modelo de *criptografía basada en atributos* (ABE), es un esquema basado en emparejamientos bilineales sobre curvas elípticas, el cual proporciona mecanismos de control de acceso a la vez que permite conservar la confidencialidad de la información. Los emparejamientos bilineales han sido estudiados por varios investigadores debido a su importancia en la construcción de diversos protocolos criptográficos. Por tal motivo, en los últimos años se han presentado diversas implementaciones enfocadas a computar emparejamientos bilineales de forma eficientemente.

Esta tesis describe el análisis, el diseño y la implementación de una biblioteca de software eficiente, que compute emparejamientos bilineales de una manera eficaz sobre un dispositivo móvil equipado con un procesador ARM Cortex A9. Así mismo, se describe el diseño e implementación del protocolo ABE utilizando la biblioteca desarrollada en este trabajo, y se presenta una aplicación demostrativa capaz de cifrar archivos a través del esquema ABE como mecanismo de control de acceso.

Abstract

The *Attributed-Based Encryption* (ABE) paradigm, is a bilinear pairing-based scheme over elliptic curves, which provides control access mechanisms and preserves data confidentiality. Bilinear pairings have been studied by several researchers due to its importance in the construction of many cryptographic protocols. For that reason, in the last few years several implementations have focused on how to implement bilinear pairings efficiently. Nevertheless, as of today the vast majority of these proposals have not addressed mobile computing environments, whose field of study is growing more and more.

This thesis describes the analysis, design and implementation of an efficient software library that computes bilinear pairings efficiently over a mobile device equipped with a ARM Cortex A9 processor. It also describes the design and implementation of the ABE protocol that utilizes the library developed in this work and presents a demonstrative application able to encrypt files using ABE as the control access mechanism.

Agradecimientos

Las palabras fluyen como el agua cuando nacen del corazón

Empezaré agradeciendo al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la oportunidad y la confianza de otorgarme su apoyo económico a través su programa de becas para la realización de mis estudios de maestría.

Agradezco al CINVESTAV, una institución de alto rendimiento, que ha sacado mi mejor potencial y me ha brindado los mejores conocimientos.

A Sofi, que se preocupa por cada uno de los alumnos, siempre nos tiene la paciencia necesaria y que de alguna manera, sin convivir demasiado, nos conoce lo suficiente.

A los doctores que me dieron clases y a mis revisores: el Dr. Morales y el Dr. de la Fraga, ejemplos de personas a seguir, que nos brindan los conocimientos más actualizados y que a su vez se preocupan por que aprendamos los suficiente. Sé que de repente les sacamos canas verdes, pero de verdad agradezco su paciencia, apoyo y motivación.

A mis amigos de UPIITA, a Luis cuyos consejos han sido muy valiosos, a mi buen amigo Víctor que siempre ha estado conmigo y entiende siempre mi estado de trabajo, y a mi súper amigo Sergio, cuyas ideas innovadoras, su amplio sentido del humor, su comprensión y opinión han sido muy importantes en mi vida.

Quiero agradecer a mis amigos y compañeros de generación, todos son grandes personas, me han enseñado mucho acerca de la convivencia y verdadera amistad, han estado en mis momentos más difíciles y me han apoyado cuando más lo he necesitado, en especial quiero agradecer a mi buen amigo Julio y a mi gran amiga Paulina, muchas gracias por los buenos momentos que hemos vivido juntos.

Agradezco a mi familia, que siempre ha apoyado y soportado, siempre han estado conmigo y me han brindado todo el amor que necesito, a mi mamá, una persona que se supera cada día, a la cual amo con todo mi corazón y que siempre está ahí para escucharme y levantarme. A mi hermano Christian, que siempre me escucha y está al pendiente, y a mi hermanito Ricardo, quien siempre me da un abrazo en los momentos que más lo necesito.

Por último, quiero dar las gracias a la persona que más me a apoyado en la realización de esta tesis, el Dr. Francisco, el cuál es una persona a la cual admiro y respeto mucho. El mejor asesor que he tenido y de los mejores profesores que conozco (de hecho uno de mis favoritos). Gracias por todo su apoyo, el voto de confianza que ha depositado en mí y sobre todo por la paciencia en aquellas ocasiones donde he fallado, de verdad, muchas gracias.

Índice general

Índice de Algoritmos

Índice de figuras

Capítulo 1

Introducción

"Si se piensa que la tecnología puede solucionar los problemas de seguridad, está claro que ni se entiende los problemas ni se entiende la tecnología"
Bruce Schneier

Con el incremento del uso de los teléfonos inteligentes, se ha dado un mayor auge en el cómputo móvil, el cual tiene como principal objetivo el procesamiento automático de información en cualquier tiempo y en cualquier lugar, mediante el acceso a las fuentes de información digital utilizando infraestructuras de comunicación inalámbrica [Zheng and Ni, 2005]. Esto implica una gran ventaja, ya que la información puede ser accedida en cualquier momento que sea requerida, mejorando el servicio en aplicaciones donde la velocidad de atención es crítica.

A pesar de las ventajas del cómputo móvil, debe tomarse en cuenta que los medios inalámbricos son altamente inseguros, es por ello que los aspectos de seguridad deben ser considerados, sobre todo en sistemas donde la información es crítica. En términos de seguridad, la criptografía es una herramienta importante contra posibles amenazas, por lo que intenta asegurar que se cumpla una serie de servicios descritos a continuación:

- *Autenticación*. Permite certificar que la identidad de los participantes y/o entidades es verdadera. La autenticación se logra verificando dichas entidades usando mecanismos como firmas digitales, certificados digitales o características biométricas. En el caso de los usuarios se pueden evaluar tres aspectos para autenticarlos; el primer aspecto es verificar algo que el usuario tiene, por ejemplo, una llave privada con la cual puede emitir firmas digitales; el segundo aspecto es poner a prueba al usuario sobre algo que sabe, esto es, pedirle una contraseña y, finalmente, el tercer aspecto es verificar algo que el usuario es, por ejemplo, analizar sus huellas dactilares o su retina.

- *Control de acceso*. Proporciona los privilegios de acceso a recursos restringidos a algunas entidades de acuerdo a ciertas reglas establecidas previamente.

- *Confidencialidad*. Asegura que la información sensible sólo puede ser consultada o manipulada por usuarios, entidades o procesos autorizados.
- *Integridad*. Da la certeza de que la información no ha sido modificada por entidades no autorizadas para hacerlo. Dentro de las posibles modificaciones están la escritura, modificación o borrado de segmentos de datos.
- *No repudio*. Ofrece protección a un usuario o entidad con respecto a que otro usuario niegue posteriormente que en realidad se realizara cierta transacción, es decir, impide la negación de una entidad sobre acciones realizadas. Esta protección se efectúa por medio de una colección de evidencias irrefutables que permitirán la resolución de cualquier disputa.

Esquemas de cifrado

La criptografía utiliza los esquemas de cifrado para garantizar los servicios de seguridad en las aplicaciones. De manera general, los esquemas de cifrado pueden ser clasificados en simétricos y asimétricos. En los esquemas simétricos se utiliza esencialmente la misma llave para cifrar y descifrar, mientras que en los asimétricos se usa un par de llaves: una pública y una privada. La principal ventaja del esquema simétrico es su bajo costo computacional, que conlleva a una alta velocidad para cifrar y/o descifrar, mientras que su principal desventaja radica en la generación, compartición y distribución de llaves. Entre los algoritmos de cifrado simétrico se encuentran DES (*Data Encryption Standard*) desarrollado por IBM en 1974 y AES (*Advanced Encryption Standard*) [Daemen and Rijmen, 2002].

El esquema asimétrico tiene como ventaja que al manejar el par de llaves pública y privada, no tiene la misma problemática del esquema simétrico en la administración de llaves; sin embargo, tiene como desventajas su alto costo computacional y el tamaño de sus llaves, las cuales requieren un incremento en el espacio de almacenamiento.

Los esquemas de cifrado asimétrico basan su seguridad en la complejidad computacional que supone resolver un problema matemático asociado a cada esquema. El representante más conocido es RSA, el cual fue desarrollado en 1977 por Rivest, Shamir y Adleman y cuya seguridad radica en el problema de la factorización de números enteros.

1.1. Antecedentes

En 1985 se propone el uso de curvas elípticas en sistemas criptográficos [Koblitz, 1987, Miller, 1985], observando que su seguridad está garantizada por la complejidad computacional del problema del logaritmo discreto en el grupo abeliano generado por los puntos de una curva elíptica, el cual, con una selección adecuada de parámetros,

es un problema matemático difícil de resolver.

A través de la introducción de las curvas elípticas en criptografía, ha sido posible obtener niveles de seguridad similares a los proporcionados por otros sistemas criptográficos, esto debido a que utilizan campos finitos de menor orden, lo que implica alcanzar niveles de seguridad con mayor velocidad, menor requerimiento de memoria y menor costo computacional.

En 1993 se describe un método para proyectar un punto de una curva elíptica hacia un campo finito utilizando el emparejamiento bilineal de Weil con propósitos destructivos [Menezes et al., 1993,Frey and Rück, 1994]. A partir del 2000 se descubren propiedades constructivas de los emparejamientos [Mitsunari et al., 2002,Sakai et al., 2000, Joux, 2004].

1.1.1. Criptografía basada en la identidad

En 1984, Shamir introdujo el concepto de *criptografía basada en la identidad* (IBE) [Shamir, 1984], para evitar los problemas de las claves públicas (certificados, autoridades de certificación) en la criptografía clásica de llave pública. En este esquema, Shamir propuso la idea de que una cadena arbitraria, tal como la dirección de correo electrónico o un número telefónico podría servir como llave pública en un esquema de criptografía asimétrica.

La idea general consiste en que si un usuario A desea enviar un mensaje cifrado a correo del usuario B, bastaría con utilizar la cadena de texto de la dirección de correo de B como llave pública para cifrar, por ejemplo: "bob@correo.com". De esta manera B recibe el mensaje cifrado y contacta a una tercera autoridad llamada "servidor de llaves" con quien se autentica y obtiene su llave privada para descifrar el mensaje (Ver Figura 1.1).

En el año 2001 Boneh y Franklin presentan una solución al problema de cómo desarrollar de forma práctica la criptografía basada en la identidad a través del uso de emparejamientos bilineales [Boneh and Franklin, 2003].

1.1.2. Criptografía basada en atributos

En 2005, se propone el *cifrado difuso* como un nuevo tipo de esquema basado en el esquema IBE [Sahai and Waters, 2005]. En este esquema se ve una identidad con un conjunto de atributos descriptivos. Este tipo de esquema permite descifrar un texto cifrado si y sólo si las identidades son lo suficientemente cercanas con respecto a una métrica fijada. Por lo tanto, un usuario con una llave secreta para una identidad ω es capaz de descifrar un texto cifrado con una llave pública ω', si y sólo si ω y ω' están dentro de cierta distancia el uno del otro de acuerdo a alguna métrica establecida. Este esquema permite la tolerancia a errores en las identidades, por lo que una de las

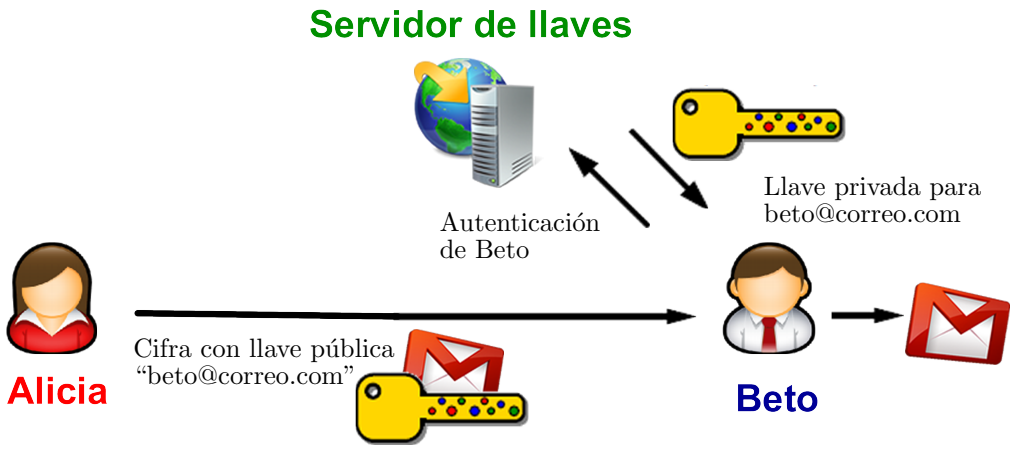


Figura 1.1: Criptografía basada en la identidad

aplicaciones propuestas es el cifrado por medio de entradas biométricas, ya que éstas contienen de manera inherente cierto nivel de ruido cada vez que son muestreadas.

En el año 2005, se introduce el término de *criptografía basada en atributos* (ABE) [Sahai and Waters, 2005], el cual permite cifrar un documento y descifrarlo únicamente por aquellos usuarios que tengan cierto conjunto de atributos. Por ejemplo, considérese que el jefe del Departamento de Computación desea cifrar un documento de manera que pueda ser visto por aquellos profesores que tienen un cargo administrativo. En este caso cifraría el documento con los atributos {“profesor”, “cargo administrativo”}. Cualquier usuario cuya identidad contenga todos los atributos descritos puede descifrar el documento.

La criptografía basada en atributos se vuelve muy útil como medio de control de acceso, ya que los atributos pueden ser vistos como privilegios; así mismo, tiene la ventaja de que al cifrar la información, ésta puede ser almacenada en cualquier repositorio. A partir de ello se han realizado varios estudios, tales como sus variantes en la política de acceso, [Bethencourt et al., 2007, Waters, 2008, Goyal et al., 2006, Attrapadung et al., 2011], una mayor eficiencia y mejoras en los algoritmos [Lewko and Waters, 2011, Pirreti et al., 2006], su uso en firmas digitales [Maji et al., 2011, Shahandashti and Safavi-Naini, 2009], etc.

Finalmente, cabe destacar que una de las aplicaciones en las que se describe la conveniencia del esquema ABE es en los expedientes clínicos electrónicos (ECE) [Akinyele et al., 2010]. El sistema de los ECE está formado por un repositorio digital que contiene los datos derivados de la atención médica de los pacientes, el cual es almacenado e intercambiado de manera segura y puede ser accedido por múltiples usuarios autorizados. La criptografía basada en atributos se propone como una solución en este sistema debido a su ventaja de disponibilidad y como medio de control de acceso a los

actores del sector salud (doctores, enfermeros, técnicos, grupo administrativo, etc), los cuales cuentan con un acceso total o parcial a los expedientes clínicos electrónicos.

1.2. Planteamiento del problema

La tendencia de las aplicaciones es proporcionar toda la información necesaria a todos los usuarios que lo soliciten, en el lugar y en el momento requerido. Sin embargo, llevar a cabo este objetivo, trae consigo ciertos problemas de seguridad, los cuales deben ser tomados en cuenta, sobre todo en sistemas que manejan información crítica.

Uno de los problemas principales en cuestiones de seguridad es el control de acceso. Bajo un enfoque tradicional, se tiene un servidor de confianza que almacena la información, la cual es adquirida por el usuario a través de un canal seguro, de acuerdo a una serie de permisos relacionados a algo que él conoce o es, por ejemplo: su cuenta de usuario y contraseña, o bien su huella dactilar. Bajo este enfoque, el funcionamiento del sistema y seguridad recae directamente en el servidor. La criptografía basada en atributos proporciona un mecanismo de control de acceso y confidencialidad, eliminando la dependencia de un servidor de confianza; esto es posible ya que el medio de control acceso se encuentra directamente asociado a la información cifrada, lo que permite que ésta pueda ser almacenada en ambientes poco seguros como sistemas en la *nube*, dispositivos móviles, etc.

En esta tesis se propone una solución a los problemas de seguridad en aplicaciones móviles a través del uso eficiente de la criptografía basada en atributos. La implementación de los servicios de seguridad, en general no es sencilla, ya que muchas técnicas son computacionalmente costosas, por lo que se debe tomar en cuenta tanto la arquitectura del dispositivo como la eficiencia en los algoritmos y técnicas para lograr un menor costo computacional.

1.3. Objetivos

Esta tesis tiene como objetivo general *desarrollar una aplicación segura para un dispositivo móvil a través de la implementación eficiente del esquema de criptografía basada en atributos*. Los objetivos particulares de este trabajo son:

1. Desarrollar una biblioteca orientada a dispositivos móviles con algoritmos eficientes para aritmética de campos finitos, multiplicación de puntos en curvas elípticas y computación de emparejamientos.

2. Implementar una aplicación prototipo en un dispositivo móvil que utilice a la criptografía basada en atributos como mecanismo de control de acceso.

1.4. Contexto de investigación

Históricamente, el problema del control de acceso ha sido formulado con base en sujetos y objetos [Zhang et al., 2002]. Los sujetos pueden ser usuarios o procesos actuando en nombre de usuarios. Los objetos son los datos o recursos en el sistema. Los permisos son un conjunto de operaciones que un sujeto puede hacer con uno o más objetos. El control de acceso involucra no sólo la protección de objetos individuales y sujetos, sino también la administración de las decisiones de control de acceso en una forma dinámica y en sistemas altamente distribuidos. Uno de los enfoques no tradicionales para resolver dicho problema es el que plantea el uso de la criptografía basada en atributos. En el esquema ABE [Sahai and Waters, 2005] los autores proponen el uso de una criptografía *difusa* basada en la identidad para resolver problemas de control de acceso a través de un conjunto de atributos. En este esquema, un usuario puede acceder a un determinado recurso únicamente si cuenta con una serie de privilegios (denominados atributos) que cumplan una política de control de acceso definida previamente.

El esquema ABE se define típicamente bajo dos formulaciones: política de acceso por texto cifrado [Bethencourt et al., 2007, Waters, 2008] y política de acceso por llave [Goyal et al., 2006, Attrapadung et al., 2011]. En la política de acceso por texto cifrado (figura 1.2), cada registro se encuentra asociado con una política de acceso que permite descifrarlo. Dichas políticas están normalmente expresadas como una fórmula lógica referenciada a una lista de atributos contenidos en una llave secreta.

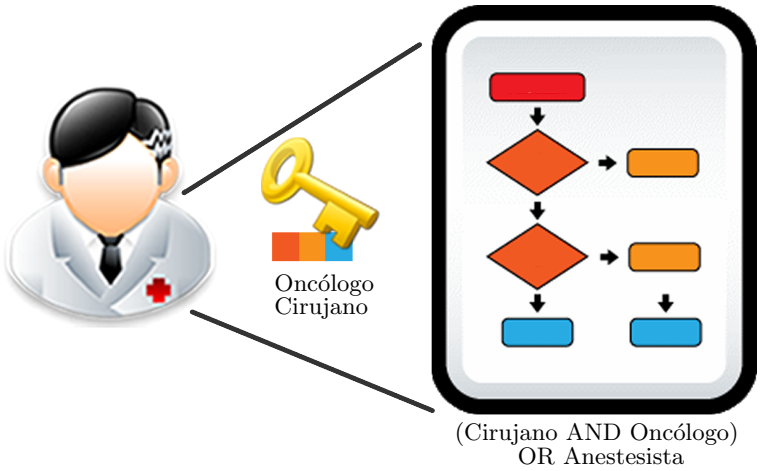


Figura 1.2: Política de acceso por texto cifrado

Dicha fórmula a su vez, puede ser expresada por un árbol de acceso, donde los nodos internos se encuentran compuestos por relaciones lógicas AND y OR, las cuales denotan conjunción y disyunción respectivamente, mientras que las hojas representan los atributos definidos en la política. Por ejemplo, suponga que un hospital define los siguientes atributos: médico general, oncólogo, enfermero, cirujano, anestesistas y paciente; de acuerdo a la siguiente política el registro únicamente podrá ser visto por un oncólogo cirujano o bien el anestesista:

(cirujano AND oncólogo) OR anestesista

En el esquema ABE con política basada en llave (figura 1.3), se invierte la relación entre el texto cifrado y la llave; es decir, los registros son etiquetados con sus atributos más relevantes, por ejemplo: resultados de laboratorio, radiografía, etc. Para permitir el acceso a la porción del registro, el propietario crea llaves específicas que contienen la fórmula de la política, determinando cuál sección del registro puede ser accedida.

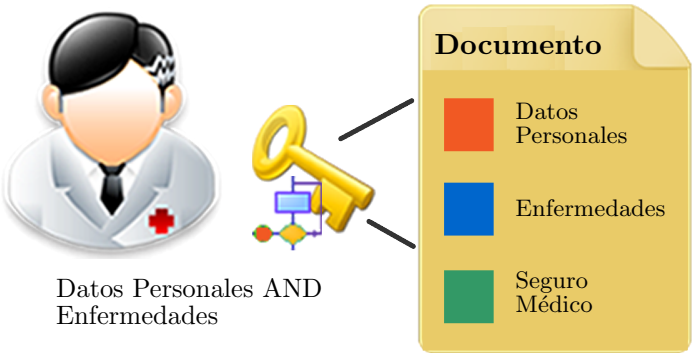


Figura 1.3: Política de acceso por llave

Estos enfoques no necesariamente pueden utilizarse de forma separada, sino que pueden combinarse en una sola primitiva denominada *política dual* [Attrapadung and Imai, 2009]. Sin embargo, no hay un estudio que muestre cuál de los dos enfoques es más eficiente de forma algorítmica para establecer una política de control de acceso.

Una propiedad fundamental de los sistemas ABE es la resistencia a colisiones, lo cual significa que un grupo de individuos no pueden combinar atributos en sus llaves secretas para así satisfacer una política de acceso. Esta propiedad es importante pero a la vez contradictoria, ya que al mismo tiempo se busca la flexibilidad en la modificación y/o actualización de atributos en las llaves en caso de ser requerido.

La *delegación*[1] es un factor importante para ambientes de cómputo distribuido seguro. Los ambientes de trabajo colaborativo y dinámico requieren que los usuarios

[1]En este contexto la delegación consiste en proporcionar atributos de un usuario a otro

asuman roles temporales. Dado que la descentralización es crítica para el control de acceso distribuido, es natural descentralizar a través de la delegación. En este aspecto, se han realizado varios estudios para los algoritmos de manejo de llaves y delegación en los esquemas ABE [Gorantla et al., 2010, Goyal et al., 2006] y en la forma de emitir o intercambiar las llaves de forma distribuida sin necesidad de una autoridad centralizada [Lewko and Waters, 2011]. Sin embargo, estas propuestas no han sido implementadas.

En cuanto a los algoritmos para la implementación de la criptografía basada en atributos, se tienen algunas mejoras en el manejo de las políticas de acceso [Zhou and Huang, 2010], ya sea por medio de un árbol o bien una matriz de acceso. A pesar de las mejoras propuestas, no existe un comparativo en cuanto a implementación para localizar el algoritmo más eficiente.

Algunos trabajos han sido publicados para los requerimientos en ambientes de salud [Zhang et al., 2002, Narayan et al., 2010, Mohan and Blough, 2010]. En éstos se plantea el uso de atributos para el control de acceso a los registros médicos, en especial se destaca el trabajo de Narayan, ya que plantea el uso de la criptografía basada en atributos como medio de control de acceso [Narayan et al., 2010].

Recientemente, investigadores del *Johns Hopkins Medical Institution* propusieron un esquema de control de expedientes clínicos utilizando emparejamientos bilineales y criptografía basada en atributos. Sin embargo, su esquema se diseñó utilizando una biblioteca de cómputo poco eficiente [Akinyele et al., 2010]. Un rediseño del protocolo, utilizando emparejamientos bilineales definidos sobre curvas no tradicionales, permitiría una mejor eficiencia y una reducción en el costo computacional.

En cuestión de ambientes móviles, Srirama y Naumenko proponen un modelo de autenticación en dispositivos móviles basado en atributos [Srirama and Naumenko, 2010]. Sin embargo, el modelo no contiene un transfondo criptográfico y por ende ninguna prueba de seguridad. La implementación de un esquema criptográfico en el ámbito de la criptografía basada en atributos y el uso de algoritmos basados en emparejamientos proporcionaría otra solución al problema de control de acceso en estos dispositivos.

Por último, cabe destacar que en el estado del arte actual, la criptografía basada en atributos presenta varios problemas de investigación a resolver, los cuales no están planteados en ninguno de los trabajos mencionados, entre ellos están la mejora de los algoritmos para modificar o actualizar las políticas de control de acceso, los mecanismos para asociar registros y usuarios, la auditabilidad en línea, y la implementación eficiente de los algoritmos ABE en dispositivos restringidos como dispositivos móviles y ubicuos, entre otros.

Implementación criptográfica

Uno de los objetivos principales de esta tesis es la implementación del esquema de criptografía basada en atributos. Este esquema se basa en emparejamientos bilineales, por lo que se busca la mayor eficiencia en su implementación.

En los últimos años, diversos investigadores han presentado implementaciones en software de emparejamientos bilineales, los cuales se destacan por las técnicas usadas en la aceleración del cálculo del mismo. En Hankerson et al. se describe las implementaciones de emparejamientos tanto en curvas elípticas ordinarias como supersingulares, así como las técnicas utilizadas para lograr la eficiencia en las implementaciones [Hankerson et al., 2009]. Asimismo, se muestra la repercusión directa de la aritmética de torres en el costo final del emparejamiento.

En Naehrig et al. se presenta los detalles de una implementación que computa el emparejamiento *óptimo ate* sobre una curva BN de 257 bits [Naehrig et al., 2010], que hasta esa fecha era la implementación más rápida. Posteriormente los trabajos propuestos por Beuchat et al. y Aranha et al. mejoran la eficiencia del emparejamiento utilizando mejores técnicas de programación, un aprovechamiento de la arquitectura y el uso de paralelización [Beuchat et al., 2009, Aranha et al., 2010, Aranha et al., 2011b]. Estos trabajos, sin embargo, dejan de lado la implementación en dispositivos móviles.

Actualmente se tiene la implementación de la función SHA-3 sobre la arquitectura ARM11 [Schwabe et al., 2012]. En cuanto a la implementación de emparejamientos sobre dispositivos móviles, únicamente se destaca el trabajo de Acar et al., el cual fue desarrollado sobre un procesador Cortex-A9 [2] [Acar et al., 2011]. A pesar de ello, este último no muestra ninguna técnica de desarrollo, sólo un reporte de tiempos a través del uso de coordenadas afines.

En cuanto a implementaciones del esquema ABE sobre dispositivos móviles se observa se destaca el trabajo de Akinyele et al., pero como se mencionó anteriormente no utiliza una biblioteca criptográfica eficiente. Por otro lado, en el trabajo desarrollado por Scott se realiza una implementación en un procesador Intel Core i5 a 2.4 GHz con los algoritmos que mejor encajan para esta implementación [Scott, 2011].

1.5. Organización de la tesis

La tesis se encuentra organizada en ocho capítulos: en el capítulo 2 se definen los conceptos generales para introducir al lector en el tema abordado en esta tesis. En el capítulo 3 se describe la artimética utilizada para el desarrollo de la biblioteca

[2] Procesador de arquitectura ARMv7

propuesta.

En el capítulo 4 y 5 se muestra a detalle el tema de curvas elípticas y emparejamientos así como la descripción de los algoritmos utilizados durante el desarrollo de la tesis.

Posteriormente, se describe en el capítulo 5 el tema de la criptografía basada en atributos, se describen las fases para su implementación, los algoritmos utilizados y su funcionamiento.

El capítulo 6 muestra detalles de la implementación en el procesador Cortex-A9 y las comparaciones con el estado del arte actual. Finalmente el capítulo 7 describe las conclusiones y trabajo futuro.

Capítulo 2

Conceptos básicos

"La imaginación es más importante que el conocimiento.
El conocimiento es limitado, mientras que la imaginación no"
Albert Einstein

El objetivo principal de este capítulo es introducir al lector en los conceptos que serán utilizados a lo largo de esta tesis. Se describen algunas definiciones de teoría de números y algebra de acuerdo con [Shoup, 2005, Fuentes-Castañeda, 2011, Lidl and Niederreiter, 1986, Menezes et al., 1996, Hankerson et al., 2004].

2.1. Grupo

Un **grupo** $(\mathbb{G}, \star, e)$ es un objeto matemático abstracto, conformado por un conjunto $\mathbb{G}$, un elemento identidad $e \in \mathbb{G}$ y una operación binaria $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$, con las siguientes propiedades:

(I) **Cerradura**: $\forall a, b \in \mathbb{G},\ a \star b \in \mathbb{G}$.

(II) **Existencia del elemento identidad**: Existe un elemento e tal que $\forall a \in \mathbb{G}$, se cumple que $a \star e = e \star a = a$.

(III) **Asociatividad**: $\forall a,\ b,\ c \in \mathbb{G}$, $a \star (b \star c) = (a \star b) \star c$.

(IV) **Elemento inverso**: $\forall a \in \mathbb{G}$ existe un elemento $\bar{a} \in \mathbb{G}$, llamado el inverso de a, tal que $a \star \bar{a} = \bar{a} \star a = e$.

El grupo es llamado **abeliano** si la operación $\star$ es conmutativa, es decir, si $\forall\ a, b \in \mathbb{G}$, se satisface la igualdad $a \star b = b \star a$.

Dado el grupo $\mathbb{G} = (\mathbb{G}, \star, e)$. Se denota la operación $\star^m(g)$ a la aplicación de m veces consecutivas el operador $\star$ sobre el elemento g consigo mismo, donde $g \in \mathbb{G}$ y $m \in \mathbb{Z}^+$. Algunas definiciones asociadas a los grupos son:

Definición 2.1. Orden del Grupo. *El orden de un grupo* $(\mathbb{G}, \star, e)$ *está definido como el número de elementos en el conjunto* $\mathbb{G}$*. Los grupos pueden tener orden finito o infinito.*

Definición 2.2. Orden de un elemento del Grupo. *Sea* $(\mathbb{G}, \star, e)$ *un grupo, el orden de* $g \in \mathbb{G}$ *es el menor entero positivo* s*, tal que* $\star^s(g) = e$*.*

Así mismo, para cualquier grupo $\mathbb{G}$ de orden finito n, el orden de cualquier elemento $g \in \mathbb{G}$ divide a n. Lo que implica que $\star^n(g) = e$.

Definición 2.3. Generador del grupo. *Dado el grupo* $\mathbb{G} = (\mathbb{G}, \star, e)$*, se dice que* $g \in \mathbb{G}$ *es un generador del grupo, si para cualquier elemento* $h \in \mathbb{G}$ *existe* $i \in \mathbb{Z}^+$*, tal que* $h = \star^i(g)$*. Al grupo generado por* g *se le denota como* $\langle g \rangle$*.*

Definición 2.4. Grupo Cíclico. *Un grupo* $\mathbb{G} = (\mathbb{G}, \star, e)$ *es cíclico si* $\mathbb{G} = \langle g \rangle$ *para algún generador* $g \in \mathbb{G}$*.*

El número de elementos generadores en un grupo cíclico finito $\mathbb{G}$ de orden n, está dado por $\varphi(n)$, donde $\varphi()$ denota la función indicatriz de Euler. Para el caso donde el grupo cíclico $\mathbb{G}$ es de orden primo p, se dice que $\mathbb{G}$ tiene $\varphi(p) = p - 1$ generadores.

Las notaciones más utilizadas en grupos son la aditiva y la multiplicativa, las cuales se describen a continuación.

Notación aditiva

Un grupo $\mathbb{G}$ es escrito de manera aditiva utilizando el operador "+" para representar la operación de grupo y el valor "0" como elemento identidad, de manera que $\mathbb{G} = (\mathbb{G}, +, 0)$. En esta notación el elemento inverso de cualquier elemento $a \in \mathbb{G}$ se representa como $-a$. Para $a, b \in \mathbb{G}$, la operación $a + (-b)$ puede representarse como $a - b$. Así mismo, la aplicación de m veces el operador $+$ sobre el elemento $b \in \mathbb{G}$ se denota como mb, donde $m \in \mathbb{Z}^+$.

Notación multiplicativa

Un grupo $\mathbb{G}$ es escrito de manera *multiplicativa* cuando la operación de grupo se denota como " $\cdot$ " y el "1" representa al elemento identidad, es decir $\mathbb{G} = (\mathbb{G}, \cdot, 1)$.

En este grupo a^{-1} representa el inverso multiplicativo de $a \in \mathbb{G}$. En general dado $a, b \in \mathbb{G}$ la operación $a \cdot b$ suele escribirse como ab, de la misma manera, la operación ab^{-1} puede representarse como a/b. La aplicación de $m \in \mathbb{Z}^+$ veces del operador $\cdot$ sobre $b \in \mathbb{G}$, se denota como b^m.

2.1.1. Subgrupo

Sea $\mathbb{G} = (\mathbb{G}, \star, e)$ un grupo y sea $\mathbb{H} \subset \mathbb{G}$. $\mathbb{H}$ se dice ser un subgrupo de $\mathbb{G}$ si:

- Para todo $a, b \in \mathbb{H}$, se tiene que $a \star b \in \mathbb{H}$.
- Dado $a \in \mathbb{H}$, entonces existe un inverso $a^{-1} \in \mathbb{H}$.
- Se tiene que $e \in \mathbb{H}$.

En otras palabras $\mathbb{H}$ forma un grupo bajo la operación $\star$ con e como elemento identidad, por lo que $(\mathbb{H}, \star, e)$ es un subgrupo de $(\mathbb{G}, \star, e)$.

2.1.2. Clase lateral

Sea $\mathbb{G} = (\mathbb{G}, \star, e)$ un grupo abeliano y $\mathbb{H} = (\mathbb{H}, \star, e)$ un subgrupo de $\mathbb{G}$. Para todo $a, b \in \mathbb{G}$ se escribe $a \equiv b \pmod{\mathbb{H}}$, si $a \star \bar{b} \in \mathbb{H}$, donde $\bar{b}$ es el inverso de b. Donde $\equiv \pmod{\mathbb{H}}$ es una *relación de equivalencia* y divide al grupo $\mathbb{G}$ en *clases de equivalencia*.

Dado $a \in \mathbb{G}$, $[a]_{\mathbb{H}}$ denota la clase de equivalencia que contiene al elemento a, la cual está definida como $[a]_{\mathbb{H}} = a \star \mathbb{H} = \{a \star h \mid h \in \mathbb{H}\}$, es decir:

$$x \in [a]_{\mathbb{H}} \iff x \equiv a \pmod{\mathbb{H}}$$

.

Las clases de equivalencia son llamadas *las clases laterales de* $\mathbb{H}$ *en* $\mathbb{G}$. El conjunto de todas las clases laterales está denotado como $\mathbb{G}/\mathbb{H}$ y forma un grupo $(\mathbb{G}/\mathbb{H}, \star, [e]_{\mathbb{H}})$, donde $[a]_{\mathbb{H}} \star [b]_{\mathbb{H}} = [a \star b]_{\mathbb{H}}$, el cual es llamado *el grupo cociente de* $\mathbb{G}$ *módulo* $\mathbb{H}$.

2.2. Anillo

Un anillo $\langle R, +, * \rangle$ está conformado por un grupo R con dos operaciones binarias definidas para sus elementos. Típicamente dichas operaciones se denotan como "+" y "$\cdot$". Las propiedades que deben cumplir los anillos son:

1. La estructura $(R, +, 0)$ es un grupo abeliano.
2. La operación "$\cdot$" es cerrada y asociativa sobre R.
3. Existe una identidad multiplicativa en R.
4. Las operaciones "+" y "$\cdot$" están relacionadas por la ley distributiva, es decir, $\forall a, b, c \in R$ se cumple que:

$$(a + b) \cdot c = a \cdot c + b \cdot c.$$

Se dice que un anillo $\langle R, +, \cdot \rangle$ es conmutativo si la operación "$\cdot$" es conmutativa, es decir, $\forall a, b \in R$, $a \cdot b = b \cdot a$.

Se dice que un elemento x del anillo es invertible si x tiene un inverso multiplicativo en $\mathbb{R}$, esto es, si existe un elemento $u \in R$ tal que:

$$x \cdot u = u \cdot x = 1$$

donde "1" es la identidad multiplicativa de R.

Ejemplos de anillos son los números enteros, los racionales y los reales bajo las operaciones de suma y multiplicación.

2.3. Campo

Un campo es una estructura algebraica $(\mathbb{F}, +, \cdot, 0, 1)$ conformada por un conjunto $\mathbb{F}$ y dos operaciones binarias: adición y producto, que satisfacen las siguientes propiedades:

(I) $\mathbb{F}^+ = (\mathbb{F}, +, 0)$ es un grupo abeliano con el "0" como unidad aditiva.

(II) $\mathbb{F}^* = (\mathbb{F} - \{0\}, \cdot, 1)$ es un grupo abeliano con el "1" como unidad multiplicativa.

(III) El producto se distribuye a ambos lados de la suma, es decir,

$$(a + b) \cdot c = a \cdot c + b \cdot c,$$

para todo $a, b, c \in \mathbb{F}$.

Por tanto, un campo es un anillo conmutativo donde todo elemento diferente de cero posee un inverso multiplicativo.

Definición 2.5. Característica de un campo. *Dado el campo $\mathbb{F}$, sea $n \in \mathbb{Z}^+$. Se dice que n es la característica de $\mathbb{F}$, si n es el menor entero positivo tal que $n \cdot 1 = \sum_{i=0}^{n-1} 1 = 0$. En caso de que no exista tal entero n, se dice que $\mathbb{F}$ es de característica 0.*

2.3.1. Campo finito

Un campo finito es un campo $\mathbb{F}$, cuyo número de elementos es finito. El orden de $\mathbb{F}$ es el número de elementos que lo conforman. $\mathbb{F}_q$ denota un campo finito con q elementos. Un campo finito $\mathbb{F}_q$ existe si y sólo si $q = p^m$, donde p es un primo y $m \geq 1$.

Se dice que $\mathbb{F}$ forma un campo finito si su característica es diferente de 0, en caso contrario es un campo infinito. Para todo campo finito de la forma $\mathbb{F}_q$, se dice que su

característica es p siempre que $q = p^m$.

Para el caso donde $q = p$, se dice que el conjunto $\mathbb{F}_p = \{0, 1, 2, ..., p-2, p-1\}$ define un campo finito $(\mathbb{F}_p, +, \cdot, 0, 1)$ bajo las operaciones suma y multiplicación módulo p.

2.3.2. Extensión de un campo finito

Definición 2.6. Polinomio irreducible. *Sea $f(z)$ un polinomio de orden $m \geq 2$, se dice que $f(z)$ es irreducible si no puede factorizarse como el producto de polinomios de grado menor a m.*

Sea p un número primo y $m \geq 2$. Se denota como $\mathbb{F}_p[z]$ al conjunto de todos los polinomios de la variable z con coeficientes en $\mathbb{F}_p$. Así mismo, sea $f(z)$ un polinomio irreducible de grado m en $\mathbb{F}_p[z]$. Los elementos de $\mathbb{F}_{p^m}$ son los polinomios en $\mathbb{F}_p[z]$ de grado a lo más $m-1$ [Hankerson et al., 2004]:

$$\mathbb{F}_{p^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 \mid a_i \in \mathbb{F}_p\}$$

donde $\mathbb{F}_{p^m}$ es un campo finito de característica p y orden p^m bajo las operaciones suma $+$ y multiplicación $\cdot$ módulo $f(z)$. De manera general $\mathbb{F}_{p^m}$ puede representarse como:

$$\mathbb{F}_{p^m} = \mathbb{F}_p[z]/f(z) \cong \text{ los polinomios en } \mathbb{F}_p[z] \text{ mod } f(z)$$

Cerradura algebráica

Sea K y L dos campos tales que $K \subseteq L$. Si $\alpha \in L$, se dice que α es *algebráica sobre* K si existe un polinomio no constante

$$f(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_0$$

con $a_0, \ldots, a_{n-1} \in K$ tal que $f(\alpha) = 0$. Se dice que L es *algebráico sobre* K, o que L es una *extensión algebráica* de K, si cualquier elemento de L es algebráico sobre K. Una cerradura algebráica de un campo K es un campo $\bar{K}$ que contiene a K tal que:

1. $\bar{K}$ es algebráica sobre K
2. Cualquier polinomio no constante $g(x)$ con coeficientes en $\bar{K}$ tiene raíces en $\bar{K}$.

Para un campo finito $\mathbb{F}_p$, con p primo, la cerradura algebraica de $\mathbb{F}_p$, denotada por $\bar{\mathbb{F}}_p$ es el conjunto infinito de todas sus extensiones, es decir:

$$\bar{\mathbb{F}}_p = \bigcup_{m \geq 1} \mathbb{F}_{p^m}$$

2.4. Torres de campos

Se le conoce como torres de campos a las construcciones formadas por campos finitos, donde cada campo finito representa un nivel de la torre el cual corresponde a una extensión de los campos de los niveles inferiores. Por ejemplo, sea $\mathbb{F}_{p^m} = \mathbb{F}_p[z]/f(z)$ una extensión de $\mathbb{F}_p$, ésta puede expresarse como $\mathbb{F}_{p^m} = \mathbb{F}_q[v]/g(v)$ tal que $\mathbb{F}_q$ es una extensión de $\mathbb{F}_p$ donde $q = p^k$ y $k|m$.

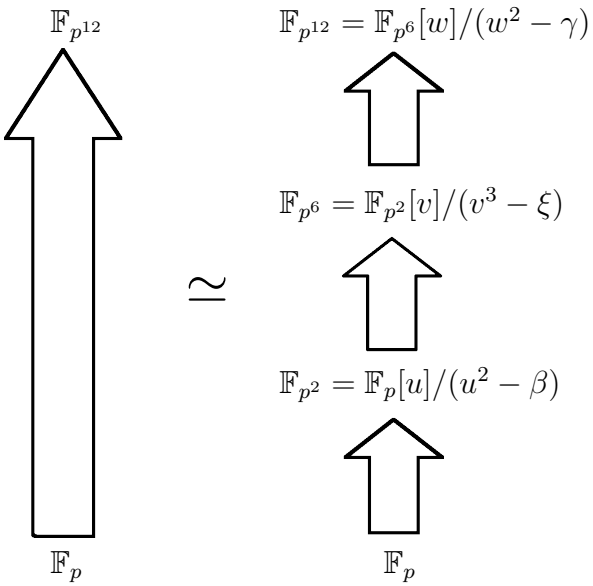


Figura 2.1: Torres de campos

El número de niveles que conforman las estructuras de las torres de campos depende directamente de la extensión máxima que se quiera construir. En general, existe más de una forma de construir una torre, sin embargo, algunas construcciones proporcionan la ventaja de tener una mayor eficiencia en las operaciones. Particularmente se dice que un campo finito $\mathbb{F}_{p^m}$ es *amable* con los emparejamientos si puede ser representado por extensiones cuadráticas y cúbicas.

Por motivos de implementación descritos en el capítulo 5, se realizó la construcción de la torre de campo para $\mathbb{F}_{p^{12}}$ de acuerdo a la Figura 2.1. Se eligió esta construcción ya que al representar cada nivel en extensiones cuadráticas y cúbicas se logra un alto nivel de eficiencia en cada extensión.

2.5. Grupo ciclotómico

En el capítulo 3 se muestra la utilidad del grupo ciclotómico, en esta sección se definen los conceptos básicos relacionados con el mismo.

Definición 2.7. Raíces de la unidad. *Dado $n \in \mathbb{N}$, las raíces n-ésimas de la unidad son las n soluciones del polinomio $z^n - 1 = 0$, las cuales se denotan como z_j.*

$$z^n - 1 = \prod_{j=0}^{n-1}(z - z_j)$$

donde el conjunto de las raíces n-ésimas de la unidad denotado como μ_n, forma un grupo cíclico $\mu_n = (\mu, \cdot, 1)$

Definición 2.8. Raíces primitivas de la unidad. *Sea $z^* \in \mu_n$, se dice que z^* es una raíz "primitiva" de la unidad, si y sólo si $\mu_n = \langle z^* \rangle$. El número de raíces primitivas de μ_n está dado por $\varphi(n)$, donde $\varphi(\cdot)$ corresponde a la función indicatriz de Euler.*

Definición 2.9. Polinomio ciclotómico. *Se define el n-ésimo polinomio ciclotómico $\Phi_n(z)$ de grado $\varphi(n)$ como:*

$$\Phi_n(z) = \prod_{k=0}^{\varphi(n)}(z - z_k^*)$$

donde z_k^* son las raíces n-ésimas primitivas de la unidad.

Dado que las raíces de $\Phi_n(z)$ forman un subconjunto μ_n, el cual es el conjunto de raíces del polinomio $z^n - 1$, entonces $\Phi_n(z) \mid z^n - 1$.

Definición 2.10. Grupo ciclotómico. *Sea p un número primo y sea $\mathbb{F}_{p^n}^*$ el grupo multiplicativo de un campo finito de característica p, el n-ésimo grupo ciclotómico $\mathbb{G}_{\Phi_n(p)}$ es el subgrupo de $\mathbb{F}_{p^n}^*$ definido por:*

$$\mathbb{G}_{\Phi_n(p)} = (\{\alpha \in \mathbb{F}_{p^n}^* \mid \alpha^{\Phi_n(p)} = 1\}, \cdot, 1)$$

2.6. Morfismos

Un *morfismo* es una proyección entre dos estructuras matemáticas. Los morfismos son frecuentemente representados como flechas entre dichas estructuras. Existen diferentes tipos de morfismos:

- Monomorfismo. Es un morfismo $f : X \to Y$ tal que para todos los morfismos $g_1, g_2 : Z \to X$, se cumple que $f \circ g_1 = f \circ g_2$.
- Isomorfismo. Sea $f : X \to Y$, f es un isomorfismo si existe un monomorfismo $g : Y \to X$.
- Endomorfismo. Un morfismo $f : X \to X$ es un endomorfismo de X.

- Automorfismo. Un automorfismo es un endomorfismo que también es un isomorfismo.

En el ámbito del álgebra (grupos, anillos, etc.) los morfismos se conocen como homomorfismos.

Capítulo 3

Aritmética de campos finitos

"La simplicidad llevada al extremo se convierte en elegancia"
Jon Franklin

La implementación eficiente de la aritmética de campos finitos es un prerrequisito para el desarrollo de esquemas basados en emparejamientos, debido a que cada una de las operaciones involucradas en la implementación del emparejamiento está conformada por operaciones aritméticas definidas sobre un campo finito.

En este capítulo se detalla la aritmética utilizada en la construcción de las torres de campos así como el costo en cada operación. La aritmética utilizada en los campos $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}$ y $\mathbb{F}_{p^{12}}$ está basada en [Beuchat et al., 2010].

3.1. Aritmética en $\mathbb{F}_p$

La aritmética en $\mathbb{F}_p$ o campo base corresponde a la capa de mayor importancia en cualquier esquema basado en campos finitos. Esta capa se encarga de alimentar a cada una de las capas superiores, por lo que una implementación eficiente de ésta se ve reflejada en las operaciones que se realicen sobre las capas superiores de las extensiones de campo.

La aritmética en el campo base está directamente relacionada con la arquitectura del dispositivo de implementación (en especial con el tamaño de palabra del procesador) y con el tamaño en bits del primo p, el cual se denota como $|p|$. En este trabajo se seleccionó un primo con un tamaño de aproximadamente 256 bits y un tamaño de palabra de 32 bits, el cual corresponde al de los dispositivos móviles actuales.

En general, un número entero de 256 bits no puede ser definido a través de los tipos de datos primitivos[1], es por ello que se utilizan otras técnicas para su uso y

[1]Tipos de datos estándares en C, tales como *short*, *long*, *int*, etc.

representación. En esta tesis, se optó por la representación a través de arreglos de elementos enteros sin signo (palabras) de 32 bits, debido a que, como se describe en el capítulo 7, corresponde a la arquitectura de los procesadores en los dispositivos móviles.

Sea $a \in \mathbb{F}_p$ con $|a| \simeq |p|$, donde $|a|$ el tamaño en bits de a. Entonces, a puede ser representado como un arreglo de n elementos:

$$a = (a_0, a_1, \ \ldots \ , a_{n-1})$$

donde $a = a_0 + a_1 2^{32} + a_2 2^{32 \cdot 2} + ... + a_{n-1} 2^{32 \cdot n-1}$, $n = [(\lfloor log_2 \ p \rfloor + 1)/32]$ y $|a_i| = 32$. En general se dice que a_0 es el elemento menos significativo de a, mientras que a_{n-1} corresponde al elemento más significativo de a. Para $|a| = 256$ y una arquitectura de 32 bits, a puede ser representado por medio de un arreglo de 8 palabras.

$$a = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$$

3.1.1. Adición Modular

Sea $a, b \in \mathbb{F}_p$, la operación adición en el campo finito $\mathbb{F}_p$ es $(a + b)$ mod p. Para realizar esta operación en números grandes (mayores de 64 bits), se realiza la suma elemento por elemento considerando el acarreo a partir del bit menos significativo, como se muestra en el Algoritmo 3.1.

Algoritmo 3.1 Adición en $\mathbb{F}_p$

Entrada: Primo p, y dos enteros a, $b \in [0, p-1]$ donde $a = (a_0, a_1, \ldots, a_{n-1})$ y $b = (b_0, b_1, \ldots, b_{n-1})$
Salida: $c = (a + b)$ mod p.

```
1: (c_0, acarreo) ← Suma(a_0, b_0)
2: for i = 1 → n − 1 do
3:     (c_i, acarreo) ← Suma_con_acarreo(a_i, b_i, acarreo)
4: end for
5: c ← (c_{n−1}, ..., c_2, c_1, c_0)
6: if existe acarreo o c > p then
7:     c ← c − p                                   (Algoritmo 3.2)
8: end if
9: return c
```

En el caso de la operación $a - b$ o resta, el procedimiento es básicamente el mismo con la diferencia de que en vez de sumas se realizan restas con préstamos, como se muestra en el Algoritmo 3.2.

3.1.2. Multiplicación

La multiplicación es la operación más relevante en la implementación de los emparejamientos bilineales y sirve de base para comparar costos en cada uno de los

Algoritmo 3.2 Sustracción en $\mathbb{F}_p$

Entrada: Primo p, y dos enteros $a,\ b \in [0, p-1]$ donde $a = (a_0, a_1, \ldots, a_{n-1})$ y $b = (b_0, b_1, \ldots, b_{n-1})$
Salida: $c = (a - b) \mod p$.

1: $(c_0, \text{préstamo}) \leftarrow \text{Resta}(a_0,\ b_0)$
2: **for** $i = 1 \rightarrow n-1$ **do**
3: $\quad (c_i, \text{préstamo}) \leftarrow \text{Resta\_con\_préstamo}(a_i,\ b_i,\ \text{préstamo})$
4: **end for**
5: $c \leftarrow (c_{n-1}, \ldots,\ c_2, c_1, c_0)$
6: **if** préstamo **then**
7: $\quad c \leftarrow c + p$ (Algoritmo 3.1)
8: **end if**
9: **return** c

campos. El principal motivo es que esta operación es la más utilizada, por lo que el costo de las operaciones aritméticas en las capas superiores depende directamente de su eficiencia. Dado que la suma y la resta son operaciones muy baratas, por lo general no suelen ser tomadas en cuenta, por otro lado, el inverso multiplicativo y la raíz cuadrada son operaciones que no se ejecutan frecuentemente por lo que sus costos tampoco son representativos.

Dado $a = (a_0, a_1, ..., a_{n-1})$ y $b = (b_0, b_1, ..., b_{n-1})$, un enfoque clásico para obtener el producto modular $c = (a \cdot b) \mod p$ consiste en realizar la multiplicación $t = a \cdot b$ y posteriormente obtener el módulo $c = t \mod p$. La multiplicación ab se realiza elemento por elemento, lo cual requiere n^2 multiplicaciones utilizando el método de la escuela:

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & t_{03} & t_{02} & t_{01} & t_{00} \\
 & & & t_{13} & t_{12} & t_{11} & t_{10} & \\
 & & t_{23} & t_{22} & t_{21} & t_{20} & & \\
 & t_{33} & t_{32} & t_{31} & t_{30} & & & \\
\hline
t_7 & t_6 & t_5 & t_4 & t_3 & t_2 & t_1 & t_0
\end{array}
$$

El producto entero $t = a \cdot b$ cumple que $|t| \simeq |a| + |b|$. Para la obtención del módulo $c = t \mod p$ se requiere realizar una división por p, la cual suele ser una operación muy costosa. Existen otros enfoques para lo obtención del producto, en esta tesis se describe el multiplicador de Montgomery, uno de los algoritmos más famosos de la aritmética computacional.

Multiplicador de Montgomery

El multiplicador de Montgomery [Montgomery, 1985] es el método más eficiente conocido para realizar la multiplicación modular. Este método reemplaza la operación de "dividir por p" por "divisiones entre r", donde $r = 2^k$ con $k - 1 < |p| < k$. Para ello se realiza una proyección de los enteros $a \in \mathbb{F}_p$ a los enteros $\tilde{a} \in \mathbb{F}_p$ utilizando la proyección: $\tilde{a} = a \cdot r \bmod p$. Al entero $\tilde{a}$ se le denomina p-residuo de a y se dice que el entero $\tilde{a}$ se encuentra en el dominio de Montgomery. Se define el producto de Montgomery de dos p-residuos como:

$$\text{MontPr}(\tilde{a}, \tilde{b}) = \tilde{a} \cdot \tilde{b} \cdot r^{-1} \bmod p$$

Encontrar a dado su p-residuo $\tilde{a}$ puede realizarse a través del producto de Montgomery por el entero unidad:

$$\text{MontPr}(\tilde{a}, 1) = \tilde{a} \cdot 1 \cdot r^{-1} \bmod p = a \bmod p$$

El multiplicador de Montgomery se muestra en el Algoritmo 3.3. Para su implementación es necesario pre-calcular p' tal que $r \cdot r^{-1} - p \cdot p' = 1$, el cual puede ser obtenido a través del algoritmo extendido de Euclides. El punto principal de este algoritmo radica en la obtención de $u = (t + (t \cdot p' \bmod r) \cdot p)/r$. Para entender su funcionamiento considere los siguientes aspectos:

1. Suponga que $m = t \cdot p' \pmod r$. Entonces $m \cdot p \equiv t \cdot p' \cdot p \pmod r$. Ahora bien, tenemos que $p' \equiv -p^{-1} \pmod r$, por lo que $t \cdot p' \cdot p \equiv -t \pmod r$. Por tanto $(t + m \cdot p) \equiv 0 \pmod r$, lo que significa que $(t + m \cdot p)$ es divisible entre r y que u es un número entero.

2. Además $u \cdot r = (t + m \cdot p) \equiv t \bmod p$, entonces $u \equiv t \cdot r^{-1} \pmod p$. Dado que $t = \tilde{a} \cdot \tilde{b}$, encontramos el valor buscado $u = \tilde{a} \cdot \tilde{b} \cdot r^{-1} \pmod p$.

3. Suponiendo que $0 \leq t < p \cdot r$, entonces $u \leq 2p$ (ya que $m < r$), por lo que la salida siempre es menor que p.

En este método, tanto la multiplicación como la división por r son operaciones rápidas, ésto se debe a que r es una potencia de 2, sin embargo, requiere del cálculo de los p-residuos[2] y de la proyección del producto de Montgomery a los enteros, por lo que este método usualmente se utiliza cuando se van a realizar varias multiplicaciones en sucesión.

Existen varias formas para implementar el algoritmo de multiplicación de Montgomery eficientemente [Koc et al., 1996], en este trabajo se describen los algoritmos SOS y CIOS, los cuales fueron implementados en la biblioteca desarrollada en esta tesis.

[2]Para realizar este cálculo se utilizó el método de Barret (Véase Apéndice A)

Algoritmo 3.3 Multiplicador de Montgomery

Entrada: Primo p, p', $r = 2^k$ y dos p-residuos $\tilde{a}$, $\tilde{b}$
Salida: MontPr$(\tilde{a}, \tilde{b})$
1: $t \leftarrow \tilde{a} \cdot \tilde{b}$
2: $u \leftarrow (t + (t \cdot p' \bmod r) \cdot p)/r$
3: **if** $u > p$ **then**
4: **return** $u - p$
5: **else**
6: **return** u
7: **end if**
8: **return** u

Método SOS

El método SOS (*Separated Operand Scanning*) realiza la multiplicación de Montgomery en dos fases, primero se obtiene el producto $a \cdot b$ como se muestra en el Algoritmo 3.4.

Algoritmo 3.4 Método SOS: Producto $a \cdot b$

Entrada: $a = (a_0, a_1, ..., a_{n-1})$ y $b = (b_0, b_1, ..., b_{n-1})$
Salida: $t = a \cdot b$ con $t = (t_0, t_1, ..., t_{2n-1})$
1: $t \leftarrow 0$
2: **for** $i = 0 \rightarrow n - 1$ **do**
3: $C \leftarrow 0$
4: **for** $j = 0 \rightarrow n - 1$ **do**
5: $(C, S) \leftarrow t_{i+j} + a_j \cdot b_i + C$
6: $t_{i+j} = S$
7: **end for**
8: $t_{i+n} = C$
9: **return** t
10: **end for**

Posteriormente se obtiene u de la fórmula $u = (t + m \cdot p)/r$ con $m = t \cdot p' \pmod r$ y $r = 2^{\omega n}$, donde ω corresponde al tamaño de palabra (32 bits) como se muestra en el Algoritmo 3.5.

Para calcular u primero se inicializa $u = t$, posteriormente se agrega $m \cdot p$ utilizando una rutina estándar de multiplicación y finalmente para dividir entre $r = 2^{n\omega}$ se ignoran los primeros n elementos de u. Dado que la reducción de $m = t \cdot p' \bmod r$ se realiza elemento por elemento se puede utilizar $p'_0 = p' \bmod 2^\omega$ en lugar de n'[3]. La función SUMA realiza la adición del elemento t_{i+n} con C y la propaga a los elementos posteriores hasta que no haya ningún acarreo generado. Una vez obtenido u se realiza

[3]Esta observación fue hecha en [Dussé and Kaliski, 1991]

Algoritmo 3.5 Método SOS: Cálculo de u

Entrada: $t = (t_0, t_1, ..., t_{2n-1})$, $p = (p_0, p_1, ..., p_{n-1})$ y p'_0
Salida: $u \leftarrow (t + (t \cdot p' \bmod r) \cdot p)/r$

```
1: for i = 0 → n − 1 do
2:    C ← 0
3:    m ← t_i · p'_0 mod 2^ω
4:    for j = 0 → n − 1 do
5:       (C, S) ← t_{i+j} + m · p_j + C
6:       t_{i+j} = S
7:    end for
8:    SUMA(t_{i+n}, C)
9: end for
10: for i = 0 → n − 1 do
11:    u_i = t_{i+n}
12: end for
13: return u
```

la comparación con p y se realiza la resta en caso de ser necesario. El costo final de este método es de $2n^2 + n$ multiplicaciones: n^2 en la obtención del producto $a \cdot b$, y $n^2 + n$ en la obtención de u.

Método CIOS

El método CIOS (*Coarsely Integrated Operand Scanning*) integra los pasos del producto y reducción. De manera más específica, en vez de realizar el producto $a \cdot b$ y posteriormente reducir, se alternan las operaciones de multiplicación y reducción dentro del ciclo principal. Este proceso es posible debido a que $m = t \cdot p' \pmod r$ depende únicamente del valor t_i el cual es calculado en la i-ésima iteración del ciclo principal como se muestra en el Algoritmo 3.6.

En este método se observa que la variable auxiliar t toma únicamente $n + 2$ elementos, a diferencia de la variable auxiliar t del método SOS que toma $2n$ elementos. El costo de este método es de $n^2 + n$ multiplicaciones, sin embargo requiere menos espacio en memoria que el método SOS.

Método de Karatsuba

El método de Karatsuba permite reducir el número de multiplicaciones para obtener el producto $a \cdot b$. Este método proviene de la idea "divide y vencerás", con lo que reduce la complejidad $O(n^2)$ del método tradicional a sólo $O(n^{log_2 3})$.

Suponga que $a = (a_0, a_1)$ y $b = (b_0, b_1)$ con $|a| = |b| = 2W$. Entonces, el producto $a \cdot b$ puede obtenerse como:

Algoritmo 3.6 Método CIOS

Entrada: $a = (a_0, a_1, ..., a_{n-1})$, $b = (b_0, b_1, ..., b_{n-1})$, $p = (p_0, p_1, ..., p_{n-1})$ y p'_0
Salida: MontPr(a, b)

1: $t = (t_0, t_1, ..., t_{n+1}) \leftarrow 0$
2: **for** $i = 0 \rightarrow n - 1$ **do**
3: $\quad C \leftarrow 0$
4: $\quad$ **for** $j = 0 \rightarrow n - 1$ **do**
5: $\quad\quad (C, S) \leftarrow t_j + a_j \cdot b_i + C$
6: $\quad\quad t_j = S$
7: $\quad$ **end for**
8: $\quad (C, S) \leftarrow t_n + C$, $t_n \leftarrow S$, $t_{n+1} \leftarrow C$
9: $\quad C \leftarrow 0$
10: $\quad m \leftarrow t_0 \cdot p'_0 \bmod 2^\omega$
11: $\quad (C, S) \leftarrow t_0 + m \cdot p_0$
12: $\quad$ **for** $j = 1 \rightarrow n - 1$ **do**
13: $\quad\quad (C, S) \leftarrow t_j + m \cdot p_j + C$
14: $\quad\quad t_{j-1} = S$
15: $\quad$ **end for**
16: $\quad (C, S) \leftarrow t_n + C$, $t_{n-1} \leftarrow S$, $t_n \leftarrow t_{n+1} + C$
17: **end for**
18: **if** $t > p$ **then**
19: $\quad$ **return** $t - p$
20: **else**
21: $\quad$ **return** t
22: **end if**

$$\begin{aligned} a \cdot b = &\ a_0 b_0 + (a_0 b_1 + a_1 b_0) 2^W + a_1 b_1 2^{2W} \\ = &\ a_0 b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) 2^W + a_1 b_1 2^{2W} \end{aligned}$$

donde se observa que el producto $a \cdot b$ puede calcularse con tres multiplicaciones en lugar de cuatro como ocurriría en el método tradicional, esto es una ventaja ya que para números grandes el costo de la suma y resta es insignificante con respecto a la multiplicación. Este método puede aplicarse de manera recursiva hasta un determinado umbral donde las multiplicaciones se realizan con el método clásico.

En esta tesis se propone el uso del método de Karatsuba para obtener el producto $a \cdot b$ en el método SOS. Sin embargo, el método se realiza sólo una vez debido que la propagación de acarreos causa un cuello de botella en su implementación, lo cual en la práctica, disminuye su eficiencia al aplicarse de forma recursiva. El costo final del método SOS de la sección anterior combinado con el método de Karatsuba es de $1.75n^2 + n$ multiplicaciones.

3.1.3. Inverso multiplicativo

El inverso multiplicativo es la operación en la que dado $a \in \mathbb{F}_p$ se busca encontrar a^{-1}. En este contexto, consiste en encontrar un elemento único $x \in \mathbb{F}_p$ tal que $a \cdot x = 1 \bmod p$. La forma más utilizada para calcular los inversos es a través del algoritmo extendido de Euclides. Debido a que las operaciones se realizan a través de la aritmética de Montgomery, el inverso multiplicativo no se calcula directamente. Para realizar esta operación se utiliza el método conocido como inverso multiplicativo de Montgomery.

Inverso multiplicativo de Montgomery

En la aritmética de Montgomery se selecciona un valor $r \geq 2^n$, donde $n = \lfloor log_2\ p \rfloor$. Sea $\tilde{a} = a \cdot r \bmod p$, el inverso multiplicativo de $\tilde{a}$ se define como $\tilde{a}^{-1} = a^{-1} \cdot r \bmod p$. El inverso multiplicativo de Montgomery [Hankerson et al., 2004] se realiza a través de dos pasos, el primero consiste en encontrar una inversión parcial $a^{-1}2^k$ con $k \in [n, 2n]$, tal y como se muestra en el Algoritmo 3.7, el cual es una modificación del algoritmo extendido de Euclides.

Algoritmo 3.7 Inversión parcial de Montgomery

Entrada: $p > 0$, $a \in [1, p-1]$ y $n = \lfloor log_2\ p \rfloor$
Salida: (x, k) tal que $n \leq k \leq 2n$ y $x = a^{-1}2^k \bmod p$

1: $u \leftarrow a$, $v \leftarrow p$, $x_1 \leftarrow 1$, $x_2 \leftarrow 0$, $k \leftarrow 0$
2: **while** $v > 0$ **do**
3: **if** v es par **then**
4: $v \leftarrow v/2$, $x_1 \leftarrow 2x_1$
5: **else if** u es par **then**
6: $u \leftarrow u/2$, $x_2 \leftarrow 2x_2$
7: **else if** $v \geq u$ **then**
8: $v \leftarrow (v-u)/2$, $x_2 \leftarrow x_2 + x_1$, $x_1 \leftarrow 2x_1$
9: **else**
10: $u \leftarrow (u-v)/2$, $x_1 \leftarrow x_2 + x_1$, $x_2 \leftarrow 2x_2$
11: **end if**
12: $k \leftarrow k + 1$
13: **end while**
14: **if** $x_1 > p$ **then**
15: **return** $x_1 \leftarrow x_1 - p$
16: **end if**
17: **return** (x_1, k)

El segundo paso consiste en encontrar el inverso multiplicativo de Montgomery como se muestra en el Algoritmo 3.8.

Algoritmo 3.8 Inverso multiplicativo de Montgomery

Entrada: Un primo $p > 2$, $n = \lceil \log_2(p) \rceil$, $\tilde{a} = a \cdot r \bmod p$, donde $r = 2^n$
Salida: $a^{-1}r \bmod p$

1: Encontrar (x, k), tal que $x = \tilde{a}^{-1}2^k \bmod p$, $n \leq k \leq 2n$ {Algoritmo 3.7}
2: **if** $k < n$ **then**
3: $\quad x \leftarrow \text{MontPr}(x, r^2)$
4: $\quad k \leftarrow x + n$
5: **end if**
6: $x \leftarrow \text{MontPr}(x, r^2)$
7: $x \leftarrow \text{MontPr}(x, 2^{2n-k})$
8: **return** x.

3.1.4. Exponenciación

Sea $a \in \mathbb{F}_p$ y $k \in \mathbb{Z}$, la exponenciación modular en el campo base se define como $a^k \bmod p$. Realizar esta operación requiere de multiplicaciones consecutivas. El Algoritmo 3.9 describe la exponenciación por medio del multiplicador Montogmery.

Algoritmo 3.9 Exponenciación de Montgomery

Entrada: $p > 0$, $r = 2^n$, $x \in [1, p-1]$, $e = (e_l, ..., e_0)_2$
Salida: $x^e \bmod p$

1: $\bar{x} \leftarrow x \cdot r \bmod p$
2: $A \leftarrow r \bmod p$
3: **for** $i = l \rightarrow 0$ **do**
4: $\quad A \leftarrow \text{MontPr}(A, A)$
5: $\quad$ **if** $e_i == 1$ **then**
6: $\quad\quad A \leftarrow \text{MontPr}(A, \bar{x})$
7: $\quad$ **end if**
8: **end for**
9: **return** $A \leftarrow \text{MontPr}(A, 1)$

Este algoritmo se conoce como método binario debido a la representación del exponente, sin embargo existen otros métodos con menor costo computacional [Hankerson et al., 2004]. En el caso de esta tesis se implementó este método debido a que la exponenciación únicamente es utilizada como bloque básico para el cómputo de la raíz cuadrada de un elemento $a \in \mathbb{F}_p$.

3.1.5. Raíz cuadrada

La raíz cuadrada en el campo base es una operación utilizada en el esquema ABE para obtener la proyección de una cadena a un punto (véase sección 6.3).

La raíz cuadrada consiste en encontrar $x \in \mathbb{F}_p$ tal que para $a \in \mathbb{F}_p$ se cumpla que $x^2 = a \pmod{p}$. Sea $g \in \mathbb{F}_p$ un generador, de acuerdo con el teorema pequeño de Fermat, tenemos que $a^{p-1} = 1 \pmod{p}$, entonces $a^{\frac{p-1}{2}} \pm 1 \pmod{p}$. Para $a = g^i$ con $i \in \mathbb{Z}^+$ se tiene que $a^{\frac{p-1}{2}} = (g^i)^{\frac{p-1}{2}} \pm 1 \pmod{p}$. Para i par se cumple que $(g^{\frac{i}{2}})^{p-1} = 1 \pmod{p}$ donde $g^{\frac{i}{2}}$ es una raíz de a, en caso contrario $(g^i)^{\frac{p-1}{2}} = -1 \pmod{p}$, lo que implica que el elemento a no tiene raíz. Por tanto, sólo la mitad de los elementos en $\mathbb{F}_p$ tienen raíces.

Existen varios algoritmos para calcular la raíz cuadrada dependiendo de la forma del primo p que se esté utilizando. De manera general, se tienen dos casos: cuando el $p \equiv 3 \pmod{4}$ y cuando $p \equiv 1 \pmod{4}$. El cálculo de la raíz cuadrada en el primer caso ($p \equiv 3 \bmod 4$) puede realizarse a través de una exponenciación, donde $x = a^{\frac{p+1}{4}}$. Ésto puede ser fácilmente comprobado a través del teorema pequeño de Fermat ($a^p \equiv a \bmod p$), entonces tenemos que:

$$x^2 = a^{\frac{p+1}{2}} = (a^{p+1})^{\frac{1}{2}}$$
$$x^2 = (a^p a)^{\frac{1}{2}} = (a^2)^{\frac{1}{2}}$$
$$\therefore x^2 = a$$

Lo cual es únicamente posible si $4|p+1$. El Algoritmo 3.10 de Shanks [Shanks, 1972] realiza el cálculo y verificación de la raíz cuadrada.

Algoritmo 3.10 Algoritmo de Shanks

Entrada: $a \in \mathbb{F}_p^*$
Salida: falso o $x \in \mathbb{F}_p^*$ tal que $x^2 = a$

1: $a_1 \leftarrow a^{\frac{p-3}{4}}$
2: $a_0 \leftarrow a_1^2 a$
3: **if** $a_0 = -1$ **then**
4: **return** falso
5: **end if**
6: $x \leftarrow a_1 a$
7: **return** x

Para el segundo caso ($p \equiv 1 \bmod 4$), la forma general de obtener esta raíz es a través del Algoritmo 3.11 de Tonelli-Skanks [Shanks, 1972, Niven et al., 1991].

3.2. Aritmética en $\mathbb{F}_{p^2}$

La aritmética en $\mathbb{F}_{p^2}$ corresponde a la segunda capa para la construcción de la torre de campos como se muestra en la figura 2.1 en la página 2.1. Se define la extensión cuadrática como:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta)$$

Algoritmo 3.11 Algoritmo de Tonelli-Shanks

Entrada: $a \in \mathbb{F}_p^*$
Salida: falso o $x \in \mathbb{F}_p^*$ tal que $x^2 = a$

PRECÓMPUTO

1: Obtenga (s, t) tal que $p - 1 = 2^s t$ donde t es impar
2: $c_0 \leftarrow 1$
3: **while** $c_0 = 1$ **do**
4: Seleccione $c \in \mathbb{F}_p^*$ de forma aleatoria
5: $z \leftarrow c^t$
6: $c_0 = c^{2^{s-1}}$
7: **end while**

CÓMPUTO

1: $\omega \leftarrow a^{\frac{t-1}{2}}$
2: $a_0 \leftarrow (\omega^2 a)^{2^{s-1}}$
3: **if** $a_0 = -1$ **then**
4: **return** falso
5: **end if**
6: $v \leftarrow s$, $x \leftarrow a\omega$, $b \leftarrow x\omega$
7: **while** $b \neq 1$ **do**
8: Encuentre el menor entero $m \geq 0$ tal que $b^{2^m} = 1$
9: $\omega \leftarrow z^{2^{v-m-1}}$, $z \leftarrow \omega^2$, $b \leftarrow bz$, $x \leftarrow x\omega$, $v \leftarrow m$.
10: **end while**
11: **return** x

donde $\beta = -1$.

Un elemento $A \in \mathbb{F}_{p^2}$ puede ser visto como $A = a_0 + a_1 u$, donde $a_0, a_1 \in \mathbb{F}_p$.

En esta sección se detallan las operaciones de adición, multiplicación, elevación al cuadrado e inverso multiplicativo, las cuales serán utilizadas por la capa superior $\mathbb{F}_{p^6}$.

3.2.1. Adición

Dados dos elementos A y $B \in \mathbb{F}_{p^2}$, el cálculo de $A + B$ requiere de 2 sumas en el campo $\mathbb{F}_p$ como se muestra en el Algoritmo 3.12.

En este caso la sustracción $A - B$ es una operación similar a la adición, la única diferencia es que en vez de sumas se realizan restas en $\mathbb{F}_p$. El Algoritmo 3.13 detalla como se realiza la sustracción en $\mathbb{F}_{p^2}$.

Algoritmo 3.12 Adición en $\mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1 u, \quad B = b_0 + b_1 u,\ A, B \in \mathbb{F}_{p^2}$
Salida: $C = A + B,\ C \in \mathbb{F}_{p^2}$
1: $c_0 \leftarrow a_0 + b_0$
2: $c_1 \leftarrow a_1 + b_1$
3: **return** $C = c_0 + c_1 u$

Algoritmo 3.13 Sustracción en $\mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1 u, \quad B = b_0 + b_1 u,\ A, B \in \mathbb{F}_{p^2}$
Salida: $C = A - B,\ C \in \mathbb{F}_{p^2}$
1: $c_0 \leftarrow a_0 - b_0$
2: $c_1 \leftarrow a_1 - b_1$
3: **return** $C = c_0 + c_1 u$

3.2.2. Multiplicación

La multiplicación de dos elementos $A, B \in \mathbb{F}_{p^2}$, se define como:

$$(a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 b_0 + a_1 b_1 \beta) + (a_0 b_1 + a_1 b_0) u,$$

Usando el método de Karatsuba, tenemos que:

$$(a_0 b_1 + a_1 b_0) = (a_0 + a_1) \cdot (b_0 + b_1) - a_0 b_0 - a_1 b_1.$$

El Algoritmo 3.14 muestra la multiplicación utilizando la técnica de Karatsuba.

Algoritmo 3.14 Multiplicación en $\mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1 u, \quad B = b_0 + b_1 u,\ A, B \in \mathbb{F}_{p^2}$
Salida: $C = A \cdot B,\ C \in \mathbb{F}_{p^2}$
1: $t_0 \leftarrow a_0 \cdot b_0$
2: $t_1 \leftarrow a_1 \cdot b_1$
3: $c_0 \leftarrow t_0 + t_1 \beta$
4: $c_1 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1) - t_0 - t_1$
5: **return** $C = c_0 + c_1 u$

Una observación hecha en [Beuchat et al., 2010] muestra que es posible realizar 2 reducciones en lugar de 3 si se divide el multiplicador de Montgomery en su producto y reducción como se describe en el Algoritmo 3.15. Una ventaja del algoritmo SOS es que maneja ambas operaciones por separado. En este algoritmo la operación **mod512** representa la reducción mostrada en el Algoritmo 3.5 mientras que la operación **mul256** representa la operación producto del método SOS. Cabe destacar que a diferencia de otros algoritmos en este campo, en este procedimiento todas las operaciones se realizan en el dominio de los enteros y no en el campo base $\mathbb{F}_p$.

Algoritmo 3.15 Multiplicación optimizada en $\mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1 u, \quad B = b_0 + b_1 u,\ A, B \in \mathbb{F}_{p^2}$
Salida: $C = A \cdot B,\ C \in \mathbb{F}_{p^2}$

1: $s \leftarrow a_0 + a_1$
2: $t \leftarrow b_0 + b_1$
3: $d_0 \leftarrow \text{mul256}(s, t)$
4: $d_1 \leftarrow \text{mul256}(a_0, b_0)$
5: $d_2 \leftarrow \text{mul256}(a_1, b_1)$
6: $d_0 \leftarrow d_0 - d_1 - d_2$
7: $c_1 \leftarrow \text{mod512}(d_0)$
8: $d_1 \leftarrow d_1 + \beta d_2$
9: $c_0 \leftarrow \text{mod512}(d_1)$
10: **return** $C = c_0 + c_1 u$

Multiplicación por b_0

Como se verá más adelante, a veces es necesario realizar la multiplicación $A \cdot B$ con $B = b_0$. El uso de los algoritmos previos derivaría en la aplicación innecesaria de multiplicaciones, por lo que en el Algoritmo 3.16 se describe la operación con un menor número de operaciones.

Algoritmo 3.16 Multiplicación por $b_0 \in \mathbb{F}_p$

Entrada: $A = a_0 + a_1 u,\ A \in \mathbb{F}_{p^2},\ b_0 \in \mathbb{F}_p$
Salida: $C = A \cdot b_0,\ C \in \mathbb{F}_{p^2}$

1: $c_0 \leftarrow a_0 \cdot b_0$
2: $c_1 \leftarrow a_1 \cdot b_0$
3: **return** $C = c_0 + c_1 u$

3.2.3. Elevación al cuadrado

La operación A^2, donde $A \in \mathbb{F}_{p^2}$, es calculada de forma análoga al método complejo de elevación al cuadrado, a través de la siguiente identidad:

$$(a_0 + a_1 u)^2 = (a_0 - \beta a_1) \cdot (a_0 - a_1) + (\beta + 1) a_0 a_1 + 2 a_0 a_1 u.$$

En el caso de $\beta = -1$ tenemos que:

$$(a_0 + a_1 u)^2 = (a_0 + a_1) \cdot (a_0 - a_1) + 2 a_0 a_1 u.$$

Por lo que la elevación al cuadrado consta entonces de 2 multiplicaciones en el campo base como se muestra en el Algoritmo 3.17.

Algoritmo 3.17 Elevación en $\mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1 u$, $A \in \mathbb{F}_{p^2}$
Salida: $C = A^2$, $C \in \mathbb{F}_{p^2}$
1: $c_0 \leftarrow a_0 - a_1$
2: $c_2 \leftarrow a_0 + a_1$
3: $c_1 \leftarrow 2a_0 a_1$
4: $c_0 \leftarrow c_0 \cdot c_2$
5: **return** $C = c_0 + c_1 u$

3.2.4. Inverso multiplicativo

El inverso de un elemento en el grupo multiplicativo del campo finito $\mathbb{F}_{p^2}$, es calculado mediante la siguiente identidad:

$$(a_0 + a_1 u)^{-1} = \frac{a_0 - a_1 u}{(a_0 - a_1 z) \cdot (a_0 + a_1 z)} = \frac{a_0}{a_0^2 - a_1^2 \beta} - \frac{a_1 u}{a_0^2 - a_1^2 \beta}.$$

Esta operación tiene un costo de 4 multiplicaciones y 1 inversión en el campo $\mathbb{F}_p$ de acuerdo con el Algoritmo 3.18.

Algoritmo 3.18 Inverso multiplicativo en $\mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1 u$, $A \in \mathbb{F}_{p^2}$
Salida: $C = A^{-1}$, $C \in \mathbb{F}_{p^2}$
1: $t_0 \leftarrow a_0^2$
2: $t_1 \leftarrow a_1^2$
3: $t_0 \leftarrow t_0 - \beta t_1$
4: $t_1 \leftarrow t_0^{-1}$
5: $c_0 \leftarrow a_0 \cdot t_1$
6: $c_1 \leftarrow -a_1 \cdot t_1$
7: **return** $C = c_0 + c_1 u$

3.3. Aritmética en $\mathbb{F}_{p^4}$

Más adelante se muestra la utilización de la aritmética en $\mathbb{F}_{p^4}$ como base para elevar al cuadrado en el grupo ciclotómico descrito en la sección 3.5. En esta sección únicamente se describe la operación *elevar al cuadrado*, ya que es la única operación utilizada en este campo por la biblioteca desarrollada en este trabajo.

Se define la extensión cuadrática como

$$\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[V]/(V^2 - \xi)$$

donde $\xi = u + 1$.

Un elemento $A \in \mathbb{F}_{p^4}$ puede ser visto como $A = a_0 + a_1 V$ donde $a_0, a_1 \in \mathbb{F}_{p^2}$. El Algoritmo 3.19 describe la operación elevación al cuadrado por medio del método de Karatsuba.

Algoritmo 3.19 Elevación al cuadrado en $\mathbb{F}_{p^4}$

Entrada: $A = a_0 + a_1 u$, $A \in \mathbb{F}_{p^4}$
Salida: $C = A^2$, $C \in \mathbb{F}_{p^4}$
1: $t_0 \leftarrow a_0^2$
2: $t_1 \leftarrow a_1^2$
3: $c_0 \leftarrow t_0 + t_1 \xi$
4: $c_1 \leftarrow a_0 + a_1$
5: $c_1 \leftarrow c_1^2 - t_0 - t_1$
6: **return** $C = c_0 + c_1 u$

3.4. Aritmética en $\mathbb{F}_{p^6}$

Esta aritmética corresponde al segundo nivel de la torre de campos. Se define la extensión cúbica como:

$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[V]/(V^3 - \xi)$$

donde $\xi = u + 1$.

Un elemento $A \in \mathbb{F}_{p^6}$ puede ser visto como $A = a_0 + a_1 V + a_2 V^2$ donde $a_0, a_1, a_2 \in \mathbb{F}_{p^2}$.

En esta sección se detallan las operaciones de adición, multiplicación, elevación al cuadrado e inverso multiplicativo, las cuales serán utilizadas por la capa superior $\mathbb{F}_{p^{12}}$.

3.4.1. Adición

Al igual que en la aritmética en $\mathbb{F}_{p^2}$, la operaciones suma y resta en $\mathbb{F}_{p^6}$ se realizan elemento por elemento como se muestra en los Algoritmos 3.20 y 3.21 respectivamente.

Algoritmo 3.20 Adición en $\mathbb{F}_{p^6}$

Entrada: $A = a_0 + a_1 V + a_2 V^2$, $B = b_0 + b_1 V + b_2 V^2$, $A, B \in \mathbb{F}_{p^6}$
Salida: $C = A + B$, $C \in \mathbb{F}_{p^6}$
1: $c_0 \leftarrow a_0 + b_0$
2: $c_1 \leftarrow a_1 + b_1$
3: $c_2 \leftarrow a_2 + b_2$
4: **return** $C = c_0 + c_1 V + c_2 V^2$

Algoritmo 3.21 Sustracción en $\mathbb{F}_{p^6}$

Entrada: $A = a_0 + a_1V + a_2V^2, \quad B = b_0 + b_1V + b_2V^2$, $A, B \in \mathbb{F}_{p^6}$
Salida: $C = A - B$, $C \in \mathbb{F}_{p^6}$
1: $c_0 \leftarrow a_0 - b_0$
2: $c_1 \leftarrow a_1 - b_1$
3: $c_2 \leftarrow a_2 - b_2$
4: **return** $C = c_0 + c_1V + c_2V^2$

3.4.2. Multiplicación

El producto de dos elementos $A = a_0 + a_1V + a_2V^2$, $B = b_0 + b_1V + b_2V^2$ en el campo $\mathbb{F}_{p^6}$ se calcula a través del método de Karatsuba, con un costo de 6 multiplicaciones en el campo $\mathbb{F}_{p^2}$ como se observa en el Algoritmo 3.22.

Cuando $B = b_0$ se realiza una operación similar a la descrita en la aritmética en $\mathbb{F}_{p^2}$ mostrada en el Algoritmo 3.23. Para el caso donde $B = b_0 + b_1V$ se utiliza el Algoritmo 3.24.

Algoritmo 3.22 Multiplicación en $\mathbb{F}_{p^6}$

Entrada: $A = a_0 + a_1V + a_2V^2, \quad B = b_0 + b_1V + b_2V^2$, $A, B \in \mathbb{F}_{p^6}$
Salida: $C = A \cdot B$, $C \in \mathbb{F}_{p^6}$
1: $v_0 \leftarrow a_0 \cdot b_0$
2: $v_1 \leftarrow a_1 \cdot b_1$
3: $v_2 \leftarrow a_2 \cdot b_2$
4: $c_0 \leftarrow ((a_1 + a_2) \cdot (b_1 + b_2) - v_1 - v_2) \cdot \xi + v_0$
5: $c_1 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1) - v_0 - v_1 + \xi \cdot v_2$
6: $c_2 \leftarrow (a_0 + a_2) \cdot (b_0 + b_2) - v_0 - v_2 + v_1$
7: **return** $C = c_0 + c_1V + c_2V^2$

Algoritmo 3.23 Multiplicación por $b_0 \in \mathbb{F}_{p^2}$

Entrada: $A = a_0 + a_1V + a_2V^2$, $A \in \mathbb{F}_{p^6}$, $b_0 \in \mathbb{F}_{p^2}$
Salida: $C = A \cdot b_0$, $C \in \mathbb{F}_{p^6}$
1: $c_0 \leftarrow a_0 \cdot b_0$
2: $c_1 \leftarrow a_1 \cdot b_0$
3: $c_2 \leftarrow a_2 \cdot b_0$
4: **return** $C = c_0 + c_1V + c_2V^2$

Algoritmo 3.24 Multiplicación por $b_0 + b_1V$

Entrada: $A = a_0 + a_1V + a_2V^2$, $A \in \mathbb{F}_{p^6}$, $b_0, b_1 \in \mathbb{F}_{p^2}$
Salida: $C = A \cdot (b_0 + b_1V)$, $C \in \mathbb{F}_{p^6}$

1: $t_0 \leftarrow a_0 \cdot b_0$
2: $t_1 \leftarrow a_1 \cdot b_1$
3: $c_0 \leftarrow ((a_1 + a_2) \cdot b_1 - t_1) \cdot \xi + t_0$
4: $c_1 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1) - t_0 - t_1$
5: $c_2 \leftarrow a_2 \cdot b_0 + t_1$
6: **return** $C = c_0 + c_1V + c_2V^2$

3.4.3. Elevación al cuadrado

El Algoritmo 3.25 describe el método basado en [Chung and Hasan, 2007] para la obtención del cuadrado en extensiones cúbicas como es el caso del campo $\mathbb{F}_{p^6}$. Este algoritmo tiene un costo de 2 multiplicaciones y 3 cuadrados en el campo $\mathbb{F}_{p^2}$.

Algoritmo 3.25 Elevación al cuadrado en $\mathbb{F}_{p^6}$

Entrada: $A = a_0 + a_1V + a_2V^2$, $A \in \mathbb{F}_{p^6}$
Salida: $C = A^2$, $C \in \mathbb{F}_{p^6}$

1: $v_4 \leftarrow 2(a_0 \cdot a_1)$
2: $v_5 \leftarrow a_2^2$
3: $c_1 \leftarrow \xi \cdot v_5 + v_4$
4: $v_2 \leftarrow v_4 - v_5$
5: $v_3 \leftarrow a_0^2$
6: $v_4 \leftarrow a_0 - a_1 + a_2$
7: $v_5 \leftarrow 2(a_1 + a_2)$
8: $v_4 \leftarrow v_4^2$
9: $c_0 \leftarrow \xi \cdot v_5 + v_3$
10: $c_2 \leftarrow v_2 + v_4 + v_5 - v_3$
11: **return** $C = c_0 + c_1w + c_2w^2$

3.4.4. Inverso multiplicativo

Sea $A = a_0 + a_1V + a_2V^2$, $A \in \mathbb{F}_{p^6}$, se define el inverso A^{-1} como:

$$A^{-1} = (a_0 + a_1V + a_2V^2)^{-1} = (A + BV + CV^2)/F,$$

donde:

$A = a_0^2 - \xi a_1 a_2$,
$B = \xi a_2^2 - a_0 a_1$,
$C = a_1^2 - a_0 a_2$, y
$F = \xi a_1 C + a_0 A + \xi a_2 B$

Esta operación se calcula a través del Algoritmo 3.26 y tiene un costo de 9 multiplicaciones, 3 cuadrados y un inverso en el campo $\mathbb{F}_{p^6}$.

Algoritmo 3.26 Inverso multiplicativo en $\mathbb{F}_{p^6}$

Entrada: $A = a_0 + a_1V + a_2V^2$, $A \in \mathbb{F}_{p^6}$
Salida: $C = A^{-1}$, $C \in \mathbb{F}_{p^6}$

1: $t_0 \leftarrow a_0^2$
2: $t_1 \leftarrow a_1^2$
3: $t_2 \leftarrow a_2^2$
4: $t_3 \leftarrow a_0 \cdot a_1$
5: $t_4 \leftarrow a_0 \cdot a_2$
6: $t_5 \leftarrow a_1 \cdot a_2$
7: $c_0 \leftarrow t_0 - \xi \cdot t_5$
8: $c_1 \leftarrow \xi t_2 - t_3$
9: $c_2 \leftarrow t_1 - t_4$
10: $t_6 \leftarrow a_0 \cdot c_0$
11: $t_6 \leftarrow t_6 + \xi \cdot a_2 \cdot c_1$
12: $t_6 \leftarrow t_6 + \xi \cdot a_1 \cdot c_2$
13: $t_6 \leftarrow t_6^{-1}$
14: $c_0 \leftarrow c_0 \cdot t_6$
15: $c_1 \leftarrow c_1 \cdot t_6$
16: $c_2 \leftarrow c_2 \cdot t_6$
17: **return** $C = c_0 + c_1V + c_2V^2$

3.5. Aritmética en $\mathbb{F}_{p^{12}}$

La aritmética en $\mathbb{F}_{p^{12}}$ corresponde a la última capa de la torre de campos. Se define la extensión cuadrática

$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[W]/(W^2 - \gamma)$$

donde $\gamma = V$.

Un elemento $a \in \mathbb{F}_{p^{12}}$ puede ser visto como $a = a_0 + a_1W$ donde $a_0, a_1 \in \mathbb{F}_{p^6}$. Para esta aritmética se implementaron las operaciones: suma y resta, multiplicación, cuadrado e inverso.

3.5.1. Adición

Dados dos elementos A y $B \in \mathbb{F}_{p^{12}}$, el cálculo de $A + B$ requiere de 2 sumas en el campo $\mathbb{F}_{p^6}$ como se muestra en el Algoritmo 3.27.

El Algoritmo 3.28 detalla cómo se realiza la sustracción en $\mathbb{F}_{p^{12}}$.

Algoritmo 3.27 Adición en $\mathbb{F}_{p^{12}}$

Entrada: $A = a_0 + a_1 W,\ \ B = b_0 + b_1 W,\ A, B \in \mathbb{F}_{p^{12}}$
Salida: $C = A + B,\ C \in \mathbb{F}_{p^{12}}$
1: $c_0 \leftarrow a_0 + b_0$
2: $c_1 \leftarrow a_1 + b_1$
3: **return** $C = c_0 + c_1 W$

Algoritmo 3.28 Sustracción en $\mathbb{F}_{p^{12}}$

Entrada: $A = a_0 + a_1 W,\ \ B = b_0 + b_1 W,\ A, B \in \mathbb{F}_{p^{12}}$
Salida: $C = A - B\ ,\ C \in \mathbb{F}_{p^{12}}$
1: $c_0 \leftarrow a_0 - b_0$
2: $c_1 \leftarrow a_1 - b_1$
3: **return** $C = c_0 + c_1 W$

3.5.2. Multiplicación

Al ser $\mathbb{F}_{p^{12}}$ una extensión cuadrática, los algoritmos son muy similares a los algoritmos de $\mathbb{F}_{p^2}$. El Algoritmo 3.29 calcula el producto de dos elementos $A, B \in \mathbb{F}_{p^{12}}$.

Algoritmo 3.29 Multiplicación en $\mathbb{F}_{p^{12}}$

Entrada: $A = a_0 + a_1 W,\ \ B = b_0 + b_1 W,\ A, B \in \mathbb{F}_{p^{12}}$
Salida: $C = A \cdot B,\ C \in \mathbb{F}_{p^{12}}$
1: $t_0 \leftarrow a_0 \cdot b_0$
2: $t_1 \leftarrow a_1 \cdot b_1$
3: $c_0 \leftarrow t_0 + t_1 \gamma$
4: $c_1 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1) - t_0 - t_1$
5: **return** $C = c_0 + c_1 W$

El Algoritmo 3.30 muestra la multiplicación por $B = b_0 + b_1 W$ donde $b_0 \in \mathbb{F}_{p^2}$ y $b_1 = b_{10} + b_{11} V + 0V^2$, la cual es utilizada en el cálculo del emparejamiento.

3.5.3. Elevación al cuadrado

Para elevar al cuadrado en $\mathbb{F}_{p^{12}}$ nuevamente se hace uso del método complejo como se muestra en el Algoritmo 3.31

3.5.4. Inverso multiplicativo

El Algoritmo 3.32 realiza la inversión de $A \in \mathbb{F}_{p^{12}}$.

Algoritmo 3.30 Multiplicación por $B = b_0+b_1W$ con $b_0 \in \mathbb{F}_{p^2}$ y $b_1 = b_{10}+b_{11}V+0V^2$

Entrada: $A = a_0 + a_1W, \quad B = b_0 + b_1W$, $A, B \in \mathbb{F}_{p^{12}}$ donde $b_0 = b_{00} + 0V + 0V^2$ y $b_1 = b_{10} + b_{11}V + 0V^2$
Salida: $C = A \cdot B$, $C \in \mathbb{F}_{p^{12}}$

1: $t_0 \leftarrow a_0 \cdot b_0$ Algoritmo 3.23
2: $t_1 \leftarrow a_1 \cdot b_1$ Algoritmo 3.24
3: $c_0 \leftarrow t_0 + t_1\gamma$
4: $t_2 \leftarrow (b_{00} + b_{10}) + b_{11}V + 0V^2$
5: $c_1 \leftarrow (a_0 + a_1) \cdot t_2$ Algoritmo 3.24
6: $c_1 \leftarrow c_1 - t_0 - t_1$
7: **return** $C = c_0 + c_1W$

Algoritmo 3.31 Elevación en $\mathbb{F}_{p^{12}}$

Entrada: $A = a_0 + a_1W$, $A \in \mathbb{F}_{p^{12}}$
Salida: $C = A^2$, $C \in \mathbb{F}_{p^{12}}$

1: $c_0 \leftarrow a_0 - a_1$
2: $c_3 \leftarrow a_0 + \gamma \cdot a_1$
3: $c_2 \leftarrow a_0 \cdot a_1$
4: $c_0 \leftarrow c_0 \cdot c_3 + c_2$
5: $c_1 \leftarrow 2c_2$
6: $c_2 \leftarrow \gamma \cdot c_2$
7: $c_0 \leftarrow c_0 + c_2$
8: **return** $C = c_0 + c_1W$

Algoritmo 3.32 Inverso multiplicativo en $\mathbb{F}_{p^{12}}$

Entrada: $A = a_0 + a_1W$, $A \in \mathbb{F}_{p^{12}}$
Salida: $C = A^{-1}$, $C \in \mathbb{F}_{p^{12}}$

1: $t_0 \leftarrow a_0^2$
2: $t_1 \leftarrow a_1^2$
3: $t_0 \leftarrow t_0 - \gamma t_1$
4: $t_1 \leftarrow t_0^{-1}$
5: $c_0 \leftarrow a_0 \cdot t_1$
6: $c_1 \leftarrow -a_1 \cdot t_1$
7: **return** $C = c_0 + c_1W$

3.6. Grupo ciclotómico $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$

La ventaja principal del grupo ciclotómico es que requiere de un menor número de operaciones en la obtención del cuadrado e inverso multiplicativo. En el cálculo de la *exponenciación final* del capítulo 5 nos encontramos que el elemento $f \in \mathbb{F}_{p^{12}}$ pertenece al grupo ciclotómico $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ como se describe en [Granger and Scott, 2010]. En esta sección se describen las operaciones: elevación al cuadrado e inverso multiplicativo en dicho grupo.

3.6.1. Inverso en el grupo ciclotómico

Los elementos del grupo ciclotómico satisfacen $\alpha^{p^6+1} = 1$. lo que quiere decir que $\alpha^{-1} = \alpha^{p^6} = \bar{\alpha}$. En otras palabras, la inversión de elementos en $\mathbb{F}_{p^{12}}$, se puede realizar mediante una conjugación. Para un $\alpha = a_0 + a_1 W \in \mathbb{F}_{p^{12}}$ el conjugado $\bar{\alpha}$ se define como:

$$\bar{\alpha} = a_0 - a_1 W$$

el cual consiste en encontrar el inverso aditivo del elemento a_1.

3.6.2. Cuadrados en el grupo ciclotómico

El campo $\mathbb{F}_{p^{12}}$ puede definirse como una extensión cúbica:

$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^4}[z]/(z^3 - \gamma)$$

En este grupo un elemento $\alpha \in \mathbb{F}_{p^{12}}$ puede ser representado como:

$$\begin{gathered} \alpha = a + bz + cz^2 \text{ con} \\ a = a_0 + a_1 V, \quad b = b_0 + b_1 V, \quad c = c_0 + c_1 V \end{gathered} \tag{3.1}$$

donde $a, b, c \in \mathbb{F}_{p^4}$ y $a_i, b_i, c_i \in \mathbb{F}_{p^2}$.

La obtención del cuadrado de un elemento $\alpha \in \mathbb{F}_{p^{12}}$ de la forma tradicional es:

$$\begin{aligned} \alpha^2 &= (a + bz + cz^2)^2 \\ &= (a^2 + 2bc\gamma) + (2ab + c^2\gamma)z + (2ac + b^2)z^2 \end{aligned} \tag{3.2}$$

La cual se realiza con 3 multiplicaciones y 3 cuadrados en $\mathbb{F}_{p^2}$.

Sin embargo, de acuerdo con los trabajos de [Granger and Scott, 2010] y [Karabina, 2010] se tiene que si α pertenece al grupo ciclotómico $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$, la obtención de su cuadrado α^2 puede realizarse con un menor número de operaciones.

Cuadrados de Granger y Scott

Sea $\alpha \in \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ un elemento del grupo ciclotómico de la forma $\alpha = a + bz + cz^2$ (ecuación 3.1), los elementos a, b, c satisfacen las siguientes identidades:

$$\begin{aligned} bc &= a^2 - \bar{a}/\gamma \\ ab &= c^2\gamma - \bar{b} \\ ac &= b^2 - \bar{c} \end{aligned}$$

donde $\bar{a}, \bar{b}$ y $\bar{c}$ corresponden a los conjugados de a, b y c respectivamente.

Si sustituimos los valores bc, ab y ac en la ecuación 3.2, obtenemos que la operación α^2 puede ser calculada con un costo de 3 cuadrados y 12 sumas en el campo $\mathbb{F}_{p^4}$, como se muestra a continuación:

$$\alpha^2 = (3a^2 - 2\bar{a}) + (3c^2\gamma + 2\bar{b})z + (3b^2 - 2\bar{c})z^2$$

Para la implementación de estos cuadrados se utilizó el Algoritmo 3.33.

Algoritmo 3.33 Elevación al cuadrado en $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$

Entrada: $A = g + hW$, $A \in \mathbb{F}_{p^{12}}$, con $g = g_0 + g_1 v + g_2 v^2$ y $h = h_0 + h_1 v + h_2 v^2$.
Salida: $C = A^2$, $C \in \mathbb{F}_{p^{12}}$

1: $t_{0,0}, t_{1,1} \leftarrow (g_0 + h_1 V)^2$
2: $t_{0,1}, t_{1,2} \leftarrow (h_0 + g_2 V)^2$
3: $t_{0,2}, tmp \leftarrow (g_1 + h_2 V)^2$
4: $t_{1,0} \leftarrow tmp \cdot \xi$
5: $c_{0,0} \leftarrow -2g_0 + 3t_{0,0}$
6: $c_{0,1} \leftarrow -2g_1 + 3t_{0,1}$
7: $c_{0,2} \leftarrow -2g_2 + 3t_{0,2}$
8: $c_{1,0} \leftarrow 2h_0 + 3t_{1,0}$
9: $c_{1,1} \leftarrow 2h_1 + 3t_{1,1}$
10: $c_{1,2} \leftarrow 2h_2 + 3t_{1,2}$
11: $c_0 \leftarrow c_{0,0} + c_{0,1} v + c_{0,2} v^2$
12: $c_1 \leftarrow c_{1,0} + c_{1,1} v + c_{1,2} v^2$
13: **return** $C = c_0 + c_1 W$

Cuadrados de Karabina

El algoritmo de Karabina propone el uso de cuadrados comprimidos [Karabina, 2010], en el cual, dado un α su cuadrado comprimido es $BV + CV^2$ donde $\alpha^2 = (A + Bz + Cz^2)$. Este algoritmo se compone de tres bloques:

1. El elemento $\alpha = a + bz + cz^2$ (ecuación 3.1) se comprime por la función $\mathcal{C}$:

$$\mathcal{C}(\alpha) = (b_0 + b_1 V)z + (c_0 + c_1 V)z^2$$

2. Se calcula $\mathcal{C}(\alpha^2) = (B_0 + B_1V)z + (C_0 + C_1V)z^2$ en base a las siguientes ecuaciones:

$$B_0 = 2b_0 + 3((c_0 + c_1)^2 - c_0^2 - c_1^2) \qquad B_1 = 3(c_0^2 + c_1^2\xi) - 2b_1$$

$$C_0 = 3(b_0^2 + b_1^2\xi) - 2c_0 \qquad C_1 = 2c_1 + 3((b_0 + b_1)^2 - b_0^2 - b_1^2)$$

Esto tiene un costo de 2 cuadrados en $\mathbb{F}_{p^4}$ como se muestra en el Algoritmo 3.34.

3. Para calcular $\alpha^2 = \mathcal{D}(\mathcal{C}(\alpha^2)) = (A_0 + A_1V) + (B_0 + B_1V)z + (C_0 + C_1V)z^2$ se obtienen los elementos A_0 y A_1 de la siguiente forma:

$$A_0 = \frac{C_1^2\xi + 3C_0^2 - 2B_1}{4B_0}, \quad A_1 = (2A_1^2 + B_0C_1 - 3B_1C_0)\xi + 1 \qquad \text{si } B_0 \neq 0$$

$$A_0 = \frac{2C_0C_1}{B_1}, \qquad A_1 = (2A_1^2 - 3B_1C_0)\xi + 1 \qquad \text{si } B_0 = 0$$

Este método de descompresión tiene un costo 3 cuadrados, 3 multiplicaciones y 1 inverso.

Algoritmo 3.34 Cuadrados comprimidos en $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$

Entrada: $A = g + hW$, $A \in \mathbb{F}_{p^{12}}$, con $g = g_0 + g_1v + g_2v^2$ y $h = h_0 + h_1v + h_2v^2$.
Salida: $\mathcal{C}(A^2)$

1: $t_{0,0}, t_{1,1} \leftarrow (h_0 + g_2V)^2$
2: $t_{0,1}, tmp \leftarrow (g_1 + h_2V)^2$
3: $t_{1,0} \leftarrow tmp \cdot \xi$
4: $c_{0,1} \leftarrow -2g_1 + 3t_{0,0}$
5: $c_{0,2} \leftarrow -2g_2 + 3t_{0,2}$
6: $c_{1,0} \leftarrow 2h_0 + 3t_{1,0}$
7: $c_{1,2} \leftarrow 2h_2 + 3t_{1,1}$
8: $c_0 \leftarrow c_{0,1}v + c_{0,2}v^2$
9: $c_1 \leftarrow c_{1,0} + c_{1,2}v^2$
10: **return** $C = c_0 + c_1W$

Debido al costo computacional en la descompresión $\mathcal{D}(\mathcal{C}(\alpha^2))$, este algoritmo sólo es útil cuando se calculan múltiples cuadrados en sucesión. En el capítulo 5 se muestra su uso en el cálculo de la *exponenciación final*.

3.7. Resumen de costos

La forma más directa de establecer los costos en la aritmética de cada una de las extensiones de campo es a través del conteo del número de operaciones requeridas en

cada caso. La tabla 3.1 presenta la notación utilizada para representar las operaciones en cada extensión:

Operación	$\mathbb{F}_p$	$\mathbb{F}_{p^2}$	$\mathbb{F}_{p^4}$	$\mathbb{F}_{p^6}$	$\mathbb{F}_{p^{12}}$	$\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$
Adición/sustracción	a	$\tilde{a}$	α	A	$\tilde{A}$	
Multiplicación	m	$\tilde{m}$		M	$\tilde{M}$	
Elevación al cuadrado		$\tilde{s}$	δ	S	$\tilde{S}$	$\tilde{\delta}$
Inversión	i	$\tilde{i}$		I	$\tilde{I}$	$\tilde{C}$
Multiplicación por β	m_β					
Multiplicación por ξ		m_ξ				
Multiplicación por γ		m_γ				

Tabla 3.1: Notación de operaciones

Por términos de conveniencia y comparación, las operaciones en $\mathbb{F}_{p^2}$ se expresarán en términos de $\mathbb{F}_p$, mientras que las operaciones en $\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}$, $\mathbb{F}_{p^{12}}$ y $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ se expresarán en términos de $\mathbb{F}_{p^2}$ como se presenta en la tabla 3.2.

Campo	Suma	Multiplicación	Cuadrado	Inversión
$\mathbb{F}_{p^2}$	$\tilde{a} = 2a$	$\tilde{m} = 3m + 5a + m_\beta$	$\tilde{s} = 2m + 3a + m_\beta$	$\tilde{i} = 4m + m_\beta + 2a + i$
$\mathbb{F}_{p^4}$	$2\tilde{a}$		$3\tilde{s} + 1m_\xi + 4\tilde{a}$	
$\mathbb{F}_{p^6}$	$3\tilde{a}$	$6\tilde{m} + 2m_\xi + 15\tilde{a}$	$2\tilde{m} + 3\tilde{s} + 2m_\xi + 9\tilde{a}$	$9\tilde{m} + 3\tilde{s} + 4m_\xi + 5\tilde{a} + \tilde{i}$
$\mathbb{F}_{p^{12}}$	$6\tilde{a}$	$18\tilde{m} + 6m_\xi + 60\tilde{a}$ $+m_\gamma$	$12\tilde{m} + 4m_\xi + 45\tilde{a}$ $+2m_\gamma$	$25\tilde{m} + 9\tilde{s} + 12m_\xi$ $+61\tilde{a} + \tilde{i} + m_\gamma$
$\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$			$9\tilde{s} + 4m_\xi + 30\tilde{a}$	$\tilde{C} \approx 3\tilde{a}$

Tabla 3.2: Resumen de costos

Capítulo 4

Curvas elípticas

"Cuanto más sabes, más te das cuenta de que no sabes nada"
Sócrates

Una curva elíptica E sobre un campo $\mathbb{F}$ se define como la gráfica correspondiente a la ecuación de Weierstrass:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4.1}$$

donde $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$. Se dice que E está definida sobre $\mathbb{F}$ debido a que los coeficientes a_1, a_2, a_3, a_4, a_6 de la ecuación (4.1) son elementos de $\mathbb{F}$. Se escribe $E/\mathbb{F}$ para enfatizar que E se encuentra definida sobre $\mathbb{F}$. Cabe aclarar que si E es definida sobre $\mathbb{F}$, entonces E está también definida sobre cualquier extensión de $\mathbb{F}$.

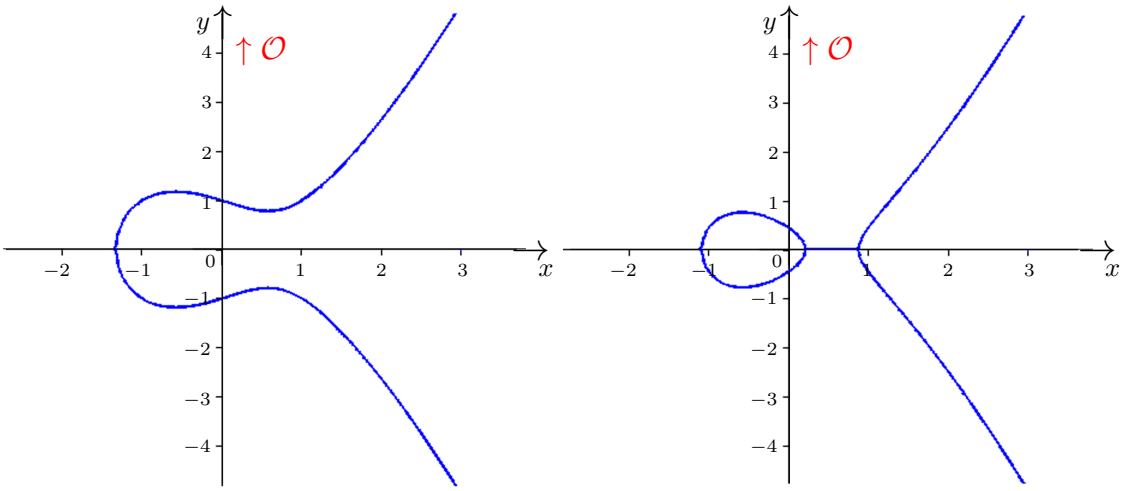
Una versión particular de la ecuación de Weierstrass es la ecuación (4.2), también conocida como la *ecuación simplificada de Weierstrass* es:

$$y^2 = x^3 + Ax + B \tag{4.2}$$

donde A y B son constantes en $\mathbb{F}$. La figura 4.1 muestra las gráficas de las curvas elípticas $E_1 : y^2 = x^3 - x + 1$ y $E_2 : y^2 = x^3 - x + 0.2$ definidas sobre $\mathbb{R}$.

4.1. Puntos de una curva elíptica

Definición 4.1. Punto al infinito. *El punto al infinito es el correspondiente al par (∞, ∞), el cual se denota por $\mathcal{O}$. Dicho punto se encuentra situado tanto en el extremo inferior como en el extremo superior del eje de las ordenadas, de manera que cualquier línea vertical interseca a $\mathcal{O}$.*



(a) $y^2 = x^3 - x + 1$ (b) $y^2 = x^3 - x + 0.2$

Figura 4.1: Curvas elípticas sobre $\mathbb{R}$

Sea $E/\mathbb{F}$ la curva elíptica definida sobre la ecuación (4.2) y $\mathbb{F}'$ cualquier extensión de $\mathbb{F}$, el conjunto de los $\mathbb{F}'$-*puntos racionales* se denota como:

$$E(\mathbb{F}') = \{(x, y) \in \mathbb{F}' \times \mathbb{F}' : y^2 - x^3 - Ax - B = 0\} \cup \{\mathcal{O}\}$$

donde los $\mathbb{F}'$-puntos racionales en E son los puntos (x, y) que satisfacen la ecuación de la curva y cuyas coordenadas x y y pertenecen a $\mathbb{F}'$. El punto al infinito es considerado como un $\mathbb{F}'$-*punto racional* para todas las extensiones $\mathbb{F}'$ de $\mathbb{F}$.

4.2. Ley de grupo

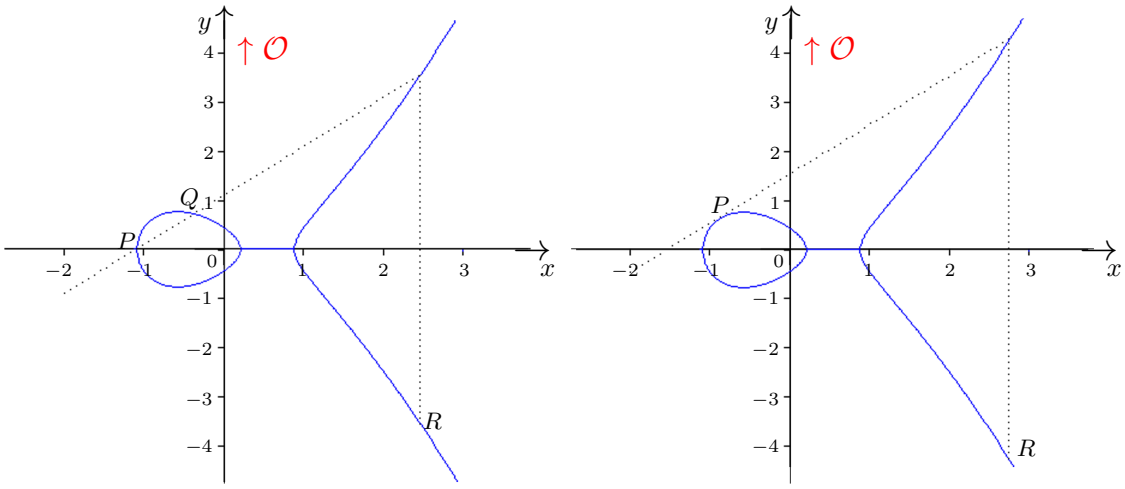
Dada la curva elíptica $E/\mathbb{F}$, el conjunto de $\mathbb{F}$-puntos racionales forma un grupo abeliano de notación aditiva cuyo elemento identidad corresponde al punto al infinito $\mathcal{O}$. Usualmente se utiliza la notación $E(\mathbb{F})$ para denotar dicho grupo.

Suma de puntos

La forma más común para explicar la operación suma es mediante su interpretación geométrica. Dada la curva elíptica $E/\mathbb{F}$ y los puntos $P = (x_1, y_1)$ y $Q = (x_1, y_1)$ donde $P, Q \in E(\mathbb{F})$, la operación $R = P + Q$ se define como:

1. Si $P \neq Q$, se traza una línea a través de los puntos P y Q hasta que la línea interseque un tercer punto en la curva elíptica, el cual se refleja sobre el eje de las abscisas, obteniendo el punto R, tal y como se representa en la figura 4.2(a)

2. Si $P = Q$, se traza una línea tangente al punto P hasta que la línea interseque un tercer punto, el cual se refleja sobre el eje x, obteniendo el punto R. Esta operación se conoce como *doblado de punto* y se ejemplifica en la figura 4.2(b)



(a) $P \neq Q$ (b) $P = Q$

Figura 4.2: Suma de puntos sobre $\mathbb{R}$

Ley de grupo

Dada la curva elíptica $E/\mathbb{F}$, donde $E : y^2 = x^3 + ax + b$, la ley de grupo se define de la siguiente manera:

(I) **Elemento Identidad.** $P + \mathcal{O} = \mathcal{O} + P = P$ para $P \in E(\mathbb{F})$.

(II) **Negativos**. Si $P = (x, y)$, $P \in E(\mathbb{F})$, entonces $(x, y) + (x, -y) = \mathcal{O}$. El punto $(x, -y)$ se denota como $-P$ y se denomina *negativo* de P. Cabe destacar que $-P$ es un punto que pertenece a $E(\mathbb{F})$.

(III) **Suma de Puntos**. Sea $P = (x_1, y_1)$, $Q = (x_2, y_2)$ donde $P, Q \in E(\mathbb{F})$ y $P \neq \pm Q$, entonces $P + Q = (x_3, y_3)$ tal que:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{y} \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \tag{4.3}$$

(IV) **Doblado de Puntos**. Sea $P = (x_1, y_1)$, $P \in E(\mathbb{F})$ donde $P \neq -P$, se define $2P = (x_3, y_3)$ tal que:

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{y} \quad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \tag{4.4}$$

Podemos deducir las ecuaciones de la suma y doblado de puntos de la siguiente manera: sea $E : y^2 = x^3 + ax + b$ la ecuación de la curva y sea $P = (x_1, y_1)$, $Q = (x_2, y_2)$ dos puntos sobre la curva elíptica tales que $Q \neq \pm P$. Podemos entonces obtener la ecuación de la recta que pasa por los dos puntos $l_{P,Q}$ como sigue:

$$l_{P,Q} : y = m(x - x_1) + y_1 \quad \text{donde} \quad m = \frac{x_2 - x_1}{y_2 - y_1} \tag{4.5}$$

Ahora bien, como se observa en la figura 4.2a, en la suma de puntos la línea $l_{P,Q}$ cruza por un tercer punto $R' = (x_3, y_3)$, por lo que podemos sustituir y de la ecuación (4.5) en la ecuación de la curva:

$$(m(x - x_1) + y_1)^2 = x^3 + ax + b$$

Desarrollando la ecuación tenemos:

$$x^3 - m^2x^2 + (a + 2x_1m^2 - 2y_1m)x + b - x_1^2m^2 + 2x_1y_1m - y_1^2 = 0 \tag{4.6}$$

Ahora bien, para un polinomio de grado 3 se satisface la siguiente igualdad:

$$(x - a)(x - b)(x - c) = x^3 - (a + b + c)x^2 + (ac + ab + bc)x + abc \tag{4.7}$$

donde a, b, c son las raíces del polinomio.

Dado que conocemos dos de las raíces, las cuales corresponden a los dos primeros puntos, el problema se resume a encontrar la tercera raíz del polinomio, por lo que podemos deducir a partir de las ecuaciones (4.6) y (4.7) que $m^2 = (x_1 + x_2 + x_3)$. Si despejamos el punto x_3 y sustituimos en la ecuación (4.5) obtenemos el punto R'. Finalmente si reflejamos este punto, obtenemos las ecuaciones mostradas en la ley de grupo.

Finalmente, para el caso del doblado de puntos tenemos que $P = Q$ (véase figura 4.2b), por lo que la pendiente m de la ecuación de la recta tangente $l_{P,P}$, se calcula a través de la derivada de la ecuación de la curva elíptica evaluada en el punto P.

4.3. Espacio proyectivo

Las fórmulas descritas en la sección 4.2 muestran que para la suma de puntos se requiere de una inversión, lo cual aritméticamente resulta ser costoso hablando de campos finitos, es por ello que una manera de ahorrar operaciones es a través de la representación de los puntos en el espacio proyectivo.

4.3.1. Coordenadas proyectivas

Sea $\mathbb{F}$ un campo y c, d dos enteros positivos. Se define la relación de equivalencia $\sim$ en el conjunto $\mathbb{F}^3 \setminus \{(0, 0, 0)\}$ como:

$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2)$ si $X_1 = \lambda^c X_2$, $Y_1 = \lambda^d Y_2$ y $Z_1 = \lambda Z_2$ para algún $\lambda \in \mathbb{F}^*$.

La clase de equivalencia que contiene a $(X, Y, Z) \in \mathbb{F}^3 \setminus \{(0, 0, 0)\}$ es:

$$(X : Y : Z) = \{(\lambda^c X, \lambda^d Y, \lambda Z) : \ \lambda \in \mathbb{F}^*\}$$

donde $(X : Y : Z)$ es el punto proyectivo, mientras que (X, Y, Z) se denomina como el representativo de $(X : Y : Z)$. El conjunto de todos los puntos proyectivos se denota como $\mathbb{P}(\mathbb{F})$. En particular, si $Z = 1$, se tiene que $(X/Z^c, Y/Z^d, 1)$ es el representativo del punto $(X : Y : Z)$, el cual es el único punto representativo con coordenada $Z = 1$. Por tanto, se tiene una correspondencia $1-1$ entre el conjunto de puntos proyectivos:

$$\mathbb{P}(\mathbb{F})^* = \{(X : Y : Z) \ : \ X, Y, Z \in \mathbb{F}, \ Z \neq 0\}$$

y el conjunto de *puntos afines*:

$$\mathbb{A}(\mathbb{F})^* = \{(x, y) \ : \ x, y \in \mathbb{F}\}$$

El conjunto de puntos proyectivos

$$\mathbb{P}(\mathbb{F})^0 = \{(X : Y : Z) \ : \ X, Y, Z \in \mathbb{F}, \ Z = 0\}$$

es denominado *línea al infinito* dado que estos puntos no corresponden a ninguno de los puntos afines.

La forma proyectiva de la ecuación de Weierstrass (4.2) de una curva elíptica $E/\mathbb{F}$ puede obtenerse reemplazando x por X/Z^c y y por Y/Z^d, y posteriormente reducir denominadores. Los puntos $\mathbb{P}(\mathbb{F})^*$ satisfacen la ecuación proyectiva de E mientras que el conjunto de puntos $\mathbb{P}(\mathbb{F})^0$ corresponden al punto al infinito $\mathcal{O}$.

4.3.2. Coordenadas jacobianas

Las coordenadas jacobianas son una particularización de las coordenadas proyectivas donde $c = 2$ y $d = 3$. Con las coordenadas jacobianas la curva E está dada por la ecuación:

$$Y^2 = X^3 + AXZ^4 + BZ^6$$

Un punto $(X_1 : Y_1 : Z_1)$ en E corresponde a un punto $(X_1/Z_1^2, Y_1/Z_1^3)$ en coordenadas afines, con $Z_1 \neq 0$. Por tanto, para pasar un punto (x, y) en coordenadas afines a coordenadas jacobianas solamente se debe agregar la coordenada $Z = 1$.

El punto al infinito $\mathcal{O}$ es representado como $(1, 1, 0)$, mientras que la negación de un punto $(X_1 : Y_1 : Z_1)$ es $(X_1 : -Y_1 : Z_1)$.

Doblado de puntos

Para realizar el doblado de puntos en coordenadas jacobianas se realiza una sustitución de x por X/Z^2 y y por Y/Z^3 en la ecuación (4.4). Por tanto, dado un punto $P = (X_1 : Y_1 : Z_1)$, el punto $2P = (X_3 : Y_3 : Z_3)$ se obtiene a través de las siguientes ecuaciones [Hankerson et al., 2004]:

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned}$$

El algoritmo para el doblado de puntos tiene un costo de seis elevaciones al cuadrado y tres multiplicaciones como se muestra en el Algoritmo 4.1

Algoritmo 4.1 Doblado de punto en coordenadas jacobianas

Entrada: $P = (X_1 : Y_1 : Z_1)$
Salida: $2P = (X_3 : Y_3 : Z_3)$
1: $t_1 \leftarrow Y_1^2$
2: $t_2 \leftarrow 4X_1 \cdot t_1$
3: $t_3 \leftarrow 8t_1^2$
4: $t_4 \leftarrow 3X_1^2 + aZ_1^4$
5: $X_3 \leftarrow t_4^2 - 2t_2$
6: $Y_3 \leftarrow t_4 \cdot (t_2 - X_3) - t_3$
7: $Z_3 \leftarrow 2Y_1 \cdot Z_1$
8: **return** $(X_3 : Y_3 : Z_3)$

Suma de puntos

Para la suma de puntos se realiza un método similar al doblado de puntos en cuanto a sustitución en la ecuación (4.3). Sin embargo, la forma más eficiente para realizar la suma es por medio del uso de coordenadas mixtas, es decir, sumar coordenadas jacobianas con coordenadas afines ($Z = 1$).

Sea $P = (X_1 : Y_1 : Z_1)$ con $Z_1 \neq 0$ y $Q = (X_2 : Y_2 : 1)$, para $P \neq \pm Q$, se tiene que la suma $R = P + Q = (X_3 : Y_3 : Z_3)$ puede obtenerse como [Hankerson et al., 2004]:

$$\begin{aligned} X_3 &= (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)(X_1 + X_2Z_1^2) \\ Y_3 &= (Y_2Z_1^3 - Y_1)(X_1(X_2Z_1^2 - X_1)^2 - X_3) - Y_1(X_2Z_1^2 - X_1)^3 \\ Z_3 &= (X_2Z_1^2 - X_1)Z_1 \end{aligned}$$

El costo de la suma de puntos tiene un costo de tres elevaciones al cuadrado y ocho multiplicaciones como se muestra en el Algoritmo 4.2

4.4. Curvas elípticas sobre campos finitos

Sea p un número primo y $q = p^n$ donde $n \in \mathbb{Z}^+$, dado un campo finito $\mathbb{F}_q$ de característica p, los $\mathbb{F}_p$-puntos racionales de una curva elíptica forman un grupo finito $E(\mathbb{F}_q)$, tal que para todo $P = (x_P, y_P)$, $P \in E(\mathbb{F}_q)$, $x_P, y_P \in \mathbb{F}_q$.

Algoritmo 4.2 Suma de puntos en coordenadas mixtas

Entrada: $P = (X_1 : Y_1 : Z_1)$ y $Q = (X_2 : Y_2 : 1)$
Salida: $R = P + Q = (X_3 : Y_3 : Z_3)$

1: $t_1 \leftarrow Z_1^2$
2: $t_2 \leftarrow Z_1 \cdot t_1$
3: $t_3 \leftarrow X_2 \cdot t_1$
4: $t_4 \leftarrow Y_2 \cdot t_2$
5: $t_5 \leftarrow t_3 - X_1$
6: $t_6 \leftarrow t_4 - Y_1$
7: $t_7 \leftarrow t_5^2$
8: $t_8 \leftarrow t_7 \cdot t_5$
9: $t_9 \leftarrow X_1 \cdot t_7$
10: $X_3 \leftarrow t_6^2 - (t_8 + 2t_9)$
11: $Y_3 \leftarrow t_6 \cdot (t_9 - X_3) - Y_1 \cdot t_8$
12: $Z_3 \leftarrow Z_1 \cdot t_5$
13: **return** $(X_3 : Y_3 : Z_3)$

La figura 4.3 muestra la curva elíptica $E : y^2 = x^3 + x + 1$ definida sobre el campo $\mathbb{F}_{17}$. Como se observa en la gráfica, cada una de las coordenadas de los puntos pertenecen al campo finito $\mathbb{F}_{17}$.

4.4.1. Orden de la curva

Sea E una curva elíptica definida sobre el campo finito $\mathbb{F}_q$, el número de puntos de $E(\mathbb{F}_q)$, denotado como $\#E(\mathbb{F}_q)$ se denomina como *orden de la curva* E sobre $\mathbb{F}_q$. Los límites del orden de la curva están definidos por el teorema de Hasse [Hankerson et al., 2004]:

Teorema 4.1. *(Hasse). Sea E una curva elíptica definida sobre $\mathbb{F}_q$, entonces*

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

El intervalo $[\, q+1-2\sqrt{q},\ q+1+2\sqrt{q}\,]$ se denomina *intervalo de Hasse*. Una alternativa al teorema de Hasse es que si E está definida sobre $\mathbb{F}_q$, entonces $\#E(\mathbb{F}_q) = q+1-t$, donde $t \leq 2\sqrt{q}$ se define como la *traza* de $E/\mathbb{F}_q$. Dado que $2\sqrt{q}$ es relativamente pequeño con respecto a q, tenemos que $\#E(\mathbb{F}_q) \approx q$.

4.4.2. Orden de un punto

Sea $E/\mathbb{F}_q$ una curva elíptica y $P \in E(\mathbb{F}_q)$ un punto de la curva elíptica. El orden del punto P es el menor entero positivo k tal que $kP = \mathcal{O}$.

La operación kP es conocida como *multiplicación escalar de punto* la cual se detalla en la sección 4.6. Es importante destacar que el orden de un punto siempre divide

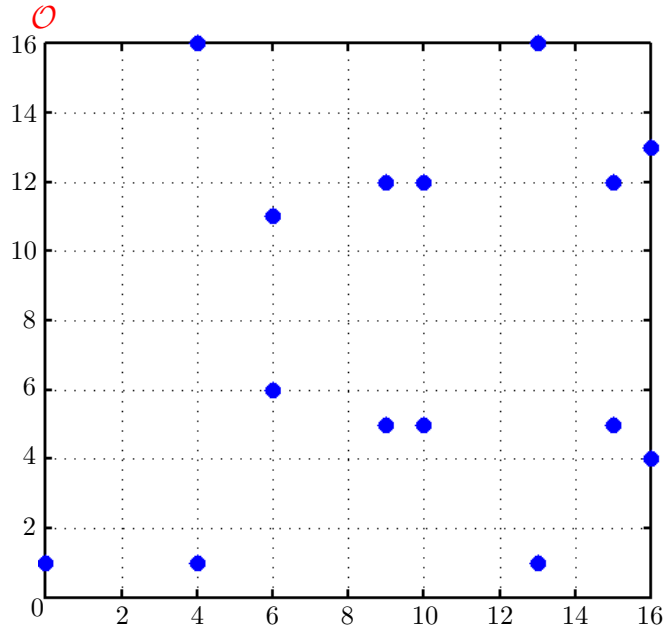


Figura 4.3: Curva elíptica $E(\mathbb{F}_{17})$

al orden de la curva, en virtud del teorema de Lagrange.

4.4.3. Puntos de torsión

Dada una curva elíptica $E/\mathbb{F}_p$, sea $\bar{\mathbb{F}}_p$ la cerradura algebráica de $\mathbb{F}_p$ y $r \in \mathbb{Z}^+$. Se define el conjunto de puntos de torsión r de $E(\bar{\mathbb{F}}_p)$, denotado como $E(\bar{\mathbb{F}}_p)[r]$, como el conjunto de puntos en $E(\bar{\mathbb{F}}_p)$ cuyo orden es r:

$$E(\bar{\mathbb{F}}_p)[r] = \{P \in E(\bar{\mathbb{F}}_p) \mid rP = \mathcal{O}\}.$$

Es importante destacar que para propósitos criptográficos r se toma como un factor primo de $E(\bar{\mathbb{F}}_p)$ suficientemente grande para alcanzar cierto nivel de seguridad.

4.4.4. Grado de encajamiento

Sean dos números primos p y r, dada la curva elíptica $E/\mathbb{F}_p$ donde $\#E(\mathbb{F}_p) = h \cdot r$ y $h \in \mathbb{Z}^+$. Se define el *grado de encajamiento* de $E/\mathbb{F}_p$ con respecto a p y r, como el menor entero positivo k tal que:

$$r \mid (p^k - 1)$$

Sea $\Phi_k(p)$ el k-ésimo polinomio ciclotómico, se cumple que $\Phi_k(p) \mid p^k - 1$ (Definición 2.9) y por tanto $r \mid \Phi_k(p)$. Dado que $p \equiv t - 1 \bmod r$, donde t es la traza de E sobre $\mathbb{F}_p$, el grado de encajamiento es también el menor entero positivo k tal que:

$$r \mid \Phi_k(t - 1)$$

4.4.5. Curva enlazada

Definición 4.2. Invariante-j. *Dada la curva elíptica* $E : y^2 = x^3 + ax + b$, *el invariante-j de* E, *denotado como* $j(E)$ *se define como:*

$$j(E) = -1728\frac{(4a)^3}{\Delta}$$

donde $\Delta = -16(4a^3+27b^2)$ *es el discriminante de la curva y el invariante-j determina el isomorfismo de* E.

Definición 4.3. Curva enlazada. *Sean* E *y* E' *dos curvas elípticas, se dice que* E' *es la curva enlazada de* E *si y sólo si ambas curvas tienen el mismo invariante-j y son isomorfas sobre la cerradura algebraica de un campo finito* $\mathbb{F}_p$.

En particular, dada una curva elíptica $E/\mathbb{F}_p$ con grado de encajamiento k, si el grupo $E(\mathbb{F}_p)$ tiene un subgrupo de orden primo r, existe una curva enlazada E' de E definida sobre el campo $\mathbb{F}_{p^{k/d}}$ donde $d|k$ y $r|\#E'(\mathbb{F}_{p^{k/d}})$ tal que las curvas E y E' son isomorfas sobre $\mathbb{F}_{p^k}$, es decir:

$$\phi : E'(\mathbb{F}_{p^{k/d}}) \longrightarrow E(\mathbb{F}_{p^k})$$

donde $d \in \mathbb{Z}^+$ se define como el grado de la curva enlazada E' [Hess et al., 2006].

4.4.6. Endomorfismo de Frobenius

Sea $E/\mathbb{F}_p$ una curva elíptica con grado de encajamiento k y sea $E(\mathbb{F}_p)$ el grupo de los $\mathbb{F}_p$-puntos racionales en la curva, tal que $E(\mathbb{F}_p)$ tiene un subgrupo de orden primo r. El endormorfismo de Frobenius se define como:

$$\pi : E(\mathbb{F}_{p^k}) \to E(\mathbb{F}_{p^k}),$$

tal que:

$$\pi(X, Y) = (X^p, Y^p) \in E(\mathbb{F}_{p^k})$$

Sea t la traza de E sobre $\mathbb{F}_p$, $\sigma(u) = u^2 - tu + p$ el polinomio característico del endomorfismo de Frobenius, es decir, para cualquier $Q \in E(\mathbb{F}_{p^k})$ se tiene que:

$$\pi^2(Q) - t\pi(Q) + pQ = \mathcal{O}.$$

Si $\sigma(u)$ se factoriza módulo r entonces:

$$\sigma(u) = (u - 1)(u - p) \bmod r$$

Por tanto existen dos conjuntos en $E(\mathbb{F}_{p^k})[r]$ definidos como $\{P \in E(\mathbb{F}_{p^k}) \mid \pi(P) = P\}$ y $\{Q \in E(\mathbb{F}_{p^k}) \mid \pi(Q) = pQ\}$ [Barreto et al., 2003]. En este sentido, el grupo $E(\mathbb{F}_p)$ corresponde al primer conjunto, ya que para cualquier punto $P = (x, y) \in E(\mathbb{F}_p)$ se cumple que $(x, y) = (x^p, y^p)$.

4.5. Curvas amables con los emparejamientos

Sea E una curva elíptica ordinaria definida sobre un campo finito $\mathbb{F}_p$. Se dice que E es *amable con los emparejamientos* [Freeman et al., 2010] si:

- Existe un número r tal que $r \geq \sqrt{p}$ y $r \mid \#E(\mathbb{F}_p)$.
- El grado de encajamiento k de la curva $E/\mathbb{F}_p$ con respecto a r es menor que $log_2(r)/8$.

Este tipo de curvas son construidas a través del método de multiplicación compleja (CM) [Goldwasser and Kilian, 1999], en el cual se fija el grado de encajamiento k y posteriormente se computan los enteros p, r y t, los cuales satisfacen las siguientes condiciones:

1. p es un número primo.
2. r es un número primo tal que $r \geq \sqrt{p}$.
3. t es primo relativo con p.
4. r divide a $p + 1 - t$
5. k es el menor entero positivo tal que $r|\Phi_k(t-1)$
6. Para $D \in \mathbb{Z}^+$ y $f \in \mathbb{Z}$ se satisface la ecuación:

$$4p - t^2 = Df^2 \tag{4.8}$$

   La cual garantiza que $t \leq 2\sqrt{p}$ y donde D es el discriminante CM.

Curvas BN

Las curvas BN (Barreto-Naehrig) son una familia de curvas elípticas ordinarias amables con los emparejamientos, estas curvas están construidas sobre un campo finito $\mathbb{F}_p$. Las curvas BN se definen por la ecuación $E : y^2 = x^3 + b$ con $b \neq 0$.

Las curvas BN definen curvas de orden primo r, es decir $\#E(\mathbb{F}_p) = r$, mientras que su grado de encajamiento es $k = 12$. La característica del campo p, el orden del grupo r y la traza de Frobenius t se encuentran parametrizados por los siguientes polinomios:

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

donde z es cualquier entero tal que $p(z)$ y $r(z)$ son números primos.

Las curvas BN cumplen con la ecuación (4.8), donde $D = 3$ y f está parametrizado como $f(z) = 6z^2 + 4z + 1$. En esta familia, la curva $E/\mathbb{F}_p : y^2 = x^3 + b$ es isomorfa a la curva enlazada de grado $d = 6$, definida como $E'/\mathbb{F}_{p^2} : Y^2 = X^3 + b/\xi$, donde $b \in \mathbb{F}_p$ y $\xi \in \mathbb{F}_{p^2}$ no tienen residuos cuadráticos ni cúbicos en $\mathbb{F}_p$ y $\mathbb{F}_{p^2}$ respectivamente.

4.6. Multiplicación escalar

La operación de multiplicación se lleva a cabo mediante un escalar k y un punto $P \in E$, denotada por kP, lo cual consiste en repetir k veces la operacion de adición sobre P.

$$kP = \underbrace{P + P + ... + P}_{k\ veces}$$

Una forma de llevar a cabo la multiplicación escalar es a través de una adaptación del *algoritmo de Horner*. En este caso, se representa el escalar k de forma binaria, el cual puede ser visto como un polinomio evaluado en 2:

$$k = k_0 2^0 + k_1 2^1 + k_2 2^2 + \ldots + k_{n-1} 2^{n-1}$$

donde $k_i \in [0, 1]$ y $n = |k|$.

Para obtener la multiplicación escalar, se recorre cada uno de los bits en k realizando el doblado de punto en cada paso y aplicando la suma de puntos si $k_i = 1$, como se muestra en el Algoritmo 4.3. Este algoritmo también es conocido como método binario de izquierda a derecha.

Algoritmo 4.3 Método binario de izquierda a derecha para multiplicación por un escalar

Entrada: $k = (k_0, k_1, ..., k_{n-1})$ y $P \in E(\mathbb{F}_p)$
Salida: $Q = kP$

```
1: Q ← O
2: for i = n − 1 → 0 do
3:     Q ← 2Q
4:     if k_i = 1 then
5:         Q ← Q + P
6:     end if
7: end for
8: return Q
```

Este método es el más sencillo para la realización de la multiplicación escalar, ejecutarlo requiere en promedio de n doblados y $n/2$ sumas (considerando que los

bits del escalar están uniformemente distribuidos).

Existen otros métodos que a cambio de cierto nivel de precómputo, reducen el número de sumas y doblados requeridos, tales como el método de ventana ω-NAF y el método *comb* [Hankerson et al., 2004]; otros métodos en cambio, aprovechan las propiedades de la curva elíptica para reducir el número de operaciones, tales como el método GLV y el método GS.

4.6.1. Método de ventana ω-NAF

Definición 4.4. Representación ω-NAF. *Sea $\omega \geq 1$ un entero positivo. La representación ω-NAF de un entero positivo k es una expresión $k = \sum_{i=0}^{l-1} k_i 2^i$, donde cada coeficiente diferente de cero k_i es un número impar, $|k_i| \leq 2^{\omega-1}$, $k_{l-1} \neq 0$. Así mismo, cumple que al menos uno de los ω coeficientes consecutivos es diferente de cero. El tamaño del ω-NAF es l.*

Para todo entero positivo k existe una única representación ω-NAF, la cual se denota como $NAF_\omega(k)$ y su tamaño es de a lo más un dígito mayor que en la representación binaria de k. Así mismo, la densidad de elementos diferentes de cero con respecto al tamaño l de la representación ω-NAF es $1/(\omega + 1)$

Ejemplo 4.1. [Hankerson et al., 2004] *Dado $k = 1122334455$, la representación binaria de k y las representaciones ω-NAF de k para $2 \leq \omega \leq 6$ son:*

$$
\begin{aligned}
(k)_2 &= 1\,0\,0\,0\;\;0\,1\,0\,1\,1\;\;1\,0\,0\,1\,0\,1\;\;0\,1\,1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,1\,1\,1\\
NAF_2(k) &= 1\,0\,0\,0\;\;1\,0\,\bar{1}\,0\,0\;\;\bar{1}\,0\,1\,0\,\bar{1}\,0\;\;\bar{1}\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\,0\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\\
NAF_3(k) &= 1\,0\,0\,0\;\;0\,0\,3\,0\,0\;\;\bar{1}\,0\,0\,1\,0\,0\;\;3\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\,0\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\\
NAF_4(k) &= 1\,0\,0\,0\;\;0\,1\,0\,0\,0\;\;7\,0\,0\,0\,0\,5\;\;0\,0\,0\,7\,0\,0\,0\,7\,0\,0\,0\,\bar{1}\,0\,0\,0\,7\\
NAF_5(k) &= 1\,0\,0\,0\,0\,\bar{15}\,0\,0\,0\,0\;\;\bar{9}\,0\,0\,0\,0\,0\,11\,0\,0\,0\,0\,0\,0\,\bar{9}\,0\,0\,0\,0\,0\,0\,0\,\bar{9}\\
NAF_6(k) &= 1\,0\,0\,0\;\;0\,0\,0\,0\,0\,23\,0\,0\,0\,0\,0\,11\,0\,0\,0\,0\,0\,0\,\bar{9}\,0\,0\,0\,0\,0\,0\,0\,\bar{9}
\end{aligned}
$$

A través del Algoritmo 4.4 puede obtenerse el $NAF_\omega(k)$ de una forma eficiente. En este algoritmo, k mods 2^ω denota el entero u cuyo valor satisface $u \equiv k \pmod{2^\omega}$ y donde $-2^{\omega-1} \leq u < 2^{\omega-1}$. Los dígitos del $NAF_\omega(k)$ son obtenidos dividiendo continuamente el entero k por 2, lo que permite la obtención del residuo r en el intervalo $[-2^{\omega-1}, 2^{\omega-1} - 1]$. Si k es impar y el residuo $r = k$ mods 2^ω es seleccionado, entonces $(k - r)/2$ será un número divisible por $2^{\omega-1}$, asegurando que los siguientes $\omega - 1$ dígitos sean cero.

El Algoritmo 4.5 realiza la multiplicación escalar por medio de una modificación del Algoritmo 4.3 usando la representación $NAF_\omega(k)$ en vez de la representación binaria de k. El costo de este algoritmo es:

$$\left[1D + (2^{\omega-2} - 1)A\right] + \left[\frac{m}{\omega + 1}A + mD\right] \tag{4.9}$$

donde $m = |k|$, D representa la operación doblado de puntos y A la operación suma de puntos.

Algoritmo 4.4 Obtención del ω-NAF de un entero positivo

Entrada: Tamaño de representación ω, entero positivo k
Salida: $NAF_\omega(k)$

1: $i \leftarrow 0$
2: **while** $k \geq 1$ **do**
3: **if** k es impar **then**
4: $k_i \leftarrow k$ mods 2^ω
5: $k \leftarrow k - k_i$
6: **else**
7: $k_i \leftarrow 0$
8: **end if**
9: $k \leftarrow k/2$
10: $i \leftarrow i + 1$
11: **end while**
12: **return** $(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$

Algoritmo 4.5 Multiplicación escalar: método ω-NAF

Entrada: Tamaño de representación ω, entero positivo k, $P \in E(\mathbb{F}_q)$
Salida: kP

1: Obtenga $NAF_\omega(k) = \sum_{i=0}^{l-1} k_i 2^i$ (Algoritmo 4.4)
2: Obtenga $P_i = iP$ para $i \in [1, 3, 5, \ldots, 2^{\omega-1} - 1]$
3: $Q \leftarrow \mathcal{O}$
4: **for** $i = l - 1 \rightarrow 0$ **do**
5: $Q \leftarrow 2Q$
6: **if** $k_i \neq 0$ **then**
7: **if** $k_i > 0$ **then**
8: $Q \leftarrow Q + P_{k_i}$
9: **else**
10: $Q \leftarrow Q - P_{-k_i}$
11: **end if**
12: **end if**
13: **end for**
14: **return** Q

En la ecuación (4.9), el primer elemento $[1D + (2^{\omega-2} - 1)A]$ corresponde al costo de precomputar los puntos P_i en el Algoritmo 4.5 mientras que el segundo elemento $\left[\frac{m}{\omega+1}A + mD\right]$ representa el número de operaciones dentro del ciclo de la línea 4. Como se observa, el costo para obtener los puntos P_i tiende a crecer exponencialmente,

por lo que se debe seleccionar un valor de ω que permita obtener un costo óptimo en las operaciones.

Una observación importante de este método es que si se conoce previamente el punto P, la obtención de los valores P_i puede realizarse previamente, por lo que el costo de la multiplicación estaría dado únicamente por $\left[\frac{m}{\omega+1}A + mD\right]$.

4.6.2. Método comb

El método *comb* es un método que reduce el costo de operaciones de manera significativa. Este método usualmente se realiza cuando la multiplicación escalar actúa sobre un punto previamente conocido, ya que requiere de cierto nivel de precómputo.

Para $k \in \mathbb{Z}^+$, sea $t = |k|$ y $d = \lceil t/\omega \rceil$, donde ω es el tamaño de ventana. El método comb (Algoritmo 4.6) funciona de la siguiente manera:

1. Se rellena con $d\omega - t$ ceros a la izquierda a la representación binaria del escalar k, de manera que $|k| = \omega d$

2. Se divide el escalar k en ω cadenas de bits, cada una con tamaño d, de tal forma que:
$$k \;=\; K^{\omega-1} \parallel \ldots \parallel K^1 \parallel K^0$$

3. Se escriben las cadenas de bits K^j de forma matricial:
$$\begin{bmatrix} K^0 \\ \vdots \\ K^{\omega'} \\ \vdots \\ K^{\omega-1} \end{bmatrix} = \begin{bmatrix} K^0_{d-1} & \ldots & K^0_0 \\ \vdots & & \vdots \\ K^{\omega'}_{d-1} & \ldots & K^{\omega'}_0 \\ \vdots & & \vdots \\ K^{\omega-1}_{d-1} & \ldots & K^{\omega-1}_0 \end{bmatrix} = \begin{bmatrix} k_{d-1} & \ldots & k_0 \\ \vdots & & \vdots \\ k_{(\omega'+1)d-1} & \ldots & k_{\omega' d} \\ \vdots & & \vdots \\ k_{\omega d-1} & \ldots & k_{(\omega-1)d} \end{bmatrix}$$

4. Se procesa cada columna del escalar a la vez

5. Para obtener una mayor aceleración, los puntos:
$$[a_{\omega-1}, \ldots, a_2, a_1, a_0]P = a_{\omega-1}2^{(\omega-1)d}P + \ldots + a_2 2^{2d}P + a_1 2^d P + a_0 P$$
son precalculados para cualquiera de las combinaciones de $(a_{\omega-1}, \ldots, a_2, a_1, a_0)$.

El costo del Algoritmo 4.6 es:
$$\left(\frac{2^\omega - 1}{2^\omega}d - 1\right)A + (d-1)D$$
donde A es el número de sumas y D el número de doblados de punto. Para $\omega \geq 2$, este algoritmo toma aproximadamente el mismo número de doblados y sumas.

Algoritmo 4.6 Multiplicación escalar: método *comb*

Entrada: Tamaño de ventana ω, entero positivo k, $P \in E(\mathbb{F}_q)$
Salida: kP

1: *Precómputo.* Obtenga $[a_{\omega-1}, \ldots, a_2, a_1, a_0]P$ para todas las combinaciones de bits $(a_{\omega-1}, \ldots, a_2, a_1, a_0)$ de tamaño ω
2: Rellene la representación binaria de k con 0's a la izquierda en caso de ser necesario y escriba $k = K^{\omega-1} \parallel \ldots \parallel K^1 \parallel K^0$, donde K^j es una cadena de bits de longitud d. Represente K_i^j al bit i del elemento K^j
3: $Q \leftarrow \mathcal{O}$
4: **for** $i = (d-1) \rightarrow 0$ **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad Q \leftarrow Q + [K_i^{\omega-1}, \ldots, K_i^1, K_i^0]P$
7: **end for**
8: **return** Q

Ejemplo 4.2. *Para $\omega = 3$, $k = 778 = (1100001010)_2$ y $P \in E(\mathbb{F}_q)$. De acuerdo al Algoritmo 4.6, el punto $kP = 778P$ se obtiene de la siguiente manera:*

1. *Se obtienen los puntos: $[a_2, a_1, a_0]P$ para todas las combinaciones de (a_2, a_1, a_0):*

$$\begin{array}{rclrcl} [0\ 0\ 0] & = & \mathcal{O} & [1\ 0\ 0] & = & 256P \\ [0\ 0\ 1] & = & P & [1\ 0\ 1] & = & 257P \\ [0\ 1\ 0] & = & 16P & [1\ 1\ 0] & = & 272P \\ [0\ 1\ 1] & = & 18P & [1\ 1\ 1] & = & 273P \end{array}$$

2. *Rellenamos k con ceros a la izquierda y separamos en $\omega = 3$ cadenas de bits: $k = 0011 \parallel 0000 \parallel 1010$, donde $K^2 = 0011$, $K^1 = 0000$ y $K^0 = 1010$.*

3. *Realizamos las iteraciones del ciclo principal para $d = 4$ y $Q = \mathcal{O}$*

   - *Iteración 1 ($i = 3$).* $Q = 2Q + [0\ 0\ 1]P = \mathcal{O} + P = P$
   - *Iteración 2 ($i = 2$).* $Q = 2Q + [0\ 0\ 0]P = 2P + \mathcal{O} = 2P$
   - *Iteración 3 ($i = 1$).* $Q = 2Q + [1\ 0\ 1]P = 4P + 257P = 261P$
   - *Iteración 4 ($i = 0$).* $Q = 2Q + [1\ 0\ 0]P = 522P + 256P = 778P$

4. *Regresamos el valor de $Q = 778P$.*

Para el caso de este trabajo y dadas las características del procesador, se optó por seleccionar $\omega = 8$, ésto debido a la facilidad en la manipulación del escalar k.

4.6.3. Método GLV

Gallant, Lambert and Vanstone [Gallant et al., 2001] introdujeron un método (denominado GLV) para acelerar la multiplicación escalar kP en $E(\mathbb{F}_p)[r]$ tomando

ciertas propiedades de la curva elíptica. Este método en su forma más simple funciona si, dado un punto P, puede tenerse conocimiento de un múltiplo no trivial de P. Esta información está disponible si existe un endomorfismo ψ eficientemente computable sobre $E/\mathbb{F}_p$ tal que $\psi(P) = \lambda P$. Por lo que es posible obtener kP de manera eficiente escribiendo $k \equiv k_0 + k_1\lambda \pmod{r}$ con $|k_i| < \sqrt{r}$ y realizando la doble multiplicación $k_0P + k_1\psi(P)$.

Este método requiere de un cierto nivel de precómputo y almacenamiento, sin embargo, su eficiencia recae en que emplea únicamente la mitad de doblados de punto que en el método binario. Así mismo, este método puede combinarse con algún método de ventana (ej. ω-NAF) para reducir el número de adiciones de punto como se muestra en el Algoritmo 4.7. El costo de este algoritmo es:

$$\left[2D + (2^{\omega-1} - 2)A\right] + \left[\frac{m}{\omega+1}A + \frac{m}{2}D\right]$$

donde $m = |k|$. En este caso, el primer sumando corresponde a la obtención de los puntos P_i y Q_i, mientras que el segundo término corresponde al costo del ciclo principal.

Algoritmo 4.7 Multiplicación escalar: método GLV

Entrada: Tamaño de ventana ω, entero positivo k, $P \in E(\mathbb{F}_p)$, endomorfismo ψ sobre $E(\mathbb{F}_p)$.
Salida: kP

1: $Q \leftarrow \psi(P)$ $(= \lambda P)$
2: Descomponga $k = u + v\lambda$ donde $|u| = |v| = l$
3: Calcule $NAF_\omega(u) = \sum_{i=0}^{l-1} u_i 2^i$ y $NAF_\omega(v) = \sum_{i=0}^{l-1} v_i 2^i$ (Algoritmo 4.4)
4: Obtenga $P_i = iP$, $Q_i = iQ$ para $i \in [\pm 1, \pm 3, \pm 5, \ldots, \pm(2^{\omega-1} - 1)]$.
5: $R \leftarrow \mathcal{O}$
6: **for** $i = l - 1 \rightarrow 0$ **do**
7: $\quad R \leftarrow 2R$
8: $\quad$ **if** $u_i \neq 0$ **then**
9: $\quad\quad R \leftarrow R + P_{u_i}$
10: $\quad$ **end if**
11: $\quad$ **if** $v_i \neq 0$ **then**
12: $\quad\quad R \leftarrow R + Q_{v_i}$
13: $\quad$ **end if**
14: **end for**
15: **return** R

Descomposición del escalar k

Un punto importante del método GLV, es la descomposición del escalar k, o bien, obtener k_0, k_1, tales que $k \equiv k_0 + k_1\lambda \pmod{r}$ con $|k_i| < \sqrt{r}$.

Definición 4.5. Rejilla (Lattice). *Sea $v_1, \ldots, v_n \in R^m$ la base del espacio vectorial V, la rejilla L generada por dicha base, es el conjunto de vectores formados por la combinación lineal de $v_1, \ldots, v_n$ con coeficientes en $\mathbb{Z}$, es decir $L \subset V$.*

La tarea de descomponer k puede obtenerse al resolver el problema del vector más cercano en una rejilla (*closest vector problem CVP*). Este problema puede ser resuelto eficientemente utilizando el método de redondeo de Babai [Babai, 1986] teniendo la base de una rejilla LLL previamente computada. Para ello, se define la rejilla modular:

$$L = \left\{ x \in Z^m : \sum_{i=0}^{m-1} x_i \lambda^i \equiv 0 \pmod{r} \right\} \tag{4.10}$$

Los $2m$ vectores $(0, \ldots, 0, r, 0, \ldots, 0)$ y $(0, \ldots, 0, \lambda, -1, 0, \ldots, 0)$ generan la rejilla L si $\text{mcd}(\lambda, r) = 1$. Se ejecuta el algoritmo LLL [Lenstra et al., 1982] para obtener la nueva base de la rejilla. Dado el escalar k, se utiliza la técnica de Babai para encontrar el vector de rejilla $x = (x_0, \ldots, x_{m-1})$ cercano a $w = (n, 0, \ldots, 0)$. Si se define $u = w - x$, entonces por definición $\sum_{i=0}^{m-1} u_i \lambda^i \equiv k \pmod{r}$. Si la base LLL es suficientemente buena, entonces los coeficientes u_i serán tales que $|u_i| \approx r^{1/m}$.

Método GLV en curvas BN

Considere los siguientes parámetros BN:

$$t = 6z^2 + 1 \quad p = 36z^4 + 36z^3 + 24z^2 + 6z + 1 \quad r = p + 1 - t$$

Se construye la curva elíptica $E_1 : y^2 = x^3 + b$ definida sobre $\mathbb{F}_p$ con $\#E(\mathbb{F}_p) = r$. Sea $\beta \in \mathbb{F}_p$, entonces la proyección $\psi : E_1 \to E_1$ definida por $(x, y) \to (\beta x, y)$ y $\mathcal{O} \to \mathcal{O}$ es un endomorfismo definido sobre $\mathbb{F}_p$. Si $P \in E(\mathbb{F}_p)$ es un punto de orden primo r, entonces ψ actúa sobre $\langle P \rangle$ como una multiplicación por λ donde $\lambda^2 + \lambda \equiv -1 \pmod{r}$. Puede observarse entonces que $\psi(P)$ puede obtenerse a través de una sola multiplicación en $\mathbb{F}_p$.

Para la descomposición del escalar k, la base para la rejilla L de la ecuación (4.10) para $\lambda = -36z^4 + 1$ es:

$$\begin{pmatrix} 6z^2 + 2z & -2z - 1 \\ -2z - 1 & -6z^2 + 4z + 1 \end{pmatrix}$$

Para descomponer el entero k, se necesita encontrar el vector x cercano a $w = (n, 0)$ en la rejilla L, para ello, primero se obtiene el vector $v = wB^{-1}$ con lo que tenemos que:

$$wB^{-1} = \begin{pmatrix} \dfrac{k(6z^2 + 4z + 1)}{r} & \dfrac{-k(2z + 1)}{r} \end{pmatrix}$$

Entonces, podemos obtener v realizando multiplicaciones enteras y divisiones por r. Finalmente obtenemos el vector $u = w - vB$ cuyas entradas son los coeficientes k_0, k_1 de la descomposición de k.

Una mejora a la descomposición del escalar puede hacerse en la obtención del vector v. Como se observó, su cálculo requiere de multiplicaciones y divisiones por r. Una manera de evitar la división, es a través de un método inspirado en la reducción de Barret (véase apéndice A), en el que utilizando cierto nivel de precómputo se realizan divisiones por 2^m en lugar de r. Este método puede aplicarse debido a que v es una aproximación, lo que permite cierto grado de error. Sea el vector $w' = [2^m, 2^m]$ con $m \geq |r|$, se obtiene el vector $v' = w'B^{-1}$:

$$v' = w'B^{-1} = (v'_0, v'_1) = \left(\left\lfloor \frac{2^m(6z^2 + 4z + 1)}{r} \right\rfloor \quad \left\lfloor \frac{-2^m(2z + 1)}{r} \right\rfloor \right)$$

Dado que v' puede ser obtenido previamente, el vector v puede entonces obtenerse a través de multiplicaciones escalares y divisiones por 2^m:

$$v = \left(\left\lfloor \frac{kv'_0}{2^m} \right\rfloor, \left\lfloor \frac{kv'_1}{2^m} \right\rfloor \right)$$

Cabe aclarar que entre mayor sea el tamaño de m, mayor será la precisión del vector v.

4.6.4. Método GS

El método GS [Galbraith and Scott, 2008] puede considerarse como una versión del método GLV, el cual aprovecha el endomorfismo de Frobenius.

Sea E una curva elíptica definida sobre $\mathbb{F}_p$ con grado de encajamiento k y E' la curva enlazada de E de grado d, por definición se tiene que E y E' son isomorfas de manera que $\phi : E'(\mathbb{F}_{p^{k/d}}) \longrightarrow E(\mathbb{F}_{p^k})$. Dado π_p el operador de Frobenius en E, tenemos que $\psi = \phi^{-1}\pi_p\phi$ es un endormorfismo de E' tal que $\psi : E'(\mathbb{F}_{p^{k/d}}) \to E'(\mathbb{F}_{p^{k/d}})$. Además, para $Q \in E'(\mathbb{F}_{p^{k/d}})$, tenemos que $\psi^k(Q) = Q$, $\psi(Q) = pQ$ y $\Phi_k(\psi)(Q) = \infty$ donde Φ_k es el k-ésimo polinomio ciclotómico.

En el caso de las curvas BN donde:

$$t = 6z^2 + 1 \quad p = 36z^4 + 36z^3 + 24z^2 + 6z + 1 \quad r = p + 1 - t$$

Se puede construir una curva elíptica $E : y^2 = x^3 + b$ sobre $\mathbb{F}_p$ de orden r. El grado de encajamiento de esta curva es 12, por lo que se tiene que la curva enlazada E' de E se encuentra definida sobre $\mathbb{F}_{p^2}$. Se define entonces $\mathbb{G}_2$ como el subgrupo $E'(\mathbb{F}_{p^2})$ de orden r.

Tenemos que $\psi = \phi^{-1}\pi_p\phi$, el cual satisface a $\psi^4 - \psi^2 + 1 = 0$. Dado que ψ actúa como una multiplicación por p y $p \equiv t - 1 \pmod{r}$, sería natural descomponer $k = k_0 + k_1(t-1)$. Sin embargo $|t-1| \not\approx r^{1/m}$, por lo que para este caso se utiliza una reducción de rejilla.

De acuerdo con [Galbraith and Scott, 2008], la base reducida para la rejilla L de la ecuación (4.10) con $\lambda = t - 1 = 6z^2$ es:

$$B = \begin{pmatrix} z+1 & z & z & -2z \\ 2z+1 & -z & -(z+1) & -z \\ 2z & 2z+1 & 2z+1 & 2z+1 \\ z-1 & 4z+2 & -(2z-1) & z-1 \end{pmatrix}$$

donde B se obtiene a través del algoritmo LLL y cumple que $Bv = 0$ para el vector columna $v = (1, \lambda, \lambda^2, \lambda^3)$.

De la misma manera que en el método GLV, se necesita encontrar el vector x más cercano a $w = (k, 0, 0, 0)$ en la rejilla L. Se encuentra el vector $v \approx wB^{-1}$ como se muestra a continuación:

$$wB^{-1} = \begin{pmatrix} \frac{k(2z^2+3z+1)}{r} & \frac{k(12z^3+8z^2+z)}{r} & \frac{k(6z^3+4z^2+z)}{r} & \frac{k(-z^2-z)}{r} \end{pmatrix}$$

Finalmente se calcula el vector $u = w - vB$ cuyos elementos u_i corresponden a los coeficientes k_i de la descomposición de k. En este caso, k puede descomponerse en 4 coeficientes tales que $k \equiv \sum_{i=0}^{3} u_i\lambda^i$, donde, todas las entradas en el vector u satisfacen que $|u_i| < r^{1/4}$.

De la misma manera que en el método GLV, se pueden evitar las divisiones por r en el cálculo de v, para ello se precomputa el vector $v' = (v'_0, v'_1, v'_2, v'_3)$ y posteriormente se realizan divisiones entre 2^m con $m \geq |r|$, donde:

$$v' = \begin{pmatrix} \frac{2^m(2z^2+3z+1)}{r} & \frac{2^m(12z^3+8z^2+z)}{r} & \frac{2^m(6z^3+4z^2+z)}{r} & \frac{2^m(-z^2-z)}{r} \end{pmatrix}$$

y

$$v = \left(\left\lfloor \frac{kv'_0}{2^m} \right\rfloor, \left\lfloor \frac{kv'_1}{2^m} \right\rfloor, \left\lfloor \frac{kv'_2}{2^m} \right\rfloor, \left\lfloor \frac{kv'_3}{2^m} \right\rfloor \right)$$

.

El Algoritmo 4.8 describe el método GS, es de destacar que para curvas BN este algoritmo requiere únicamente de la cuarta parte de doblados de punto que en el algoritmo binario. Así mismo, se observa que se utiliza el método de ventana ω-NAF para reducir así mismo el número de adiciones de punto. El costo de este algoritmo es:

$$\left[4D + 4(2^{\omega-2} - 1)A\right] + \left[\frac{m}{\omega+1}A + \frac{m}{4}D\right]$$

donde $m = |k|$, A es el número de adiciones de punto y D el número de doblados de punto.

Algoritmo 4.8 Multiplicación escalar: método GS

Entrada: Tamaño de ventana ω, entero positivo k, $Q \in E(\mathbb{F}_{p^2})$, endomorfismo $\psi = \phi^{-1}\pi_p\phi$ sobre $E(\mathbb{F}_{p^2})$.

Salida: kQ

1: $R_0 \leftarrow Q$ $(= \lambda^0 Q)$
2: $R_1 \leftarrow \psi(Q)$ $(= \lambda^1 Q)$
3: $R_2 \leftarrow \psi^2(Q)$ $(= \lambda^2 Q)$
4: $R_3 \leftarrow \psi^3(Q)$ $(= \lambda^3 Q)$
5: Descomponga $k = k_0 + k_1\lambda + k_2\lambda^2 + k_3\lambda^3$ donde $|k_i| = l$
6: Calcule $NAF_\omega(k_i) = \sum_{j=0}^{l-1} k_{ij}2^j$ (Algoritmo 4.4)
7: Obtenga $R_i^j = iR_j$ para $i \in [\pm 1, \pm 3, \pm 5, \ldots, \pm(2^{\omega-1} - 1)]$.
8: $R \leftarrow \mathcal{O}$
9: **for** $i = l - 1 \rightarrow 0$ **do**
10: $R \leftarrow 2R$
11: **if** $k_{0i} \neq 0$ **then**
12: $R \leftarrow R + R^0_{k_{0i}}$
13: **end if**
14: **if** $k_{1i} \neq 0$ **then**
15: $R \leftarrow R + R^1_{k_{1i}}$
16: **end if**
17: **if** $k_{2i} \neq 0$ **then**
18: $R \leftarrow R + R^2_{k_{2i}}$
19: **end if**
20: **if** $k_{3i} \neq 0$ **then**
21: $R \leftarrow R + R^3_{k_{3i}}$
22: **end if**
23: **end for**
24: **return** R

Capítulo 5

Emparejamientos bilineales

"El auténtico conocimiento es conocer la extensión de la propia ignorancia"
Confucio

En la actualidad los emparejamientos bilineales se han vuelto atractivos en el campo de la criptografía, ya que sus propiedades han permitido la creación de novedosos esquemas criptográficos, como es el caso de la criptografía basada en atributos (véase capítulo 6).

En este capítulo se presenta una introducción a los emparejamientos bilineales, posteriormente se presenta una breve descripción de las operaciones involucradas en los mismos y finalmente se detallan las funciones principales para su implementación.

Introducción a los emparejamientos bilineales

Sea $\mathbb{G}_1 = (\mathbb{G}_1, +, 0)$, $\mathbb{G}_2 = (\mathbb{G}_2, +, 0)$ y $\mathbb{G}_T = (\mathbb{G}_T, \cdot, 1)$ grupos cíclicos de orden primo $r \in \mathbb{Z}^+$, se define el emparejamiento bilineal $\hat{e}$ como la proyección:

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

el cual cumple con las siguientes características:

- Computable. $\hat{e}$ puede computarse eficientemente.
- No es degenerado. Para todo $a \in \mathbb{G}_1$ existe un $c \in \mathbb{G}_2$ tal que $\hat{e}(a, c) \neq 1$ y $a, c \neq 0$.
- Bilineal. Dados los elementos $a, b \in \mathbb{G}_1$ y $c, d \in \mathbb{G}_2$ donde $a, b, c, d \neq 0$, se tiene que $\hat{e}(a + b, c) = \hat{e}(a, c) \cdot \hat{e}(b, c)$ y que $\hat{e}(a, c + d) = \hat{e}(a, c) \cdot \hat{e}(a, d)$, entonces:

  $$\hat{e}(a + a, c) = \hat{e}(a, c + c) = \hat{e}(a, c) \cdot \hat{e}(a, c) = \hat{e}(a, c)^2$$

  En general, para $k \in [1, r - 1]$ se cumple que:

  $$\hat{e}(ka, c) = \hat{e}(a, kc) = \hat{e}(a, c)^k$$

En la práctica, los elementos del grupo $\mathbb{G}_1$ y $\mathbb{G}_2$ son determinados por los eigenespacios del endomorfismo de Frobenius π sobre alguna curva elíptica $E/\mathbb{F}_p$ con grado de encajamiento $k > 1$ [Geovandro et al., 2011]. Considere entonces la curva $E/\mathbb{F}_p$ de orden $h \cdot r$, donde r es un número primo y $h \in \mathbb{Z}^+$. Sea E' la curva enlazada de grado d tal que $\phi : E'(\mathbb{F}_{p^{k/d}}) \to E(\mathbb{F}_{p^k})$, entonces podemos generar los grupos $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$ de la siguiente manera [Fuentes-Castañeda, 2011]:

- $\mathbb{G}_1$ es el eigenespacio-1 de π en $E(\mathbb{F}_{p^k})$, el cual corresponde al grupo cíclico generado por los puntos de torsión r de $E(\mathbb{F}_p)$.
- Sea $Q' \in E(\mathbb{F}_{p^{k/d}})[r]$ y $Q = \phi(Q')$, entonces $\mathbb{G}_2$ es el grupo cíclico generado por Q, es decir $\mathbb{G}_2 = \langle Q \rangle$, donde $\mathbb{G}_2$ es el eigenespacio-p de π en $E(\mathbb{F}_{p^k})$
- $\mathbb{G}_T$ es un subgrupo de $\mathbb{F}^*_{p^k}$, formado por el conjunto de las r-ésimas raíces primitivas de la unidad en el grupo cíclico $\mathbb{F}^*_{p^k}$. Este grupo se denota como $\mathbb{F}^\times_{p^k}$.

Seguridad en los emparejamientos

Como se definió anteriormente, el emparejamiento toma como entradas elementos de grupos cíclicos aditivos y entrega como salida un elemento del grupo cíclico multiplicativo. Por tanto, para que un esquema basado en emparejamientos se considere seguro, el problema del logaritmo discreto debe ser computacionalmente *difícil de resolver* tanto en el grupo $\mathbb{F}^\times_{p^k}$, como en el grupo formado por los puntos de torsión de la curva elíptica E.

En el caso del grupo $E(\cdot)[r]$, el mejor ataque conocido es el algoritmo paralelizado de Pollard rho [Pollard, 1978], cuya complejidad es $\mathcal{O}(\sqrt{r})$, mientras que para el grupo $\mathbb{F}^\times_{p^k}$ el mejor algoritmo conocido es el cálculo de índices, cuya complejidad es $\mathcal{O}(\exp(1.92 \cdot (\ln p^k)^{1/3} \cdot (\ln \ln p^k)^{2/3}))$ [Adleman and Huang, 1999].

Por tanto, la seguridad del emparejamiento se mide con respecto a $\log_2(r)$ y $\log_2(p^k)$, es por ello que es necesario escoger curvas con un subgrupo de torsión r cuya longitud en bits sea suficientemente grande, para hacer inviable el ataque del logaritmo discreto en la curva. Asimismo, es necesario seleccionar el grado de encajamiento k de manera que también garantice la seguridad en el grupo multiplicativo. Sin embargo, no se pueden escoger dichos valores demasiado grandes, ya que se comprometería la eficiencia las operaciones aritméticas. Las *curvas amables con los emparejamientos*[1] son grupos de curvas que garantizan la seguridad del emparejamiento y permiten la eficiencia de los cálculos en el emparejamiento.

[1] Véase capítulo 4

5.1. Funciones racionales de la curva elíptica

Dada la curva elíptica $E/\mathbb{F}_p$ y $\bar{\mathbb{F}}_p$ la cerradura algebraica de $\mathbb{F}_p$, se dice que f es una función racional en $E/\mathbb{F}_p$ si existe un punto $P = (x, y)$, $P \in E(\bar{\mathbb{F}}_p)$ tal que $f(x, y) \neq \infty$. El conjunto de funciones racionales en $E/\mathbb{F}_p$ está denotado por $\bar{\mathbb{F}}_p(E)$, mientras que para todo $f \in \bar{\mathbb{F}}_p(E)$ se cumple que $f(P)$ es un elemento en el conjunto $\{\bar{\mathbb{F}}_p \cup \infty\}$ [Washington, 2008].

Sea P un punto en la curva elíptica $E/\mathbb{F}_p$ y $f \in \bar{\mathbb{F}}_p(E)$ una función racional. Se dice que f tiene un *cero* en P si $f(P) = 0$. De la misma manera, se dice que f tiene un *polo* en P si $f(P) = \infty$. En general, el número de polos y ceros en f es finito.

Para todo $P \in E/\mathbb{F}_p$ existe una función *uniformadora* u tal que $u(P) = 0$. En general, la evaluación de f en el punto P puede ser escrita como:

$$f(P) = (u(P))^m \cdot g(P)$$

donde $g \neq \{0, \infty\}$ y m es el orden de f en P, el cual se denota como $\text{ord}_p(f)$. Entonces, si $m > 0$ tenemos que f tiene un cero en P mientras que si $m < 0$ se dice que f tiene un polo en P.

Ejemplos de funciones racionales son la línea tangente a un punto P en la curva elíptica E, denotado como $\ell_{P,P}$, y la línea secante a E en los puntos P y Q, la cuál se denota como $\ell_{P,Q}$.

5.2. Divisores

Sea $E/\mathbb{F}$ una curva elíptica definida sobre $\mathbb{F}_p$. Para cada punto $P \in E(\bar{\mathbb{F}}_p)$ se define un símbolo formal $[P]$. Un *divisor* $\mathcal{D}$ sobre $E/\mathbb{F}_p$ es una combinación lineal de estos símbolos con coeficientes en $\mathbb{Z}$:

$$\mathcal{D} = \sum_j a_j[P_j], \quad a_j \in \mathbb{Z}$$

Un divisor es por tanto un elemento del grupo abeliano generado por los símbolos $[P]$ [Washington, 2008]. El grupo de divisores se denota como $\text{Div}(E)$. Las operadores que definen a un divisor son el *soporte*, el *grado* y la *suma*, los cuales se definen a continuación:

- Soporte:
$$\text{supp}(\mathcal{D}) = \{P \in E \mid a_j \neq 0\}$$
- Grado:
$$\deg(\mathcal{D}) = \sum_j a_j \ \in \mathbb{Z}$$

- Suma:

$$\text{sum}(\mathcal{D}) = \sum_j a_j P_j \ \in E(\bar{\mathbb{F}}_p)$$

La operación suma utiliza la ley de grupo sobre E para agregar los puntos que están dentro de los símbolos. Los divisores de grado 0 forman un subgrupo importante de $\text{Div}(E)$, el cual se denota como $\text{Div}^0(E)$.

Divisor de una función

Dada una curva elíptica $E/\mathbb{F}_p$ y $f \in \bar{\mathbb{F}}_p(E)$, el divisor de una función f está definido como:

$$\text{div}(f) = \sum_{P \in E} \text{ord}_p(f)[P] \ \in \text{Div}(E)$$

Un divisor $\mathcal{D}$ sobre una curva elíptica $E/\mathbb{F}_q$ con $\deg(\mathcal{D}) = 0$ y $\text{sum}(\mathcal{D}) = 0$, se denomina *divisor principal* si existe una función racional $f \in \bar{\mathbb{F}}_p(E)$ tal que $\mathcal{D} = \text{div}(f)$.

En general, dadas las funciones f y $g \in \bar{\mathbb{F}}_p(E)$, los divisores principales cumplen con las siguientes propiedades:

- $\text{div}(f \cdot g) = \text{div}(f) + \text{div}(g)$
- $\text{div}(f/g) = \text{div}(f) - \text{div}(g)$
- $\text{div}(f) = 0$ si y sólo si f es una función constante.

Dados los divisores $\mathcal{D}_1$ y $\mathcal{D}_2 \in \text{Div}(E)$, se dice que son linealmente independientes, lo cual se denota como $\mathcal{D}_1 \sim \mathcal{D}_2$, si $\mathcal{D}_1 - \mathcal{D}_2$ es una función principal. Entonces, dado $\mathcal{D}_1 \sim \mathcal{D}_2$, podemos escribir $\mathcal{D}_1 = \mathcal{D}_2 + \text{div}(f)$ para alguna función racional f.

Para $f \in \bar{\mathbb{F}}_p(E)^*$ una función racional diferente de cero y un punto $P \in E(\bar{\mathbb{F}}_p)$, se cumple que:

- Si P es un cero de f, entonces $\text{ord}_p(f) > 0$
- Si P es un polo de f, entonces $\text{ord}_p(f) < 0$
- Si P no es un polo ni un cero, entonces $\text{ord}_p(f) = 0$

Sea f una función racional y $\mathcal{D} = \sum_j a_j[P_j]$ un divisor tal que $\text{supp}(\text{div}(f)) \cap \text{supp}(\mathcal{D}) = \emptyset$. La función f puede entonces expresarse como:

$$f(\mathcal{D}) = \prod_{P_j \in \text{supp}(\mathcal{D})} f(P_j)^{a_j}$$

5.3. Emparejamiento de Tate

Definición 5.1. Función de Miller *[Miller, 2004]. Una función de Miller de longitud $s \in \mathbb{Z}$ denotada como $f_{s,R}$ es una función racional en $\bar{\mathbb{F}}_p(E)$ con divisor $div(f_{s,R}) = s[R] - [sR] - (s-1)[\mathcal{O}]$*

Lema 5.1. *Sea $f_{s,R}$ una función de Miller y v_R la línea vertical que corta a la curva elíptica E en el punto R. Para todo $a, b \in \mathbb{Z}$ se cumple que:*

- $f_{a+b,R} = f_{a,R} \cdot f_{b,R} \cdot \ell_{aR,bR}/v_{(a+b)R}$
- $f_{ab,R} = f_{bR}^{a} \cdot f_{a,bR}$
- $f_{1,R} = c$, *donde c es una constante.*

Dados los puntos $P \in \mathbb{G}_1$ y $Q \in \mathbb{G}_2$, consideremos el divisor $\mathcal{D}_q \sim [Q] - [\mathcal{O}]$ y la función de Miller $f_{r,P}$. El emparejamiento reducido de Tate $\hat{t}$ no degenerativo y bilineal se define como:

$$\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T : (P, Q) \mapsto f_{r,P}(Q)^{(p^k-1)/r}$$

Sea $g = f_{r,P}(Q)$ un elemento en el grupo multiplicativo $\mathbb{F}_{p^k}^*$, el emparejamiento Tate requiere el cálculo de la exponenciación $g^{\frac{p^k-1}{r}}$ tal que:

$$(g^{\frac{p^k-1}{r}})^r = 1$$

por tanto, $\hat{t}(P, Q)$ es un elemento del subgrupo de las r-ésimas raíces primitivas de la unidad en $\mathbb{F}_{p^k}^*$.

Emparejamiento *ate*

Dada la curva elíptica $E/\mathbb{F}_p$ con grado de encajamiento k y orden $\#E(\mathbb{F}_p) = h \cdot r = p + 1 - t$, donde t es la traza de E sobre $\mathbb{F}_p$. Sea E' la curva enlazada de E de grado d. Dados los puntos $P \in E(\mathbb{F}_p)[r]$ y $Q \in E'(\mathbb{F}_{p^{k/d}})[r]$, el emparejamiento *ate* $\hat{a}$ [Hess et al., 2006] se define como:

$$\hat{a}(Q, P) = f_{t-1,Q}(P)^{(p^k-1)/r}$$

5.4. Ciclo de Miller

Uno de los puntos principales en la construcción del emparejamiento es la evaluación de la función de Miller $f_{r,p}$ (Definición 5.1). Sea $r = (r_{l-1}, \ldots, r_1, r_0)_2$, de acuerdo con el lema 5.1 podemos construir $f_{r,p}$ de manera recursiva utilizando el método de *doblado y suma* de líneas:

$$f_{a+1,R} = f_{a,R} \cdot \ell_{aR,R}/v_{(a+1)R} \quad \text{y} \quad f_{2a,R} = f_{a,R}^2 \cdot \ell_{aR,aR}/v_{(2a)R} \tag{5.1}$$

donde las líneas verticales $v_{(a+1)R}$ y $v_{(2a)R}$ pueden ser omitidas de la ecuación (5.1) de acuerdo con [Barreto et al., 2003]. El ciclo de Miller se muestra en el Algoritmo 5.1.

Algoritmo 5.1 Ciclo de Miller

Entrada: $r = (r_{l-1}, \ldots, r_1, r_0)_2$, $Q, P \in E(\bar{\mathbb{F}}_p)$ tal que $P \neq Q$
Salida: $f_{r,Q}(P)$
1: $T \leftarrow Q$, $f \leftarrow 1$
2: **for** $i = l - 2 \rightarrow 0$ **do**
3: $\quad f \leftarrow f^2 \cdot \ell_{T,T}(P)$, $T \leftarrow 2T$
4: $\quad$ **if** $r_i = 1$ **then**
5: $\quad\quad f \leftarrow f \cdot \ell_{T,Q}(P)$, $T \leftarrow T + Q$
6: $\quad$ **end if**
7: **end for**
8: **return** f

Aritmética del ciclo de Miller en curvas BN

En el Algoritmo 5.1, observamos que se necesita de la evaluación de la línea tangente $\ell_{T,T}$ y del doblado del punto T así como la evaluación de la línea secante $\ell_{T,P}$ y suma de puntos $T + P$.

De acuerdo con el trabajo [Costello et al., 2010], la forma más eficiente para realizar estas operaciones es a través del uso de coordenadas proyectivas estándar. Las coordenadas estándar son aquellas coordenadas proyectivas (ver sección 4.3 en la página 46) donde los parámetros c y d tienen un valor de 1, por lo que un punto $(X_1 : Y_1 : Z_1)$ en una curva elíptica E en coordenadas estándar corresponde al punto $(X_1/Z_1, Y_1/Z_1)$ en coordenadas afines.

Sea $E/\mathbb{F}_p$ con grado de encajamiento $k = 12$ la cual se encuentra definida como $E : y^2 = x^3 + b$, tenemos que su forma proyectiva es $Y^2Z = X^3 + bZ^3$. Sea E' la curva enlazada de E de orden $d = 6$, la cual se define como $E' : y^2 = x^3 + b'$, dado un punto $T = (X_1 : Y_1 : Z_1) \in E'(\mathbb{F}_{p^2})$, podemos calcular $2T = (X_3 : Y_3 : Z_3)$ en base a las siguientes fórmulas [Aranha et al., 2011a]:

$$\begin{aligned} X_3 &= \frac{X_1 Y_1}{2}(Y_1^2 - 9b'Z_1^2) \\ Y_3 &= \left[\frac{1}{2}(Y_1^2 + 9b'Z_1^2)\right]^2 - 27b'^2 Z_1^4 \\ Z_3 &= 2Y_1^3 Z_1 \end{aligned}$$

así mismo la línea tangente $\ell_{T,T}$ evaluada en $P = (x_P, y_P) \in E(\mathbb{F}_p)$ está dada por:

$$\ell_{T,T}(P) = -2Y_1 Z_1 y_P + 3X_1 x_P w + (3b'Z_1^2 - X_1^2)w^3 \in \mathbb{F}_{p^{12}}$$

Tanto el doblado como la evaluación de la línea $\ell_{T,T}$ pueden realizarse simultáneamente como se muestra en el Algoritmo 5.2.

Algoritmo 5.2 Doblado de punto y evaluación de la línea tangente

Entrada: $Q = (X_1, Y_1, Z_1)$, $P = (x_P, y_P)$ donde $Q \in E(\mathbb{F}_{p^2})$ y $P \in E(\mathbb{F}_p)$
Salida: $T = 2Q$ y $\ell_{Q,Q}(P)$

1: $A \leftarrow X_1 \cdot Y_1/2$
2: $B \leftarrow Y_1^2$
3: $C \leftarrow Z_1^2$
4: $E \leftarrow 3b'C$
5: $F \leftarrow 3E$
6: $G \leftarrow (B + F)/2$
7: $H \leftarrow (Y_1 + Z_1)^2 - (B + C)$
8: $X_3 \leftarrow A \cdot (B - F)$
9: $Y_3 \leftarrow G^2 - 3E^2$
10: $Z_3 \leftarrow B \cdot H$
11: $l_0 \leftarrow -H \cdot y_p$ (Algoritmo 3.16)
12: $l_1 \leftarrow 3X_1^2 \cdot x_p$ (Algoritmo 3.16)
13: $l_2 \leftarrow E - B$
14: **return** $2Q = (X_3, Y_3, Z_3)$ y $\ell_{Q,Q}(P) = l_0 + l_1 w + l_2 w^3$

De la misma forma, dados los puntos $T = (X_1, Y_1, Z_1), Q = (X_2, Y_2, 1) \in E'(\mathbb{F}_{p^2})$ y $P = (x_P, y_P) \in E(\mathbb{F}_p)$, se puede calcular la suma de puntos $R = T + Q = (X_3, Y_3, Z_3)$ y $\ell_{T,Q}(P)$ en base a las siguientes fórmulas [Aranha et al., 2011a]:

$$\begin{aligned} \ell_{T,Q}(P) &= \lambda y_P - \theta x_P w + (\theta X_2 - \lambda Y_2) w^3, \\ X_3 &= \lambda(\lambda^2 + Z_1\theta^2 - 2X_1\lambda^2) \\ Y_3 &= \theta(3X_1\lambda^2 - \lambda^3 - Z_1\theta^2) - Y_1\lambda^3 \\ Z_3 &= Z_1\lambda^3 \end{aligned}$$

donde $\theta = Y_1 - Y_2 Z_1$ y $\lambda = X_1 - X_2 Z_1$. El Algoritmo 5.3 describe cómo realizar la suma de puntos $R = T + Q$ y evaluación de la línea secante $\ell_{T,Q}(P)$ de manera simultánea.

Otro punto importante del algoritmo de Miller es la multiplicación del elemento f por la evaluación de la línea en las instrucciones 3 y 5 del Algoritmo 5.1. La evaluación de las líneas $\ell_{Q,Q}$ y $\ell_{T,Q}$ definen un elemento en el grupo $\mathbb{F}_{p^k}^*$ con la mitad de los coeficientes en 0, por lo que cualquier producto de $f \in \mathbb{F}_{p^{12}}$ por $\ell_{Q,Q}$ y $\ell_{T,Q}$ se conoce como *multiplicación dispersa*, y puede realizarse a través del Algoritmo 3.30 de la página 38.

5.5. Exponenciación final

Como se observó en las secciones anteriores, se requiere calcular $f^{(p^k-1)/r} \in \mathbb{F}_{p^k}^{\times}$. Esta operación se conoce como *exponenciación final* y nos garantiza que el resultado

Algoritmo 5.3 Adición de puntos y evaluación de la línea secante

Entrada: $T = (X_1, Y_1, Z_1), Q = (X_2, Y_2, 1)$, $P = (x_P, y_P)$ donde $T, Q \in E(\mathbb{F}_{p^2})$ y $P \in E(\mathbb{F}_p)$

Salida: $R = T + Q$ y $\ell_{T,Q}(P)$

1: $A \leftarrow Y_2 \cdot Z_1$
2: $B \leftarrow X_2 \cdot Z_1$
3: $\theta \leftarrow Y_1 - A$
4: $\lambda \leftarrow X_1 - B$
5: $C \leftarrow \theta^2$
6: $D \leftarrow \lambda^2$
7: $E \leftarrow D \cdot \lambda$
8: $F \leftarrow Z_1 \cdot C$
9: $G \leftarrow X_1 \cdot D$
10: $H \leftarrow E + F - 2G$
11: $I \leftarrow Y_1 \cdot E$
12: $J \leftarrow \theta \cdot X_2 - \lambda \cdot Y_2$
13: $X_3 \leftarrow \lambda \cdot H$
14: $Y_3 \leftarrow \theta \cdot (G - H) - I$
15: $Z_3 \leftarrow Z_1 \cdot E$
16: $l_0 \leftarrow \lambda \cdot y_p$ (Algoritmo 3.16)
17: $l_1 \leftarrow -\theta \cdot x_p$ (Algoritmo 3.16)
18: $l_2 \leftarrow J$
19: **return** $R = T + Q = (X_3, Y_3, Z_3)$ y $\ell_{T,Q}(P) = l_0 + l_1 w + l_2 w^3$

del emparejamiento sea un valor único.

Realizar la exponenciación final directamente resulta ser muy costoso computacionalmente. Una forma más eficiente de efectuar esta operación es representando $(p^k - 1)/r$ como producto de dos exponentes:

$$\frac{p^k - 1}{r} = \frac{p^k - 1}{\Phi_k(p)} \cdot \frac{\Phi_k(p)}{r}$$

donde $\Phi_k(p)$ es el k-ésimo polinomio ciclotómico evaluado en p. Para el caso de las curvas BN donde $k = 12$ tenemos que:

$$\frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}$$

Para $f = f_0 + f_1 W$, $f \in \mathbb{F}_{p^{12}}$, se cumple que $f^{p^6} = \bar{f}$, donde $\bar{f} = f_0 - f_1 W$ es el conjugado de f. Entonces, la evaluación del primer exponente $(p^6 - 1)$ es:

$$f^{p^6 - 1} = \bar{f} \cdot f^{-1}$$

el cual puede ser calculado por medio de una multiplicación e inversión en $\mathbb{F}_{p^{12}}$. Una ventaja de esta operación es que el elemento $g = f^{p^6 - 1}$ se convierte en un elemento del

grupo ciclotómico $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ cuyas operaciones tienen un menor costo computacional (véase sección 3.6, en la página 39). La exponenciación por (p^2+1) puede realizarse por medio de una multiplicación y a través del uso del operador de Frobenius (sección 5.5.1 en la página 71). Finalmente al cálculo de f^e donde $e=(p^4-p^2+1)/r$ se le conoce como *parte difícil de la exponenciación final*, ésto se debe a que requiere un mayor número de operaciones que en los otros dos exponentes.

5.5.1. Operador de Frobenius

Durante el cálculo de la exponenciación final, es necesario realizar varias elevaciones de un elemento $f \in \mathbb{F}_{p^{12}}$ a la p-ésima potencia. Una manera de reducir el costo de esta operación es a través del uso del operador de Frobenius.

Sea $f = g + hw$ tal que $f \in \mathbb{F}_{p^{12}}$, $h, g \in \mathbb{F}_{p^6}$ donde $g = g_0 + g_1 v + g_2 v^2$, $h = h_0 + h_1 v + h_2 v^2$ y $g_i, h_i \in \mathbb{F}_{p^2}$. Si representamos $\mathbb{F}_{p^{12}}$ como una extensión de orden seis del campo $\mathbb{F}_{p^2}$, es decir $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[u]/(W^6-\xi)$, tenemos que f puede ser escrito como:

$$f = g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5$$

así mismo, para cualquier elemento $a = a_0 + a_1 u \in \mathbb{F}_{p^2}$ se cumple que:

$$a^{p^{2i}} = a \quad \text{y} \quad a^{p^{(2i-1)}} = \bar{a} \quad \forall i \in \mathbb{Z}^+$$

donde $\bar{a} = a_0 - a_1 u$ es el conjugado de a.

Ahora bien dada la igualdad $W^p = \xi^{(p-1)/6} W$, puede escribirse $(W^i)^p = \gamma_{1,i} W^i$ con $\gamma_{1,i} = \xi^{i(p-1)/6}$ para $i \in [1,5]$. Entonces f^p puede calcularse como:

$$\begin{aligned} f^p &= (g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5)^p \\ &= \hat{g_0} + \hat{h_0} W^P + \hat{g_1} W^{2P} + \hat{h_1} W^{3P} + \hat{g_2} W^{4P} + \hat{h_2} W^{5P} \\ &= \hat{g_0} + \hat{h_0}\gamma_{1,1} W^P + \hat{g_1}\gamma_{1,2} W^2 + \hat{h_1}\gamma_{1,3} W^3 + \hat{g_2}\gamma_{1,4} W^4 + \hat{h_2}\gamma_{1,5} W^5 \end{aligned} \tag{5.2}$$

La ecuación (5.2) tiene un costo de 5 multiplicaciones y 5 conjugados en $\mathbb{F}_{p^2}$. De la misma manera, se pueden calcular f^{p^2} y f^{p^3}, las cuales son requeridas en el cálculo de la parte difícil del emparejamiento.

Cabe destacar que para llevar a cabo el cálculo de f^p, f^{p^2} y f^{p^3}, es necesario contar con las constantes $\gamma_{1,i} = \xi^{i(p-1)/6}$, $\gamma_{2,i} = \gamma_{1,i} \cdot \bar{\gamma}_{1,i}$ y $\gamma_{3,i} = \gamma_{1,i} \cdot \gamma_{2,i}$ para $i \in [1,5]$. Dado que estos valores dependen únicamente de ξ y p, pueden ser calculados *fuera de línea* antes de realizar el emparejamiento.

Los Algoritmos 5.4, 5.5 y 5.6 realizan el cálculo de f^p, f^{p^2} y f^{p^3} respectivamente. Para cada caso, f es un elemento de la extensión $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[u]/(W^6-\xi)$.

Algoritmo 5.4 Operador de Frobenius para calcular f^p

Entrada: $f = g + hw$, $f \in \mathbb{F}_{p^{12}}$, con $g = g_0 + g_1 v + g_2 v^2$ y $h = h_0 + h_1 v + h_2 v^2$, $g, h \in \mathbb{F}_{p^6}$

Salida: $f^p \in \mathbb{F}_{p^{12}}$

1: Calcule $\gamma_{1,i} = \xi^{i(p-1)/6}$ para $i = 1, 2, 3, 4, 5$
2: $t_1 \leftarrow \bar{h_0} \cdot \gamma_{1,1}$
3: $t_2 \leftarrow \bar{g_1} \cdot \gamma_{1,2}$
4: $t_3 \leftarrow \bar{h_1} \cdot \gamma_{1,3}$
5: $t_4 \leftarrow \bar{g_2} \cdot \gamma_{1,4}$
6: $t_5 \leftarrow \bar{h_2} \cdot \gamma_{1,5}$
7: $c_0 \leftarrow \bar{g_0} + t_2 v + t_4 v^2$
8: $c_1 \leftarrow t_1 + t_3 v + t_5 v^2$
9: **return** $c \leftarrow c_0 + c_1 w$

Algoritmo 5.5 Operador de Frobenius para calcular f^{p^2}

Entrada: $f = g + hw$, $f \in \mathbb{F}_{p^{12}}$, con $g = g_0 + g_1 v + g_2 v^2$ y $h = h_0 + h_1 v + h_2 v^2$, $g, h \in \mathbb{F}_{p^6}$

Salida: $f^{p^2} \in \mathbb{F}_{p^{12}}$

1: Calcule $\gamma_{2,i} = \gamma_{1,i} \cdot \gamma_{1,i}^p$ para $i = 1, 2, 3, 4, 5$
2: $t_1 \leftarrow h_0 \cdot \gamma_{2,1}$
3: $t_2 \leftarrow g_1 \cdot \gamma_{2,2}$
4: $t_3 \leftarrow h_1 \cdot \gamma_{2,3}$
5: $t_4 \leftarrow g_2 \cdot \gamma_{2,4}$
6: $t_5 \leftarrow h_2 \cdot \gamma_{2,5}$
7: $c_0 \leftarrow g_0 + t_2 v + t_4 v^2$
8: $c_1 \leftarrow t_1 + t_3 v + t_5 v^2$
9: **return** $c \leftarrow c_0 + c_1 w$

Algoritmo 5.6 Operador de Frobenius para calcular f^{p^3}

Entrada: $f = g + hw$, $f \in \mathbb{F}_{p^{12}}$, con $g = g_0 + g_1 v + g_2 v^2$ y $h = h_0 + h_1 v + h_2 v^2$, $g, h \in \mathbb{F}_{p^6}$

Salida: $f^{p^3} \in \mathbb{F}_{p^{12}}$

1: Calcule $\gamma_{3,i} = \gamma_{1,i} \cdot \gamma_{2,i}$ para $i = 1, 2, 3, 4, 5$
2: $t_1 \leftarrow \bar{h_0} \cdot \gamma_{3,1}$
3: $t_2 \leftarrow \bar{g_1} \cdot \gamma_{3,2}$
4: $t_3 \leftarrow \bar{h_1} \cdot \gamma_{3,3}$
5: $t_4 \leftarrow \bar{g_2} \cdot \gamma_{3,4}$
6: $t_5 \leftarrow \bar{h_2} \cdot \gamma_{3,5}$
7: $c_0 \leftarrow \bar{g_0} + t_2 v + t_4 v^2$
8: $c_1 \leftarrow t_1 + t_3 v + t_5 v^2$
9: **return** $c \leftarrow c_0 + c_1 w$

5.5.2. Parte *difícil* de la exponenciación final

Como se mencionó anteriormente, la parte *difícil* de la exponenciación final corresponde al cálculo de $f^{(p^4-p^2+1)/r}$ para $f \in \mathbb{F}_{p^{12}}$.

Para el caso de este trabajo, se realizó la exponenciación a $d = (p^4 - p^2 + 1)/r$ de acuerdo a la mejora descrita en el trabajo [Fuentes-Castañeda et al., 2011], en el cual es posible reducir el número de operaciones utilizando un múltiplo d' de d siempre que $r \nmid d$.

En el caso de las curvas BN (véase sección 4.5, página 52) con parámetros:

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

el exponente $d(z) = (p(z)^4 - p(z)^2 + 1)/r(z)$ puede expresarse como:

$$\begin{aligned} d(z) &= (-36z^4 - 30z^2 + 18z - 2) &+ \\ &= (-36z^3 - 18z^2 - 12z + 1)p(z) &+ \\ &= (6z^2 + 1)p(z)^2 &+ \\ &= p(z)^3 \end{aligned}$$

A través del uso de rejillas y el algoritmo LLL [Lenstra et al., 1982] se obtiene el múltiplo $d'(z) = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3 = (12z^3 + 6z^2 + 2z)d(z)$, donde:

$$\begin{aligned} \lambda_0 &= 12z^3 + 12z^2 + 6z + 1 \\ \lambda_1 &= 12z^3 + 6z^2 + 4z \\ \lambda_2 &= 12z^3 + 6z^2 + 6z \\ \lambda_3 &= 12z^3 + 6z^2 + 4z - 1 \end{aligned}$$

La exponenciación de $g^{d'(z)}$ requiere del cálculo de las siguientes variables temporales:

$$f^z \mapsto f^{2z} \mapsto f^{4z} \mapsto f^{6z} \mapsto f^{6z^2} \mapsto f^{12z^2} \mapsto f^{12z^3}$$

las cuales requieren de 3 exponenciaciones por z, 3 cuadrados y 1 multiplicación $\mathbb{F}_{p^{12}}$. Finalmente, dadas las variables $a = g^{12z^3} \cdot g^{6z^2} \cdot g^{6z}$ y $b = a \cdot (g^{2z})^{-1}$, la exponenciación $g^{d'(z)}$ se calcula de la siguiente manera:

$$g^{d'(z)} = \left[a \cdot g^{6z^2} \cdot g\right] \cdot [b]^p \cdot [a]^{p^2} \cdot \left[b \cdot g^{-1}\right]^{p^3} \in \mathbb{F}_{p^{12}}^{\times}$$

Dado que $g \in \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$, el costo de $g^{d'(z)}$ es de 3 aplicaciones del operador de Frobenius, 3 exponenciaciones por z, 10 multiplicaciones en $\mathbb{F}_{p^{12}}$ y tres cuadrados en el grupo $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$. El Algoritmo 5.7 lista todos los pasos para realizar la exponenciación final.

Algoritmo 5.7 Exponenciación final

Entrada: $h \in \mathbb{F}_{p^{12}}$
Salida: $h^{(p^{12}-1)/r} \in \mathbb{F}_{p^{12}}$

1: $f_1 \leftarrow \bar{h}$
2: $f_2 \leftarrow h^{-1}$
3: $f \leftarrow f_1 \cdot f_2$
4: $f \leftarrow f^{p^2} \cdot f$ (Algoritmo 5.5)
5: $f_c \leftarrow \bar{f}$
6: $t_1 \leftarrow f^z$
7: $t_2 \leftarrow t_1^2$ (Algoritmo 3.33)
8: $t_3 \leftarrow t_1 \cdot t_2$
9: $t_4 \leftarrow t_3^2$ (Algoritmo 3.33)
10: $t_5 \leftarrow t_4^z$
11: $t_6 \leftarrow t_5^z$
12: $t_7 \leftarrow t_6^2$ (Algoritmo 3.33)
13: $a \leftarrow t_7 \cdot t_5 \cdot t_4$
14: $t_8 \leftarrow \bar{t}_2$
15: $t_9 \leftarrow (b \cdot f_c)$
16: $b \leftarrow a \cdot t_8$
17: $g \leftarrow a \cdot t_5 \cdot f$
18: $l_1 \leftarrow b^p$ (Algoritmo 5.4)
19: $l_2 \leftarrow a^{p^2}$ (Algoritmo 5.5)
20: $l_3 \leftarrow t_9^{p^3}$ (Algoritmo 5.6)
21: $g \leftarrow g \cdot l_1 \cdot l_2 \cdot l_3$
22: **return** g

Un punto importante en la exponenciación final es el cálculo de $f^z \in \mathbb{F}_{p^{12}}$. En general, z es seleccionada de manera que su peso de Hamming sea bajo, es decir, se busca que su representación binaria tenga pocos elementos de valor 1. Ésto representa una ventaja, ya que al ser f un elemento del grupo ciclotómico $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$, puede efectuarse esta exponenciación de manera eficiente utilizando los cuadrados de Karabina (véase sección 3.6.2, en la página 40).

El Algoritmo 5.8 realiza la exponenciación por z, donde $\mathcal{C}(g)$ calcula el cuadrado comprimido de $g \in \mathbb{F}_{p^{12}}$ mientras que $\mathcal{D}$ realiza la descompresión.

5.6. Emparejamiento óptimo *ate* para curvas BN

Como se detalló anteriormente, la evaluación de la función de Miller $f_{s,p}$ puede ser calculada a través del ciclo de Miller, donde el número de iteraciones depende directamente del entero s. En el caso del emparejamiento de Tate, tenemos que el número de iteraciones está relacionado con el orden r de la curva elíptica, mientras

Algoritmo 5.8 Exponenciación por z en $\mathbb{F}_{p^{12}}$

Entrada: $f \in \mathbb{F}_{p^{12}}$, y $z = \sum_{i=0}^{l-1} z_i 2^i$
Salida: $C = f^z \in \mathbb{F}_{p^{12}}$

1: $j \leftarrow 0$
2: $C \leftarrow 1$
3: **if** $z_0 = 1$ **then**
4: $\quad C \leftarrow f$
5: **end if**
6: $g \leftarrow f$
7: **for** $i = 1 \rightarrow l - 1$ **do**
8: $\quad g \leftarrow \mathcal{C}(g)$ (Algoritmo 3.34)
9: $\quad$ **if** $z_i \neq 0$ **then**
10: $\quad\quad c_j \leftarrow g$
11: $\quad\quad j \leftarrow j + 1$
12: $\quad$ **end if**
13: **end for**
14: **for** $i = j - 1 \rightarrow 0$ **do**
15: $\quad C \leftarrow C \cdot \mathcal{D}(c_i)$
16: **end for**
17: **return** C

que para el emparejamiento *ate*, observamos que el número de iteraciones depende de la traza t de la curva. Dado que $t \approx \sqrt{r}$, el emparejamiento *ate* representa una mejora ya que el número de iteraciones se reduce a la mitad.

Una mejora al emparejamiento *ate* para curvas BN, es el conocido como *emparejamiento óptimo ate* [Vercauteren, 2010], el cual está dado por:

$$\begin{aligned} \hat{a}_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 &\rightarrow \mathbb{G}_T \\ (Q, P) &\mapsto \left[f_{6z+2}(P) \cdot \ell_{(6z+2)Q,\pi(Q)}(P) \cdot \ell_{(6z+2)Q+\pi(Q),\pi^2(Q)}(P) \right]^{(p^{12}-1)/r} \end{aligned}$$

donde $\pi : (x, y) \mapsto (x^p, y^p)$ es el endomorfismo de Frobenius en el punto Q (véase sección 4.4.6, en la página 51). Dado que $z \approx \sqrt[4]{t}$, el emparejamiento óptimo *ate* sólo requiere una cuarta parte de iteraciones en el ciclo de Miller en relación con el emparejamiento Tate.

El Algoritmo 5.9 muestra el emparejamiento óptimo *ate* para curvas BN de acuerdo con el trabajo [Beuchat et al., 2010], donde las operaciones de adición y sustracción de puntos son permitidas en el cálculo del emparejamiento.

Algoritmo 5.9 Emparejamiento óptimo *ate* para curvas BN

Entrada: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$
Salida: $\hat{a}_{opt}(Q, P)$

1: Escribir $s = 6z + 2$ como $s = \sum_{i=0}^{l-1}$ con $s_i \in \{-1, 0, 1\}$
2: $T \leftarrow Q$, $f \leftarrow 1$
3: **for** $i = l - 2 \rightarrow 0$ **do**
4: $\quad f \leftarrow f^2 \cdot \ell_{T,T}(P)$, $T \leftarrow 2T$
5: $\quad$ **if** $s_i = 1$ **then**
6: $\quad\quad f \leftarrow f \cdot \ell_{T,Q}(P)$, $T \leftarrow T + Q$
7: $\quad$ **else if** $s_i = -1$ **then**
8: $\quad\quad f \leftarrow f \cdot \ell_{T,-Q}(P)$, $T \leftarrow T - Q$
9: $\quad$ **end if**
10: **end for**
11: $Q_1 \leftarrow \pi(Q)$
12: $Q_2 \leftarrow \pi^2(Q)$
13: $f \leftarrow f \cdot \ell_{T,Q_1}(P)$, $T \leftarrow T + Q_1$
14: $f \leftarrow f \cdot \ell_{T,-Q_2}(P)$, $T \leftarrow T - Q_2$
15: $g \leftarrow f^{(p^{12}-1)/r}$
16: **return** g

5.7. Exponenciación en el grupo $\mathbb{G}_T$

En el capítulo 4 se mostraron varias técnicas para implementar la multiplicación escalar de puntos (véase sección 4.6, en la página 53). De la misma forma, estos métodos pueden utilizarse para la realizar la exponenciación en el grupo multiplicativo G_T, con la diferencia de que en vez de realizar suma y doblado de puntos, se realizan multiplicaciones y elevaciones al cuadrado en $\mathbb{F}_{p^{12}}^{\times}$.

Esta operación es necesaria en algunos esquemas criptográficos basados en emparejamientos, como es el caso de la criptografía basada en atributos (capítulo 6). En este trabajo se implementaron dos opciones para realizar la exponenciación en $\mathbb{G}_T$.

En la primera implementación se utilizó el método GS. Para este caso, ψ corresponde al endomorfismo de frobenius π, el cual actúa como una exponenciación por p. Dado que el grupo $\mathbb{F}_{p^{12}}^{\times}$ es de orden r para curvas BN, entonces $p = t - 1 \pmod{r}$, por lo que la descomposición del escalar es la misma que en el método descrito en la sección 4.6.4, página 60. El algoritmo muestra la exponenciación en el grupo $\mathbb{G}_T$ utilizando el método GS, donde $\lambda = t - 1$.

Algoritmo 5.10 Exponenciación en $\mathbb{G}_T$: Método GS

Entrada: Tamaño de ventana ω, entero positivo k donde $|k| = 4\ell$, $g \in \mathbb{F}_{p^{12}}^{\times}$, endomorfismo π.
Salida: $g^k \in \mathbb{F}_{p^{12}}^{\times}$

1: $r_0 \leftarrow g$ $(= \lambda^0 Q)$
2: $r_1 \leftarrow \pi(g)$ $(= \lambda^1 Q)$
3: $r_2 \leftarrow \pi^2(g)$ $(= \lambda^2 Q)$
4: $r_3 \leftarrow \pi^3(g)$ $(= \lambda^3 Q)$
5: Descomponga $k = k_0 + k_1\lambda + k_2\lambda^2 + k_3\lambda^3$ donde $|k_i| = l$
6: Calcule $NAF_\omega(k_i) = \sum_{j=0}^{l-1} k_{ij}2^j$ (Algoritmo 4.4)
7: Obtenga r_j^i para $i \in [\pm 1, \pm 3, \pm 5, \ldots, \pm(2^{\omega-1} - 1)]$.
8: $f \leftarrow 1$
9: **for** $i = l - 1 \rightarrow 0$ **do**
10: $f \leftarrow f^2$
11: **if** $k_{0i} \neq 0$ **then**
12: $f \leftarrow f \cdot r_0^{k_{0i}}$
13: **end if**
14: **if** $k_{1i} \neq 0$ **then**
15: $f \leftarrow f \cdot r_1^{k_{1i}}$
16: **end if**
17: **if** $k_{2i} \neq 0$ **then**
18: $f \leftarrow f \cdot r_2^{k_{2i}}$
19: **end if**
20: **if** $k_{3i} \neq 0$ **then**
21: $f \leftarrow f \cdot r_3^{k_{3i}}$
22: **end if**
23: **end for**
24: **return** f

La segunda implementación se realizó por medio del método *comb*, esto debido a que se tiene un conocimiento previo de un elemento $g \in \mathbb{F}_{p^{12}}^{\times}$, lo que permite que el precómputo se realice *fuera de línea*. El Algoritmo 5.11 realiza la exponenciación en el grupo $\mathbb{G}_T$ utilizando el método *comb*.

Cabe destacar que tanto en el Algoritmo 5.10 como en el Algoritmo 5.11 se utilizó el método ω-NAF (véase sección 4.6.1, en la página 54) con el fin de ahorrar multiplicaciones.

Algoritmo 5.11 Exponenciación en $\mathbb{G}_T$: Método *comb*

Entrada: Tamaño de ventana ω, entero positivo k, $g \in \mathbb{F}_{p^{12}}^{\times}$

Salida: $g^k \in \mathbb{F}_{p^{12}}^{\times}$

1: *Precómputo.* Obtenga $g^{[a_{\omega-1},\ldots,a_2,a_1,a_0]}$ para todas las combinaciones de bits $(a_{\omega-1},\ldots,a_2,a_1,a_0)$ de tamaño ω
2: Rellene la representación binaria de k con 0's a la izquierda en caso de ser necesario y escriba $k = K^{\omega-1} \parallel \ldots \parallel K^1 \parallel K^0$, donde K^j es una cadena de bits de longitud d. Represente K_i^j al bit i del elemento K^j
3: $f \leftarrow 1$
4: **for** $i = d-1 \rightarrow 0$ **do**
5: $\quad f \leftarrow f^2$
6: $\quad f \leftarrow f \cdot g^{[K_i^{\omega-1},\ldots,K_i^1,K_i^0]}$
7: **end for**
8: **return** f

Capítulo 6

Criptografía basada en atributos (ABE)

"La inteligencia consiste no sólo en el conocimiento,
sino también en la destreza de aplicar los conocimientos en la práctica"
Aristóteles

En el capítulo 1 se da una breve introducción a la criptografía basada en atributos, mientras que en los capítulos 2, 3, 4 y 5 se presentan los conceptos y aritmética relacionados con los emparejamientos. En este capítulo se describe la criptografía basada en atributos detallando su funcionamiento desde el punto de vista de los emparejamientos bilineales.

6.1. Introducción al esquema ABE

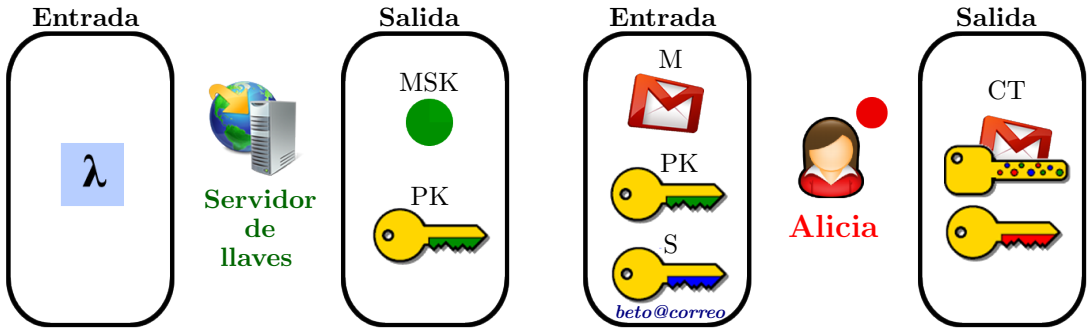
La criptografía basada en atributos es un esquema criptográfico en el que un usuario puede acceder a un determinado recurso si y sólo si, cuenta con un conjunto de privilegios (atributos) que puedan satisfacer una política de acceso previamente establecida. Este esquema se vuelve muy útil como medio de control de acceso, ya que usuarios con los mismos atributos pueden acceder a un mismo recurso.

El esquema ABE puede realizarse mediante el empleo del *cifrado difuso* [Sahai and Waters, 2005], en el cual se propone el uso de emparejamientos bilineales para su realización, donde el cifrado difuso se toma como un caso particular de la criptografía basada en la identidad.

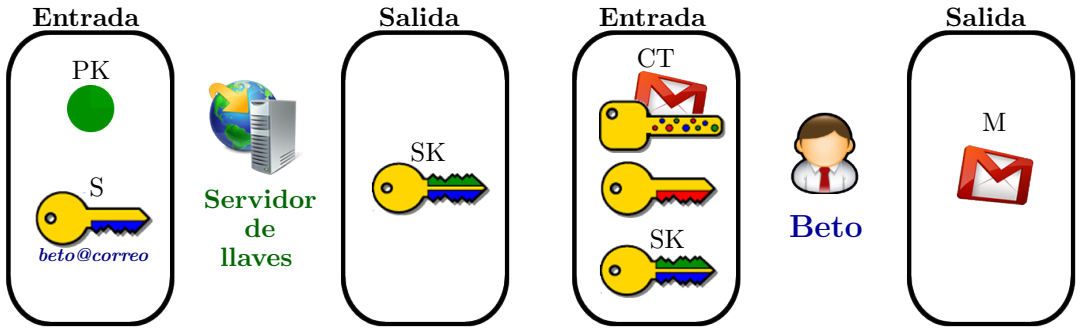
Criptografía Basada en la Identidad (IBE)

Como se mencionó en el capítulo 1, la criptografía basada en la identidad es un esquema que permite utilizar una cadena de texto arbitraria como llave pública para

cifrar un archivo. Este esquema plantea la comunicación de dos entidades denominadas Alicia y Beto de manera segura. En este caso, Alicia desea enviar un mensaje cifrado a Beto utilizando una cadena que identifique a Beto (por ejemplo su correo). Beto recibe el mensaje y lo descifra utilizando la llave privada proporcionada por una autoridad denominada "Servidor de llaves". Por tanto, este esquema está conformado de cuatro fases (figura 6.1), las cuales se describen a continuación:



(a) Inicialización (b) Cifrado



(c) Generación de llave (d) Descifrado

Figura 6.1: Fases de la criptografía basada en la identidad

a) **Inicialización(λ)**. El servidor de llaves toma como entrada el parámetro de seguridad λ y genera como salida su llave pública PK y su llave secreta maestra MSK.

b) **Cifrado(PK,M,S)**. Alicia toma la llave pública del servidor PK, el mensaje M y una cadena arbitraria S. Genera un secreto k y la llave pública para la cadena S, finalmente cifra el mensaje M obteniendo el texto cifrado CT.

c) **Generación de llave(MSK, S)**. El servidor de llaves recibe una solicitud con la cadena de texto S, toma la llave secreta maestra MSK y genera la llave privada SK para la cadena S.

d) **Descifrado**(CT, SK). Beto recibe el texto cifrado CT, si su llave privada SK corresponde a la llave pública con el que fue cifrado CT, entonces Beto puede descifrar CT y así obtener el mensaje M.

La figura 6.2 muestra la manera de implementar la criptografía basada en la identidad utilizando emparejamientos.

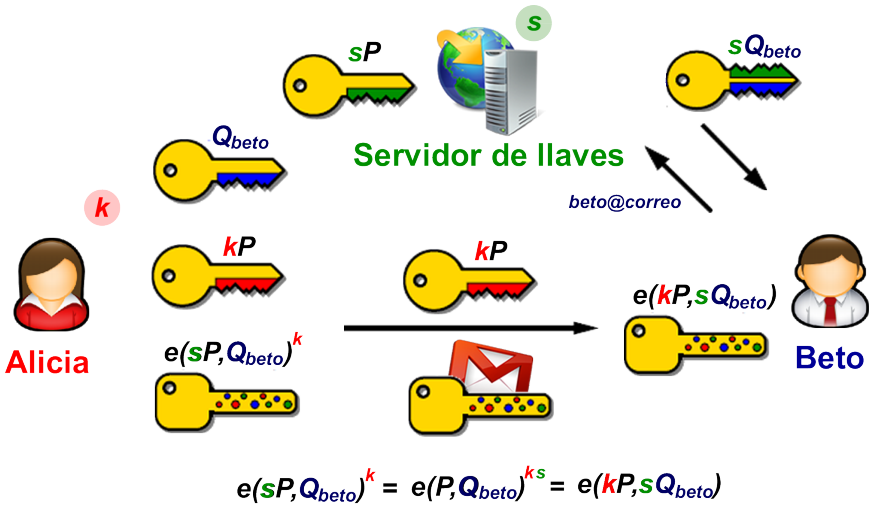


Figura 6.2: Implementación del esquema IBE utilizando emparejamientos

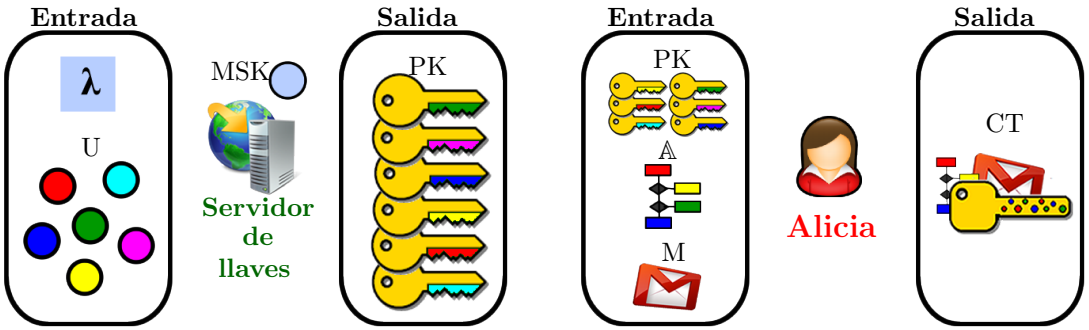
De manera análoga al esquema IBE, la criptografía basada en atributos plantea un esquema de comunicación segura entre Alicia y Beto a través del uso de atributos. De igual manera, este esquema se conforma de cuatro algoritmos [Waters, 2008], los cuales se ejemplifican en la figura 6.3 y se describen a continuación:

1. **Inicialización**$(\lambda,\ U)$. Este algoritmo toma como entrada el parámetro de seguridad λ y el universo de atributos. Genera la llave maestra MSK y genera como salida el conjunto de parámetros públicos PK.

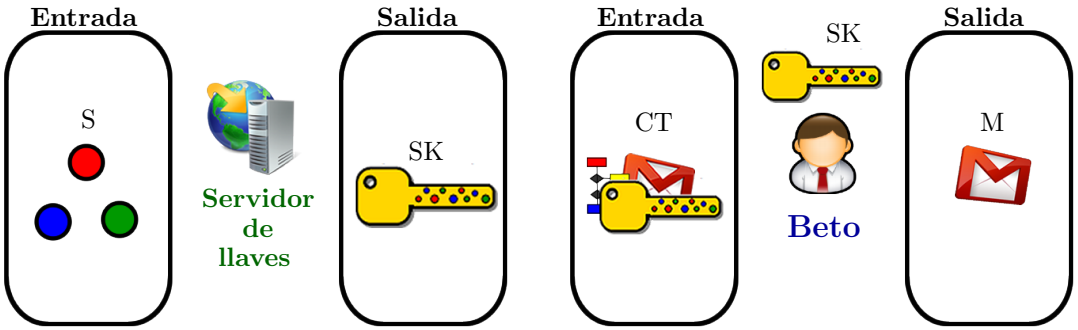
2. **Cifrado**$(PK,M,\mathbb{A})$. Recibe como entrada el conjunto de parámetros públicos PK, el mensaje M y la política de acceso $\mathbb{A}$ sobre el universo de atributos. El algoritmo cifra entonces el mensaje M obteniendo CT, de forma que únicamente el usuario que posea el conjunto de atributos que satisfaga la política de acceso $\mathbb{A}$, sea capaz de descifrar el mensaje. Se supone entonces que CT incluye $\mathbb{A}$.

3. **Generación de llave**(MSK, S). Toma como entrada la llave maestra MSK y un conjunto de atributos S que describen la llave. El algoritmo genera entonces la llave privada SK

4. **Descifrado**(PK, CT, SK). Este algoritmo toma como entrada los parámetros públicos PK, el texto cifrado CT que contiene la política $\mathbb{A}$ y la llave secreta SK, la cual es la llave privada para el conjunto de los atributos S. Si el conjunto de atributos S satisface la política de acceso $\mathbb{A}$, entonces el algoritmo es capaz de descifrar el texto cifrado CT y obtener el mensaje M



(a) Inicialización (b) Cifrado



(c) Generación de llave (d) Descifrado

Figura 6.3: Fases de la criptografía basada en atributos

La forma de implementación del esquema ABE se ejemplifica en la figura 6.4. En las secciones posteriores se mostrará cómo implementar este esquema a través del uso de los emparejamientos bilineales.

6.2. Política de acceso

Como se mencionó anteriormente, el esquema ABE necesita una política de acceso para poder descifrar un documento, en general, esta política se describe a través de una fórmula booleana monótona, por ejemplo: $(A \vee C) \wedge B$, donde $\vee$ representa una

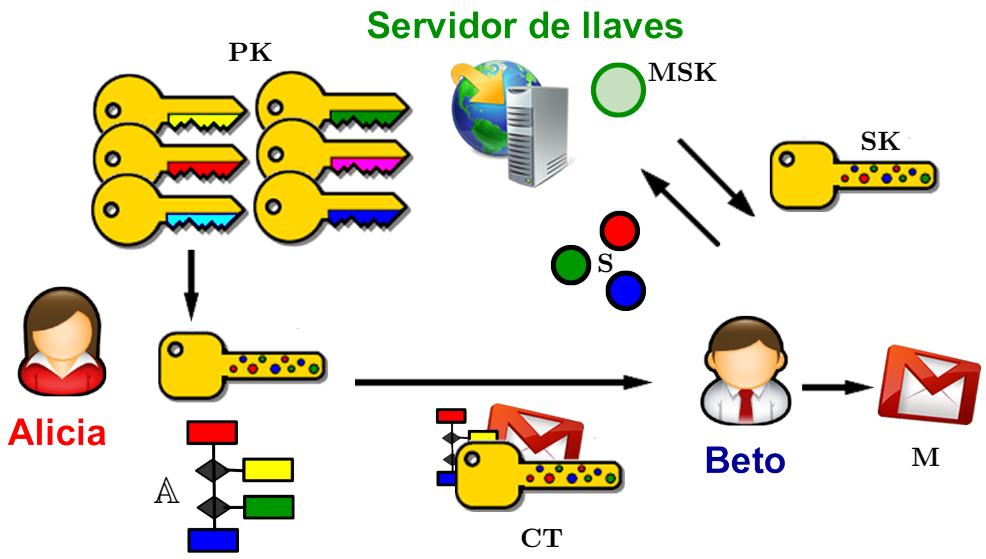


Figura 6.4: Criptografía basada en atributos

operación lógica OR, mientras que $\wedge$ es una operación lógica AND.

En general, la fórmula booleana puede ser representada a través de una estructura monótona de acceso (a la cual nos referiremos de ahora en adelante como estructura de acceso). Tanto la estructura de acceso como la fórmula pueden realizarse utilizando un esquema LSSS (linear secret-sharing scheme). Esto es importante ya que así como la fórmula booleana describe la estructura de acceso, su equivalente LSSS es utilizado para cifrar y descifrar el mensaje.

A continuación se presentan los conceptos generales sobre los esquemas LSSS de acuerdo con el trabajo descrito en [Beimel, 1996].

6.2.1. Esquemas LSSS

Definición 6.1. Estructura de Acceso monótono. *Sea* $\{P_1, P_2, \ldots, P_n\}$ *un conjunto de objetos. Una colección* $\mathcal{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}}$ *es monótona si para dos conjuntos* B, C *donde* $B \in \mathcal{A}$ *y* $B \subseteq C$, *se cumple que* $C \in \mathbb{A}$ $\forall B, C$. *Una estructura de acceso es una colección* $\mathbb{A}$ *de conjuntos no vacíos de* $\{P_1, P_2, \ldots, P_n\}$, *es decir* $\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}} \setminus \{\emptyset\}$. *Los conjuntos en* $\mathbb{A}$ *son denominados "conjuntos autorizados", mientras que los conjuntos que no están en* $\mathbb{A}$ *son llamados "conjuntos no autorizados".*

En este contexto, el rol de los objetos es tomado por los atributos, por tanto, la estructura de acceso $\mathbb{A}$ contiene el conjunto autorizado de atributos.

Definición 6.2. Esquema lineal de secreto compartido *(LSSS - Linear Secret-Sharing Scheme). Un esquema secreto* Π *sobre un conjunto de objetos* $\mathcal{P}$ *es llamado lineal (sobre* $\mathbb{F}_p$*) si:*

1. *La contribución de cada objeto forma un vector sobre* $\mathbb{F}_p$*.*
2. *Existe una matriz* M *llamada "matriz generadora de contribución" para* Π*. Suponga que* M *está formada por* ℓ *filas y* n *columnas. Para todo* $i = 1, \ldots, \ell$*, la* i*-ésima fila de* M *es etiquetada por un objeto en* $\mathcal{P}$ *como* $\rho(i)$*, donde* ρ *es una función que va desde* $\{1, \ldots, \ell\}$*. Cuando consideramos el vector columna* $v = (s, r_2, \ldots, r_n)$ *donde* $s \in \mathbb{F}_p$ *es el secreto a ser compartido y* $r_2, \ldots, r_n \in \mathbb{F}_p$ *son seleccionados aleatoriamente, entonces* Mv *es el vector de* ℓ *contribuciones del secreto* s *de acuerdo con* Π*. Donde la contribución* $(Mv)_i$ *pertenece a al grupo* $\rho(i)$*.*

De acuerdo con [Beimel, 1996], cada LSSS posee la propiedad de *reconstrucción lineal*. Suponga que Π es un LSSS para la estructura de acceso $\mathbb{A}$. Sea $S \in \mathbb{A}$ un conjunto autorizado, y defina $I \subset \{1, 2, \ldots, \ell\}$ como $I = \{i : \rho(i) \in S\}$. Entonces, existe un grupo de constantes $\{\omega_i \in \mathbb{F}_p\}_{i \in I}$ tal que para cualquier contribución válida $\{\lambda_i\}$ del secreto s de acuerdo con Π, cumple que $\sum_{i \in I} \omega_i \lambda_i = s$. Estas constantes $\{\omega_i\}$ pueden ser encontradas en tiempo polinomial de acuerdo con el tamaño de la matriz generadora de contribución M. Asimismo, dada la propiedad de LSSS, no existe ninguna constante $\{\omega_i\}$ para un conjunto no autorizado.

Ejemplo 6.1. *Sea* M *una matriz generadora de contribución de tamaño* $\ell \times n$ *y* $v = (s, r_2, \ldots, r_n)$ *el vector columna. Si representamos a la matriz* M *como:*

$$M = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\ell 1} & a_{\ell 2} & \ldots & a_{\ell n} \end{bmatrix}$$

tenemos que las contribuciones $\lambda_i = M_i v$*, están dadas como:*

$$\begin{array}{rcccc} \lambda_1 = & a_{11}s + a_{12}r_2 + & \ldots & +a_{1n}r_n \\ \lambda_2 = & a_{21}s + a_{22}r_2 + & \ldots & +a_{2n}r_n \\ \vdots & & & \\ \lambda_l = & a_{\ell 1}s + a_{\ell 2}r_2 + & \ldots & +a_{\ell n}r_n \end{array}$$

Asimismo, para realizar la reconstrucción consideramos el conjunto de $\{\omega_i\}$*, tal que* $\sum_{i \in I} \omega_i \lambda_i = s$*, entonces:*

$$\begin{array}{rccc} \omega_1[a_{11}s + a_{12}r_2 + & \ldots & +a_{1n}r_n] & + \\ \omega_2[a_{21}s + a_{22}r_2 + & \ldots & +a_{2n}r_n] & + \\ & \vdots & & + \\ \omega_l[a_{\ell 1}s + a_{\ell 2}r_2 + & \ldots & +a_{\ell n}r_n] & = s \end{array}$$

Reorganizando los elementos tenemos que:

$$\begin{array}{rl} s & [a_{11}\omega_1 + a_{21}\omega_2 + \ldots + a_{\ell 1}\omega_\ell] \quad + \\ r_2 & [a_{12}\omega_1 + a_{22}\omega_2 + \ldots + a_{\ell 2}\omega_\ell] \quad + \\ & \vdots \quad + \\ r_n & [a_{1n}\omega_1 + a_{2n}\omega_2 + \ldots + a_{\ell n}\omega_\ell] \quad = s \end{array}$$

Entonces, el problema de encontrar los elementos ω_i se reduce a resolver un sistema lineal de ecuaciones:

$$\begin{array}{rl} a_{11}\omega_1 + a_{21}\omega_2 + \ldots + a_{\ell 1}\omega_\ell & = 1 \\ a_{12}\omega_1 + a_{22}\omega_2 + \ldots + a_{\ell 2}\omega_\ell & = 0 \\ \vdots & = 0 \\ a_{1n}\omega_1 + a_{2n}\omega_2 + \ldots + a_{\ell n}\omega_\ell & = 0 \end{array}$$

el cual depende únicamente de la matriz M.

Ejemplo 6.2. *Considere la siguiente matriz de contribución:*

$$M = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \end{bmatrix}$$

De aquí se deduce que el sistema de ecuaciones está dado por:

$$\begin{array}{rrrrl} \omega_1 + & \omega_2 + & \omega_3 + & \omega_4 & = 1 \\ \omega_1 + & 2\omega_2 + & 2\omega_3 + & 2\omega_4 & = 0 \\ & & \omega_3 + & 2\omega_4 & = 0 \end{array}$$

donde una solución es:

$$\omega_1 = 2 \quad \omega_2 = 1 \quad \omega_3 = -4 \quad \omega_4 = 2$$

Como se observó en el esquema LSSS, éste requiere de la matriz generadora de contribución M y de la función de etiquetado ρ. Este par $\mathcal{M} = (M, \rho)$ se conoce como *programa de extensión monótona MSP (Monotone Span Program)* [Karchmer and Wigderson, 1993].

6.2.2. Matriz LSSS

Como se describió en la sección anterior, podemos utilizar el esquema LSSS para manejar la estructura de acceso. Para ello es necesario generar la matriz de contribución, a la cual llamaremos matriz LSSS, a partir de la fórmula booleana. Liu y Cao muestran es su trabajo [Liu and Cao, 2010], una manera de obtener la matriz LSSS, para lo cual se definen algunos conceptos.

Fórmula booleana formateada. Una fórmula booleana puede verse como un árbol de acceso, donde los nodos son compuertas lógicas, mientras que las hojas son los atributos. Una fórmula booleana puede a su vez expresarse como $(F_1, F_2, \ldots, F_n, t)$, la cual es llamada fórmula formateada. En este sentido, el nodo raíz es una compuerta-(n,t)[1] y sus hijos son $F_1, F_2, \ldots, F_n$, donde F_i puede ser un atributo u otra compuerta de la misma forma. Por ejemplo

$$\begin{aligned}((A \wedge B) \vee (B \wedge C) \vee (A \wedge C)) \wedge (A \vee D) \wedge E) \\ = ((A, B, C, 2), (A, D, 1), E, 3) \\ = (F_1, F_2, F_3, 3)\end{aligned}$$

donde $F_1 = (A, B, C, 2)$, $F_2 = (A, D, 1)$, $F_3 = E$ y $t = 3$.

Por simplicidad, la política de acceso estará dada de acuerdo a la fórmula booleana formateada descrita anteriormente. Así mismo, el etiquetado de los atributos se realizará de acuerdo a su orden de aparición en la fórmula de izquierda a derecha, por ejemplo, para la fórmula $((A, B, C, 2), (A, D, 1), E, 3)$, los atributos A, B, C, A, D, E se etiquetarán como $1, 2, 3, 4, 5, 6$ respectivamente.

Definición 6.3. Inserción de MSP. *Sea $\mathcal{A}_1$ y $\mathcal{A}_2$ dos estructuras de acceso definidas sobre un conjunto de objetos $\mathcal{P}_1$ y $\mathcal{P}_2$ respectivamente y sea $P_z \in \mathcal{P}_1$. Se define la inserción de $\mathcal{A}_2$ sobre el objeto P_z en $\mathcal{A}_1$, denotada como $\mathcal{A}_1(P_z \to \mathcal{A}_2)$, como una estructura de acceso definida sobre el conjunto $\mathcal{P}_3 = (\mathcal{P}_1 \setminus \{P_z\}) \cup \mathcal{P}_2$, tal que para $G \subseteq \mathcal{P}_3$, se tiene que:*

$$G \in \mathcal{A}_1(P_z \to \mathcal{A}_2) \iff \begin{cases} (G \cap \mathcal{P}_1) \in \mathcal{A}_1, \ \acute{o} \\ ((G \cap \mathcal{P}_1) \cup \{P_z\} \in \mathcal{A}_1 \ y \ G \cap \mathcal{P}_2 \in \mathcal{A}_2) \end{cases}$$

En otras palabras, $\mathcal{A}_1(P_z \to \mathcal{A}_2)$ es la estructura de acceso $\mathcal{A}_1$ con el objeto P_z "reemplazado" por los conjuntos de $\mathcal{A}_2$.

Teorema 6.1. *Sea $\mathcal{A}_1$ y $\mathcal{A}_2$ dos estructuras de acceso definidas sobre un conjunto de objetos $\mathcal{P}_1$ y $\mathcal{P}_2$ con los MSPs $\mathcal{M}_1$ y $\mathcal{M}_2$ respectivamente. Sea a su vez m_1 el tamaño de $\mathcal{M}_1$ y m_2 el tamaño de $\mathcal{M}_2$. Dado $P_z \in \mathcal{P}_1$, existe un MSP $\mathcal{M}$ que computa la estructura de acceso $\mathcal{A}_1(P_z \to \mathcal{A}_2)$ de tamaño igual a $m_1 + (m_2 - 1)q$, donde q es el número de filas pertenecientes a P_z en $\mathcal{M}_1$*

Sea $M^{(1)}$ y $M^{(2)}$ las matrices de las MSPs $\mathcal{M}_1$ y $\mathcal{M}_2$ respectivamente (matrices LSSS). Utilizando el teorema 6.1 y de acuerdo con [Liu and Cao, 2010], la inserción puede realizarse como sigue:

[1] esta compuerta da verdadero cuando al menos t de sus n hijos dan verdadero, por lo que $t \leq n$

Sea $q = 2$, $M^{(1)} = \begin{pmatrix} \bar{M}^{(11)} \\ \mathbf{v}_1 \\ \bar{M}^{(12)} \\ \mathbf{v}_2 \\ \bar{M}^{(13)} \end{pmatrix}$ una matriz de $m_1 \times d_1$ y $M^{(2)} = \begin{pmatrix} u^{(2)} & \tilde{M}^{(2)} \end{pmatrix}$ una matriz de $m_2 \times d_2$, donde $u^{(2)} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{m_2} \end{pmatrix}$. Dada la operación $\mathbf{v}_i \otimes u^{(2)} = \begin{pmatrix} \mathbf{v}_i u_1 \\ \mathbf{v}_i u_2 \\ \vdots \\ \mathbf{v}_i u_{m_2} \end{pmatrix}$ para $i = 1 \ldots q$. Suponga que las filas de $\mathbf{v}_1$ y $\mathbf{v}_2$ pertenecen a P_z Entonces, para realizar la inserción, primero se realiza el "reemplazo" de $\mathbf{v}_1$ con $M^{(2)}$ obteniendo la nueva matriz $M^{(1)}$ dada como:

$$M^{(1)} = \begin{pmatrix} \bar{M}^{(11)} & 0 \\ \mathbf{v}_1 \otimes u^{(2)} & \tilde{M}^{(2)} \\ \bar{M}^{(12)} & 0 \\ \mathbf{v}_2 & 0 \\ \bar{M}^{(13)} & 0 \end{pmatrix}$$

El siguiente reemplazo se realiza ahora sobre la nueva fila ($\mathbf{v}_2\mathbf{0}$) con la matriz $M^{(2)}$ obteniendo:

$$M^{(1)} = \begin{pmatrix} \bar{M}^{(11)} & 0 & 0 \\ \mathbf{v}_1 \otimes u^{(2)} & \tilde{M}^{(2)} & 0 \\ \bar{M}^{(12)} & 0 & 0 \\ \mathbf{v}_2 \otimes u^{(2)} & 0 \otimes u^{(2)} & \tilde{M}^{(2)} \\ \bar{M}^{(13)} & 0 & 0 \end{pmatrix} = \begin{pmatrix} \bar{M}^{(11)} & 0 & 0 \\ \mathbf{v}_1 \otimes u^{(2)} & \tilde{M}^{(2)} & 0 \\ \bar{M}^{(12)} & 0 & 0 \\ \mathbf{v}_2 \otimes u^{(2)} & 0 & \tilde{M}^{(2)} \\ \bar{M}^{(13)} & 0 & 0 \end{pmatrix}$$

Y ésta es la matriz equivalente del MSP que computa la estructura de acceso $\mathcal{A}_1(P_z \rightarrow \mathcal{A}_2)$.

Matriz del MSP para la estructura de acceso con umbral. Dada una estructura de acceso con umbral (n, t), es posible construir la matriz MSP-(n, t) sobre el campo $\mathbb{F}_p$ (con $p > n + 1$) como:

$$M = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 2^2 & \ldots & 2^{t-1} \\ 1 & 3 & 3^2 & \ldots & 3^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \ldots & n^{t-1} \end{pmatrix}$$

donde para cualquier $S \subseteq \{1, 2, \ldots, n\}$, $(1, 0, \ldots, 0) \in \text{span}(M_S)$ siempre que $\mid S \mid \geq t$.

El Algoritmo 6.1, describe cómo convertir una fórmula booleana formateada F en una matriz LSSS M. En este algoritmo M es una matriz de $m \times d$ definida sobre $\mathbb{F}_p$,

mientras que L es un vector de m coordenadas, donde cada coordenada corresponde a un atributo o a una fórmula formateada. En este caso, la i-ésima coordenada de L etiqueta a la i-ésima coordenada de M. Finalmente, para realizar las inserciones, se hace uso de la matriz especial MSP-(n, t).

Algoritmo 6.1 Generación de la matriz LSSS

Entrada: Fórmula booleana formateada F

Salida: Matriz LSSS M que realice la estructura de acceso de F, donde la i-ésima fila de M está etiquetada por el i-ésimo atributo de F.

1: $M \leftarrow (1)$, $L \leftarrow (F)$
2: $m \leftarrow 1$, $d \leftarrow 1$
3: **while** existan fórmulas en L **do**
4: Represente L como $L = (L_1, L_2, \ldots, L_m)$ y M como una matriz de $m \times d$.
5: Encuentre la primera coordenada en L que sea una fórmula y etiquete ésta como L_z donde $L_z = F_z = (F_{z1}, F_{z2}, \ldots, F_{zn}, t)$
6: Obtenga los n hijos de L_z, es decir $F_{z1}, F_{z2}, \ldots, F_{zn}$ y su valor umbral t.
7: Ejecute la "inserción" de la matriz especial MSP-(n, t) en la z-ésima fila de M, obteniendo la nueva matriz de tamaño $(m - 1 + n) \times (d - 1 + t)$.
8: $L \leftarrow (L_1, L_2, \ldots, L_{z-1}, F_{z1}, F_{z2}, \ldots, F_{zn}, L_{z+1}, \ldots L_m)$
9: $m \leftarrow m - 1 + n$
10: $d \leftarrow d - 1 + t$
11: **end while**
12: **return** M

Ejemplo 6.3. **[Liu and Cao, 2010]**. *Dado el campo $\mathbb{F}_p = \mathbb{F}_{47}$, considere la siguiente estructura de acceso:*

$$((A \wedge B) \vee (B \wedge C) \vee (C, D, E, 2)) \wedge (E, F, G, H, 3)$$

A partir de la estructura de acceso se construye la fórmula booleana formateada:

$$\begin{gathered}((A \wedge B) \vee (B \wedge C) \vee (C, D, E, 2)) \wedge (E, F, G, H, 3) \\ = ((B \wedge (A \vee C)) \vee (C, D, E, 2)) \wedge (E, F, G, H, 3) \\ = (((B, (A, C, 1), 2), (C, D, E, 2), 1), (E, F, G, H, 3), 2)\end{gathered}$$

Tomando la fórmula $(((B, (A, C, 1), 2), (C, D, E, 2), 1), (E, F, G, H, 3), 2)$ como entrada, el algoritmo funciona como sigue:

1. $M = (1)$, $L = (((B, (A, C, 1), 2), (C, D, E, 2), 1), (E, F, G, H, 3), 2)$
2. $M = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$, $L = \begin{pmatrix} ((B, (A, C, 1), 2), (C, D, E, 2), 1) \\ (E, F, G, H, 3) \end{pmatrix}$
3. $M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$, $L = \begin{pmatrix} (B, (A, C, 1), 2) \\ (C, D, E, 2) \\ (E, F, G, H, 3) \end{pmatrix}$

4. $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \\ 1 & 2 & 0 \end{pmatrix}$, $L = \begin{pmatrix} B \\ (A, C, 1) \\ (C, D, E, 2) \\ (E, F, G, H, 3) \end{pmatrix}$

5. $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \\ 1 & 2 & 0 \end{pmatrix}$, $L = \begin{pmatrix} B \\ A \\ C \\ (C, D, E, 2) \\ (E, F, G, H, 3) \end{pmatrix}$

6. $M = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 2 \\ 1 & 1 & 0 & 3 \\ 1 & 2 & 0 & 0 \end{pmatrix}$, $L = \begin{pmatrix} B \\ A \\ C \\ C \\ D \\ E \\ (E, F, G, H, 3) \end{pmatrix}$

7. $M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 3 & 0 & 0 \\ 1 & 2 & 0 & 0 & 1 & 1 \\ 1 & 2 & 0 & 0 & 2 & 4 \\ 1 & 2 & 0 & 0 & 3 & 9 \\ 1 & 2 & 0 & 0 & 4 & 16 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \end{pmatrix}$, $L = \begin{pmatrix} B \\ A \\ C \\ C \\ D \\ E \\ E \\ F \\ G \\ H \end{pmatrix}$

Si $S_1 = \{A, B, F, G, H\}$ *es un conjunto autorizado, tenemos que las filas pertenecientes a* S_1 *son las correspondientes a* $\{v_1, v_2, v_8, v_9, v_{10}\}$*. Para estas filas las constantes de reconstrucción son* $\{\omega_1 \equiv 4,\ \omega_2 \equiv -2,\ \omega_8 \equiv -6,\ \omega_9 \equiv 8,\ \omega_{10} \equiv -3\}$ *mod* 47*.*

Para el caso donde $S_2 = \{D, E, F, G\}$ *es un conjunto autorizado,* S_2 *es propietario de las filas* $\{v_5, v_6, v_7, v_8, v_9\}$*, mientras que sus constantes de reconstrucción son* $\{\omega_5 \equiv 6,\ \omega_6 \equiv -4,\ \omega_7 \equiv -3,\ \omega_8 \equiv 3,\ \omega_9 \equiv -1\}$ *mod* 47*.*

6.3. Proyección de una cadena a un punto

Como se mostró en la criptografía basada en la identidad, es posible utilizar una cadena de texto arbitraria S como la llave pública de un usuario. Para ello, es necesario convertir esta cadena a un objeto que pueda ser utilizado en el ámbito de

emparejamientos, en este caso, un punto sobre una curva elíptica.

Tanto en el esquema IBE como en el esquema ABE, la proyección de la cadena se realiza hacia un punto P en el grupo aditivo $\mathbb{G}_1$, para ello, se utiliza la función de picadillo $H : \{0,1\}^* \to \mathbb{G}_1$, donde la cadena de texto S es vista como una secuencia de bits de longitud variable.

Sea H_1 una función de picadillo conocida[2] $H_1 : \{0,1\}^* \to \mathbb{F}_p$ y sea $E/\mathbb{F}_p$ una curva elíptica definida como $E : y^2 = x^3 + b$. Definamos la función denominada **MapToPoint** como:

$$\begin{aligned} H_2 : \mathbb{F}_p &\to \mathbb{G}_1 \\ x &\to P \in E(\mathbb{F}_p) \end{aligned}$$

la cual puede ser implementada de acuerdo con el Algoritmo 6.2.

Algoritmo 6.2 Función MapToPoint [Boneh et al., 2004]

Entrada: $x \in \mathbb{F}_p$
Salida: $P \in E(\mathbb{F}_p)$

1: $i \leftarrow 0$
2: **while** $i > p$ **do**
3: $\quad x_0 \leftarrow x + i$
4: $\quad c \leftarrow x^3 + b$
5: $\quad$ **if** c es un residuo cuadrático **then**
6: $\quad\quad y_0 \leftarrow c^{1/2}$
7: $\quad\quad$ **return** $P = (x_0, y_0)$
8: $\quad$ **end if**
9: **end while**
10: **return** $\mathcal{O}$

Entonces, la función de picadillo $H : \{0,1\}^* \to \mathbb{G}_1$ para realizar la proyección de una cadena a un punto, se define como $H = H_2(H_1(S))$.

6.4. Algoritmos

Ahora que se tiene conocimiento de los esquemas LSSS y la proyección de la cadena a un punto, se detallarán los algoritmos descritos en la sección 6.1 para el esquema ABE, a través del uso de la multiplicación escalar y emparejamientos bilineales sobre curvas elípticas.

6.4.1. Inicialización

Este algoritmo toma como entrada el parámetro de seguridad λ y el universo de atributos U. A partir del parámetro de seguridad se definen los grupos $\mathbb{G}_1$ y $\mathbb{G}_2$ de

[2]En el caso de esta tesis se ocupó la función de picadillo SHA-2

orden primo r, así como los generadores $P \in \mathbb{G}_1$ y $Q \in \mathbb{G}_2$. Se generan los puntos $H_1, H_2, \ldots, H_U \in \mathbb{G}_1$ a partir al universo de atributos y finalmente se seleccionan dos enteros aleatorios $a, \alpha \in \mathbb{F}_r$.

La llave pública se publica como:

$$\text{PK} = \quad P, Q, e(Q,P)^{\alpha}, aP, H_1, \ldots, H_U$$

Mientras que la autoridad establece $\text{MSK} = \alpha P$ como la llave secreta maestra.

Este algoritmo tiene un costo de un emparejamiento, dos multiplicaciones escalares y U funciones MapToPoint.

6.4.2. Cifrado

El algoritmo de cifrado toma como entrada los parámetros públicos PK y el mensaje $\mathcal{M}$ a ser cifrado. Además, toma como entrada una estructura de acceso LSSS (M, ρ), donde la función ρ asocia las filas de M con los atributos. Sea M una matriz de $\ell \times n$. El algoritmo selecciona un vector aleatorio $v = (s, y_2, \ldots, y_n) \in \mathbb{Z}_r^n$, los cuales serán utilizados para compartir el exponente s. Para $i = 1$ hasta ℓ, se calcula $\lambda_i = v \cdot M_i$, donde M_i es el vector correspondiente a la i-ésima fila de M. Además, el algoritmo elige de manera aleatoria los enteros $r_1, \ldots, r_\ell \in \mathbb{Z}_r$.

Entonces la cifra CT es publicada como:

$$\text{CT} = \begin{cases} C = \mathcal{M} e(Q,P)^{\alpha s}, C' = sQ, \\ (C_1 = \lambda_1(aP) - r_1 H_{\rho(1)}, D_1 = r_1 Q), \ldots, (C_\ell = \lambda_\ell(aP) - r_\ell H_{\rho(\ell)}, D_1 = r_\ell Q) \end{cases}$$

la cual es enviada junto con la descripción de (M, ρ).

En este algoritmo se conocen previamente los puntos Q, P y aP, por lo que puede utilizarse el método *comb* para el cálculo de las multiplicaciones escalares aplicadas a éstos puntos (véase sección 4.6.2). De la misma manera, puede aplicarse este método para la exponenciación del vector $e(Q,P)^{\alpha}$ por el entero s. En el caso de los puntos H_i, a pesar de ser puntos conocidos, no se consideró la implementación utilizando el método *comb*, esto debido a que el universo de atributos tiene un tamaño variable, lo cual se ve reflejado directamente en el costo del almacenamiento en el precómputo. Para este caso y dado que los puntos H_i pertenecen al grupo $\mathbb{G}_1$ se utilizó el método GLV (véase sección 4.6.3).

Entonces, el costo de este algoritmo es de una exponenciación con vector conocido en $\mathbb{G}_T$, una multiplicación con punto conocido en $\mathbb{G}_2$, ℓ multiplicaciones con punto conocido en $\mathbb{G}_1$, ℓ multiplicaciones en el grupo $\mathbb{G}_2$, ℓ multiplicaciones en el grupo $\mathbb{G}_1$ y una multiplicación en el grupo $\mathbb{G}_T$

6.4.3. Generación de llaves

El algoritmo de generación de llaves toma como entrada la llave maestra MSK = αP y un conjunto S de atributos. El algoritmo primero selecciona un número aleatorio $t \in \mathbb{F}_r$, entonces crea la llave privada como:

$$\text{SK} = \{\ K = \alpha P + t(aP),\quad L = tQ,\quad \forall x \in S\ K_x = tH_x\ \}$$

Sea n el número de atributos en S, dado que los puntos aP y Q son conocidos, tenemos que el costo de este algoritmo es de 1 multiplicación con punto conocido en $\mathbb{G}_1$, 1 multiplicación con punto conocido en $\mathbb{G}_2$, n multiplicaciones en $\mathbb{G}_1$ y 1 suma de puntos en $\mathbb{G}_1$. Para las dos primeras multiplicaciones se utilizó el método *comb*, mientras que para las demás se utilizó el método GLV.

6.4.4. Descifrado

El algoritmo de descifrado toma como entradas la cifra CT para la estructura de acceso (M, ρ) y la llave privada SK para el conjunto S. Suponga que S satisface la estructura de acceso y defina $I \subset \{1, 2, \ldots, \ell\}$, como $I = \{i : \rho(i) \in S\}$. Entonces, sean $\{\omega_i \in \mathbb{Z}\}_{i \in I}$ el conjunto de constantes tales que si λ_i es una contribución válida de cualquier secreto s conforme a M, entonces $\sum_{i \in I} \omega_i \lambda_i = \Delta s$ para $\Delta \in \mathbb{F}_r$.

El algoritmo primero calcula:

$$\left(e(L, \textstyle\sum_{i \in I} \omega_i C_i) \prod_{i \in I} e(D_i, \omega_i K_{\rho(i)})\right)^{\frac{1}{\Delta}} / e(C', K) = e(P, Q)^{-\alpha s}$$

Finalmente toma este valor y lo multiplica por C obteniendo así el mensaje $\mathcal{M}$.

En este algoritmo, la variable Δ garantiza que las contantes ω_i sean enteros de longitud pequeña, es decir, $\mid \omega_i \mid < 64$ bits, por lo que sus multiplicaciones pueden calcularse utilizando un método de ventana (ω-NAF). A este tipo de multiplicaciones las denominaremos *multiplicaciones cortas*

Si definimos n como el número de elementos en I, entonces este algoritmo tiene un costo de $2n$ multiplicaciones cortas en $\mathbb{G}_1$, $n + 2$ emparejamientos, una inversión en $\mathbb{G}_T$ (la cual es un conjugado), $n + 2$ multiplicaciones en $\mathbb{G}_T$ y una exponenciación en el grupo $\mathbb{G}_T$, la cual puede realizarse utilizando el método GS descrito en la sección 5.7.

Justificación del Algoritmo

El algoritmo de descifrado está dado por la siguiente ecuación:

$$\left(e(L, \sum_{i \in I} \omega_i C_i) \prod_{i \in I} e(D_i, \omega_i K_{\rho(i)})\right)^{\frac{1}{\Delta}} / e(C', K) \tag{6.1}$$

En el cifrado tenemos que:

$$\begin{aligned} C &= \mathcal{M}e(Q,P)^{\alpha s} \\ C' &= sQ \\ C_i &= \lambda_i(aP) - r_i H_{\rho(i)} \\ D_i &= r_i Q) \end{aligned} \tag{6.2}$$

Mientras que la llave privada se tiene que:

$$\begin{aligned} K &= \alpha P + t(aP) = (\alpha + at)P \\ L &= tQ \\ K_i &= tH_i \end{aligned} \tag{6.3}$$

Entonces, sustituyendo los valores de las ecuaciones (6.2) y (6.3) en los emparejamientos de la ecuación (6.1) obtenemos:

$$\begin{aligned} e(L, \textstyle\sum_{i\in I} \omega_i C_i) &= e(tQ, \textstyle\sum_{i\in I} \omega_i(\lambda_i aP - r_i H_{\rho(1)})) \\ &= e(tQ, \Delta asP - \textstyle\sum_{i\in I} \omega_i r_i H_{\rho(i)}) \\ e(D_i, \omega_i K_{\rho(i)}) &= e(r_i Q, \omega_i t H_{\rho(i)}) \\ e(C', K) &= e(sQ, (\alpha + at)P) \end{aligned}$$

Dadas las propiedades de bilinealidad en los emparejamientos tenemos que:

$$\begin{aligned} e(L, \textstyle\sum_{i\in I} \omega_i C_i) &= e(tQ, \Delta asP) e(tQ, -\textstyle\sum_{i\in I} \omega_i r_1 H_{\rho(1)}) \\ &= e(Q,P)^{\Delta ast} e(Q, \textstyle\sum_{i\in I} \omega_i r_i H_{\rho(i)})^{-t} \\ &= e(Q,P)^{\Delta ast} \textstyle\prod_{i\in I} e(Q, \omega_i r_i H_{\rho(i)})^{-t} \\ &= e(Q,P)^{\Delta ast} \textstyle\prod_{i\in I} e(Q, H_{\rho(i)})^{-\omega_i r_i t} \\ e(D_i, \omega_i K_{\rho(i)}) &= e(Q, H_{\rho(i)})^{r_i \omega_i t} \\ e(C', K) &= e(Q,P)^{\alpha s} e(Q,P)^{ast} \end{aligned} \tag{6.4}$$

Sustituyendo los emparejamientos de la ecuación (6.4) en (6.1) obtenemos:

$$\left(e(L, \textstyle\sum_{i\in I} \omega_i C_i) \prod_{i\in I} e(D_i, \omega_i K_{\rho(i)})\right)^{\frac{1}{\Delta}} / e(C', K)$$

$$= \left(e(Q,P)^{\Delta ast} \textstyle\prod_{i\in I} e(Q, H_{\rho(i)})^{-\omega_i r_i t} \prod_{i\in I} e(Q, H_{\rho(i)})^{r_i \omega_i t}\right)^{\frac{1}{\Delta}} / \left(e(Q,P)^{\alpha s} e(Q,P)^{ast}\right)$$
$$= e(Q,P)^{ast} / \left(e(Q,P)^{\alpha s} e(Q,P)^{ast}\right)$$

$$= e(Q,P)^{-\alpha s}$$

6.4.5. Delegación

En algunos sistemas puede ser necesario que un usuario necesite proporcionar acceso a otro, como es en el caso de algunos sistemas distribuidos. En este contexto, el esquema ABE permite la delegación de atributos en las llaves entre dos usuarios de acuerdo al algoritmo de delegación.

El algoritmo de delegación de llaves toma como entrada los parámetros públicos, la llave privada $\mathrm{SK} = \{K, L, \forall x \in S\ K_x\}$ y un conjunto de atributos $S' \subset S$. Entonces el algoritmo selecciona un número aleatorio $s \in \mathbb{F}_r$, creando la nueva llave SK' para el conjunto de atributos S' como:

$$\mathrm{SK}' = \{\ K' = K + s(aP),\quad L = L + sQ,\quad \forall x \in S'\ K'_x = K_x + sH_x\ \}$$

Podemos demostrar que SK' es una llave válida para el conjunto S'. De acuerdo con el algoritmo de generación de llaves tenemos que:

$$\mathrm{SK} = \{\ K = \alpha P + t(aP),\quad L = tQ,\quad \forall x \in S\ K_x = tH_x\ \}$$

Entonces, SK' está dado por:

$$\begin{aligned}\mathrm{SK}' = &\ \{\ K' = \alpha P + t(aP) + s(aP),\quad L = tQ + sQ,\quad \forall x \in S'\ K'_x = tH_x + sH_x\ \}\\ = &\ \{\ K' = \alpha P + (t+s)(aP),\quad L = (t+s)Q,\quad \forall x \in S'\ K'_x = (t+s)H_x\ \}\end{aligned}$$

Sea $u = (t + s)$, entonces:

$$\mathrm{SK}' = \{\ K' = \alpha P + u(aP),\quad L = uQ,\quad \forall x \in S'\ K'_x = uH_x\ \}$$

Por tanto SK' es una llave válida de acuerdo con el algoritmo de generación de llaves.

Sea n el número de atributos en S', dado que P y Q son parámetros previamente conocidos, tenemos que el costo del algoritmo de delegación de llaves es de 1 multiplicación con punto conocido en $\mathbb{G}_1$, 1 multiplicación con punto conocido en $\mathbb{G}_2$, n multiplicaciones en $\mathbb{G}_1$, $n + 1$ sumas de punto en $\mathbb{G}_1$ y n sumas de punto en el grupo $\mathbb{G}_2$. Como ya se mencionó anteriormente, las multiplicaciones con punto conocido pueden realizarse utilizando el método *comb*, mientras que para las demás puede utilizarse el método GLV.

Capítulo 7

Implementación y resultados

"La diferencia entre la teoría y la práctica es que, en teoría,
no hay diferencia entre la teoría y la práctica"
Richard Moore

Como se mencionó a lo largo de estos capítulos, el objetivo principal de esta tesis es la implementación eficiente de la criptografía basada en atributos en un dispositivo móvil. Uno de los puntos importantes para lograr este objetivo, es la implementación de las funciones criptográficas involucradas de manera eficiente, tales como la multiplicación escalar de puntos (capítulo 4), el emparejamiento bilineal (capítulo 5) y la proyección de una cadena a un punto (capítulo 6). Dado que estas funciones están implementadas sobre campos finitos, es necesario que también su aritmética (capítulo 3) sea desarrollada eficientemente.

En este capítulo se presentan los detalles de implementación de la biblioteca criptográfica sobre emparejamientos y curvas elípticas, así como su uso en el esquema ABE.

7.1. Implementación en Cortex-A9

A lo largo de esta tesis se desarrolló una biblioteca sobre un procesador Cortex-A9, el cual corresponde a la arquitectura ARMv7 utilizada en los dispositivos móviles actuales. Uno de los puntos importantes de esta arquitectura es que el procesador trabaja con palabras de 32 bits y a su vez permite el uso de las instrucciones NEON[1].

La biblioteca entonces fue desarrollada bajo el lenguaje de programación C, la cual fue utilizada sobre una aplicación en android, por lo que para su compilación se utilizó el NDK[2] (Native Development Kit) de android. La figura 7.1 muestra el modelo de implementación de la biblioteca desarrollada. En las secciones posteriores se detalla

[1]http://www.arm.com/products/processors/technologies/neon.php (04/10/2012)
[2]http://developer.android.com/tools/sdk/ndk/index.html (4/10/2012)

la implementación de la aritmética utilizando las instrucciones NEON así como las técnicas implementadas de reducción displicente y *multiemparejamiento*.

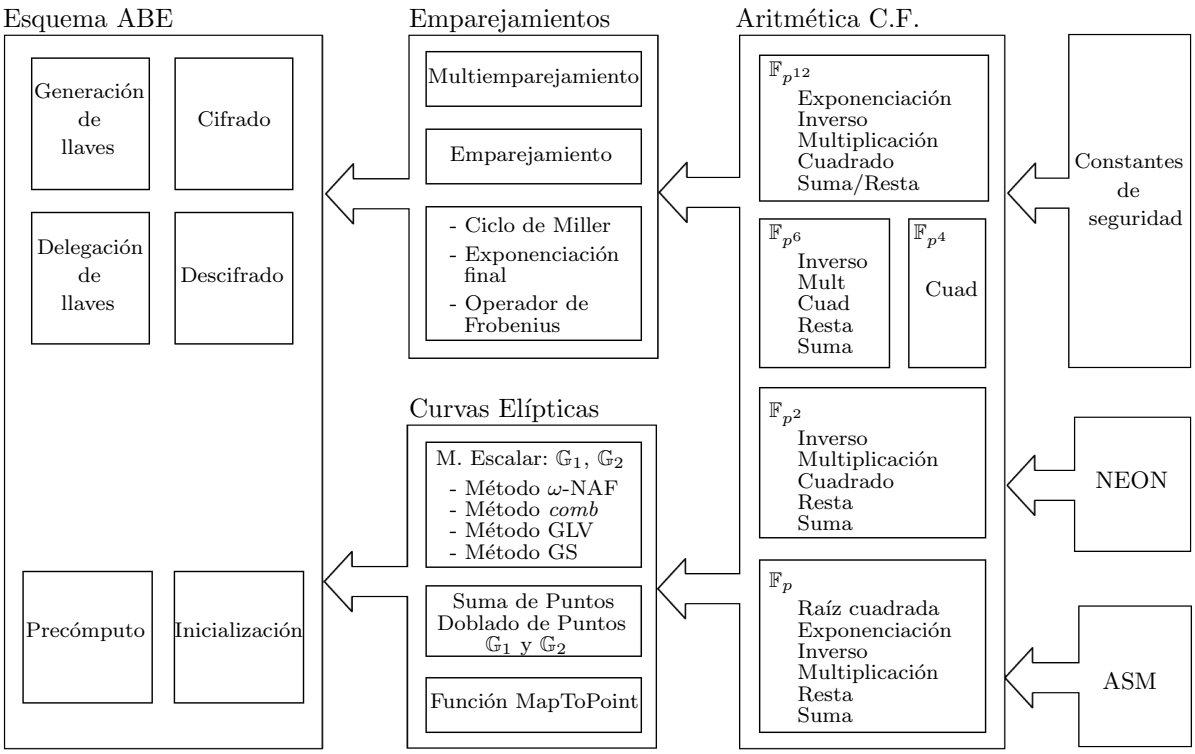


Figura 7.1: Modelo de implementación

7.1.1. Uso de instrucciones NEON

La tecnología NEON es una arquitectura de 128 bits que trabaja sobre el modelo SIMD (Single Instruction, Multiple Data), la cual es una extensión para la serie de procesadores ARM Cortex™-A. Esta arquitectura consta de 32 registros de 64 bits (*doubleword*), los cuales pueden verse como 16 registros de 128 bits (*quadword*).

Las instrucciones NEON realizan un procesamiento "SIMD comprimido" (véase figura 7.2) de manera que:

- Los registros son considerados como vectores de elementos de un mismo tipo de dato.
- Los tipos de datos pueden ser de: 8 bits, 16 bits, 32 bits, 64 bits y de punto flotante de precisión simple. Todos ellos con signo o sin signo.
- Las instrucciones ejecutan la misma operación en todas las líneas.

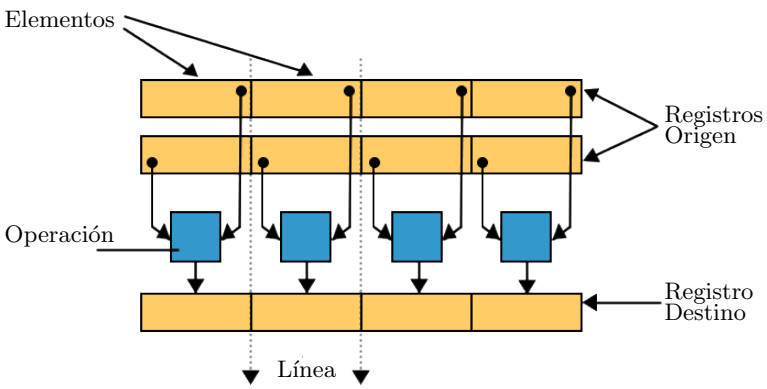


Figura 7.2: Procesamiento de las instrucciones NEON

Entre las operaciones de NEON se encuentran:

- Suma y resta.
- Multiplicación con acumulación opcional
- Máximos y mínimos correspondientes a operaciones de selección de línea.
- Aproximación de la raíz cuadrada inversa.
- Instrucciones para el cargado de estructuras datos.

Existen dos formas para utilizar la tecnología NEON, la primera es utilizando las instrucciones directamente en lenguaje ensamblador, y la otra es a través de las funciones y tipos de datos *intrinsic*. En esta tesis se optó por la segunda opción, ésto debido a que las funciones *intrinsic* proporcionan una funcionalidad similar a la de utilizar código ensamblador incrustado en otro código de más alto nivel, y tienen la ventaja de que sus funciones son similares a las funciones en C o C++, con la diferencia de que éstas son reemplazadas durante la compilación por una secuencia de instrucciones de bajo nivel.

Cada función *intrinsic* tiene la siguiente estructura:

$$< nombre\_operacion >< bandera > \_ < tipo\_de\_dato >$$

donde la bandera es un dato opcional denotado por q, el cual indica que la operación trabaja con vectores de 128 bits, en caso de no aparecer dicha bandera, la operación trabaja con vectores de 64 bits. Algunos ejemplos son:

uint32x2_t **vadd_u32** (uint32x2_t, uint32x2_t)
int32x2_t **vsub_s32** (int32x2_t, int32x2_t)
uint8x16_t **vandq_u8** (uint8x16_t, uint8x16_t)

donde la primera instrucción realiza dos sumas de enteros sin signo de 32 bits, la segunda instrucción realiza dos restas de enteros con signo de 32 bits, mientras que la tercera realiza la operación lógica AND sobre 16 enteros sin signo de 8 bits, es decir, trabaja con un vector de 128 bits.

En este trabajo las principales instrucciones utilizadas fueron:

uint64x2_t vmull_u32 (uint32x2_t, uint32x2_t)
uint64x2_t vmlal_u32 (uint64x2_t, uint32x2_t, uint32x2_t)

donde la primera instrucción nos permite realizar dos multiplicaciones enteras de 32 bits cuyo resultado se guarda en registros de 64 bits, mientras que la segunda nos permite además sumar un entero de 64 bits a cada multiplicación.

Estas instrucciones fueron utilizadas debido a la ventaja que representan, ya que hacen posible realizar dos multiplicaciones por instrucción, sin embargo, el cargado y almacenamiento de las variables suele ser muy costoso, ya que implica pasar de la arquitectura NEON a la arquitectura ARM, es decir, pasar de los registros de 64/128 bits a los registros de 32 bits. Esto suele ser costoso ya que para realizar esta operación debe buscarse un conjunto de registros consecutivos de 32 bits que puedan almacenar los 64/128 bits de los registros de la arquitectura NEON.

Por tanto, para tomar una ventaja real de las instrucciones NEON es conveniente evitar el uso compartido de variables entre ambas arquitecturas, lo cual puede lograrse si no existe una dependencia en los datos utilizados. Debido a ello, las instrucciones NEON no pudieron ser implementadas en el multiplicador en $\mathbb{F}_p$, sin embargo, tratando de mejorar la aritmética en $\mathbb{F}_{p^2}$, se propuso en esta tesis la implementación de dos multiplicaciones independientes en $\mathbb{F}_p$ utilizando la tecnología NEON, lo cual resultó ser menos costoso que implementarlas por separado.

En este sentido y para el campo finito $\mathbb{F}_p$ con $|p| = 256$ bits, se definieron 3 funciones a través de la tecnología NEON:

1. mul_{NEON}. Realiza dos multiplicaciones independientes en el campo $\mathbb{F}_p$ utilizando el método CIOS (véase sección 3.1.2). Para ello, esta función recibe como entrada 4 elementos en el campo primo, y entrega como salida 2 elementos en $\mathbb{F}_p$, es decir, dados $a, b, c, d, f, g \in \mathbb{F}_p$, se define la función como:

   $$(f, g) \leftarrow mul_{NEON}(a, b, c, d)$$

   donde $f = a \cdot b \bmod p$ y $g = c \cdot d \bmod p$.

2. $mule_{NEON}$. Realiza dos multiplicaciones enteras. Esta función recibe como entrada 4 elementos de 256 bits y obtiene como salida dos elementos de 512 bits, es decir:

   $$(F, G) \leftarrow mule_{NEON}(a, b, c, d)$$

con $a, b, c, d \in \mathbb{F}_p$, $F = a \cdot b$ y $G = c \cdot d$, donde $|G| = |F| = 512$ bits.

3. red_{NEON}. Realiza la reducción módulo p de acuerdo al método SOS (véase sección 3.1.2 en la página 20). Esta función recibe como entradas 2 argumentos de 512 bits y regresa como salida 2 elementos en el campo $\mathbb{F}_p$, como se muestra a continuación:
$$(f, g) \leftarrow red_{NEON}(F, G)$$
donde $f, g \in \mathbb{F}_p$ y $|G| = |F| = 512$ bits.

Estas funciones fueron implementadas en los algoritmos de la aritmética en $\mathbb{F}_{p^2}$ (véase sección 3.2, página 28). Los Algoritmos 7.1 y 7.2 muestran el uso de las funciones NEON en los algoritmos de multiplicación y elevación al cuadrado en $\mathbb{F}_{p^2}$ respectivamente. Para estos casos se consideró $\beta = -1$

Algoritmo 7.1 Multiplicación optimizada en $\mathbb{F}_{p^2}$ (NEON)

Entrada: $A = a_0 + a_1 u, \quad B = b_0 + b_1 u \in \mathbb{F}_{p^2}$
Salida: $C = A \cdot B \in \mathbb{F}_{p^2}$
1: $s \leftarrow a_0 + a_1$
2: $t \leftarrow b_0 + b_1$
3: $(d_0, d_1) \leftarrow mule_{NEON}(s, t, a_0, b_0)$
4: $d_2 \leftarrow \text{mul256}(a_1, b_1)$
5: $d_0 \leftarrow d_0 - d_1 - d_2$
6: $d_1 \leftarrow d_1 - d_2$
7: $(c_1, c_0) \leftarrow red_{NEON}(d_0, d_1)$
8: **return** $C = c_0 + c_1 u$

Algoritmo 7.2 Elevación al cuadrado en $\mathbb{F}_{p^2}$ (NEON)

Entrada: $A = a_0 + a_1 u \in \mathbb{F}_{p^2}$
Salida: $C = A^2 \in \mathbb{F}_{p^2}$
1: $c_0 \leftarrow a_0 - a_1$
2: $c_2 \leftarrow a_0 + a_1$
3: $(c_1, c_0) \leftarrow mul_{NEON}(a_0, a_1, c_0, c_2)$
4: $c_1 \leftarrow 2c_1$
5: $c_0 \leftarrow c_0 - c_1$
6: **return** $C = c_0 + c_1 u$

Estas funciones, también fueron utilizadas en el doblado y suma de puntos en coordenadas jacobianas[3] bajo el grupo $\mathbb{G}_1$, esto debido a la independencia de las multiplicaciones de sus algoritmos. En este sentido, los Algoritmos 7.3 y 7.4 describen el doblado y suma de puntos utilizando la tecnología NEON, para este caso se consideró $a = 0$.

[3]véase sección 4.3.2

Algoritmo 7.3 Doblado de punto en coordenadas jacobianas (NEON)

Entrada: $P = (X_1 : Y_1 : Z_1) \in \mathbb{G}_1$
Salida: $2P = (X_3 : Y_3 : Z_3) \in \mathbb{G}_1$
1: $(t_1, t_4) \leftarrow mul_{NEON}(Y_1, Y_1, 3X_1, X_1)$
2: $(t_2, t_3) \leftarrow mul_{NEON}(4X_1, t_1, 4t_1, 2t_1)$
3: $t_3 \leftarrow 2t_3$
4: $(X_3, Z_3) \leftarrow mul_{NEON}(t_4, t_4, 2Y_1, Z_1)$
5: $X_3 \leftarrow X_3 - 2t_2$
6: $Y_3 \leftarrow t_4 \cdot (t_2 - X_3) - t_3$
7: **return** $(X_3 : Y_3 : Z_3)$

Algoritmo 7.4 Suma de puntos en coordenadas mixtas (NEON)

Entrada: $P = (X_1 : Y_1 : Z_1)$ y $Q = (X_2 : Y_2 : 1) \in \mathbb{G}_1$
Salida: $R = P + Q = (X_3 : Y_3 : Z_3) \in \mathbb{G}_1$
1: $t_1 \leftarrow Z_1^2$
2: $(t_2, t_3) \leftarrow mul_{NEON}(Z_1, t_1, X_2, t_1)$
3: $t_5 \leftarrow t_3 - X_1$
4: $(t_4, t_7) \leftarrow mul_{NEON}(Y_2, t_2, t_5, t_5)$
5: $(t_8, t_9) \leftarrow mul_{NEON}(t_7, t_5, t_7, X_1)$
6: $t_6 \leftarrow t_4 - Y_1$
7: $(X_3, Z_3) \leftarrow mul_{NEON}(t_6, t_6, Z_1, t_5)$
8: $X_3 \leftarrow X_3 - (t_8 + 2t_9)$
9: $(Y_3, t_0) \leftarrow mul_{NEON}(t_6, t_9 - X_3, Y_1, t_8)$
10: $Y_3 \leftarrow Y_3 - t_0$
11: **return** $(X_3 : Y_3 : Z_3)$

7.1.2. Reducción displicente

Como se observó en la sección 3.1.2, una multiplicación modular puede separarse en dos partes, una que realiza la multiplicación entera $t = a \cdot b$ y otra que realiza la operación de reducción t mod p. En este sentido, la reducción displicente es una técnica que permite un ahorro en las reducciones de las operaciones pertenecientes a las extensiones del campo $\mathbb{F}_p$.

Sea $k = 2^i 3^j$, $i, j \in \mathbb{Z}^+$, de acuerdo con el teorema 1 de [Aranha et al., 2011a], dado $a, b \in \mathbb{F}_{p^k}$, el producto $c = a \cdot b$, $c \in \mathbb{F}_{p^k}$ puede ser computado con $3^i 6^j$ multiplicaciones enteras y $2^i 3^j$ reducciones módulo p para cualquier k. Esta disminución en el número de reducciones se conoce como *reducción displicente*.

El Algoritmo 3.15 de la página 31 muestra un claro ejemplo del uso de la reducción displicente, ya que para el campo $\mathbb{F}_{p^2}$ donde $k = 2^1 3^0$, tenemos que el costo del algoritmo es de tres multiplicaciones enteras y dos reducciones módulo p.

El Algoritmo 7.5 muestra un ejemplo de cómo se implementaría esta técnica en la multiplicación en $\mathbb{F}_{p^6}$, para este caso suponga que $\times_2$ corresponde a una multiplicación entera en $\mathbb{F}_{p^2}$, mientras que red_2 indica una reducción en $\mathbb{F}_{p^2}$. Cabe destacar que para realizar las multiplicaciones enteras basta con retirar la operación de reducción en los algoritmos de multiplicación, asegurando en el análisis que no ocurren ceros de desbordamiento.

Algoritmo 7.5 Multiplicación en $\mathbb{F}_{p^6}$ (reducción displicente)

Entrada: $A = a_0 + a_1V + a_2V^2, \quad B = b_0 + b_1V + b_2V^2 \in \mathbb{F}_{p^6}$
Salida: $C = A \cdot B \in \mathbb{F}_{p^6}$

1: $v_0 \leftarrow a_0 \times_2 b_0$
2: $v_1 \leftarrow a_1 \times_2 b_1$
3: $v_2 \leftarrow a_2 \times_2 b_2$
4: $C_0 \leftarrow ((a_1 + a_2) \times_2 (b_1 + b_2) - v_1 - v_2) \cdot \xi + v_0$
5: $C_1 \leftarrow (a_0 + a_1) \times_2 (b_0 + b_1) - v_0 - v_1 + \xi \cdot v_2$
6: $C_2 \leftarrow (a_0 + a_2) \times_2 (b_0 + b_2) - v_0 - v_2 + v_1$
7: $c_0 \leftarrow red_2(C_0)$
8: $c_1 \leftarrow red_2(C_1)$
9: $c_2 \leftarrow red_1(C_2)$
10: **return** $C = c_0 + c_1V + c_2V^2$

Podemos corroborar que el Algoritmo 7.5 cumple con el teorema **??**, ya que cada multiplicación $\times_2$ tiene un costo de tres multiplicaciones enteras, mientras que cada operación red_2 tiene un costo de 2 reducciones, por tanto, el costo de este algoritmo es de 18 multiplicaciones enteras y 6 reducciones.

Cabe destacar que en la biblioteca desarrollada, se utilizó reducción displicente en las operaciones de multiplicación y elevación al cuadrado sobre todas las extensiones de campo. Así mismo, se aplicó esta técnica en las evaluaciones de la línea en el ciclo de Miller (véase 5.4 de la página 67) y en la suma de puntos de una curva elíptica.

7.1.3. Multiemparejamiento

En el esquema ABE, observamos que el algoritmo de descifrado (véase sección 6.4.4) requiere de un producto de emparejamientos, este producto puede realizarse de una forma más eficiente si se comparten las funciones involucradas en la implementación de los emparejamientos. En este caso, observamos que las operaciones que pueden ser compartidas son aquellas que no se ven involucradas directamente por los elementos del grupo $\mathbb{G}_1$ y $\mathbb{G}_2$, es decir, la elevación al cuadrado en el ciclo de Miller y la exponenciación final. Este método se conoce como multiemparejamiento, el cual se presenta en el Algoritmo 7.6.

Otro método para mejorar el desempeño del emparejamiento puede realizarse si se tiene previo conocimiento de un elemento Q en el grupo $\mathbb{G}_2$, ya que los resultados de

Algoritmo 7.6 Multiemparejamiento para curvas BN

Entrada: $P_1, P_2, \ldots, P_n \in \mathbb{G}_1$, $Q_1, Q_2, \ldots, Q_n \in \mathbb{G}_2$
Salida: $\prod_{i=1}^{n} e(Q_i, P_i)$

1: Escribir $s = 6z + 2$ como $s = \sum_{j=0}^{l-1}$ con $s_j \in \{-1, 0, 1\}$
2: $f \leftarrow 1$
3: **for** $i = 0 \rightarrow n$ **do**
4: $\quad T_i \leftarrow Q_i$
5: **end for**
6: **for** $j = l - 2 \rightarrow 0$ **do**
7: $\quad f \leftarrow f^2$
8: $\quad$ **for** $i = 0 \rightarrow n$ **do**
9: $\quad\quad f \leftarrow f \cdot \ell_{T_i,T_i}(P_i)$, $T_i \leftarrow 2T_i$
10: $\quad\quad$ **if** $s_j = 1$ **then**
11: $\quad\quad\quad f \leftarrow f \cdot \ell_{T_i,Q_i}(P_i)$, $T_i \leftarrow T_i + Q_i$
12: $\quad\quad$ **else if** $s_j = -1$ **then**
13: $\quad\quad\quad f \leftarrow f \cdot \ell_{T_i,-Q_i}(P_i)$, $T_i \leftarrow T_i - Q_i$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end for**
17: **for** $i = 0 \rightarrow n$ **do**
18: $\quad R_1 \leftarrow \pi(Q_i)$
19: $\quad R_2 \leftarrow \pi^2(Q_i)$
20: $\quad f \leftarrow f \cdot \ell_{T_i,R_1}(P)$, $T_i \leftarrow T_i + R_1$
21: $\quad f \leftarrow f \cdot \ell_{T_i,-R_2}(P)$, $T_i \leftarrow T_i - R_2$
22: **end for**
23: $g \leftarrow f^{(p^{12}-1)/r}$
24: **return** g

las funciones las líneas pueden ser computadas previamente, por lo que únicamente se realiza la evaluación de éstas en el punto $P \in \mathbb{G}_1$, donde la evaluación de cualquier línea $\ell_{\cdot,\cdot} = l_0 + l_1 w + l_2 w^3$ en el punto $P = (x_P, y_P)$ estaría dado como:

$$\ell_{\cdot,\cdot}(P) = l_0 y_P + l_1 x_P w + l_2 w^3$$

En este trabajo se utilizaron ambas técnicas en el algoritmo de descifrado del esquema ABE (sección 6.4.4 de la página 92), donde L es el punto conocido de la llave de usuario SK.

7.2. Complejidad computacional

En esta sección se presentan los costos de las funciones implementadas en la biblioteca; éstos se encuentran divididos en tres partes: la primera describe el esfuerzo computacional para efectuar el emparejamiento, la segunda los costos para realizar la

multiplicación escalar, y la última presenta la complejidad computacional para efectuar los algoritmos del esquema ABE. Tanto el costo del emparejamiento como el costo de la multiplicación escalar se dará en términos de la aritmética en $\mathbb{F}_{p^2}$, esto debido a que las operaciones sobre el campo $\mathbb{F}_p$ se ven afectadas directamente por el uso de la tecnología NEON.

7.2.1. Costo del emparejamiento

Para calcular el costo total del emparejamiento, separaremos los costos en base a las funciones involucradas en su implementación, es decir, el ciclo de Miller y la exponenciación final.

Un punto importante en el cálculo de costos es el uso de la técnica de reducción displicente, por lo que fueron consideradas las operaciones de multiplicación entera, cuadrado entero y reducción en $\mathbb{F}_{p^2}$, las cuales fueron denotadas como $\tilde{m}_E$, $\tilde{s}_E$ y $\tilde{r}_E$ respectivamente, donde $\tilde{m} = \tilde{m}_E + \tilde{r}_E$ y $\tilde{s} = \tilde{s}_E + \tilde{r}_E$.

Campo	Suma	Multiplicación	Cuadrado	Inversión
$\mathbb{F}_{p^2}$	$\tilde{a} = 2a$	$\tilde{m} = 3m_E + 2r_E + 8a + m_\beta$	$\tilde{s} = 2m_E + 2r_E + 3a + m_\beta$	$\tilde{i} = 2m_E + 1r_E + 2m + m_\beta + 2a + i$
$\mathbb{F}_{p^4}$	$2\tilde{a}$		$3\tilde{s} + 1m_\xi + 4\tilde{a}$	
$\mathbb{F}_{p^6}$	$3\tilde{a}$	$6\tilde{m}_E + 3\tilde{r}_E + 2m_\xi + 24\tilde{a}$	$2\tilde{m} + 3\tilde{s} + 2m_\xi + 9\tilde{a}$	$3\tilde{m} + 6\tilde{m}_E + 3\tilde{s}_E + 4\tilde{r}_E + 4m_\xi + 10\tilde{a} + \tilde{i}$
$\mathbb{F}_{p^{12}}$	$6\tilde{a}$	$18\tilde{m}_E + 6\tilde{r}_E + 7m_\xi + 96\tilde{a}$	$12\tilde{m}_E + 6\tilde{r}_E + 6m_\xi + 63\tilde{a}$	$3\tilde{m} + 22\tilde{m}_E + 9\tilde{s}_E + 13\tilde{r}_E + 13m_\xi + 79\tilde{a} + \tilde{i}$
$\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$			$9\tilde{s} + 4m_\xi + 30\tilde{a}$	$3\tilde{a}$

Tabla 7.1: Resumen de costos en las extensiones de campo de $\mathbb{F}_{p^{12}}$ (reducción displicente)

La tabla 7.1 muestra los costos funcionales en la aritmética de las torres de campo utilizando reducción displicente, la cual es una modificación a la tabla 3.2 presentada en el capítulo 3. En este caso, las sumas por elementos de 512 bits se consideran como 2 sumas en el campo $\mathbb{F}_{p^2}$, mientras que la multiplicación por γ es equivalente a la multiplicación por ξ.

Ciclo de Miller

Para estimar los costos del ciclo de Miller en el Algoritmo 5.9, debemos estimar los costos de las operaciones involucradas en su implementación, los cuales se presentan en la tabla 7.2. En esta tabla se muestran los costos de la multiplicación dispersa, el doblado de puntos y evaluación de la línea tangente, la suma de puntos y evaluación de la línea secante, así como el costo del operador de Frobenius sobre el punto Q.

Operación	**Costo**
Multiplicación dispersa $f \cdot \ell_{\cdot,\cdot}(P)$	$13\tilde{m}_E + 6\tilde{r}_E + 3m_\xi + 50\tilde{a}$
Doblado de un punto y evaluación de la línea tangente. $\ell_{T,T}(P)$ y $2T$	$3\tilde{m} + 4\tilde{s} + 2\tilde{s}_E + 1\tilde{r}_E + 19\tilde{a} + 4m$
Suma de un punto y evaluación de la línea secante. $\ell_{T,Q}(P)$ y $T + Q$	$7\tilde{m} + 2\tilde{s} + 4\tilde{m}_E + 2\tilde{r}_E + 10\tilde{a} + 4m$
Operador de Frobenius en Q $\pi(Q)$ o $\pi^2(Q)$	$4m$

Tabla 7.2: Resumen de costos de las operaciones del ciclo de Miller

Ahora bien, para el caso de este trabajo se seleccionó el parámetro $z = -2^{62} - 2^{55} - 1$, por lo que el valor $s = 6z + 2$ de la línea 1 del Algoritmo 5.9, está dado por: $s = -(2^{64} + 2^{63} + 2^{57} + 2^{56} + 2^2)$, el cual es un número con signo de 65 bits y un peso de Hamming de 5 bits. Para evitar el signo, basta con efectuar un conjugado al final del ciclo de miller, al cual tiene un costo de $3\tilde{a}$. Entonces, el costo del ciclo de Miller en términos de funciones es:

Operaciones del ciclo de Miller	$= 63$	(elevaciones al cuadrado en $\mathbb{F}_{p^{12}}$ + multiplicaciones dispersas evaluaciones de línea y doblado de punto) +
	5	(multiplicaciones dispersas + evaluaciones de línea y suma de puntos) +
	2	(operadores frobenius + multiplicaciones dispersas + evaluaciones de línea y suma de puntos)+ 1 conjugado en $\mathbb{F}_{p^{12}}$.

Por lo que su costo funcional de acuerdo con las tablas 7.1 y 7.2 es:

$$\begin{aligned}\text{Costo del ciclo de Miller} &= 63\,(3\tilde{m} + 4\tilde{s} + 25\tilde{m}_E + 2\tilde{s}_E + 13\tilde{r}_E + 9m_\xi + 132\tilde{a} + 4m) + \\ &\quad 5\,(7\tilde{m} + 2\tilde{s} + 17\tilde{m}_E + 8\tilde{r}_E + 3m_\xi + 60\tilde{a} + 4m) + \\ &\quad 2\,(7\tilde{m} + 2\tilde{s} + 17\tilde{m}_E + 8\tilde{r}_E + 3m_\xi + 60\tilde{a} + 8m) + 3\tilde{a} \\ &= 238\tilde{m} + 266\tilde{s} + 1694\tilde{m}_E + 126\tilde{s}_E + 875\tilde{r}_E + 588m_\xi + 8739\tilde{a} + 288m\end{aligned}$$

Ahora bien, cuando se tiene un punto conocido (P.C.) en el grupo $\mathbb{G}_2$, el costo de la evaluación de la líneas es de $4m$ y no se ejecutan los operadores frobenius, por lo que el costo final del ciclo de Miller es de:

$$1666\tilde{m}_E + 798\tilde{r}_E + 588m_\xi + 7469\tilde{a} + 280m$$

Exponenciación final

Para obtener el costo de la exponenciación final (E.F.), se contabilizaron las operaciones del Algoritmo 5.7, por lo que su costo operacional en el campo $\mathbb{F}_{p^{12}}$ es:

$$\begin{array}{rcl} \text{Costo de la E.F.} & = & 3 \text{ exponenciaciones por } z + 12 \text{ multiplicaciones} + \\ & & 3 \text{ elevaciones al cuadrado en } \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2}) + \\ & & 4 \text{ operadores de Frobenius} + \\ & & 1 \text{ inverso} + 1 \text{ conjugado} \end{array}$$

Para estimar el esfuerzo computacional de las exponenciaciones por $z = -(2^{62} + 2^{55}+1)$, utilizamos el Algoritmo 5.8. En este caso, se toma el absoluto de z y posteriormente se aplica un conjugado para obtener el inverso. La complejidad computacional para calcular cada cuadrado ciclotómico $\mathcal{C}$ es:

$$6\tilde{s} + 3m_\xi + 24\tilde{a}$$

Para el caso de la línea 15 del Algoritmo 5.8, se requiere de dos descompresiones $\mathcal{D}$, donde cada una tiene un costo de:

$$3\tilde{s} + 3\tilde{m} + 2m_\xi + 12\tilde{a} + \tilde{i}$$

sin embargo, utilizando el truco de Montgomery para inversiones simultáneas [Montgomery, 1987], tenemos que la complejidad de la descompresión de ambas líneas es:

$$9\tilde{m} + 6\tilde{s} + 4m_\xi + 24\tilde{a} + \tilde{i}$$

Finalmente se requieren de dos multiplicaciones en $\mathbb{F}_{p^{12}}$ por las descompresiones obtenidas en las líneas $14-16$ del Algoritmo 5.8, por lo que el costo de una exponenciación por z es:

$$\begin{array}{rl} & 62\ (6\tilde{s} + 3m_\xi + 24\tilde{a}) + 2(18\tilde{m}_E + 6\tilde{r}_E + 7m_\xi + 96\tilde{a}) + (9\tilde{m} + 6\tilde{s} + 4m_\xi + 24\tilde{a} + \tilde{i}) \\ = & 9\tilde{m} + 378\tilde{s} + 36\tilde{m}_E + 12\tilde{r}_E + 204m_\xi + 1704\tilde{a} + \tilde{i} \end{array}$$

De acuerdo con los Algoritmos 5.4, 5.5 y 5.5 observamos que la complejidad computacional de aplicar los operadores de Frobenius es de $5\tilde{m}$, sin embargo, de acuerdo a los parámetros seleccionados el costo fue de $5\tilde{m}+\tilde{a}$, $8m+\tilde{a}$ y $3\tilde{m}+4\tilde{a}$ para p, p^2 y p^3 respectivamente. Utilizando lo anterior y aplicando los datos de la tabla 7.1, podemos estimar el esfuerzo computacional de la exponenciación final como:

$$\begin{array}{rcl} \text{Costo de la E.F.} & = & 3\ (9\tilde{m} + 378\tilde{s} + 36\tilde{m}_E + 12\tilde{r}_E + 204m_\xi + 1704\tilde{a} + \tilde{i}) + \\ & & 12\ (18\tilde{m}_E + 6\tilde{r}_E + 7m_\xi + 96\tilde{a}) + \\ & & 3\ (9\tilde{s} + 4m_\xi + 30\tilde{a}) + (8\tilde{m} + 8m + 6\tilde{a}) + \\ & & 3\tilde{m} + 22\tilde{m}_E + 9\tilde{s}_E + 13\tilde{r}_E + 13m_\xi + 79\tilde{a} + \tilde{i} + 3\tilde{a} \\ & = & 38\tilde{m} + 1161\tilde{s} + 346\tilde{m}_E + 9\tilde{s}_E + 121\tilde{r}_E + 721m_\xi + 8m + 6442\tilde{a} + 4\tilde{i} \end{array}$$

Entonces, el costo del emparejamiento dado por el ciclo de miller y la exponenciación final es:

$$276\tilde{m} + 1427\tilde{s} + 2040\tilde{m}_E + 135\tilde{s}_E + 996\tilde{r}_E + 1309m_\xi + 15181\tilde{a} + 288m + 4\tilde{i}$$

Ahora bien, cuando el emparejamiento tiene un punto conocido (P. C.), tenemos su costo se reduce a:

$$51\tilde{m} + 1161\tilde{s} + 2012\tilde{m}_E + 9\tilde{s}_E + 919\tilde{r}_E + 1309m_\xi + 13911\tilde{a} + 288m + 4\tilde{i}$$

La tabla 7.3 muestra los tiempo de ejecución de las operaciones involucradas en el emparejamiento, los cuales fueron medidos sobre dos dispositivos móviles.

		ARMv7 Cortex A9 Dual Core			
Campo	Operación	Galaxy Tab 10.1 nVidia Tegra 2	Galaxy Note Exynos 4		
		@1.0Ghz	@1.4Ghz	@1.4Ghz NEON	$\times 1.4 \sim$ @1.0Ghz NEON
$\mathbb{F}_{p^2}$	add	0.169	0.121	0.113	0.158
	sub	0.145	0.103	0.110	0.154
	mul	3.410	2.443	1.811	2.535
	cua	2.405	1.720	1.433	2.006
	inv	39.25	28.01	26.85	37.59
$\mathbb{F}_{p^6}$	add	0.529	0.376	0.358	0.501
	sub	0.456	0.337	0.346	0.484
	mul	20.65	14.76	12.12	16.97
	cua	16.13	11.55	9.387	13.14
	inv	76.05	53.84	47.01	65.81
$\mathbb{F}_{p^{12}}$	add	1.027	0.732	0.700	0.980
	sub	0.883	0.668	0.668	0.935
	mul	62.16	44.33	36.10	50.54
	cua	44.79	31.89	26.04	36.46
	inv	146.0	104.7	87.55	122.6
$\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$	cua	26.07	18.84	13.49	18.89
	inv	0.589	0.454	0.485	0.679
Ciclo de Miller		8313	5963	4807	6730
Exponenciación Final		5269	3291	2891	4047
Emparejamiento		13582	9727	7698	10777
Emparejamiento P.C.		11560	8375	6701	9382

Tabla 7.3: Tiempos de ejecución en μs del emparejamiento y funciones asociadas

En la segunda columna se muestran los tiempos de operación tomados sobre la tableta *Samsung Galaxy Tab 10.1* con procesador **nVidia Tegra 2@1Ghz**, la segunda muestra la ejecución sobre el celular *Samsung Galaxy Note* con procesador **Exynos 4@1.4Ghz**, la tercera columna muestra los tiempos sobre el mismo celular utilizando la tecnología NEON, mientras que la última columna muestra un aproximado en microsegundos para el caso que el procesador del celular tuviera una frecuencia de 1Ghz.

7.2.2. Costo de la multiplicación escalar y exponenciación en $\mathbb{G}_T$

Debido a que tanto la multiplicación escalar de puntos como la exponenciación ocupan las mismas técnicas para su implementación (véase sección 4.6, página 53), el análisis de la complejidad computacional de ambas son presentados en esta sección. Para ello, primero se deben conocer las operaciones involucradas para cada caso. En el caso de la multiplicación escalar tenemos la suma y doblado de puntos, cuyos costos se presentan en la tabla 7.4; mientras que para la exponenciación en $\mathbb{G}_T$ tenemos la multiplicación en $\mathbb{F}_{p^{12}}$ y la elevación al cuadrado en el grupo ciclotómico $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$, cuyos costos pueden verse en la tabla 7.1.

Grupo	**Doblado de Puntos (D)**	**Suma de Puntos (A)**	**Inverso (I)**
$\mathbb{G}_1$	$7m + 10a$	$11m + 7a$	a
$\mathbb{G}_2$	$3\tilde{m} + 4\tilde{s} + 10\tilde{a}$	$5\tilde{m} + 6\tilde{s} + 7\tilde{a}$	$\tilde{a}$

Tabla 7.4: Costo de las operaciones de la multiplicación escalar de puntos

La tabla 7.5 muestra los costos aproximados para la multiplicación escalar y exponenciación en $\mathbb{G}_T$ utilizando los métodos: ω-NAF, *comb*, GLV y GS para un escalar y exponente de 256 bits. Note que para el caso de la exponenciación se utilizó la notación de la tabla 3.1.

Grupo	**Operación**	**Costo funcional**	**Costo operacional**
$\mathbb{G}_1$	ω-NAF ($\omega = 3$)	$65A + 257D$	$2514m + 3025a$
	GLV ($\omega = 3$)	$66A + 130D$	$1636m + 1570a$
	comb ($\omega = 8$)	$31A + 31D$	$558m + 527a$
$\mathbb{G}_2$	ω-NAF ($\omega = 3$)	$65A + 257D$	$1096\tilde{m} + 1418\tilde{s} + 3023\tilde{a}$
	GS ($\omega = 3$)	$68A + 68D$	$544\tilde{m} + 680\tilde{s} + 1148\tilde{a}$
	comb ($\omega = 8$)	$31A + 31D$	$248\tilde{m} + 310\tilde{s} + 527\tilde{a}$
$\mathbb{G}_T$	ω-NAF ($\omega = 3$)	$65\tilde{M} + 257\tilde{\delta} + 2C$	$2313\tilde{s} + 1170\tilde{m}_E + 390\tilde{r}_E + 1483n_\xi + 13956\tilde{a}$
	GS ($\omega = 3$)	$68\tilde{M} + 68\tilde{\delta} + 8C$	$612\tilde{s} + 1224\tilde{m}_E + 408\tilde{r}_E + 748n_\xi + 8592\tilde{a}$
	comb ($\omega = 8$)	$31\tilde{M} + 31\tilde{\delta}$	$279\tilde{s} + 558\tilde{m}_E + 186\tilde{r}_E + 341n_\xi + 3906\tilde{a}$

Tabla 7.5: Costo de la multiplicación escalar en $\mathbb{G}_1$, $\mathbb{G}_2$ y exponenciación en $\mathbb{G}_T$

La tabla 7.6 muestra los tiempos de implementación de las funciones implementadas para la multiplicación escalar de puntos en $\mathbb{G}_1$ y $\mathbb{G}_2$ así como la exponenciación en el grupo $\mathbb{G}_T$. De la misma forma que en la tabla 7.3, los costos están expresados en micro-segundos sobre dos dispositivos móviles.

		ARMv7 Cortex A9 Dual Core			
Campo	Operación	Galaxy Tab 10.1 nVidia Tegra 2	Galaxy Note Exynos 4		
		@1.0Ghz	@1.4Ghz	@1.4Ghz NEON	×1.4 ∼ @1.0Ghz NEON
$\mathbb{G}_1$	Multiplicación				
	NAF	3010	2144	1890	2657
	GLV	1977	1409	1247	1746
	Comb	626	448	393	550
$\mathbb{G}_2$	Multiplicación				
	NAF	8036	5716	4578	6409
	GS	4190	2992	2348	3287
	Comb	1745	1244	970	1358
$\mathbb{G}_T$	Exponenciación				
	NAF	10299	7316	5607	7850
	GS	5998	4273	3412	4094
	Comb	2727	1940	1539	2155

Tabla 7.6: Tiempos de implementación de la multiplicación escalar y exponenciación

7.2.3. Costo del multiemparejamiento

De acuerdo con el Algoritmo 7.6, observamos que para $n = 1$, el multiemparejamiento es equivalente al de un emparejamiento, ahora bien, para $n > 1$ tenemos que por cada emparejamiento extra, se omite la exponenciación final y el costo de la elevación al cuadrado en el ciclo de Miller (véase Algoritmo 5.1). La tabla 7.7 muestra los costos por cada elemento en el multiemparejamiento.

Número de Emparejamientos	Costo	Aproximación en $\mathbb{F}_p$
$n = 1$	$280\tilde{m}+1436\tilde{s}+2148\tilde{m}_E+135\tilde{s}_E+996\tilde{r}_E+1309m_\xi+15175\tilde{a}+288m+4\tilde{i}$	$10652m + 32968a + 4i$
$n > 1$ + (c/u)	$238\tilde{m}+266\tilde{s}+938\tilde{m}_E+126\tilde{s}_E+497\tilde{r}_E+210m_\xi+4767\tilde{a}+288m$	$4474m + 9954$
+ P.C. (c/u)	$910\tilde{m}_E+420\tilde{r}_E+210m_\xi+3500\tilde{a}+280m$	$2100m + 7420a$

Tabla 7.7: Costo del Multiemparejamiento

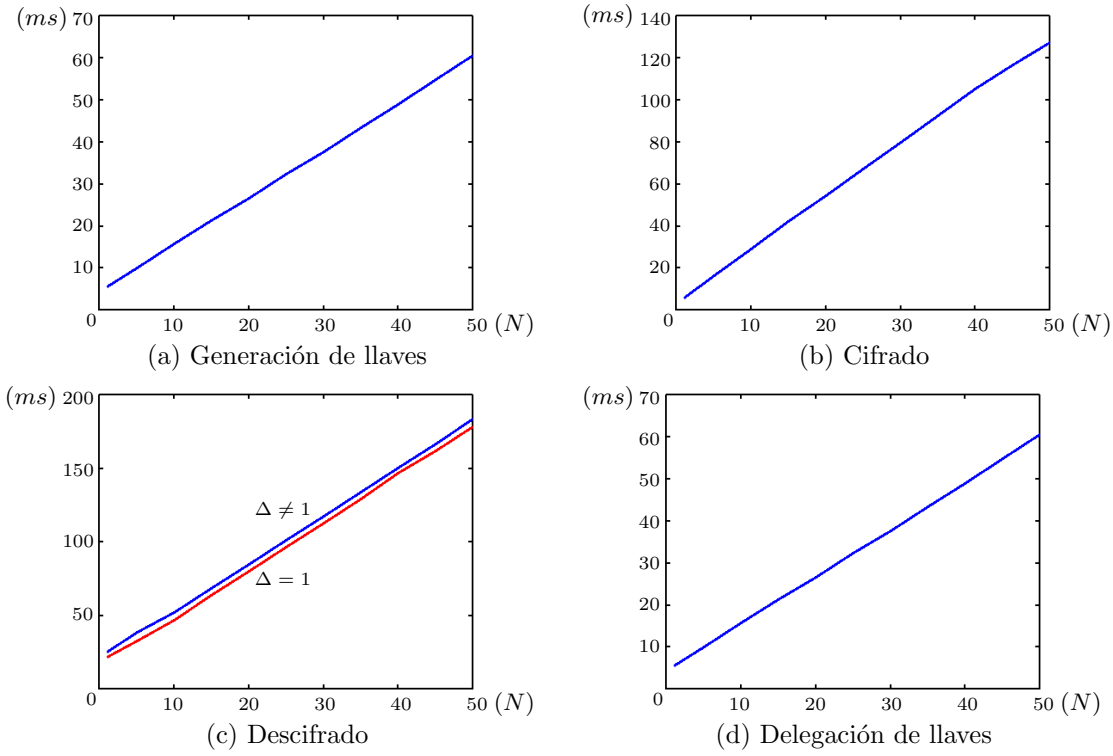
7.2.4. Costo del esquema ABE

Los costos del esquema ABE se contabilizaron de acuerdo a sus algoritmos: generación de llaves, delegación de llaves, cifrado y descifrado. El costo de la inicialización no fue considerado debido a que éste se realiza una sola vez. La tabla 7.8 muestra un resumen de los costos funcionales de los algoritmos de acuerdo con la sección 6.4 y la siguiente notación:

e	Emparejamiento	M_e^n	Multiemparejamiento de n elementos
Sm	Multiplicación escalar (ω-NAF) en $\mathbb{G}_1$	SM	Multiplicación escalar (ω-NAF) en $\mathbb{G}_2$
Sm_C	Multiplicación escalar P.C. en $\mathbb{G}_1$	SM_C	Multiplicación escalar P.C. en $\mathbb{G}_2$
Sm_G	Multiplicación escalar (GLV) en $\mathbb{G}_1$	SM_G	Multiplicación escalar (GS) en $\mathbb{G}_2$
E	Exponenciación (ω-NAF)	E_C	Exponenciación (*comb*)
E_S	Exponenciación (GS)	ℓ	Número de Atributos
A_1	Adición de Puntos en $\mathbb{G}_1$	A_2	Adición de Puntos en $\mathbb{G}_2$
$\tilde{M}$	Multiplicación en $\mathbb{F}_{p^{12}}$	C	Conjugado

Operación	**Costo**
Generación de llaves	$Sm_C + SM_C + \ell \cdot Sm_G + A_1$
Cifrado	$E_C + SM_C + \ell \cdot SM_G + \ell \cdot Sm_C + \ell \cdot Sm_G + \tilde{M}$
Descifrado	$2\ell \cdot m + e + M_e^{\ell+1} + \tilde{M} + C$
Delegación de llaves	$Sm_C + SM_C + \ell \cdot Sm_G + (\ell + 1) \cdot A_1 + \ell \cdot A_2$

Tabla 7.8: Costo funcional de los algoritmos del esquema ABE



(a) Generación de llaves (b) Cifrado

(c) Descifrado (d) Delegación de llaves

Figura 7.3: Tiempos de ejecución (ms) vs. número de atributos (N)

La figura 7.3 muestra las gráficas de tiempos de los cuatro algoritmos sobre el dispositivo Galaxy Note (ARM v7) Exynos 4 Cortex-A9 a 1.4 GHz utilizando la tecnología NEON.

Cabe aclarar que en la tabla 7.8 tanto la operación m, como la operación M, se realizan sobre escalares de un tamaño menor de 64 bits, por lo que sus costos con respecto a las tablas 7.5 y 7.6 serían de aproximadamente la cuarta parte de aquellos. Así mismo, la exponenciación en el algoritmo de descifrado puede o no realizarse, ya que depende directamente de la política de acceso.

7.3. Comparación con otras implementaciones

A pesar del auge de los esquemas basados en emparejamientos bilineales, no se tienen muchas implementaciones de éstos sobre dispositivos móviles. En esta tesis se presenta la comparación del emparejamiento con dos trabajos: [Acar et al., 2011] y [Grewal et al., 2012]. En el primero se desarrolla una biblioteca con coordenadas afines, mientras que el segundo se utilizan coordenadas proyectivas. La tabla 7.9 muestra los costos comparativos de la biblioteca desarrollada en ambos trabajos.

Trabajo	**Procesador**	**Tiempos de operación** $[\mu s]$ **para** 254 **bits**						
		$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML	FE	O-$\hat{a}$(.)
[Acar et al., 2011]	Tegra 2[a]	1.42	8.18	5.20	26.61	26,320	24,690	51,010
[Grewal et al., 2012]	Apple A5[b]	0.25	3.48	2.88	19.19	8,338	5,483	13,821
	TI OMAP[c]	0.13	2.81	2.11	14.05	6,859	4,382	11,241
	(ASM)	0.10	2.46	2.07	13.79	6,147	3,758	9,905
Este trabajo	Tegra 2[a]	0.17	3.41	2.41	39.25	8,313	5,269	13,582
	Exynos[d]	0.12	2.44	1.72	28.01	5,963	3,291	9,727
	(NEON)	0.11	1.81	1.43	26.85	4,807	2,891	7,698

a. NVidia Tegra 2 (ARM v7) Cortex-A9 a 1.0 GHz (C)

b. iPad 2 (ARM v7) Apple A5 Cortex-A9 a 1.0 GHz (C)

c. Galaxy Nexus (ARM v7) TI OMAP 4460 Cortex-A9 a 1.2 GHz (Dos versiones: C y ASM

d. Galaxy Note (ARM v7) Exynos 4 Cortex-A9 a 1.4 GHz (Dos versiones: C y NEON)

Tabla 7.9: Comparativa de los tiempos medidos con la literatura

Para hacer una comparación más equitativa, la tabla 7.10 muestra el aproximado en ciclos de reloj para el emparejamiento con la mejor implementación de cada trabajo.

Trabajo	**Ciclo de Miller**	**Exponenciación Final**	**Emparejamiento**
[Acar et al., 2011]	26,320	24,690	51,010
[Grewal et al., 2012] (ASM)	7,376	4,510	11,886
Este Trabajo (NEON)	6,730	4,047	10,777

Tabla 7.10: Comparativa con la literatura con aproximación en ciclos de reloj (1×10^3)

En cuanto al esquema ABE sobre dispositivos móviles, se encontró el trabajo de [Akinyele et al., 2010], sin embargo, en este trabajo únicamente se implementa el algoritmo de descifrado y no presenta las cifras exactas de los tiempos de ejecución. Sin embargo, con el objetivo de hacer futuras comparaciones y de manera análoga al trabajo [Scott, 2011], medimos los tiempos para 6 atributos sobre cada función como se presenta en la tabla 7.11. Note que se tomaron los casos para $\Delta = 1$ y para $\Delta > 1$.

Procesador	**Generación de llaves**	**Cifrado**	**Descifrado**	**Delegación de llaves**
NVidia Tegra 2 (ARM v7) Cortex-A9 a 1.0 GHz				
($\Delta = 1$)	18.34	31.83	63.87	18.44
($\Delta > 1$)			74.14	
Galaxy Note (ARM v7) Exynos 4 Cortex-A9 a 1.4 GHz				
($\Delta = 1$)	13.05	22.68	45.58	13.16
($\Delta > 1$)		52.81		
Galaxy Note (ARM v7) Exynos 4 Cortex-A9 a 1.4 GHz (NEON)				
($\Delta = 1$)	11.52	18.89	36.62	11.57
($\Delta > 1$)			42.22	

Tabla 7.11: Tiempos en milisegundos del esquema ABE para 6 atributos

7.4. Aplicación móvil

Como se mostró anteriormente, se desarrolló una biblioteca que implementa el esquema ABE. Ahora bien, para darle un uso a la biblioteca se realizó una aplicación sobre el sistema operativo Android. Ésta aplicación se basó en el programa PGP (*Pretty Good Privacy*)[4], el cual permite proteger la información distribuida a través de Internet mediante el uso de criptografía de llave pública.

De la misma manera que en el programa PGP, se construyó un sistema híbrido, es decir, se utilizó tanto criptografía simétrica como asimétrica, con la diferencia que en el caso de la criptografía asimétrica, se utilizó el esquema ABE, por lo que el acceso al documento se realiza a través de atributos y no por identidades.

Para proteger un documento, éste se cifra utilizando criptografía simétrica con una llave de sesión. Finalmente se protege la llave de sesión utilizando un esquema asimétrico, que en este caso sería el esquema de criptografía basada en atributos, en donde cualquier usuario tendría acceso a la llave de sesión si y sólo si, cumple con los atributos necesarios que satisfagan una política de acceso.

En el caso de la criptografía simétrica se utilizó el estándar AES, el cual se encuentra proporcionado por las bibliotecas del kit de desarrollo de android SDK. Mientras

[4]http://www.ietf.org/rfc/rfc4880.txt (04/10/2012)

que para el caso de la criptografía asimétrica se utilizó la biblioteca desarrollada en este trabajo. Para tener un intermediario entre la aplicación y la biblioteca se utilizó el kit de desarrollo nativo (NDK) de android. En el apéndice B se presenta la instalación del ambiente de desarrollo, mientras que en el apéndice C se presenta el modelo de capas utilizado en la aplicación.

De acuerdo con lo anterior, se desarrolló una aplicación como prueba de concepto del esquema ABE, la cual utiliza los principales atributos involucrados en el Departamento de Computación, tales como: investigadores, estudiantes, administrativos, etc. Por tanto, la aplicación proporciona un medio de administración de llaves, así como la posibilidad de realizar el cifrado y el descifrado de documentos dentro del dispositivo móvil, de acuerdo a una política dada. El apéndice D presenta el manual de usuario de la aplicación junto con la interfaz gráfica utilizada para cada caso.

Capítulo 8

Conclusiones

"Si piensas que la criptografía es la solución a tu problema,
es que realmente no conoces tu problema"
Peter G. Neumann

En este trabajo se estudió, analizó e implementó el esquema de criptografía basada en atributos (ABE) sobre una plataforma especialmente diseñada para dispositivos móviles. Para ello, se desarrolló una biblioteca de software eficiente que permite el cálculo de las operaciones criptográficas involucradas, tales como el emparejamiento bilineal, multiplicación escalar de puntos y la proyección de una cadena a un punto.

8.1. Resumen de resultados

En este capítulo se presentan las principales conclusiones obtenidas a lo largo del desarrollo de este trabajo, las cuales, a manera de resumen se describen a continuación:

- Una biblioteca criptográfica eficiente debe garantizar que los esquemas desarrollados sobre ésta sean a su vez efectivos y eficientes, por lo que, en esta tesis se diseñó e implementó una biblioteca que realiza las funciones criptográficas utilizadas por el esquema ABE, de manera que fueran desarrolladas eficientemente de acuerdo a la literatura actual y que a su vez, aprovechara las características de los procesadores utilizados en los dispositivos móviles. En el caso de este trabajo, se utilizó la tecnología NEON en los procesadores ARM Cortex-A9, la cual permite realizar múltiples operaciones utilizando una solo instrucción.
- La biblioteca criptográfica desarrollada genera tres funciones principales: el emparejamiento bilineal, la multiplicación escalar de puntos y la proyección de una cadena a un punto (véase figura 7.1, página 96). Estas funciones son las base de varios sistemas criptográficos basados en emparejamientos y dependen directamente de la aritmética de campos finitos. En el caso del emparejamiento, es

necesaria una torre de campos sobre $\mathbb{F}_{p^{12}}$, mientras que para las curvas elípticas definidas sobre $\mathbb{G}_1$ y $\mathbb{G}_2$ fue necesaria la aritmética en $\mathbb{F}_p$ y $\mathbb{F}_{p^2}$ respectivamente.

- La implementación eficiente de la aritmética de campos finitos depende principalmente de tres aspectos, el primero corresponde a las características del procesador, por ejemplo, el tamaño de palabra; el segundo aspecto es el nivel de seguridad que le queremos dar al sistema, el cual determina el tamaño de los parámetros que vamos a utilizar, por ejemplo, en este trabajo se proporcionan 127 bits de seguridad, por lo que necesitamos un número primo de 254 bits; finalmente el tercero está relacionado directamente con las funciones y prácticas de programación, es decir, buscar el algoritmo más eficiente, el primo que produzca menos operaciones, la reorganización de las operaciones y la reducción displicente.

- En cuanto a las características del procesador ARM Cortex-A9, el tamaño de palabra es de 32 bits. Como se mencionó anteriormente, esto repercute directamente en la biblioteca, ya que como se mostró en la página 20 de la sección 3.1.2, el tamaño de palabra proporciona el número de productos que se deben de aplicar para obtener la multiplicación modular en $\mathbb{F}_p$. Otro punto importante en cuanto al procesador, es el manejo de la tecnología NEON, la cual permite realizar múltiples operaciones. La principal aportación en el uso de NEON fue la implementación de un multiplicador dual en $\mathbb{F}_p$, es decir, una función que realiza dos multiplicaciones al mismo tiempo. Por lo que la aritmética en $\mathbb{F}_{p^2}$ y las operaciones de las curvas elípticas en $\mathbb{G}_1$ tuvieron que ser adaptada como se muestra en la sección 7.1.1 de la página 96. Debido a que no todos los dispositivos móviles poseen esta tecnología, se realizaron dos versiones de la biblioteca, una utilizando la tecnología NEON y otra que no la utiliza; ésta última fue implementada tal y como se describe en el capítulo 3. La tabla 7.3 de la página 106 muestra los tiempos obtenidos de las operaciones en las extensiones de campo con el uso de las dos bibliotecas, a partir de ello podemos decir que el uso de NEON proporciona un ahorro del 20 % en el costo del emparejamiento. En la multiplicación escalar de puntos presenta un ahorro aproximado del 12 % en $\mathbb{G}_1$ y 21 % en $\mathbb{G}_2$, mientras que en la exponenciación en $\mathbb{G}_T$ presenta un ahorro del 20 % como se muestra en la tabla 7.6 de la página 108. De esto se concluye que, explotar las características del procesador es un punto importante para garantizar un menor costo en las operaciones.

- Para realizar la biblioteca, se consideró el uso de curvas BN con 127 bits de seguridad. Como se mencionó en la sección 4.5 de la página 52, las curvas BN son parametrizadas con la variable z para la obtención del primo p, el cual proporciona la característica del campo, y el primo r, el cual proporciona el orden de las curvas elípticas. En este trabajo se tomó $z = -2^{62} - 2^{55} - 1$. Esta selección es importante ya que afecta directamente la torre de campos, principalmente en la selección de los polinomios irreducibles y está directamente relacionado con el número de operaciones de la exponenciación final. La principal ventaja de

esta selección fue que las operaciones con las variables β y ξ (véase capítulo 3), pueden realizarse con simples sumas. Finalmente, dada esta selección, se tiene que $p \equiv 3 \pmod 4$, por lo en la proyección de una cadena un punto, la raíz cuadrada en $\mathbb{F}_p$ puede realizarse con una exponenciación.

- Algunas técnicas de programación permiten la reducción en el número de operaciones en la aritmética de campos finitos, por ejemplo, el uso de los métodos SOS y CIOS, los cuales realizan la multiplicación modular con un número reducido de operaciones en el dominio de Montgomery; estos métodos destacan en importancia ya que el costo de la biblioteca depende directamente de la eficiencia del multiplicador en el campo $\mathbb{F}_p$. Otro método en cuanto a torres de campos es la reducción displicente, la cual proporciona un ahorro en las reducciones como se muestra en la tabla 7.1 de la página 103, donde el número de reducciones en cada operación es menor al número de multiplicaciones enteras, esto se debe a que la reducción displicente evita la reducción modular hasta ser necesariamente requerida, logrando una disminución de su uso. De acuerdo con lo anterior, se logro que el costo del emparejamiento observamos tuviera un ahorro del 30 % de reducciones, tal como se observa en la sección 7.2.1 de la página 103.

- En la implementación de la multiplicación escalar de puntos, se utilizaron coordenadas jacobianas, ya que no requieren del cálculo de inversos multiplicativos en la implementación de la suma y doblado de puntos, esto es importante ya que como se mencionó en la sección 4.3 de la página 46, la obtención del inverso multiplicativo es una operación costosa en comparación con la multiplicación en los campos finitos, en el caso de esta tesis es de 27 veces más caro para $\mathbb{F}_p$ y 18 veces para $\mathbb{F}_{p^2}$. Por lo que, utilizando coordenadas jacobianas tenemos un ahorro de 24 y 15 multiplicaciones en el doblado de puntos en $\mathbb{G}_1$ y $\mathbb{G}_2$ respectivamente, mientras que para la suma de puntos se tiene un ahorro de 19 multiplicaciones en $\mathbb{G}_1$ y 10 multiplicaciones en $\mathbb{G}_2$ aproximadamente.

- Dependiendo del esquema criptográfico, pueden ser necesarios diferentes tipos de multiplicaciones escalares de punto para curvas elípticas. En este trabajo se implementaron cuatro tipos diferentes, los cuales reducen el número de suma y doblado de puntos con respecto al método binario descrito en la sección 4.6 de la página 53. El primer multiplicador es el conocido por el método *comb*, el cual fue utilizado en multiplicaciones con punto conocido (P.C.) tanto para $\mathbb{G}_1$ como para $\mathbb{G}_2$; este método requiere cierto precómputo para su implementación, sin embargo reduce significativamente el número de sumas y doblados de punto. El segundo multiplicador utiliza el método GLV, el cual utiliza las características de la curva en $\mathbb{G}_1$ para reducir el número de doblados de punto. El tercer multiplicador fue el GS, el cual se utiliza para las curvas definidas sobre $\mathbb{G}_2$ y al igual que el método GLV, aprovecha las características de la curva elíptica para reducir el número de doblados. Finalmente el cuarto multiplicador fue el ω-NAF, el cual fue utilizado para el caso donde el escalar k en la multiplicación

de punto kP es un número de longitud pequeña, es decir, menor a 64 bits y se utiliza tanto para $\mathbb{G}_1$ como para $\mathbb{G}_2$. La tabla 7.5 de la página 107 muestra los costos de cada uno de estos métodos.

- Para disminuir el costo de la descomposición del escalar en los métodos GLV y GS se utilizó un método que permite realizar divisiones por 2^m en lugar de divisiones por r. Como se mencionó anteriormente, una división puede llegar a ser costosa, sin embargo, para el caso de que m sea seleccionada de acuerdo con el tamaño de palabra del procesador, esta operación se vuelve una simple asignación de valores, lo cuál es muy barato computacionalmente.

- Dado que el emparejamiento utiliza la aritmética en $\mathbb{F}_{p^{12}}$, fue necesario del uso de algoritmos que permitieran el menor número de operaciones; por ejemplo, la aplicación de operaciones dentro del grupo ciclotómico en la exponenciación final, con lo cual se logró una reducción significativa en los costos de implementación, como se muestra en la tabla 7.1 de la página 103. Así mismo, en el ciclo de Miller, el uso de las coordenadas estándar en la evaluación de las líneas en lugar de proyectivas permitió un ahorro en la evaluación de la línea tangente y doblado de punto de 2 elevaciones al cuadrado en $\mathbb{F}_{p^2}$, que en el en ciclo de Miller equivale a 126 elevaciones al cuadrado, dado que esta operación se repite 63 veces.

- Los algoritmos de la criptografía basada en atributos fueron implementados utilizando la biblioteca criptográfica desarrollada como se muestra en la figura 7.1 de la página 96. Cada uno de los algoritmos descritos utiliza las funciones desarrolladas en la biblioteca. Se observa que según sea el caso, se utilizan los diferentes tipos de multiplicadores escalares. En el algoritmo de inicialización, se define la llave pública y privada que se van a utilizar, en este caso varios de los puntos generados pueden ser tomados como puntos conocidos. Dado que el algoritmo de cifrado toma la llave pública, se observa que algunas multiplicaciones escalares pueden realizarse con el método *comb* y otras con el método GLV. Para la generación de llaves se utilizaron los métodos *comb* y GLV. Finalmente el descifrado utilizó el método ω-NAF debido a que los coeficientes utilizados tienen un tamaño menor de 64 bits.

- Una forma de optimizar el algoritmo de descifrado es a través del multiemparejamiento, el cual permite el ahorro de las elevaciones al cuadrado en el campo $\mathbb{F}_{p^{12}}$ en el ciclo de Miller y de la exponenciación final. Esto se debe a que en lugar de ejecutar estas operaciones por cada emparejamiento, sólo se realizan una sola vez, la tabla 7.7 de la página 108 muestra el ahorro de implementar esta técnica.

- De acuerdo con los resultados obtenidos la tabla 7.10 de la página 110, se obtuvo una eficiencia de aproximadamente el 10 % con respecto a la mejor implementación del emparejamiento de la literatura actual, por lo que se concluye que

una selección eficiente de los parámetros y algoritmos, el aprovechamiento de las características del procesador y el uso de algunas técnicas de programación permite una implementación eficiente de una biblioteca criptográfica para dispositivos móviles.

- Finalmente se concluye que la criptografía basada en atributos es un esquema que puede utilizarse como un medio de control de acceso, donde la información puede almacenarse de manera segura en un ambiente no seguro, como se implementó en este trabajo (véase apéndice D). En este sentido, la aplicación fue desarrollada en el ámbito del Departamento de Computación, donde cualquier usuario puede cifrar y descifrar documentos con respecto a una política de acceso dada, y donde los atributos manejados en las llaves fueron descripciones de los actores del departamento, tales como estudiante, maestría, doctorado, investigador, etc.

8.2. Trabajo futuro

El trabajo futuro de este trabajo se resume en los siguientes puntos:

1. Mejorar la biblioteca utilizando técnicas de paralelización en sus operaciones, principalmente en la implementación de la multiplicación escalar de puntos y en el multiemparejamiento.

2. Implementar el multiplicador polinomial descrito en [González-Díaz, 2010] para obtener el producto $c = a \cdot b \bmod p$, expresando los operandos como polinomios de la variable z de las curvas BN. Utilizar la tecnología NEON para disminuir el número de operaciones y ajustar la biblioteca en caso de ser éste más eficiente que los métodos descritos en este trabajo.

3. Mejorar el multiplicador utilizando ensamblador en lugar de las instrucciones *intrinsic* de NEON.

4. Adaptar la biblioteca de manera que permita diversos tipos de curvas y parámetros, ya sea para mejorar su eficiencia en ciertas operaciones o bien, para ofrecer un nivel mayor de seguridad.

5. Buscar nuevos endomorfismos en las curvas que permitan realizar un menor número de operaciones en la multiplicación escalar de punto.

6. Localizar y desarrollar aplicaciones donde la criptografía basada en atributos pueda ser utilizada como mecanismo de control de acceso seguro o bien, como medio de acceso biométrico.

7. Implementar otros esquemas criptográficos tales como firma corta, criptografía basada en la identidad, firma a ciegas, etc. sobre dispositivos móviles utilizando la biblioteca desarrollada.

Apéndice A

Reducción de Barret

La reducción de Barret A.1 encuentra x mod p para dos enteros positivos x y p dados [Hankerson et al., 2004]. Para ello, requiere del precómputo de:

$$\mu = \left\lfloor \frac{b^{2k}}{p} \right\rfloor$$

donde b es seleccionada como una potencia de dos cercana al tamaño de palabra del procesador. Por lo que, este algoritmo representa una ventaja si varias de las reducciones se realizan con un mismo módulo.

El algoritmo de Barret se basa en lo siguiente: dado $x = Qp + R$ donde $0 \leq R < p$, entonces $Q = \left\lfloor \frac{x}{p} \right\rfloor$ puede ser escrito como:

$$Q = \left\lfloor \left(x/b^{k-1}\right) \left(b^{2k}/p\right) \left(1/b^{k+1}\right) \right\rfloor$$

Algoritmo A.1 Reducción de Barret

Entrada: p, $b \geq 3$, $k = \lfloor log_b\ p \rfloor + 1$, $0 \leq z < b^{2k}$, y $\mu = \lfloor b^{2k}/p \rfloor$
Salida: z mod p

1: $q \leftarrow \left\lfloor \lfloor z/b^{k-1} \rfloor \cdot \mu / b^{k+1} \right\rfloor$
2: $r \leftarrow (z \bmod b^{k+1}) - (q \cdot p \bmod b^{k+1})$
3: **if** $r < 0$ **then**
4: $\quad r \leftarrow r + b^{k+1}$
5: **end if**
6: **while** $r \geq p$ **do**
7: $\quad r \leftarrow r - p$
8: **end while**
9: **return**

Para entender el algoritmo A.1 consideremos lo siguiente:

- En la línea 1 se tiene que:

$$0 \leq q = \left\lfloor \frac{\left\lfloor \frac{z}{b^{k-1}} \right\rfloor \cdot \mu}{b^{k+1}} \right\rfloor \leq \left\lfloor \frac{z}{p} \right\rfloor = Q$$

  donde q es una aproximación de Q tal que $Q \leq q + 2$

- Las líneas 2-5 garantizan que $r = z - qp \pmod{b^{k+1}}$.

- Dado que $0 \leq z - Qp < p$, entonces $0 \leq z - qp \leq z - (Q-2)p < 3p$, por lo que en la línea 6 requiere a lo más de 2 sustracciones.

Apéndice B

Instalación del ambiente de trabajo

Como se mencionó en el capítulo 7, la aplicación se implementó sobre un dispositivo móvil con sistema operativo Android. Dos herramientas importantes para desarrollar sobre este sistema son el SDK (Software Development Kit) y el NDK (Native Development Kit), los cuales pueden ser descargados directamente de la página de android (*http://developer.android.com/*) como se muestra en la figura B.1. La primera herramienta permite desarrollar aplicaciones sobre el sistema operativo y su lenguaje de programación es Java. La segunda herramienta nos permite desarrollar aplicaciones en código nativo, es decir C y C++, por lo que se toma ventaja del rendimiento de estos lenguajes.

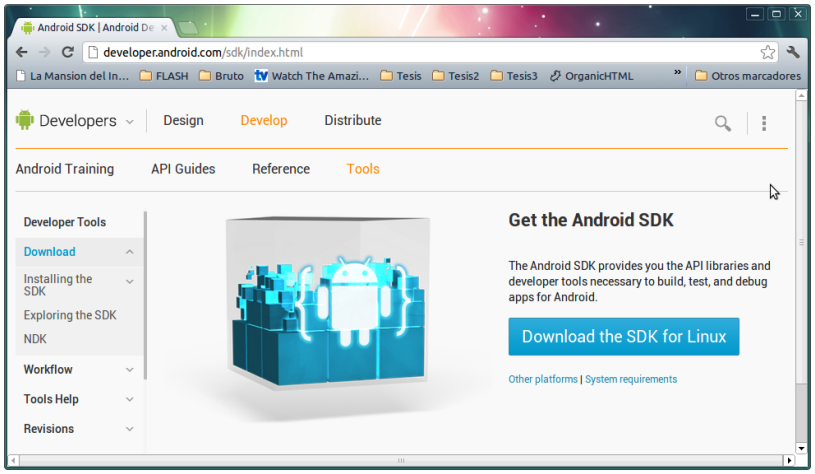


Figura B.1: Descarga del SDK de android

Para empezar a utilizar el SDK es necesaria una plataforma de desarrollo, en este caso se utilizó Eclipse en su versión 3.7.2, sin embargo se puede utilizar la versión 3.6 o cualquier otra superior. Posteriormente, se descarga el *plugin* para eclipse ADT (Android Developer Tools). Para ello se selecciona el menú **Help**, **Install New**

Software, y se agrega el sito:

https://dl-ssl.google.com/android/eclipse/

como se muestra en la figura B.2. Aparecerá una lista de herramientas de desarrollo las cuales deberán ser seleccionadas para ser descargadas e instaladas. Una vez instalado el *plugin* se configura indicando la ubicación del SDK.

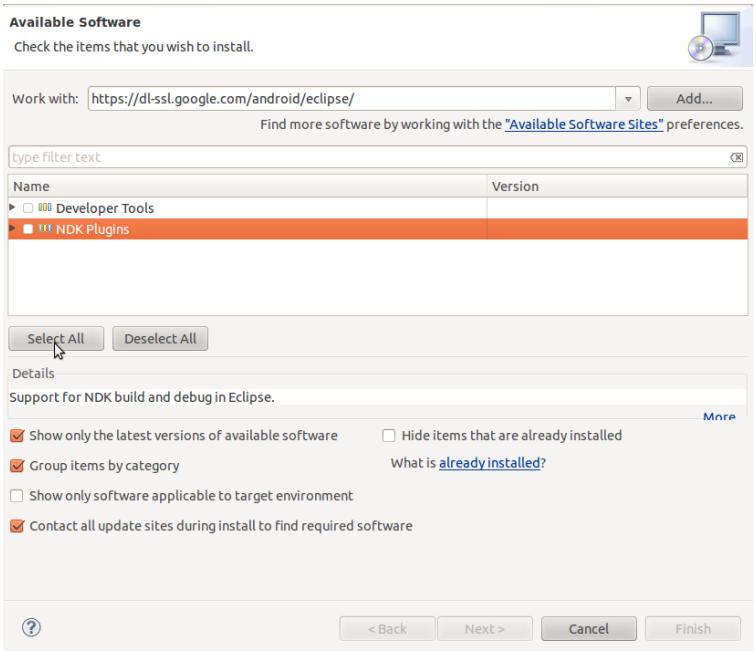


Figura B.2: Instalación del *plugin* ADT en Eclipse

Ahora bien, para empezar a trabajar sobre un dispositivo debemos seleccionar el menú **Window** y el submenú **Android SDK Manager**. Esto proporciona la lista de las versiones de android y las herramientas utilizadas para cada una (véase figura B.3). Finalmente se seleccionan las plataformas y herramientas deseadas para ser instaladas.

Una vez hecho esto, se tiene el ambiente de trabajo necesario para trabajar con el SDK sobre cualquier versión de android (véase figura B.4).

Para trabajar con el NDK, éste se debe descargar y descomprimir. Para compilar código nativo se debe crear un directorio llamado **jni** dentro del directorio raíz de la aplicación (<proyecto>). Dentro de la carpeta jni se debe colocar todo el código nativo. Para compilar, supongamos que <ndk> correponde a la dirección del NDK, entonces, para ejecutar el código, nos situamos en el directorio del proyecto y utilizamos la instrucción ndk-build como se muestra a continuación

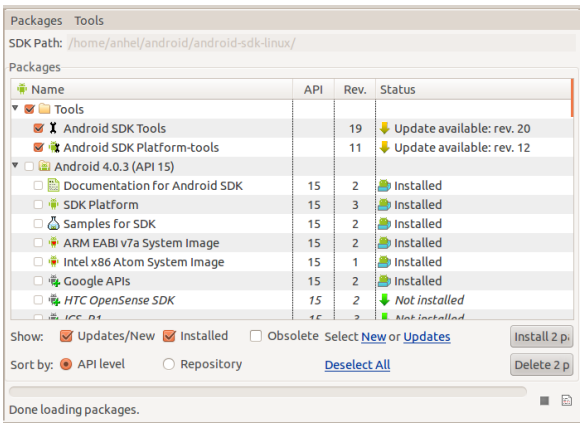


Figura B.3: Android SDK Manager

```
cd <proyecto>
<ndk>/ndk-build
```

Una forma de aplicar sólo la instrucción ndk-build es colocando la dirección <ndk> en el PATH[1].

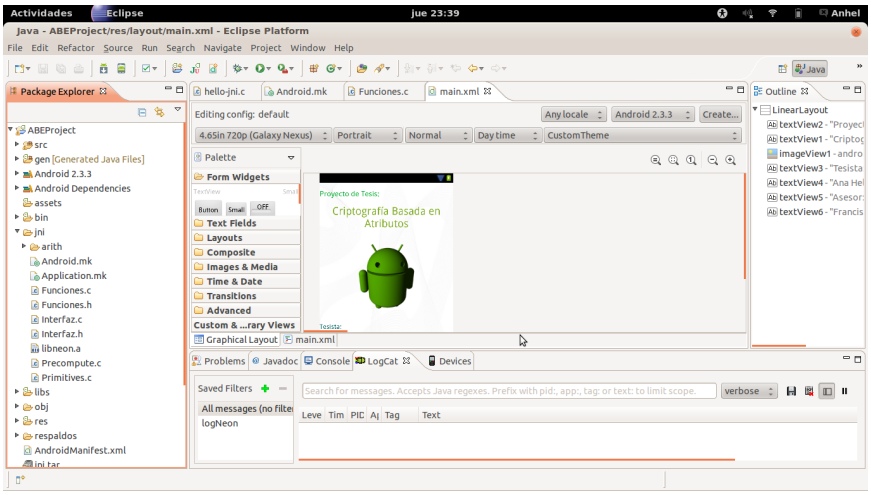


Figura B.4: Ambiente de desarrollo en Eclipse

[1]Si se está trabajando sobre Linux

Apéndice C

Modelo de capas

El modelo de capas en esta sección se refiere a qué funciones fueron implementadas utilizando las herramientas de desarrollo SDK y NDK. La figura C.1 muestra de forma gráfica cómo fue desarrollada la biblioteca y qué herramienta fue utilizada en cada caso.

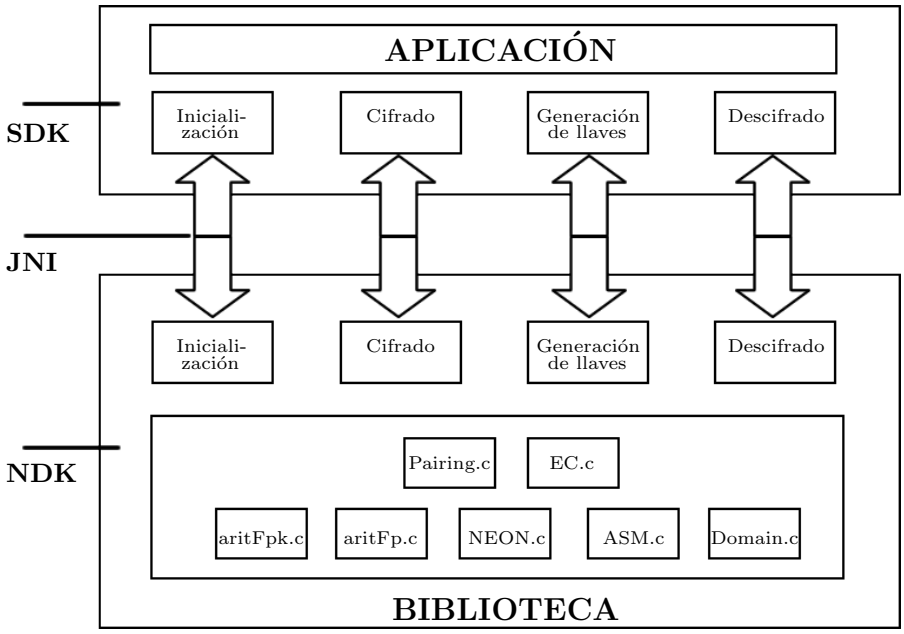


Figura C.1: Modelo de capas

Como se observa, se utilizó el NDK para compilar la biblioteca y ésta se encuentra formada por diferentes secciones de código, los cuales corresponden a las capas del modelo de implementación descrito en la sección 7.1. Así mismo, se utilizó el JNI (Java Native Interface) como intermediario entre las funciones compiladas por el NDK y la aplicación desarrollada con el SDK.

Apéndice D

Manual de usuario

Como se mencionó en la sección 7.4 la aplicación es sistema análogo a PGP[1] para el Departamento de Computación. De acuerdo a lo anterior, el programa se divide en tres secciones: Manejo de llaves, cifrado y descifrado (véase figura D.1).

Figura D.1: Aplicación: Menú principal

D.1. Manejo de llaves

La sección de manejo de llaves muestra como pantalla de inicio la lista de las llaves creadas indicando los atributos que éstas contienen. Éstas llaves pueden ser eliminadas presionando el botón X como se muestra en la figura D.2.

De acuerdo con el esquema ABE, para realizar el manejo de llaves se tienen dos opciones, la primera es crear directamente la llave a través de la llave maestra (MSK)

[1]http://www.ietf.org/rfc/rfc4880.txt (04/10/2012)

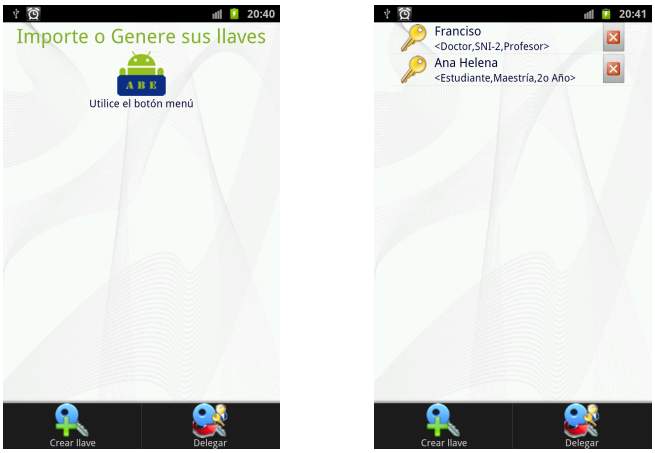


Figura D.2: Aplicación: Manejo de llaves

o bien, realizar una delegación de llaves. De manera que la aplicación cuenta con esas dos opciones para generar una llave (véase figura D.2).

D.1.1. Crear llave

Para crear una llave se presenta la pantalla de la figura D.3, en ésta se solicita el nombre de usuario y un botón para agregar los atributos que ésta permitirá, en este caso, atributos de los actores del departamento, tales como doctores, investigadores, estudiantes, administrativos, etc. Al final se agrega un botón aceptar. Al presionar este botón se solicitará la contraseña para la llave maestra y la contraseña del usuario.

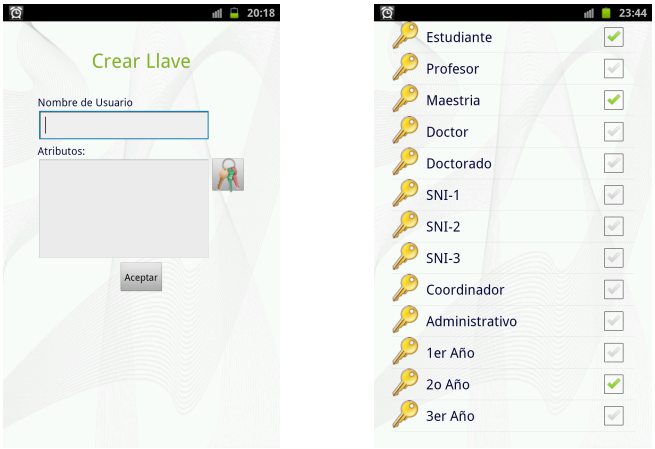


Figura D.3: Aplicación: Crear llave

D.1.2. Delegar llave

De manera similar a la creación de llaves, éste solicita el nombre del usuario y los atributos. Sin embargo, los atributos pertenecen a la llave seleccionada. Cuando se presiona el botón aceptar lo que solicita ahora es la contraseña del usuario de la llave original y posteriormente la contraseña del usuario de la nueva llave (véase figura D.4).

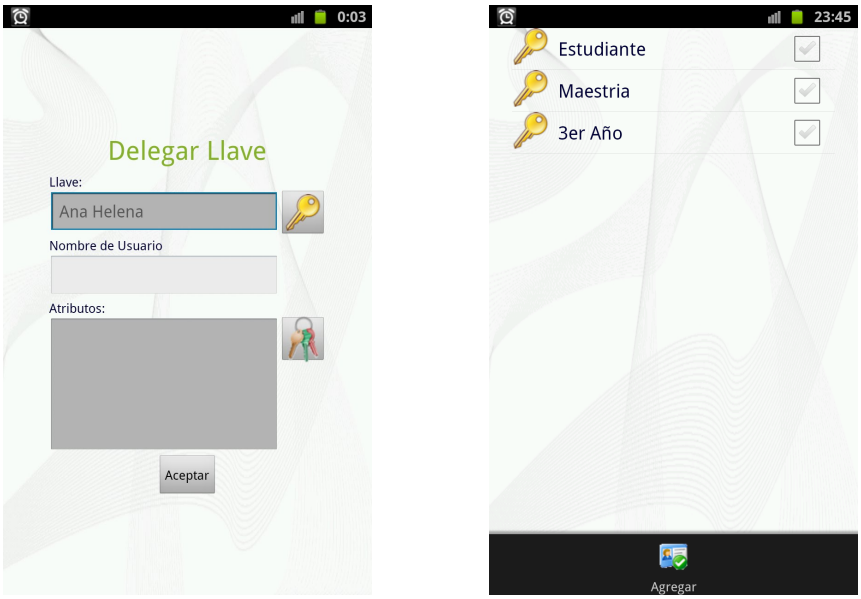
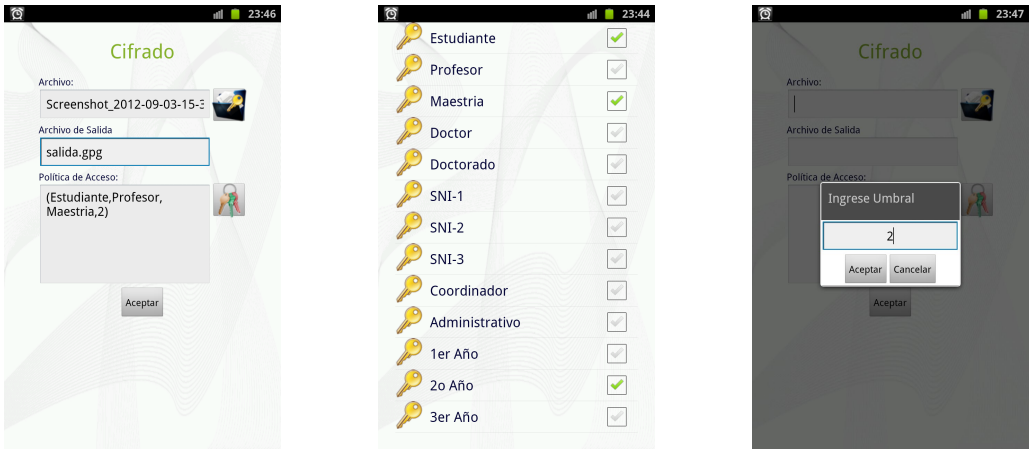


Figura D.4: Aplicación: Delegar llave

D.2. Cifrado

Como se observa en la figura D.5a, para realizar el cifrado se selecciona un archivo proveniente de la tarjeta de memoria del dispositivo móvil, posteriormente se debe indicar un nombre para el archivo de salida y por último se agrega la política de acceso. Al presionar el botón aceptar, se valida la política de acuerdo a los atributos existentes y se realiza el cifrado del archivo, el cuál se guardará en la misma carpeta del archivo seleccionado. En caso de que la política sea inválida, se enviará un mensaje de error. El botón a la izquierda de la caja de texto de la política sirve de ayuda al usuario para generar la política de acceso, ya que muestra la lista de atributos con los cuales puede cifrar (figura D.5b) y el umbral que puede utilizar en ellos (figura D.5c).



a) b) c)

Figura D.5: Aplicación: Cifrado

D.3. Descifrado

Finalmente la sección de descifrado solicita el archivo y la llave que se utilizarán para descifrar, así como el nombre del archivo de salida (véase figura D.6). Al presionar el botón aceptar, se realiza el descifrado del documento el cuál se colocará en la misma carpeta del archivo cifrado. En caso de que no se satisfaga la política de acceso el programa envía una notificación de error.

Figura D.6: Aplicación: Descifrado

Bibliografía

[Acar et al., 2011] T. Acar, K. Lauter, M. Naehrig, and D. Shumow, *Affine Pairings on ARM*, IACR Cryptology ePrint Archive, Vol. 2011, p. 243, 2011.

[Adleman and Huang, 1999] L. M. Adleman and M.-D. A. Huang, *Function Field Sieve Method for Discrete Logarithms over Finite Fields*, Information and Computation, Vol. 151, No. 1-2, pp. 5–16, ScienceDirect, May/June, 1999.

[Akinyele et al., 2010] J. A. Akinyele, C. Lehmann, M. Green, M. Pagano, Z. Peterson, and A. Rubin, *Self-Protecting Electronic Medical Records Using Attribute-Based Encryption*, In SPSM '11 Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices A. Bhattacharya, P. Dasgupta and W. Enck (Eds.), ACM, pp. 75–86, Chicago, Illinois, USA, October 17-21, 2010.

[Aranha et al., 2011a] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, *Faster Explicit Formulas for Computing Pairings over Ordinary Curves*, In Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vol. 6632 of Lecture Notes in Computer Science K.G. Paterson (Ed.), Springer, pp. 48–68, Tallinn, Estonia, May 15-19, 2011a.

[Aranha et al., 2011b] D. F. Aranha, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez, *Parallelizing the Weil and Tate Pairings*, In Cryptography and Coding - 13th IMA International Conference, IMACC 2011, Vol. 7089 of Lecture Notes in Computer Science. L. Chen (Ed.), Springer, pp. 275–295, Oxford, UK, December 12-15, 2011b.

[Aranha et al., 2010] D. F. Aranha, J. López, and D. Hankerson, *High-Speed Parallel Software Implementation of the η_T Pairing*, In Topics in Cryptology - CT-RSA 2010, The Cryptographers'Track at the RSA Conference 2010, Vol. 5985 of Lecture Notes in Computer Science. J. Pieprzyk (Ed.), Springer, pp. 89–105, San Francisco, California, USA, March 1-5, 2010.

[Attrapadung and Imai, 2009] N. Attrapadung and H. Imai, *Conjunctive Broadcast and Attribute-Based Encryption*, In Pairing-Based Cryptography - Pairing 2009, Third International Conference, Vol. 5671 of Lecture Notes in Computer Science.

H. Shacham and B. Waters (Eds.), Springer, pp. 248–265, Palo Alto, California, USA, August 12-14, 2009.

[Attrapadung et al., 2011] N. Attrapadung, B. Libert, and E. de Panafieu, *Expressive Key-Policy Attribute-Based Encryption with Constant-Size Ciphertexts*, In Public Key Cryptography 2011 (PKC 2011), Vol. 6571 of Lecture Notes in Computer Science. D. Catalano, N. Fazio, R. Gennaro and A. Nicolosi (Eds.), Springer, pp. 90–108, Taormina, Italy, March 6-9, 2011.

[Babai, 1986] L. Babai, *On lovász'lattice reduction and the nearest lattice point problem*, Combinatorica, Vol. 6, No. 1, pp. 1–13, Springer, March, 1986.

[Barreto et al., 2003] P. S. L. M. Barreto, B. Lynn, and M. Scott, *On the Selection of Pairing-Friendly Groups*, In Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Vol. 3006 of Lecture Notes in Computer Science M. Matsui and R.J. Zuccherato (Eds.), Springer, pp. 17–25, Ottawa, Canada, August 14-15, 2003.

[Beimel, 1996] A. Beimel, *Secure Schemes for Secret Sharing and Key Distribution*, PhD Thesis, Israel Institute of Technology, June, 1996.

[Bethencourt et al., 2007] J. Bethencourt, A. Sahai, and B. Waters, *Ciphertext-Policy Attribute-Based Encryption*, In Proceedings of the 2007 IEEE Symposium on Security and Privacy B. Pfitzmann and P. McDaniel (Eds.), IEEE Computer Society, pp. 321–334, Oakland, California, USA, May 20-23, 2007.

[Beuchat et al., 2010] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, *High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves*, In Pairing-Based Cryptography - Pairing 2010 - 4th International Conference, Vol. 6487 of Lecture Notes in Computer Science M. Joye, A. Miyaji and A. Otsuka (Eds.), Springer, pp. 21–39, Yamanaka Hot Spring, Japan, December 13-15, 2010.

[Beuchat et al., 2009] J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez, *Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves*, In CANS, Vol. 5888 of Lecture Notes in Computer Science. J.A. Garay, A. Miyaji and A. Otsuka (Eds.), Springer, pp. 413–432, Kanazawa, Japan, December 12-14, 2009.

[Boneh and Franklin, 2003] D. Boneh and M. Franklin, *Identity-Based Encryption from the Weil Pairing*, SIAM Journal of Computing, Vol. 32, No. 3, pp. 586–615, Society for Industrial and Applied Mathematics, June 2003.

[Boneh et al., 2004] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the weil pairing*, J. Cryptology, Vol. 17, No. 4, pp. 297–319, Springer, September, 2004.

[Chung and Hasan, 2007] J. Chung and M. A. Hasan, *Asymmetric Squaring Formulas*, In Proceedings of the 18th IEEE Symposium on Computer Arithmetic P. Kornerup and J.M. Muller (Eds.), IEEE Computer Society, pp. 113–122, Montpellier, France, June 25-27, 2007.

[Costello et al., 2010] C. Costello, T. Lange, and M. Naehrig, *Faster Pairing Computations on Curves with High-Degree Twists*, In Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Vol. 6056 of Lecture Notes in Computer Science P.Q. Nguyen and D. Pointcheval (Eds.), Springer, pp. 224–242, Paris, France, May 26-28, 2010.

[Daemen and Rijmen, 2002] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, Information Security and Cryptography, 1st Edition, Springer, Leuven, Belgium, 2002.

[Dussé and Kaliski, 1991] S. R. Dussé and B. S. Kaliski, Jr., *A cryptographic library for the Motorola DSP56000*, In Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology, EUROCRYPT '90, Vol. 473 of Lecture Notes in Computer Science. I.B. Damgård (Ed.), Springer, pp. 230–244, Aarhus, Denmark, May 21-24, 1991.

[Freeman et al., 2010] D. Freeman, M. Scott, and E. Teske, *A Taxonomy of Pairing-Friendly Elliptic Curves*, J. Cryptology, Vol. 23, No. 2, pp. 224–280, Springer, April, 2010.

[Frey and Rück, 1994] G. Frey and H.-G. Rück, *A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves*, Mathematics of Computation, Vol. 62, No. 206, p. 865, American Mathematical Society, May 1994.

[Fuentes-Castañeda, 2011] L. Fuentes-Castañeda, *Estudio y Análisis de Emparejamientos Bilineales Definidos sobre Curvas Ordinarias con Alto Nivel de Seguridad*, MsCs Thesis, Department of Computer Science, Cinvestav, December, 2011.

[Fuentes-Castañeda et al., 2011] L. Fuentes-Castañeda, E. Knapp, and F. Rodríguez-Henríquez, *Faster hashing to* $\mathbb{G}_2$, In Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Vol. 7118 of Lecture Notes in Computer Science A. Miri and S. Vaudenay (Eds.), Springer, pp. 412–430, Toronto, ON, Canada, August 11-12, 2011.

[Galbraith and Scott, 2008] S. D. Galbraith and M. Scott, *Exponentiation in Pairing-Friendly Groups Using Homomorphisms*, In Pairing-Based Cryptography - Pairing 2008, Second International Conference, Vol. 5209 of Lecture Notes in Computer Science S.D. Galbraith and K.G. Paterson (Eds.), Springer, pp. 211–224, Egham, UK, September 1-3, 2008.

[Gallant et al., 2001] R. P. Gallant, R. J. Lambert, and S. A. Vanstone, *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms*, In Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Vol. 2139 of Lecture Notes in Computer Science J. Kilian (Ed.), Springer, pp. 190–200, Santa Barbara, California, USA, August 19-23, 2001.

[Geovandro et al., 2011] C. C. F. P. Geovandro, M. A. S. Jr., M. Naehrig, and P. S. L. M. Barreto, *A family of implementation-friendly BN elliptic curves*, Journal of Systems and Software, Vol. 84, No. 8, pp. 1319–1326, ScienceDirect, August, 2011.

[Goldwasser and Kilian, 1999] S. Goldwasser and J. Kilian, *Primality Testing Using Elliptic Curves*, J. ACM, Vol. 46, No. 4, pp. 450–472, ACM, July, 1999.

[González-Díaz, 2010] J. González-Díaz, *Diseño e Implementación Eficiente del Emparejamiento Óptimo "Ate"*, MsCs Thesis, Department of Computer Science, Cinvestav, September, 2010.

[Gorantla et al., 2010] M. C. Gorantla, C. Boyd, and J. M. G. Nieto, *Attribute-Based Authenticated Key Exchange*, In Proceeding ACISP'10 Proceedings of the 15th Australasian conference on Information security and privacy, Vol. 6168 of Lecture Notes in Computer Science R. Steinfeld and P. Hawkes (Eds.), Springer-Verlag, pp. 300–317, Sydney, Australia, July 5-7, 2010.

[Goyal et al., 2006] V. Goyal, O. Pandey, A. Sahai, and B. Waters, *Attribute-based encryption for finegrained access control of encrypted data*, In Proceeding of the 13th ACM conference on Computer and Communications Security. A. Juels, R.N. Wright and S.D.C. di Vimercati (Eds.), ACM, pp. 89–98, Alexandria, Virginia, USA, October 30 - November 3, 2006.

[Granger and Scott, 2010] R. Granger and M. Scott, *Faster squaring in the cyclotomic subgroup of sixth degree extensions*, In In PKC'10 Proceedings of the 13th international conference on Practice and Theory in Public Key Cryptography, Vol. 6056 of Lecture Notes in Computer Science P.Q. Nguyen and D. Pointcheval (Eds.), Springer, pp. 209–223, Paris, France, May 26-28, 2010.

[Grewal et al., 2012] G. Grewal, R. Azarderakhsh, P. Longa, S. Hu, and D. Jao, *Efficient implementation of bilinear pairings on arm processors*, IACR Cryptology ePrint Archive, Vol. 2012, p. 408, 2012.

[Hankerson et al., 2009] D. Hankerson, A. Menezes, and M. Scott, *Software implementation of pairings (Chapter 12)*, In Identity-based Cryptography, Vol. 2 of Cryptology and Information Security M. Joye and G. Neven (Eds.), IOS Press, p. 188–206, Ámsterdam, The Netherlands, December 2009.

[Hankerson et al., 2004] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, 1st Edition, Springer-Verlag New York, Inc., Secaucus, New Jersey, USA, 2004.

[Hess et al., 2006] F. Hess, N. P. Smart, and F. Vercauteren, *The Eta Pairing Revisited*, IEEE Transactions on Information Theory, Vol. 52, No. 10, pp. 4595–4602, IEEE Information Theory Society, October, 2006.

[Joux, 2004] A. Joux, *A One Round Protocol for Tripartite Diffie-Hellman*, Journal of Cryptology, Vol. 17, No. 4, pp. 263–276, Springer, January 2004.

[Karabina, 2010] K. Karabina, *Squaring in cyclotomic subgroups*, IACR Cryptology ePrint Archive, Vol. 2010, p. 542, 2010.

[Karchmer and Wigderson, 1993] M. Karchmer and A. Wigderson, *On span programs*, In Structure in Complexity Theory Conference, IEEE Computer Society Press, pp. 102–111, San Diego, California, USA, May 18-21 1993.

[Koblitz, 1987] N. Koblitz, *Elliptic Curve Cryptosystems*, Mathematics of Computation, Vol. 48, No. 177, pp. 203–209, American Mathematical Society, January 1987.

[Koc et al., 1996] C. K. Koc, T. Acar, and B. S. K. Jr., *Analyzing and Comparing Montgomery Multiplication Algorithms*, IEEE Micro, Vol. 16, No. 3, pp. 26–33, IEEE Computer Society Press, June, 1996.

[Lenstra et al., 1982] A. K. Lenstra, H. W. Lenstra, and L. Lovász, *Factoring polynomials with rational coefficients*, Mathematische Annalen, Vol. 261, No. 4, pp. 515–534, Springer, December, 1982.

[Lewko and Waters, 2011] A. Lewko and B. Waters, *Unbounded HIBE and Attribute-Based Encryption*, In Proceedings of Advances in Cryptology, EUROCRYPT 2011, Vol. 6632 of Lecture Notes in Computer Science. K. Paterson (Ed.), Springer, pp. 547–567, Tallinn, Estonia, May 15-19, 2011.

[Lidl and Niederreiter, 1986] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, 2nd Edition, Cambridge University Press, New York, New York, USA, 1986.

[Liu and Cao, 2010] Z. Liu and Z. Cao, *On efficiently transferring the linear secret-sharing scheme matrix in ciphertext-policy attribute-based encryption*, IACR Cryptology ePrint Archive, Vol. 2010, p. 374, 2010.

[Maji et al., 2011] H. Maji, M. Prabhakaran, and M. Rosulek, *Attribute-Based Signatures*, In Topics in Cryptology CT-RSA, Vol. 6558 of Lecture Notes in Computer Science. A. Kiayias (Ed.), Springer, pp. 376–392, San Francisco, California, USA, February 14-18, 2011.

[Menezes et al., 1993] A. Menezes, T. Okamoto, and S. A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory, Vol. 39, No. 5, pp. 1639–1646, IEEE Computer Society Press, September 1993.

[Menezes et al., 1996] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st Edition, CRC Press, Inc., Boca Raton, Florida, USA, 1996.

[Miller, 1985] V. S. Miller, *Use of Elliptic Curves in Cryptography*, In Proceedings on Advances in cryptology CRYPTO 85, Vol. 218 of Lecture Notes in Computer Science. H. Williams (Ed.), Springer, pp. 417–426, Santa Barbara, California, USA, August 18-22, 1985.

[Miller, 2004] V. S. Miller, *The Weil Pairing, and Its Efficient Calculation*, J. Cryptology, Vol. 17, No. 4, pp. 235–261, Springer, January, 2004.

[Mitsunari et al., 2002] S. Mitsunari, R. Sakai, and M. Kasahara, *A New Traitor Tracing*, EICE Trans Fundam Electron Commun Comput Sci (Inst Electron Inf Commun Eng), Vol. E85-A, No. 2, pp. 481–484, Japan Science and Technology Agency, October 2002.

[Mohan and Blough, 2010] A. Mohan and D. M. Blough, *An attribute-based authorization policy framework with dynamic conflict resolution*, In Proceedings of the 9th Symposium on Identity and Trust on the Internet, IDTRUST '10 K. Klingenstein and C.M. Ellison (Eds.), ACM, pp. 37–50, Gaithersburg, Maryland, USA, April 6-7, 2010.

[Montgomery, 1985] P. L. Montgomery, *Modular Multiplication without Trial Division*, Mathematics of Computation, Vol. 44, No. 170, pp. 519–521, American Mathematical Society, April, 1985.

[Montgomery, 1987] P. L. Montgomery, *Speeding the Pollard and Elliptic Curve Methods of Factorization*, Mathematics of Computation, Vol. 48, No. 177, pp. 243–264, American Mathematical Society, January, 1987.

[Naehrig et al., 2010] M. Naehrig, R. Niederhagen, and P. Schwabe, *New Software Speed Records for Cryptographic Pairings*, In Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Vol. 6212 of Lecture Notes in Computer Science M. Abdalla and P.S.L.M. Barreto (Eds.), Springer, pp. 109–123, Puebla, México, August 8-11, 2010.

[Narayan et al., 2010] S. Narayan, M. Gagne, and R. S. Naini, *Privacy preserving EHR system using attribute-based infrastructure*, In CCSW '10: Proceeding of the 2010 ACM Workshop on Cloud computing security workshop E. Al-Shaer,

A.D. Keromytis and V. Shmatikov (Eds.), ACM, pp. 47–52, Chicago, Illinois, USA, October 04 - 08, 2010.

[Niven et al., 1991] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*, 5th Edition, Wiley, New York, New York, USA, 1991.

[Pirreti et al., 2006] M. Pirreti, P. Traynor, P. McDaniel, and B. Waters, *Secure attribute-based systems*, In Proceeding of ACM Conference on Computer and Communications Security R.N. Wright, S.D.C. di Vimercati and V. Shmatikov (Eds.), ACM, pp. 99–112, Alexandria, Virginia, USA, October 30 - November 3, 2006.

[Pollard, 1978] J. Pollard, *Monte Carlo methods for Index Computation (mod p)*, Mathematics of Computation, Vol. 32, No. 143, pp. 918–924, American Mathematical Society, July, 1978.

[Sahai and Waters, 2005] A. Sahai and B. Waters, *Fuzzy identity-based encryption*, In Advances in Cryptology, EUROCRYPT '05, Vol. 3494 of Notes in Computer Science. R. Cramer (Ed.), Springer, pp. 457–473, Aarhus, Denmark, May 22-26, 2005.

[Sakai et al., 2000] R. Sakai, K. Oghishi, and M. Kasahara, *Cryptosystems based on pairing*, In Symposium on Cryptography and Information Security (SCIS2000), Springer, Okinawa, Japan, January 26-28 2000.

[Schwabe et al., 2012] P. Schwabe, B.-Y. Yang, and S.-Y. Yang, *SHA-3 on ARM11 processors*, In Progress in Cryptology - AFRICACRYPT 2012, Vol. 7374 of Lecture Notes in Computer Science. A. Mitrokotsa and S. Vaudenay (Eds.), Springer, Ifrane, Morocco, July 10-12, 2012.

[Scott, 2011] M. Scott, *On the Efficient Implementation of Pairing-Based Protocols*, In Cryptography and Coding - 13th IMA International Conference, IMACC 2011, Vol. 7089 of Lecture Notes in Computer Science. L. Chen (Ed.), Springer, pp. 296–308, Oxford, UK, December 12-15, 2011.

[Shahandashti and Safavi-Naini, 2009] S. Shahandashti and R. Safavi-Naini, *Threshold Attribute-Based Signatures and Their Application to Anonymous Credential Systems*, In Progress in Cryptology, AFRICACRYPT 2009, Vol. 5580 of Lecture Notes in Computer Science. B. Preneel (Ed.), Springer, pp. 198–216, Gammarth, Tunisia, June 21-25, 2009.

[Shamir, 1984] A. Shamir, *Identity-Based Cryptosystems and Signature Schemes*, In Advances in Cryptology, CRYPTO '84, Vol. 196 of Lecture Notes in Computer Science G.R. Blakley and D. Chaum (Eds.), Springer, pp. 47–53, Santa Barbara, California, USA, August 19-22, 1984.

[Shanks, 1972] D. Shanks, *Five number-theoretic algorithms*, In Proceedings of the Second Manitoba Conference on Numerical Mathematics R.S.D. Thomas and H.C. Williams (Eds.), Utilitas Mathematica, pp. 51–70, Winnipeg, Manitoba, October 5-7 1972.

[Shoup, 2005] V. Shoup, *A computational introduction to number theory and algebra*, 2nd Edition, Cambridge University Press, New York, New York, USA, 2005.

[Srirama and Naumenko, 2010] S. N. Srirama and A. Naumenko, *Secure Communication and Access Control for Mobile Web Service Provisioning*, CoRR. The Computing Research Repository, Vol. abs/1007/3649, Cornell University Library, July 2010.

[Vercauteren, 2010] F. Vercauteren, *Optimal pairings*, IEEE Transactions on Information Theory, Vol. 56, No. 1, pp. 455–461, IEEE Information Theory Society, January, 2010.

[Washington, 2008] L. C. Washington, *Elliptic Curves: Number Theory and Criptograhpy*, 2nd Edition, Chapman & Hall/CRC, University of Maryland, College Park, USA, 2008.

[Waters, 2008] B. Waters, *Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization*, In PKC'11 Proceedings of the 14th international conference on Practice and theory in public key cryptography, Vol. 6571 of Lecture Notes in Computer Science. D. Catalano, N. Fazio, R. Gennaro and A. Nicolosi (Eds.), Springer, pp. 53–70, Taormina, Italy, March 6-9, 2008.

[Zhang et al., 2002] L. Zhang, G. Ahn, and B. Chu, *A role-based delegation framework for healthcare information systems*, In SACMAT '02 Proceedings of the seventh ACM symposium on Access control models and technologies. R. Sandhu and E. Bertino (Eds.), ACM, pp. 125–134, Monterey, California, USA, June 3-4, 2002.

[Zheng and Ni, 2005] P. Zheng and L. Ni, *Smart Phone and Next Generation Mobile Computing*, 1st Edition, Morgan Kaufmann, Burlington, Massachusetts, USA, 2005.

[Zhou and Huang, 2010] Z. Zhou and D. Huang, *On Efficient Ciphertext-Policy Attribute Based Encryption and Broadcast Encryption*, In Proceedings of the 17th ACM conference on Computer and communications security E. Al-Shaer, A.D. Keromytis and V. Shmatikov (Eds.), ACM, pp. 753–755, Chicago, Illinois, USA, October 4-8, 2010.