Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Computación

Diseño de un compilador para protocolos criptográficos

Tesis que presenta:

Rogelio Vargas Márquez

Para obtener el grado de:

Maestro en Ciencias en Computación

Director de la Tesis:
Dr. Francisco Rodríguez Henríquez

México, D. F. Diciembre, 2013

Agradecimientos

En primer lugar quiero decir que por falta de espacio no me es posible nombrar a todas las personas que me apoyaron a lo largo de estos dos años.

Gracias a mi compañera de vida Lucia Rodríguez Cruz y a mi hijo Johann Yael Vargas Rodríguez, su gran comprensión los ayudo a soportar la distancia y las dificultades que enfrentamos. Saber que siempre estuvieron ahí en las buenas y en las malas fue el mayor impulso para concluir mi trabajo.

A mi madre Ma. Mercedes Márquez Arzate, a mi padre Rubén Vargas Blancas y a mi hermano Rubén Vargas Márquez, quienes me brindaron su apoyo incondicional durante este tiempo y cuidaron de Lucy y Johann el tiempo que estuve ausente.

A mi director de tesis el Dr. Francisco Rodríguez Henríquez le agradezco por los conocimientos que me brindo, por el voto de confianza que ha depositado en mí y sobre todo por la paciencia que tuvo en las ocasiones donde he fallado.

A el Dr. Luis Julian Dominguez Perez le agradezco por todo el apoyo que me dio, por guiarme durante mi estancia en Tamaulipas y por todo el tiempo que me otorgó para concluir con éxito este trabajo.

A mis sinodales la Dra. María de Lourdes López García y el Dr. Carlos Artemio Coello Coello por sus contribuciones a mi trabajo de tesis.

A mis profesores del Departamento de Computación en Zacatenco, a Sofi por todos los favores que me hizo, a todos mis compañeros en Zacatenco en especial a Eduardo, Marco y Michel.

A todo el personal de Laboratorio de Tecnologías de Información en Cd. Victoria. A mis compañeros de aquellos rumbos, en especial a Carlos, Ana, Arturo y Daniel.

A mis amigos de toda la vida y de la UAM Azcapotzalco por su apoyo y preocupación.

Agradezco 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.

Y por supuesto al Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAVIPN) por permitirme ser parte de esta institución.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Las características que poseen los emparejamientos bilineales han permitido la construcción de novedosos esquemas criptográficos. Cuando se requiere el uso de dichos esquemas surgen dos tareas: la primera consiste en diseñar dicho esquema o protocolo, el cual tendrá como objetivo proporcionar diferentes servicios de seguridad. La segunda tarea consiste en implementar el protocolo mediante el uso de algún lenguaje de programación.

Ciertos protocolos basados en emparejamientos se pueden implementar en más de una manera, sobre todo cuando la propuesta de los diseñadores utiliza una definición general del emparejamiento (emparejamiento tipo I), y de los componentes de alrededor. Esta situación tiene el riesgo de presentar una implementación diferente a la versión original. Además, el diseño de un protocolo debe incluir una descripción detallada de las etapas que lo componen, y es importante destacar las primitivas criptográficas usadas.

Esta tesis describe el análisis, diseño e implementación de un compilador para protocolos criptográficos. El compilador recibe como entrada la descripción de un protocolo criptográfico basado en emparejamientos bilineales, y genera a la salida la implementación eficiente en el lenguaje Magma y lenguaje C/C++. Para la implementación hacemos uso de las curvas Barreto-Naerigh otorgando un nivel de seguridad de 126 bits. Así mismo, proponemos el lenguaje para la descripción de protocolos, usado como entrada del compilador.

Abstract

The features provided by the bilinear pairings, allow efficient construction of some cryptographic schemes. When the use of these schemes is required, there are two tasks: the first involves the designs of these schemes or protocols, which aim to provide different security services. The second task is to implement the protocol using a programming language.

Certain pairing based protocols can be implemented in more than one way, specially when the designers proposal uses a general definition of the pairing (type I pairing), and for the components around them. This situation has the risk to present different implementation versions of the scheme. In addition, the design of a protocol must include a detailed description of the component steps, highlighting the required cryptographic primitives inside.

This thesis describes the analysis, design and implementation of a compiler for cryptographic protocols. The compiler takes as input the description of a cryptographic protocol based on bilinear pairings, and generates an efficient implementation in Magma and C/C++. For the implementation part we used the Barreto-Naerigh curves at a security level of 126 bits. Furthermore, we propose a language for the description of protocols, wich was used as part of input to the compiler.

1

Introducción

Generalmente, el intercambio de información entre dos o más entidades se realiza a través de canales de comunicación inseguros, por lo tanto, la información intercambiada no cuenta con garantía alguna de privacidad o integridad. Por este motivo, se requieren de mecanismos o servicios que brinden seguridad durante el proceso de comunicación. Principalmente se busca comprobar la autenticidad de las entidades involucradas, garantizar la privacidad de las mismas, asegurar la integridad de la información y proporcionar pruebas de la integridad y origen de la información. Dichos mecanismos pueden ser construidos mediante herramientas criptográficas.

Este capítulo tiene la finalidad de brindar al lector una breve introducción de conceptos tales como: seguridad, criptografía y protocolos criptográficos, los cuales ayudarán a una mejor comprensión de este trabajo de tesis.

1.1 Nociones básicas de seguridad y criptografía

La creciente evolución en las comunicaciones ha promovido el desarrollo de aplicaciones, como: el cómputo nube, las redes sociales, las transacciones electrónicas, las firmas electrónicas y el uso del dinero electrónico. Estas aplicaciones requieren de un intercambio y acceso seguro a la información, por lo cual se debe cumplir con una serie de servicios o mecanismos de seguridad descritos en seguida [Vieites, 2006]:

- *Confidencialidad*: Garantiza que solo la información sensible pueda ser leída o manipulada por los usuarios, entidades o procesos autorizados.
- *Autenticación*: Asegura que cada entidad participante es realmente quien dice ser. La autenticación de cada entidad se logra utilizando mecanismos tales como: firmas y certificados digitales, o características biométricas de los participantes.
- *Integridad*: Se encarga de dar certeza de que la información sensible no sea modificada por una entidad no autorizada. Dentro de las posibles modificaciones que se pueden realizar, tenemos: la escritura, la modificación o el borrado de segmentos de la información.
- *No repudio*: Asegura que una entidad que emita alguna información, no pueda negar dicha emisión (no repudio de origen) o que la entidad que recibe dicha información no pueda negar que la recibió (no repudio de destino).
- *Disponibilidad*: Brinda medios para que las entidades interesadas y autorizadas tengan acceso a la información disponible.

El intercambio de información entre sistemas informáticos implica la existencia de una arquitectura de comunicaciones, la cual comúnmente es estructurada en niveles o capas. Cada uno de estos niveles realizará un subconjunto de las funcionalidades propias necesarias para el intercambio de información. Es necesario integrar las funcionalidades propias de la seguridad en

las arquitecturas de comunicaciones. Este proceso de integración implicará la implementación de mecanismos, servicios y funciones de seguridad. El resultado final será lo que denominaremos *Arquitectura de Seguridad*.

Los servicios de seguridad utilizan los mecanismos implementados en dicha arquitectura. Un mecanismo de seguridad implica un intercambio de unidades de información entre entidades, además de la realización de una serie de funciones aplicadas a la información, por ejemplo: el cifrado de datos, o un acuerdo de llaves. Este intercambio entre entidades, que tiene por objeto implementar un servicio de seguridad, lo denominaremos *protocolo de seguridad*.

Por lo tanto, cuando se dice que una red de datos es segura, se espera que proporcione los servicios anteriormente mencionados, para que se garantice la distribución de la información de manera segura, logrando que los datos lleguen al destinatario de manera íntegra, que los contenidos no son revelados, y que mantenga privacidad de las entidades involucradas, entre otras. De manera genérica, estos mecanismos de seguridad podrían ser específicos, o se podrían integrar con los mecanismos propios de las arquitecturas de comunicaciones, con el fin de ofertar al usuario final servicios de seguridad.

La criptología es la ciencia que trata los problemas teóricos relacionados con la seguridad en el intercambio de mensajes cifrados entre dos entidades a través de un medio de comunicación.

De manera general, la criptología está dividida en dos grandes áreas: la criptografía y el criptoanálisis [Paar and Pelzl, 2010] (véase figura 1.1).

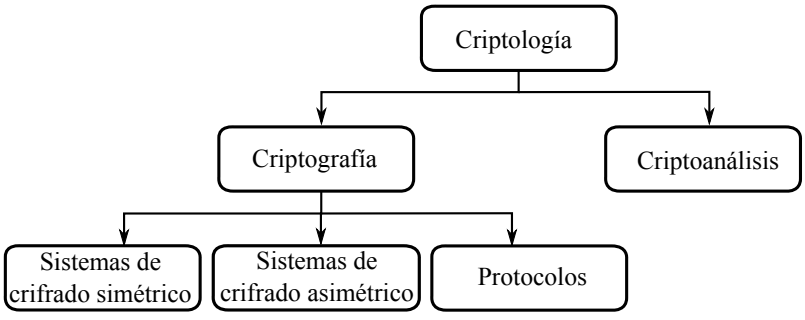


Figura 1.1: Clasificación general de la criptología

La criptografía es la ciencia de la escritura secreta con el objetivo de ocultar el significado de un mensaje. El criptoanálisis es la ciencia y a veces arte de romper sistemas criptográficos. El criptoanálisis es realmente importante para sistemas criptográficos modernos: sin gente que trate de romper nuestros métodos criptográficos, nunca sabremos si realmente son seguros o no.

1.2 Esquemas de cifrado

Uno de los objetivos de la criptografía consiste en diseñar sistemas basados en técnicas matemáticas que logren obtener una comunicación segura en canales que no lo son [Trappe and Washington, 2006]. La criptografía da lugar a diferentes tipos de sistemas criptográficos que permiten asegurar los principios de la seguridad informática. Así pues, los protocolos de seguridad permiten llevar al cabo las metas de seguridad. La criptografía es la principal herramienta matemática de la que se sirven dichos protocolos, los cuales brindan los servicios necesarios para proteger tanto a los datos como a las entidades involucradas en la comunicación.

De manera general, los esquemas de cifrado pueden ser clasificados en simétricos y asimétricos. El primero, también es conocido como criptografía de llave privada, este tipo de esquemas utilizan una única llave que es usada para el cifrado y el descifrado. Los esquemas asimétricos, o también conocidos como criptografía de llave pública, utilizan un par de llaves complementarias denominadas llave pública y llave privada respectivamente.

Actualmente la combinación de estas técnicas criptográficas forma la base de los mecanismos de seguridad [Stallings, 2002] que se utilizan en cualquier aplicación que requiera un intercambio seguro de información. En seguida se describe brevemente en que consiste cada tipo.

1.2.1 Criptografía simétrica o de llave privada

En este esquema se supone que las partes involucradas son las únicas entidades que poseen la llave secreta (emisor y receptor), con la cual es posible cifrar y descifrar los mensajes enviados entre dos o más partes. La fortaleza de los algoritmos de llave privada radica principalmente en que la llave solo es conocida por las entidades que desean transmitir información de manera confidencial. Cabe destacar que este tipo de criptografía es simple y eficiente de implementar, pero uno de los principales problemas con este esquema es la distribución de la llave [Menezes et al., 1996]. En la figura 1.2 se muestra el esquema de criptografía de llave privada.

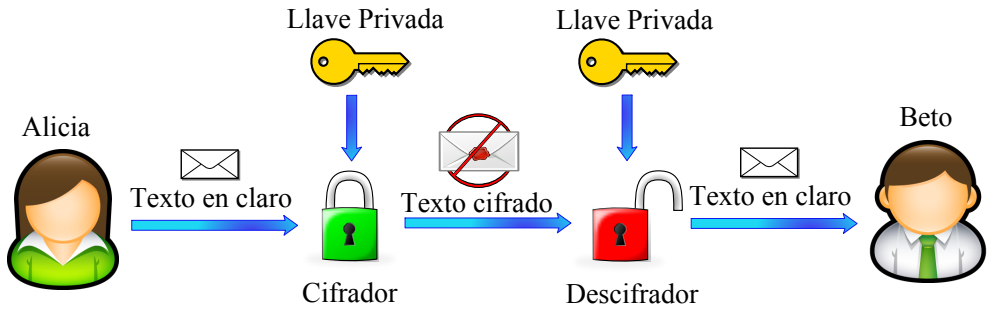


Figura 1.2: Esquema de cifrado de llave privada.

El problema de distribución de llaves, de manera general, se divide en dos problemáticas: la primera se presenta cuando se establece una llave para la comunicación entre dos entidades, ya se requiere que el intercambio de ésta sea seguro, es decir, que la llave se mantenga en secreto. Es mucho más fácil para un atacante intentar interceptar una llave que probar las todas posibles combinaciones del espacio de llaves.

La segunda consiste en que a medida que crece el número de participantes aumenta el número de veces que se debe repetir la operación de intercambio. En general, la distribución de llaves de manera segura se vuelve muy complicada cuando se realiza entre un número considerablemente grande de entidades.

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].

1.2.2 Criptografía asimétrica o de llave pública

En este tipo de sistemas, cada entidad posee dos llaves distintas: una llave pública y una llave privada. Como su nombre lo indica, la llave pública puede ser conocida por cualquiera. Por lo general la llave pública será la que se utilice para realizar el proceso de cifrado. Por otra parte, la llave que es utilizada para el descifrado de la información es conocida como llave privada, y será conocida únicamente por el dueño de la misma. Ambas claves están relacionadas matemáticamente, y deberá ser computacionalmente imposible conocer la clave privada a partir de la pública. En el contexto de firmas digitales, la llave privada se usa para firmar y la llave pública para la verificación de la firma. En la figura 1.3 se muestra el esquema de criptografía de llave pública.

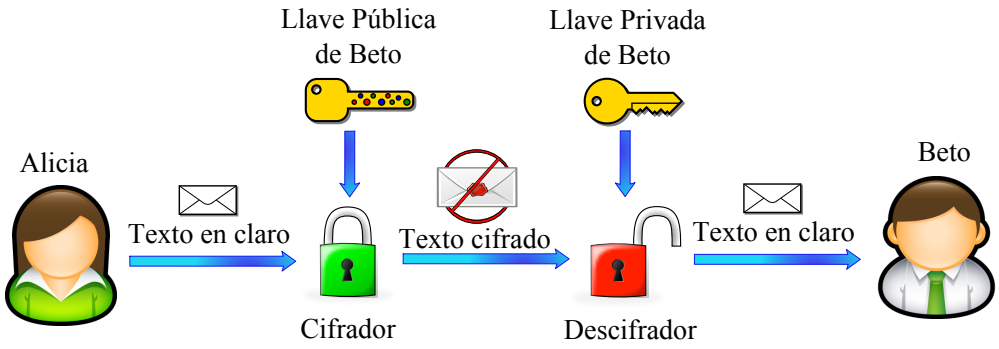


Figura 1.3: Esquema de cifrado de llave pública.

Una de las primitivas criptográficas usada dentro de este tipo de esquemas es el emparejamiento bilineal. Introducidos por André Weil [Weil, 1948] en 1948, los emparejamientos bilineales sobre curvas elípticas han servido para la propuesta de nuevos protocolos criptográficos, debido a las propiedades constructivas descubiertas por Joux [Joux, 2004], Sakai [Sakai et al., 2000] y Mitsunari [Mitsunari et al., 2002].

El principal interés de este trabajo de tesis se enfoca en los protocolos basados en emparejamientos bilineales, los cuales caen dentro de la clasificación del tipo de esquemas asimétricos.

1.3 Protocolos Criptográficos

En cualquier escenario donde se realizan transacciones electrónicas existen diferentes entidades, las reglas a seguir que establecen una comunicación segura las denominaremos como *protocolos criptográficos*, .

Un *protocolo criptográfico* es un algoritmo distribuido definido por una secuencia de pasos que especifican, de manera precisa, las acciones requeridas por dos o más entidades para llevar a cabo un objetivo de seguridad específicos [Menezes et al., 1996]. Los protocolos criptográficos usan funciones de criptografía en algunos o todos los pasos que los componen.

Con base en la funcionalidad, se pueden clasificar los protocolos dependiendo de los servicios de seguridad que se cubren [Dutta et al., 2004], entre los principales tenemos:

- *Cifrado de datos*. Es el proceso por el que una información legible es transformada, mediante un algoritmo, en información ilegible.
- *Firma digital*. Es el método que permite asociar la identidad de una entidad a un documento como autor del mismo.
- *El acuerdo de llaves*. Es la manera en que las entidades participantes se ponen de acuerdo en el valor de una información secreta, donde dos o más participantes requieren establecer un secreto en común.
- *La criptografía con umbral*. Es útil para eliminar las fallas en la centralización y proporciona resistencia a la corrupción por parte del adversario de un cierto porcentaje de las entidades que integran un sistema distribuido.

1.4 Problemas de complejidad en los que se basan los protocolos

Una diferencia conceptual entre primitivas y protocolos criptográficos se presenta en las garantías de seguridad que brinda cada uno. Las primitivas criptográficas están diseñados para preservar la seguridad en algunos escenarios bien definidos, mientras que los protocolos deben resistir diferentes ataques dirigidos a objetivos diferentes.

En esta sección se enlistan los problemas con una complejidad matemática y computacional que son difíciles de resolver, dichos problemas son asociados a los protocolos, ya que mediante la dificultad para resolverlos es que los protocolos brindan la seguridad que se desea. Entenderemos que un problema difícil, es aquel para el cual no se conoce algún algoritmo en tiempo polinomial para resolverlo [Menezes et al., 1996].

1.4.1 Problema de logaritmo discreto

Discrete Logarithm Problem (DPL). Sea $\mathbb{G}$ un grupo cíclico multiplicativo de orden $n-1$, g un elemento primitivo de $\mathbb{G}$, y $\alpha \in \mathbb{G}$. Dados g y α, el problema del logaritmo discreto consiste en encontrar $x \in \mathbb{G}$, tal que $\alpha = g^x$, donde $1 \leq x \leq n-1$.

Elliptic Curve Discrete Logarithm Problem (ECDLP). Sean P, Q puntos sobre una curva elíptica E. Dados P y Q, el problema de logaritmo discreto en curvas elípticas consiste en encontrar d, tal que $Q = [d]P$, donde d es menor que el orden del punto, y $[d]P$ es el resultado de hacer la suma del punto P consigo mismo d veces.

1.4.2 Problemas de Diffie-Hellman

Diffie-Hellman Problem o Computational Diffi-Hellman Problem (DHP o CDHP). Sean $\mathbb{G}$ un grupo cíclico multiplicativo de orden n y $g \in \mathbb{G}$ un elemento primitivo. Dados g, g^a y g^b, el

Problema Computacional de Diffie-Hellman consiste en encontrar un elemento $h \in \mathbb{G}$ tal que $h = g^{ab}$.

Elliptic Curve Diffie-Hellman Problem (ECDHP). Sean P, Q puntos sobre una curva elíptica E. Dados los puntos $P, [a]P, [b]P \in E$, el problema de Diffie-Hellman sobre curvas elípticas consiste en encontrar el punto $Q = [ab]P$.

Bilinear Dffie-Hellman Problem (BDHP). Sean $\hat{e}$ un emparejamiento bilineal $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, donde $\mathbb{G}_1$, $\mathbb{G}_2$ son grupos aditivos y $\mathbb{G}_T$ un grupo multiplicativo. Además, $a, b, c \in \mathbb{Z}_n^*$, $P \in \mathbb{G}_1$ y $Q \in \mathbb{G}_2$ puntos sobre la curva elíptica E. Dados P, aP, bP, cP, el Problema de Diffie-Hellman sobre emparejamientos consiste en encontrar $\hat{e}(P, Q)^{abc}$.

Decisional Diffie-Hellman Problem (DDHP). Sean $a, b, c \in \mathbb{Z}_n^*$, $\mathbb{G}$ un grupo multiplicativo de orden n, y $g \in \mathbb{G}$ un elemento primitivo. Dados g, g^a, g^b, g^c, el problema de decisión de Diffie-Hellman consiste en determinar si $g^c = g^{ab} \bmod n$.

Elliptic Curve Decisional Diffie-Hellman Problem (ECDDHP), Sean $a, b, c \in Z_n^*$ y P un punto sobre la curva elíptica E. Dados los puntos $P, [a]P, [b]P, [c]P$, el problema de decisión Diffie-Hellman sobre curvas elípticas pide determinar si el punto $[c]P = [ab]P$

Decisional Bilinear Diffie-Hellman Problem (DBDHP). Sean $\hat{e}$ un emparejamiento en $(\mathbb{G}_1, \mathbb{G}_T)$, $\mathbb{G}_1$ es un grupo aditivo y $\mathbb{G}_T$ un grupo multiplicativo. Además, $a, b, c \in \mathbb{Z}_n^*$. Dados P, $[a]P$, $[b]P$, $[c]P$, el problema de decisión de Diffie-Hellman sobre emparejamientos bilineales consiste en determinar si $\hat{e}(P, P)^c = \hat{e}(P, P)^{ab}$, que se puede reducir a determinar si $c \equiv ab \bmod n$.

Gap Diffie-Hellman Problem (GDHP). Los grupos para los cuales el DDHP sea fácil y el CDHP sea difícil, se conocen como grupos con el problema de brecha de Diffie-Hellman.

1.4.3 Relación entre los problemas

Para tratar de entender la dificultad para encontrar una solución a los problemas antes mencionados, en las siguientes líneas se muestra la relación entre los problemas. La notación

utilizada: *Problema_1* $\rightarrow$ *Problema_2*, la cual indica que al encontrar una solución al *Problema_1*, el *Problema_ 2* se vuelve fácil, pero esta notación no indica, que si el *Problema_1* es difícil el *Problema_2* también lo será, esto dependerá del grupo.

$$DLP \rightarrow CDHP \rightarrow DDHP$$

$$DLP \text{ en } \mathbb{G}_1 \rightarrow CDHP \text{ en } \mathbb{G}_1 \rightarrow DDHP \text{ en } \mathbb{G}_1$$

$$DLP \text{ en } \mathbb{G}_T \rightarrow CDHP \text{ en } \mathbb{G}_T \rightarrow DDHP \text{ en } \mathbb{G}_T$$

$$DLP \text{ en } \mathbb{G}_1 \text{ y } DLP\mathbb{G}_T \rightarrow BDHP \rightarrow DBDHP$$

1.5 Motivación

El enfrentamiento constante entre la criptografía y el criptoanálisis[1] ha provocado que día a día se busquen nuevos métodos para la protección de la información, los cuales sean más robustos y sofisticados, buscando cumplir con las necesidades de velocidad y seguridad que se requieren hoy en día.

En medio del desarrollo tecnológico, provocado por esta competencia, surgen la criptografía de curvas elípticas, y el uso de los emparejamientos bilineales. El uso de los emparejamientos permite la implementación de esquemas y protocolos de cifrado y de firma digital, como: el protocolo de Diffie-Hellman tripartito [Joux, 2004] , la criptografía basada en la identidad [Boneh and Franklin, 2001], los esquemas de firma corta [Boneh et al., 2001] y muchos más [Dutta et al., 2004].

La evolución de aplicaciones de comunicación y de comercio electrónico, ha requerido la introducción de nuevos protocolos criptográficos que logren satisfacer los diferentes servicios de seguridad necesarios para estas aplicaciones. Por tal motivo, resulta útil contar con herramientas

[1] Es la parte de la criptología que se dedica a el estudio de sistemas criptográficos con el fin de encontrar debilidades en los sistemas y verificar su seguridad.

que faciliten las tareas durante el proceso de diseño e implementación de los nuevos esquemas. Por ejemplo, puede resultar útil para un diseñador de protocolos criptográficos el contar con un lenguaje que le permita modelar en detalle los bloques básicos, las primitivas criptográficas, las interacciones entre entidades y el intercambio de mensajes que forman parte de la descripción de protocolos criptográficos modernos. Así mismo, para el implementador de protocolos, resulta importante contar con una base mediante la cual se pueda construir la implementación final de un protocolo de manera eficiente.

Al momento de trabajar con protocolos criptográficos, se tiene el problema del diseño, el cual requiere de un trabajo previo y riguroso antes de poder realizar la pruebas del mismo. Por otro lado, al momento de implementar un protocolo, por lo general, no se cuentan con herramientas que faciliten la tarea. Mediante el uso de técnicas para la descripción o especificación, se puede detallar un sistema y su funcionamiento mediante el uso de un lenguaje abstracto de alto nivel. Dentro de la literatura de especificación protocolos no se presenta una manera estandarizada para describirlos, la forma en que se realiza dicha descripción depende exclusivamente del diseñador del protocolo. La manera en que se interpreta e implementa el protocolo queda en total libertad del implementador.

Por lo tanto, en este trabajo de tesis se desea generar la implementación de protocolos criptográficos basados en emparejamientos bilineales, partiendo de una descripción formal del protocolo. Por lo cual es necesario contar con un lenguaje capaz de describir las operaciones abstractas que se realizan, junto con las entidades que conforman dicho protocolo. La propuesta de dicho lenguaje y la herramienta encargada de realizar el proceso de traducción fueron desarrolladas como parte de este trabajo de investigación. La finalidad de dicha herramienta es transformar la descripción de entrada dado un lenguaje de descripción de protocolos, a el código fuente de un programa en lenguaje Magma y C/C++ de la descripción proporcionada, y en este caso, compilar para un equipo de escritorio con un procesador de 64 bits. Las ideas planteadas en [Dominguez Perez and Scott, 2009] se usaron como referencia en este caso.

1.6 Metodología

En esta sección se describe de manera breve las etapas del proceso que se realizó durante este trabajo de tesis; se puede comparar el proceso con el modelo ciclo de vida en espiral [Boehm, 1986]. El proceso está dividido en tres etapas, las cuales se describen en seguida:

1. En una primer etapa se realizó un estudio de los protocolos basados en emparejamientos bilineales, junto con las primitivas criptográficas involucradas. La finalidad fue la de identificar las diversas primitivas criptográficas utilizadas en la implementación de los mismos, y diseñar una propuesta para el lenguaje de descripción de protocolos.

2. Posteriormente, se desarrolló un compilador para el lenguaje de descripción de protocolos. Para tal efecto, se utilizó ANTLR (véase la sección 5.1.2). Nuestro compilador recibe como entrada de datos un archivo con la especificación de un protocolo, utilizando nuestra propuesta de lenguaje para descripción de protocolos, y genera como salida la implementación en lenguaje Magma, y en lenguaje C/C++ de dicho protocolo.

3. Finalmente en una tercer etapa, se implementaron las primitivas criptográficas identificadas en la primer fase del proyecto.

   Además de éstas primitivas, se implementaron las funciones extras identificadas en diversos protocolos, junto con aquellas necesarias para la entrada y salida de datos requerida por los mismos.

Las interacciones que se tienen entre éstas etapas pueden verse ilustradas en la figura 1.4. El proceso busca hacer más robusto el poder del compilador mediante cada etapa de pruebas.

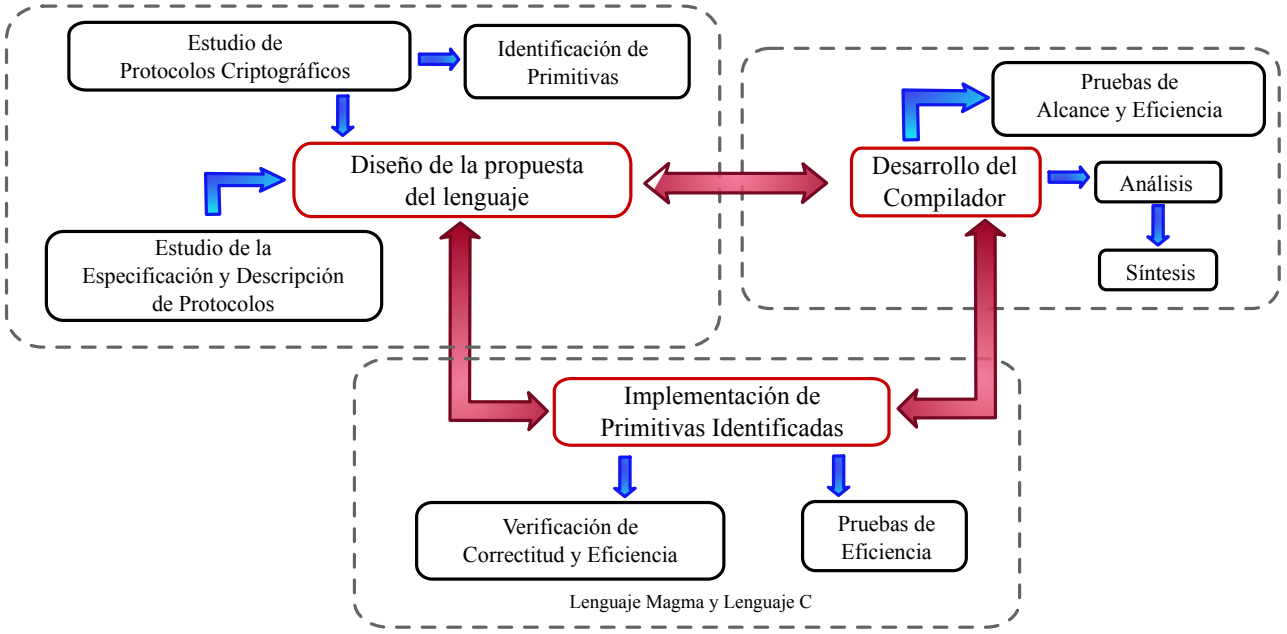


Figura 1.4: Metodología del proceso de trabajo.

1.7 Objetivos

Ésta tesis tiene como objetivo general *desarrollar un compilador que tome como entrada una descripción de un protocolo criptográfico basado en emparejamientos bilineales, y genere a su salida la implementación eficiente en lenguaje C/C++ y lenguaje Magma de dicha descripción.*

Los objetivos particulares de este trabajo son:

1. Brindar el diseño de una propuesta de un lenguaje para la descripción de protocolos criptográficos basados en emparejamientos bilineales.

2. Verificar el potencial del compilador desarrollado, mediante la verificación de la traducción del código.

1.8 Organización de la tesis

El contenido de la tesis ha sido organizado en seis capítulos: el capítulo 2 retoma los conceptos generales de la teoría de compiladores, los cuales son indispensables para la comprensión de este trabajo.

En el capítulo 3 se proporciona una visión general de las principales primitivas criptográficas utilizadas en el desarrollo de este trabajo de tesis. El capítulo 4 brinda una visión de los protocolos criptográficos basados en emparejamientos bilineales, proporcionando un análisis de cómo se especifican o describen dichos protocolos dentro de la literatura.

En el capítulo 5 se muestra el diseño del compilador desarrollado, junto con el análisis que se realizó para dar una propuesta de un lenguaje para descripción de protocolos. También, se muestran los resultados obtenidos sobre la generación automática de código, y se da una visión general del lenguaje propuesto.

Finalmente el capítulo 6 contiene un resumen de los resultados más relevantes obtenidos, las conclusiones, y se enlistan las propuestas de trabajo futuro.

2

Teoría de Compiladores

En este trabajo de tesis se desarrolló una herramienta capaz de traducir la descripción de un protocolo criptográfico a código en lenguaje Magma o lenguaje C/C++. Este capítulo tiene el objetivo de brindar al lector la teoría básica sobre los traductores.

2.1 Introducción

Un traductor [Aho et al., 1998] es un programa (o conjunto de programas) que lee un programa escrito en un lenguaje *fuente*, y lo traduce a un programa equivalente escrito en otro lenguaje, el lenguaje *objeto*. Como parte importante de este proceso de traducción, el compilador debe informar al usuario la presencia de errores en el programa fuente (véase figura 2.1).

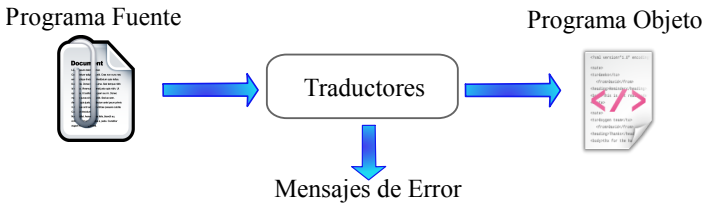


Figura 2.1: Un compilador

2.2 Tipos de traductores

Desde el inicio de la computación, ha existido un gran abismo en la forma como las personas expresan sus necesidades y la forma en que un computador es capaz de interpretar esas instrucciones. Cuando aplicamos la teoría de autómatas dentro de diferentes campos y áreas de la informática, podemos obtener distintos tipos de traductores:

- *Traductores de idioma*: Este tipo de programas traduce un idioma a otro, por ejemplo del español al inglés.

- *Compiladores*: Son aquellos traductores que tienen como entrada la sentencia de algún lenguaje formal y como salida tienen un programa ejecutable. El nombre de "compilador" se utiliza sobre todo para los programas que traducen el código fuente de un lenguaje de programación de alto nivel a uno de bajo nivel. También genera un archivo objeto en lugar de un programa ejecutable final.

- *Intérpretes*: En esencia, es un compilador, la única diferencia es que la salida de los intérpretes es una ejecución y no un programa ejecutable, es decir, el programa de entrada se lee y se ejecuta a la vez. No se produce un resultado físico, a diferencia del caso de los compiladores, que se produce un código objeto.

- *Preprocesadores*: Permiten modificar el programa fuente antes de la verdadera compilación. Hacen uso de macroinstrucciones y directivas de compilación. Los preprocesadores suelen

actuar de manera transparente para el programador, pudiendo incluso considerarse que son una fase preliminar del compilador.

- *Intérpretes de comandos*: Un intérprete de comandos traduce sentencias simples a invocaciones de programas de una biblioteca. Son utilizados comúnmente en los sistemas operativos.

- *Conversores fuente-fuente*: Permiten traducir de un lenguaje de alto nivel a otro de alto nivel, con lo que se consigue una mayor portabilidad en los programas de alto nivel.

- *Compilador cruzado*: Es un tipo de compilador que genera código para ser ejecutado en una máquina diferente a donde se realiza el proceso de compilación.

- *Metacompilador*: Es sinónimo de compilador de compiladores y se refiere a un programa que recibe como entrada las especificaciones del lenguaje para el que se desea obtener un compilador y genera como salida el compilador para ese lenguaje.

- *Descompilador*: es un programa que acepta como entrada código máquina y lo traduce a un lenguaje de alto nivel, realizando el proceso inverso a la compilación.

2.3 Estructura de un compilador

Conceptualmente, un compilador opera en fases, cada una de las cuales transforma el programa fuente de una representación a otra. El proceso de compilación está dividido en dos partes: análisis y síntesis. La parte de análisis divide a el programa fuente en sus elementos componentes y genera una representación intermedia del mismo. Por otra parte, la síntesis construye el programa de salida partiendo de la representación intermedia. En la figura 2.2 puede observarse la descomposición típica de un compilador. En la práctica, pueden agruparse

algunas de las fases y las representaciones intermedias entre las fases agrupadas no necesitan ser construidas explícitamente.

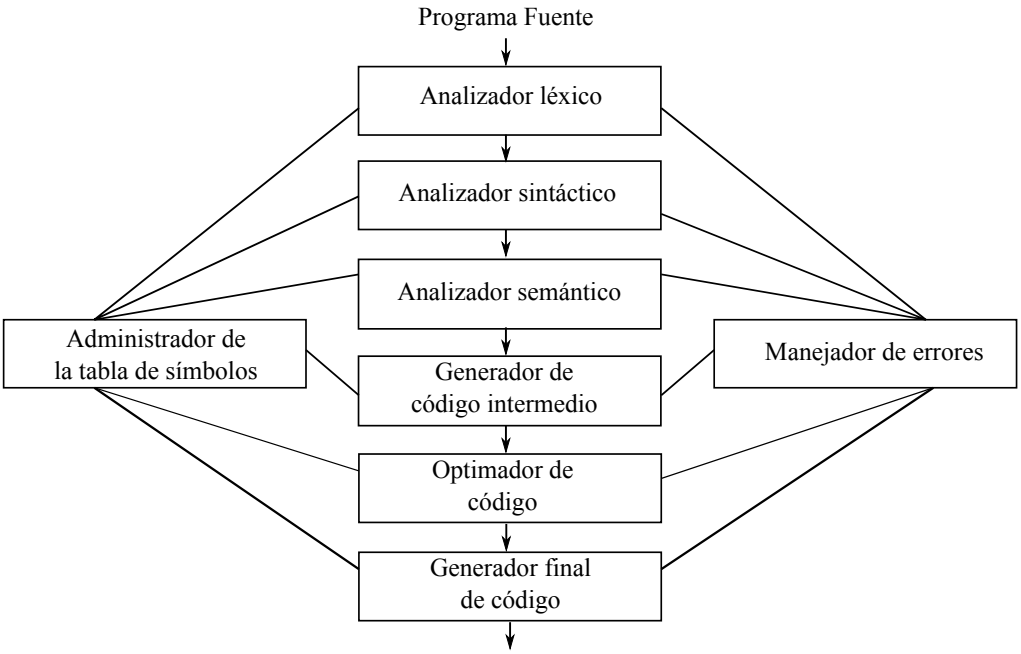


Figura 2.2: Etapas de un compilador.

2.3.1 Análisis

Para poder realizar la transformación de lenguaje, el compilador debe realizar un análisis de qué es lo que se ha escrito en el lenguaje de programación establecido, a través de tres criterios fundamentales:

- *Análisis léxico*: Verifica que el lenguaje sea correctamente aplicado. Durante esta etapa, se divide el programa fuente en los componentes básicos del lenguaje a compilar. Cada componente básico es una subsecuencia de caracteres del programa fuente, y pertenece a una categoría gramatical, es decir: números, identificadores, palabras reservadas, signos de puntuación, etc.

- *Análisis sintáctico*: Revisa que la programación esté ordenada, sin fallas en sus variables,

es decir, comprueba que la estructura de los componentes básicos sea correcta según las reglas gramaticales del lenguaje que se compila.

- *Análisis semántico*: Se verifica que el resultado de lo que ha sido programado tiene una correlación con la orden que ha querido ser ejecutada. Comprueba que el programa fuente respete las directrices del lenguaje que se compila, es decir, todo lo relacionado con el significado: verificación de tipos, rangos de valores, existencia de variables, etc.

Cualquiera de estas fases puede emitir mensajes de error derivados de fallos cometidos por el programador en la redacción de los textos fuente, podría decirse que mientras más errores controle un compilador, menos problemas dará un programa en tiempo de ejecución.

2.3.2 Síntesis

Una vez terminada la etapa de análisis, comienza una etapa en la cual se sintetiza el código de salida, es decir, se construye el programa objeto deseado (equivalente semánticamente al fuente). Dicha tarea generalmente es dividida en tres pasos:

- *Generación de código Intermedio*: Se genera una representación intermedia explícita del programa fuente. Se puede considerar esta representación intermedia como un programa para una máquina abstracta. Esta representación debe tener dos propiedades importantes; debe de ser fácil de producir y fácil de traducir al programa objeto.

- *Optimización de código*: Esta fase tiene como objetivo principal generar una versión mejorada del código intermedio, de modo que resulte un código máquina más rápido para ejecutar, con la finalidad de buscar un mayor rendimiento y tener menos errores.

- *Generación de código*: Esta es la ultima fase en el proceso de compilación. Se da lugar a la creación de un código de salida equivalente al programa fuente, que por lo general consiste en código máquina o código ensamblador.

2.3.3 Tabla de símbolos

Durante las etapas del análisis y síntesis, una función esencial del compilador es registrar los identificadores utilizados en el programa fuente, además, se debe reunir la información sobre los distintos atributos de cada identificador. Estos atributos pueden proporcionar información sobre la memoria asignada a un identificador, su tipo, su ámbito (la parte del programa donde tiene validez) y, en el caso de los procedimientos, cosas como el número y tipos de los argumentos.

Una *tabla de símbolos* es una estructura que contiene un registro por cada identificador, con los campos para los atributos de cada identificador. La estructura debe permitir encontrar rápidamente el registro en cada identificador. Además, se debe poder almacenar o consultar velozmente los datos de los registros.

2.3.4 Manejador de errores

En cada fase se pueden encontrar errores. Sin embargo, después de detectar un error, cada fase debe tratar de una manera ese error, para poder continuar con la compilación, permitiendo la detección de más errores en el programa fuente. Un compilador que se detiene al encontrar el primer error, no resulta tan útil como debiera.

En las etapas del análisis sintáctico y semántico por lo general manejan un gran porcentaje de los errores manejados por el compilador. Por ejemplo, en la fase léxica se puede detectar errores donde los caracteres en la entrada no representen ningún componente léxico válido. Durante el análisis semántico, el compilador intenta detectar construcciones que no tengan una estructura sintáctica correcta, por ejemplo, intentar sumar dos identificadores de diferente tipo de dato.

2.4 Expresiones regulares

Una expresión regular es un patrón que permite encontrar determinados caracteres, palabras o cadenas. Las expresiones regulares se escriben en un lenguaje formal que puede ser interpretado por un procesador de expresiones regulares.

Las expresiones regulares son usadas por muchos editores de textos y lenguajes de programación para buscar y manipular textos basándose en patrones. Una expresión regular se encuentra completamente definida mediante el conjunto de cadenas con las que concuerda. A pesar de que las expresiones regulares estén muy extendidas por el mundo de Unix, no existe un lenguaje estándar de expresiones regulares.

Definición 2.1 (Alfabeto). *Conjunto de caracteres que aparecen en el documento a revisar, para generalizar nos referiremos, siempre a todo el juego ASCII de caracteres.*

Definición 2.2 (Lenguaje Universal). *Conjunto de todas las posibles secuencias de caracteres de ese alfabeto.*

El Lenguaje Universal es demasiado amplio y no permite ningún tipo de restricciones a la hora de definir las secuencias de caracteres. Por eso nos interesa definir lenguajes más restrictivos que nos permitan localizar solamente aquellas cadenas de texto (secuencias de caracteres del alfabeto) que nos interesan.

Una expresión regular nos sirve para definir lenguajes, imponiendo restricciones sobre las secuencias de caracteres que se permiten en el lenguaje que estamos definiendo. Por tanto, la expresión estará formada por el conjunto de caracteres del alfabeto original, más un pequeño conjunto de caracteres extra, a estos caracteres se les conoce como *meta-caracteres*, los cuales nos permiten definir las restricciones. El conjunto de meta-caracteres más utilizado se puede apreciar en la tabla 2.1.

Nombre	Carácter	Significado
Cierre	'*'	El elemento precedente debe aparecer 0 o más veces
Cierre positivo	'+'	El elemento precedente debe aparecer 1 o más veces
Comodín	'.'	Un carácter cualquiera excepto salto de línea
Condicional	'?'	El elemento precedente es opcional
OR	'|'	Operador OR entre dos elementos. En el lenguaje aparecerá o uno u otro.
Operador de rango	'–'	Dentro de un conjunto de caracteres escrito entre corchetes, podemos especificar un rango
Salto de línea	'$\backslash n$'	Carácter de salto de línea
Tabulador	'$\backslash t$'	Carácter de tabulación
	[···]	Conjunto de caracteres admitidos
	[^ ···]	Conjunto de caracteres no admitidos
	(···)	Agrupación de varios elementos

Tabla 2.1: Meta-caracteres principales.

2.5 Gramáticas libres de contexto

Estas gramáticas, conocidas también como gramáticas de tipo 2 o gramáticas independientes del contexto, son las que generan los lenguajes libres o independientes del contexto. Los lenguajes libres del contexto son aquellos que pueden ser reconocidos por un autómata de pila[1] determinístico o no determinístico.

Como toda gramática se definen mediante una cuádrupla $G = (N, T, P, S)$, tal que

- N es un conjunto finito de símbolos no terminales. Los terminales son los símbolos básicos con que se forman las cadenas.

- T es un conjunto finito de símbolos terminales. Los no terminales son variables sintácticas que denotan conjuntos de cadena, las cuales ayudan a definir el lenguaje generado por la gramática.

[1]Un autómata con pila es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

- P es un conjunto finito de producciones de la forma $A \to \alpha$, donde A es un no terminal y $\alpha \in (N \cup T)^*$. Las producciones especifican cómo se pueden combinar los terminales y no terminales para formar cadenas.

- S es una variable designada, llamada el símbolo inicial.

Ejemplo 2.1. *Supongamos que usamos E para identificar una expresión. La siguiente gramática $\mathcal{A}$ identifica expresiones aritméticas*

$$E \to E + E | E * E | (E) | id$$

entonces, formalmente está gramática $\mathcal{A}$ se expresa como: $\mathcal{A} = (\{E\}, \{+, *, (.)\}, P, E)$.

2.5.1 BNF

Las gramáticas libres del contexto se escriben, frecuentemente, utilizando una notación conocida como BNF (Backus-Naur Form). BNF es la técnica más común para definir la sintaxis de los lenguajes de programación.

En esta notación se deben seguir las siguientes convenciones:

- Los no terminales se escriben entre paréntesis angulares <>.

- Los terminales se representan con cadenas de caracteres sin paréntesis angulares.

- El lado izquierdo de cada regla debe tener únicamente un no terminal (ya que es una gramática libre del contexto)

- El símbolo $\to$, se lee "se define como" o "se reescribe como".

2.5.2 Árbol de derivación

Un *árbol*[2] *de derivación* permite mostrar gráficamente cómo se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera ese lenguaje.

El árbol de derivación tiene las siguientes propiedades:

- El *nodo raíz*[3] está rotulado con el símbolo distinguido de la gramática.
- Cada *hoja*[4] corresponde a un símbolo terminal o un símbolo no terminal.
- Cada *nodo interior*[5] corresponde a un símbolo no terminal.

Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como rótulo uno de los símbolos de la cadena.

2.5.3 Gramáticas ambiguas

Una gramática es ambigua si permite construir dos o más árboles de derivación distintos para la misma cadena. Por lo tanto, para demostrar que una gramática es ambigua, lo único que se necesita es encontrar una cadena que tenga más de un árbol de derivación. Una gramática en la cual, para toda cadena generada w, todas las derivaciones de w tienen el mismo árbol de derivación, es no ambigua. En algunos casos, dada una gramática ambigua, se puede encontrar otra gramática que produzca el mismo lenguaje pero que no sea ambigua.

[2] Un árbol es un conjunto de puntos, llamados nodos, unidos por líneas, llamadas arcos.
[3] La raíz es el único nodo distinguido que no tiene arcos incidentes.
[4] Una hoja es un nodo que no tiene hijos.
[5] Los nodos interiores son todos aquellos que no son una hoja.

3

Primitivas Criptográficas

Las primitivas criptográficas son funciones o algoritmos de bajo nivel que se utilizan con frecuencia en el desarrollo de esquemas criptográficos, son los ladrillos mediante los cuales se construyen los protocolos criptográficos. Las primitivas criptográficas incluyen, pero no se limitan a: las funciones de cifrado, funciones picadillo, y la generación de números aleatorios.

Durante el desarrollo de este trabajo de tesis se estudiaron diversos protocolos criptográficos basados en emparejamientos, el objetivo consistió en identificar las principales primitivas criptográficas utilizadas por los mismos. Por tal motivo, en este capítulo se presentan las principales primitivas identificadas.

3.1 Multiplicación escalar

Dado un punto P y un escalar $k \in \mathbb{Z}$ se trata de calcular el punto Q aplicando k-veces el operador $+$ sobre el punto P, la operación se denota como $Q = [k]P$

$$Q = [k]P = \underbrace{P + P + \cdots + P}_{k \text{ veces}}$$

El orden de un punto P es el entero más pequeño k tal que $[k]P = \mathcal{O}$. El orden de cualquier punto está siempre bien definido, y divide el orden de la curva $\#E(\mathbb{F}_q)$. Esto garantiza que si r y ℓ son enteros, entonces $[r]P = [\ell]P$ si y sólo si $r = \ell \bmod k$, donde k tiene que ser primo para cumplir está propiedad de manera unívoca.

Una forma sencilla de llevar a cabo la multiplicación por un escalar es mediante una adaptación del conocido método de "elevar al cuadrado y multiplicar" para exponenciación modular.

La idea consiste en representar el escalar k en forma binaria, y recorrer los bits de éste mientras se dobla el punto Q incondicionalmente, y realizando sumas de puntos si el bit actual o evaluado es 1, tal como se muestra en el algoritmo 3.1.

Algoritmo 3.1 Método de izquierda a derecha para la multiplicación escalar

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

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

Cabe resaltar que en el caso de este trabajo de tesis la multiplicación escalar quedará

denotada como $[k]P$.

El método de sumas y doblados presentado en el algoritmo 3.1 tiene en promedio un costo de n doblados y $n/2$ sumas, si consideramos que los bits del escalar están uniformemente distribuidos. Existen otros métodos para realizar la multiplicación escalar que a cambio de realizar un precómputo buscan mejorar la eficiencia de la implementación.

3.1.1 Método de ventana ω-NAF

Definición 3.1 (Representación ω-NAF). *Sea $\omega \leq 1$ 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 K_i diferente de cero es un número impar $|k_i| 2^{\omega-1}$, $k_{l-1} \neq 0$. 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 hasta un dígito mayor que en la representación binario de k, Así mismo, la densidad de elementos diferentes de cero con respecto al tamaño de l de la representación ω-NAF es $1/(\omega + 1)$.

Utilizando el algoritmo 3.2 puede obtenerse el $NAF_\omega(k)$ de una forma eficiente. En este algoritmo $k \bmod 2^\omega$ 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 al entero k entre 2, lo que permite la obtención del residuo r en el intervalo $[-2^{\omega-1}, 2^{\omega-1} - 1]$. Si k es un número impar y el residuo $r = k \bmod 2^\omega$ es seleccionado, entonces $(k - e)/2$ será un número divisible por $2^{\omega-1}$, asegurando que los siguientes dígitos sean cero.

El algoritmo 3.3 presenta la versión modificada del método de izquierda a derecha para la multiplicación (véase el algoritmo 3.1). El algoritmo realiza la multiplicación usando la transformación ω-NAF(k) en lugar de la representación binaria de k. El costo del algoritmo

Algoritmo 3.2 Obtención del ω-NAF de entero positivo

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

$i \leftarrow 0$
while $k \geq 1$ **do**
  if k es impar **then**
    $k_i \leftarrow k \bmod 2^\omega$, $k \leftarrow k - k_i$
  else
    $k_i \leftarrow 0$
  end if
  $k \leftarrow k/2$, $i \leftarrow i + 1$
end while
return $(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$

es:

$$[1D + (2^{\omega-2} - 1)A] + [(m/(\omega + 1))A + mD]$$

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

Algoritmo 3.3 Multiplicación escalar: Método ω-NAF

Entrada: Entero positivo k, $P \in E(\mathbb{F}_q)$
Salida: $[k]P$

Usar el algoritmo 3.2 para computar $NAF(k) = \sum_{i=0}^{i-1} k_i 2^i$
Obtener $P_i = [i]P$ para $i \in [1, 2, 5, \ldots, 2^{\omega-1} - 1]$
$Q \leftarrow \mathcal{O}$
for $i = l - 1$ **to** 0 **do**
  $Q \leftarrow [2]Q$
  if $k_i \neq 0$ **then**
    if $k_i > 0$ **then**
      $Q \leftarrow Q + P_{k_i}$
    else
      $Q \leftarrow Q - P_{-k_i}$
    end if
  end if
end for
return Q

3.1.2 Método comb

El método *comb* es un método que reduce el número de operaciones de manera significativa. Este método usualmente se realiza cuando previamente se cuenta con un punto conocido, debido al precómputo que se requiere.

Para $k \in \mathbb{Z}^+$, sea $t = |k|$ y $d = [t/\omega]$, donde ω es el tamaño de ventana. El funciona comb funciona de la siguiente manera:

1. Se rellena con $d\omega - t$ ceros a la izquierda de 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 un tamaño d, de tal forma que:

$$k = K^{\omega-1} \parallel \cdots \parallel K^1 \parallel K^0$$

3. Se escriben las cadenas de bits K^j de forma matricial:

$$\begin{bmatrix} K^0 \\ \vdots \\ K^{\omega'} \\ \vdots \\ K^{\omega-1} \end{bmatrix} = \begin{bmatrix} K^0_{d-1} & \dots & K^0_0 \\ \vdots & & \vdots \\ K^{\omega'}_{d-1} & \dots & K^{\omega'}_0 \\ \vdots & & \vdots \\ K^{\omega-1}_{d-1} & \dots & K^{\omega-1}_0 \end{bmatrix} = \begin{bmatrix} k_{d-1} & \dots & k_0 \\ \vdots & & \vdots \\ k_{(\omega'+1)d-1} & \dots & k_{\omega' d} \\ \vdots & & \vdots \\ k_{wd-1} & \dots & k_{(\omega-1)d} \end{bmatrix}$$

4. Se procesa cada columna del escalar a la vez.

5. Para obtener mayor aceleración, los puntos:

$$[a_{\omega-1}, \dots, a_2, a_1, a_0]P = a_{\omega-1}2^{(\omega-1)d}P + \cdots + a_2 2^{2d}P + a_1 2^d P + a_0 P$$

se precalculan para cualquier combinación $(a_{\omega-1}, \dots, a_2, a_1, a_0)$

El costo de la multiplicación escalar utilizando el algoritmo 3.4 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 puntos.

Algoritmo 3.4 Multiplicación escalar: Método comb

Entrada: Entero positivo k, $P \in E(\mathbb{F}_q)$, tamaño de ventana ω
Salida: $[k]P$

1: Se precomputa $[a_{\omega-1},\ldots,a_2,a_1,a_0]P$ para todas las combinaciones de bits $a_{\omega-1},\ldots,a_2,a_1,a_0$ de tamaño ω
2: Rellenar la representación binaria de k con ceros a la izquierda en caso de ser necesario y se escribe $k = K^{\omega-1} \parallel \cdots \parallel K^1 \parallel K^0$, donde K^j es una cadena de longitud d. Representar K_i^j al bit i del elemento K^j.
3: $Q \leftarrow \mathcal{O}$
4: **for** $i = d-1$ **to** 0 **do**
5: $\quad Q \leftarrow [2]Q$
6: $\quad Q \leftarrow Q + [K_i^{\omega-1},\ldots,K_i^1,K_i^0]P$
7: **end for**
8: **return** Q

3.1.3 Método GLV

El método GLV [Gallant et al., 2001] acelera la multiplicación escalar $[k]P$ en $E(\mathbb{F}_p)[r]$ utilizando 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 computar $[k]P$ de manera eficiente escribiendo $k \equiv k_0 + k_1\lambda \pmod{r}$ con $k_i < \sqrt{r}$ y realizando la doble multiplicación $[k_0]P + [k_1]\psi(P)$ simultáneamente.

La eficiencia de este método recae en que emplea únicamente la mitad de doblados de punto que en el método binario, a costo de un precómputo y almacenamiento. Combinando este método con algún método de ventana, por ejemplo ω-NAF, se puede reducir el número de adiciones de

punto como se muestra en el algoritmo 3.5. El costo de este algoritmo es:

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

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 sumando corresponde al costo del ciclo principal del algoritmo.

Algoritmo 3.5 Multiplicación escalar: Método GLV

Entrada: Entero positivo k, $P \in E(\mathbb{F}_q)$, tamaño de ventana ω, endomorfismo ψ sobre $E(\mathbb{F}_p)$
Salida: $[k]P$

1: $Q \leftarrow \psi(P)(= [\lambda]P)$
2: Descomponer $k = u + v\lambda$ donde $|u| = |v| = l$
3: Calcular $NAF_\omega(u) = \sum_{i=0}^{l-1} u_i^i$ y $NAF_\omega(v) = \sum_{i=0}^{l-1} v_i^i$
4: Obtener $P_i = [i]P, Q_i = [i]Q$ para $i \in [\pm 1, \pm 3, \pm 5, \ldots, \pm(2^{\omega-1} - 1)]$
5: $R \leftarrow \mathcal{O}$
6: **for** $i = l - 1$ **to** 0 **do**
7: $\quad R \leftarrow [2]R$
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

3.1.4 Método GS

El método GS [Galbraith and Scott, 2008] es una versión generalizada del método GLV, la cual aprovecha el endormofismo de Frobenius en los grupos $\mathbb{G}_2$ y $\mathbb{G}_T$.

En el caso de las curvas BN se hace la descomposición $k \equiv k_0 + k_1\lambda + k_2\lambda^2 + k_3\lambda^3 (\bmod r)$. En este caso, el homomorfismo es $\psi^i = \phi\pi^i\phi^{-1}$, donde $\phi : E' \to E$ es el endomorfismo utilizado para tomar un punto de la curva enlazada, y π^i es la p-ésima potencia del endomorfismo de

Frobenius. Entonces la multiplicación escalar queda de la forma:

$$[k]P = [n_1]P + [n_2]\psi(P) + [n_3]\psi^2(P) + [n_4]\psi^3(P).$$

En el caso de otras curvas, el grado del polinomio está regido por la función indicatriz de Euler (véase definición A.1) y un valor característico de la curva(grado de encajamiento). El algoritmo 3.6 describe el método GS, utilizando el método de ventana ω-NAF para reducir el número de adiciones. El costo computacional de este algoritmo es:

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

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

3.2 Emparejamiento Bilineal

Sea r un número primo, $\mathbb{G}_1$ y $\mathbb{G}_2$ dos grupos aditivos con identidad $\mathcal{O}$, $\mathbb{G}_T$ un grupo multiplicativo con identidad 1 y $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = r$.

Definición 3.2. *Un emparejamiento bilineal sobre los grupos* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ *es la transformación*

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

- **Computabilidad**: Se requiere que exista una forma eficiente de calcular $\hat{e}$.
- **No degenerado**:

Para cada $P \in \mathbb{G}_1$ con, $P \neq \mathcal{O}$, existe un $Q \in \mathbb{G}_2$ tal que $\hat{e}(P, Q) \neq 1$

Para cada $Q \in \mathbb{G}_w$ con, $Q \neq \mathcal{O}$, existe un $P \in \mathbb{G}_1$ tal que $\hat{e}(P, Q) \neq 1$

$\hat{e}(P, Q) = 1$, si $P = \mathcal{O}$ o $Q = \mathcal{O}$

Algoritmo 3.6 Multiplicación escalar: Método GS

Entrada: Entero positivo k, $P \in E(\mathbb{F}_2)$, tamaño de ventana ω, endomorfismo $\psi = \phi^{-1}\pi_p\phi$ sobre $E(\mathbb{F}_2)$

Salida: $[k]Q$

1: $R_0 \leftarrow Q$ $(= \lambda^0 Q)$
2: $R_1 \leftarrow \phi(Q)$ $(= \lambda^1 Q)$
3: $R_2 \leftarrow \phi^2(Q)$ $(= \lambda^2 Q)$
4: $R_3 \leftarrow \phi^3(Q)$ $(= \lambda^3 Q)$
5: Descomponer $k \equiv k_0 + k_1\lambda + k_2\lambda^2 + k_3\lambda^3$ donde $|k_i| = l$
6: Calcular $NAF_\omega(k_i) = \sum_{i=0}^{l-1} k_{ij}2^j$
7: Obtener $R_i^j = [i]R_j$ para $i \in [\pm 1, \pm 3, \pm 5, \ldots, \pm(2^{\omega-1} - 1)]$
8: $R \leftarrow \mathcal{O}$
9: **for** $i = l - 1$ **to** 0 **do**
10: $R \leftarrow [2]R$
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

- **Bilinealidad**: Sea $a, b \in \mathbb{Z}$, para cada $P, P' \in \mathbb{G}_1$, $Q, Q' \in \mathbb{G}_2$, se cumple que:

$$\hat{e}(P + P', Q) = \hat{e}(P, Q) \cdot \hat{e}(P', Q)$$

$$\hat{e}(P, Q + Q') = \hat{e}(P, Q) \cdot \hat{e}(P, Q')$$

$$\hat{e}([a]P, Q) = \hat{e}(P, [a]Q) = \hat{e}(P, Q)^a$$

$$\hat{e}([a]P, [b]Q) = \hat{e}([b]P, [a]Q) = \hat{e}([ab]P, Q) = \hat{e}(P, [ab]Q) = \hat{e}(P, Q)^{ab}$$

Considérese una curva elíptica ordinaria $E/\mathbb{F}_p$ con grado de encajamiento k y un grupo $E(\mathbb{F}_p)$ de orden $h \cdot r = p + 1 - t$ donde r es un número primo y $h \in \mathbb{Z}^+$. Dada la curva enlazada E' de grado d tal que $E'(\mathbb{F}_{p^{k/d}})$ tiene un subgrupo de orden r, supongamos que E y E' son isomórficas sobre el campo $\mathbb{F}_{p^k}$, es decir, $\phi : E'(\mathbb{F}_{p^{k/d}}) \to E(\mathbb{F}_{p^k})$

Bajo estas condiciones estamos interesados en emparejamientos bilineales sobre los grupos $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$, tales que:

- $\mathbb{G}_1$ es el eigenespacio-1 de π en $E(\mathbb{F}_{p^k})$, es decir, es el grupo cíclico escrito de manera aditiva, formado por los puntos de torsión r en la curva elíptica $E(\mathbb{F}_p)$

- $\mathbb{G}_2$ es el eigenespacio-p de π en $E(\mathbb{F}_{p^k})$. Sean $Q' \in E'(\mathbb{F}_{p^{k/p}})[r]$ tal que $\mathbb{G}'_2 = \langle Q' \rangle$ y sea $Q = \phi(Q')$, entonces $\mathbb{G}_2$ es el grupo cíclico generado por el punto Q, es decir $\mathbb{G}_2 = \langle Q \rangle$

- $\mathbb{G}_T$ es un subgrupo de $\mathbb{F}^*_{p^k}$, escrito de manera multiplicativa, el cual está formado por el conjunto de las r-ésimas raíces de la unidad en el grupo cíclico $\mathbb{F}^*_{p^k}$.

3.2.1 Emparejamiento de Tate

Definición 3.3 (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 $\overline{\mathbb{F}}_p(E)$ con divisor $div(f_{s.R}) = s[R] - (s-1)[\mathcal{O}]$*

Lema 3.1. *Sea $f_{s,R}$ una función de Miller y υ_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:*

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

Definición 3.4 (Emparejamiento de Tate). *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 del Miller $f_{r,P}$. El emparejamiento reducido de Tate $\hat{t}$ no degenerativo y bilineal queda definido de la siguiente manera:*

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

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 $\mathbb{F}_{p^k}^*$. A este operación se le denomina *exponenciación final*

Definición 3.5 (Emparejamiento ate). *[Hess et al., 2006]. 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 $Q \in E'(\mathbb{F}_{k/d})[r]$, el emparejamiento* **ate** *se define como:*

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

3.2.2 Ciclo de Miller

Es necesaria la evaluación de la función de Miller (Definición 3.3) durante la construcción del emparejamiento. Sea $r = (r_{l-1}, \ldots, r_1, r_0)_2$ de acuerdo con el Lema 3.1 podemos construir

$f_{r,p}$, de manera recursiva utilizando el método de *suma* y *doblado* de líneas:

$$f_{a+1,R} = f_{a,R} \cdot \ell_{aR,R}/\upsilon_{(a+1)R} \quad \text{y} \quad f_{2a,R} = f_{a,R}^2 \cdot \ell_{aR,aR}/\upsilon_{(2a)R}. \tag{3.1}$$

con base en [Barreto et al., 2004] las líneas verticales $\upsilon_{(a+1)R}$ y $\upsilon_{(2a)R}$ pueden ser omitidas de la ecuación 3.1. El cálculo del ciclo de Miller se muestra en el algoritmo 3.7

Algoritmo 3.7 Ciclo de Miller

Entrada: $r = (r_{l-1}, \ldots, r_1, r_0)_2$, $Q, P \in E(\overline{\mathbb{F}}_p)$ tal que $P \neq Q$
Salida: $f_{r,Q}(P)$

1: $T \leftarrow Q$, $f \leftarrow 1$
2: **for** $i = l-2$ **to** 0 **do**
3: $\quad f \leftarrow f^2 \cdot \ell_{T,T}$, $T \leftarrow 2T$
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

De manera informal podemos decir que el cálculo del emparejamiento está compuesto de dos funciones: la primera es la evaluación del ciclo de Miller, y la segunda, es la *exponenciación final*, la cual consiste en la potenciación $(p^k - 1)/r$ en $\mathbb{F}_p^k$.

A partir de las mejoras reportadas por Barreto et al. en [Barreto et al., 2002], diversos autores han propuesto modificaciones al ciclo de Miller y a la exponenciación final para mejorar la eficiencia del cómputo del emparejamiento. Los emparejamientos *ate* [Hess et al., 2006], *R-ate* [Choie et al., 2005] y *ate óptimo* [Vercauteren, 2008, Beuchat et al., 2010, Wang et al., 2009] son versiones modificadas del emparejamiento Tate y forman la familia de emparejamientos de Tate. El propósito de dichos emparejamientos es reducir el número de operaciones que se requieren durante la evaluación de la función de Miller.

3.2.3 Emparejamiento óptimo ate sobre curvas BN

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. Para el emparejamiento Tate, el número de iteraciones está relacionado con el orden r de la curva elíptica, mientras que para el emparejamiento *ate*, se puede observar que las iteraciones necesarias dependen de la traza t de la curva. Dado que $t \approx \sqrt{r}$, el emparejamiento *ate* es una mejor opción para el cómputo, ya que el número de iteraciones se reduce a la mitad.

Una mejora al emparejamiento *ate* para las curvas BN A.11, es el conocido como emparejamiento *óptimo ate* [Vercauteren, 2008] el cual está dado por:

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

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

El algoritmo 3.8 muestra el emparejamiento óptimo *ate* para curvas BN de acuerdo con el trabajo [Beuchat et al., 2010]. El objetivo principal del ciclo de Miller (líneas 3 a la 13) es, primero construir la función racional $f_{6z+2,Q}$ asociada al punto Q y al entero $6z + 2$, para posteriormente evaluarla en el punto P. Después (lineas 14 a la 16) se evalúan las funciones $\ell_{[6z+2]Q,\pi_p(Q)}(P)$ y $\ell_{[6z+2]Q+\pi_p(Q),\pi_p^2(Q)}(P)$, que se refieren a las lineas que cruzan por los puntos $[6z + 2]Q$ y $\pi_p(Q)$ para la primera función y $[6z + 2]Q + \pi_p(Q)$ y $\pi_p^2(Q)(P)$ para la segunda. Finalmente se calcula la exponenciación final (línea 17).

Algoritmo 3.8 Emparejamiento óptimo *ate* sobre curvas BN

Entrada: $P \in \mathbb{G}_1$ y $Q \in \mathbb{G}_2$
Salida: $a_{opt}(Q, P)$

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** $i = l - 2$ **to** 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

3.3 Multiemparejamiento

En algunos protocolos el producto de emparejamientos es requerido. En el cifrado basado en atributos para el algoritmo de descifrado (véase la sección 4.4.6) se requiere de un producto de emparejamientos. Este producto se puede realizar de una manera más eficiente si en la implementación de los emparejamientos se comparten las funciones involucradas.

Se puede notar que las funciones que se pueden compartir son aquellas que no tienen relación directa con los elementos de los grupos $\mathbb{G}_1$ y $\mathbb{G}_2$, en este caso, la elevación al cuadrado en el ciclo de Miller y la exponenciación final. A este método se le conoce como multiemparejamiento o emparejamiento simultáneo, el cual se presenta en el algoritmo 3.9.

Algoritmo 3.9 Multiemparejamiento sobre curvas BN

Entrada: $P_1, P_2, \ldots, P_n \in \mathbb{G}_1$ y $Q_1, Q_2, \ldots, Q_n \in \mathbb{G}_2$
Salida: $\prod_{i=1}^{n} \hat{e}(Qi, Pi)$

Escribir $s = 6z + 2$ como $s = \sum_{i=0}^{l-1}$ con $s_i \in \{-1, 0, 1\}$
$f \leftarrow 1$
for $i = 0$ **to** n **do**
    $T_i \leftarrow Q_i$
end for
for $j = l - 2$ **to** 0 **do**
    $f \leftarrow f^2$
    for $i = 0$ **to** n **do**
        $f \leftarrow f \cdot \ell_{T_i,T_i}(P_i)$, $T_i \leftarrow 2T_i$
        if $s_j = 1$ **then**
            $f \leftarrow f \cdot \ell_{T_i,Q_i}(P_i)$, $T_i \leftarrow T_i + Q_i$
        else if $s_j = -1$ **then**
            $f \leftarrow f \cdot \ell_{T_i,-Q_i}(P_i)$, $T \leftarrow T_i - Q_i$
        end if
    end for
end for
for
    do$R_1 \leftarrow \pi(Q_i)$
    $R_2 \leftarrow \pi^2(Q_i)$
    $f \leftarrow f \cdot \ell_{T_i,R_1}(P)$, $T_i \leftarrow T_i + R_1$
    $f \leftarrow f \cdot \ell_{T_i,-R_2}(P)$, $T_i \leftarrow T_i + R_2$
end for
$g \leftarrow f^{p^{12}-1}/r$
return g

3.4 Proyección de una cadena a un punto

En la implementación de protocolos basados en identidades se utilizan curvas elípticas amables con los emparejamientos. Dichas implementaciones, al menos, requieren usar dos grupos $\mathbb{G}_1$ y $\mathbb{G}_2$, ambos de orden r.

En este caso, con la criptografía basada en identidades, 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 los emparejamientos, en este caso se requiere de un punto único sobre una curva elíptica.

Normalmente en los esquemas basados en identidades como en los basados en atributos, la proyección de la cadena se realiza hacia un punto P en el grupo aditivo $\mathbb{G}_1$, para ello, se utiliza una función picadillo H_1.

Definición 3.6 (Función MapToPoint). *La función picadillo denominada MapToPoint es una transformación de la forma:*

$$H_1 : \{0,1\}^* \to \mathbb{G}_1$$

Sea $E : y^2 = x^3 + b$ una curva elíptica ordinaria BN, definida sobre un campo finito $\mathbb{F}_p$, el punto base $P \in E(\mathbb{F}_p)$, H una función de picadillo conocida $H : \{0,1\}^* \to \mathbb{F}_p$ y el grupo abeliano $\mathbb{G}_1 = \langle P \rangle$ de orden r.

Podemos construir H_1 usando una función picadillo estándar H, en el caso de este trabajo de tesis se utiliza SHA-2 [NIST, 2008]. Se aplica la función H para la coordenada x tal que $x \in \mathbb{F}_p$, después se obtiene la coordenada y calculando $\sqrt{x^3 + b} = u$. Una vez obtenidas las coordenadas (x, y) se debe verificar que el punto recién generado pertenece al grupo $\mathbb{G}_1$, si no es el caso, se buscarán nuevas coordenadas hasta encontrar un punto que satisfaga la pertenencia al grupo.

Entonces, la función de picadillo $H_1 : \{0,1\}^* \to \mathbb{G}_1$ para realizar la proyección de una cadena a un punto, se define como $H_1 = H_1(H(S))$. Boneh et al. en [Boneh and Franklin, 2001]

muestran un algoritmo para realizar dicho proceso (véase el algoritmo 3.10).

Algoritmo 3.10 Función MapToPoint

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** $P = \mathcal{O}$

4

Protocolos Criptográficos

Los protocolos criptográficos son el elemento esencial en la planificación e implementación de políticas de seguridad en redes de datos. Como es sabido, un protocolo criptográfico implica el intercambio de información entre entidades en un entorno distribuido, con el fin de implementar mecanismos y funciones de seguridad y, por lo tanto, ofrecer a las aplicaciones finales servicios de seguridad.

El presente capítulo tiene el objetivo de proporcionar conceptos relacionados con los protocolos criptográficos. Se retoma una forma en que éstos pueden ser clasificados. Además, se mencionan los problemas sobre los cuales se garantiza la seguridad durante el intercambio de información. Para finalizar, se explican los principales protocolos estudiados durante el desarrollo de este trabajo de tesis, destacando los detalles que sirven para la generación automática de código.

4.1 Introducción

Retomando la definición dada por Menezes en [Menezes et al., 1996], podemos decir que un protocolo criptográfico es un algoritmo distribuido definido por una secuencia de pasos, dicho de otra manera, es la descripción de las acciones requeridas por dos o más entidades para llevar a cabo uno o más objetivos de seguridad específicos. Dichos protocolos sirven para llevar a cabo los servicios de seguridad como el de autenticación de entidades o la confidencialidad de la información.

Los protocolos criptográficos usan funciones de criptografía en algunos o todos los pasos que los componen. Para que un protocolo criptográfico que involucra intercambio de datos tenga una ejecución exitosa, se requiere de una definición precisa de cada una de las etapas que conforman el protocolo, de las funciones o primitivas involucradas, y la descripción de las entidades involucradas. Además de la definición, se requiere que que el protocolo y los participantes cumplan las siguientes premisas:

- Todas las entidades involucradas deben conocer los pasos del protocolo de antemano.
- El protocolo debe ser aceptado por todos los participantes.
- El protocolo no debe tener ambigüedades.
- Se debe definir qué hacer para cada circunstancia posible que se presente.
- No debe ser posible realizar una acción que no este definida dentro del protocolo.

Por otra parte, algunos de los intercambios de información pueden tener como objetivo la distribución de claves de sesión, con el fin de que las entidades puedan cifrar la información y constituir un canal seguro entre ellas, éstos serán los protocolos de distribución de claves. Otros intercambios pueden tener por objetivo no sólo la distribución de claves, sino asegurar

la integridad de la información transmitida y la identidad del origen de la información. Éstos serán los denominados protocolos de autenticación.

4.2 Clasificación

De manera individual, cada protocolo cubre diferentes objetivos y servicios, por tal motivo, en esta sección se otorga una visión general para su clasificación para los diferentes tipos de protocolos.

El ámbito de los protocolos criptográficos es lo suficientemente amplio como para hacer necesario el establecimiento de criterios de clasificación de los mismos. Dependiendo de el número de participantes que intervienen en el intercambio de mensajes, podemos distinguir entre protocolos *bipartitos* y *multipartitos* En los primeros, el número de participantes directamente implicados se limita a dos, mientras que en los otros sólo existe la restricción sobre finitud de los mismos.

Otra clasificación se puede realizar en función de si existe un intercambio recíproco de información entre los participantes o si por el contrario, están establecidas las figuras de emisor y receptor. Siguiendo este razonamiento, realizamos la siguiente diferenciación:

- Si durante el desarrollo del algoritmo un usuario tiene el papel de emisor y su contrario intenta obtener la entrada del primero a partir de una transformación de ésta, se trata de un protocolo *unilateral* o *unidireccional*. Esto es lo que sucede en protocolos tales como la transferencia inconsciente o las demostraciones de conocimiento nulo.

- Si por el contrario, todos los participantes juegan idénticos papeles siendo emisor y receptor a la vez, entonces se trata de un protocolo *multilateral*, *multidireccional* o de *intercambio mutuo*. Esta situación se produce en protocolos de firmado.

4.3 Propiedades

Aunque el diseño de cada protocolo está pensado resolver una situación en particular, se debe garantizar el cumplimento de ciertas propiedades generales [Menezes et al., 1996]. En la tabla 4.1 se pueden apreciar cuestiones fundamentales que deben ser garantizadas antes, durante y después de la ejecución, esto depende de la finalidad que se quiere lograr con cada protocolo.

Es importante mencionar que dentro de la tabla 4.1 hay cuatro objetivos los cuales destacan por la importancia que tienen con respecto a los otros. Estos son la *privacidad*, la *integridad*, la *autenticación* y el *no repudio*.

4.4 Protocolos basados en emparejamientos bilineales

La criptografía basada en emparejamientos surgió como un ataque presentado por Menezes, Okamoto y Vanstone [Menezes et al., 1991]. Dicho trabajo describía cómo usar el emparejamiento de Weil para reducir el problema del logaritmo discreto en un grupo cíclico, al problema de logaritmo discreto en el grupo de las raíces de unidad. Tiempo después, los emparejamientos fueron empleados con fines más constructivos. Pocos años después, a partir del trabajo realizado por Joux [Joux, 2004], se han generado gran variedad de protocolos que aprovechan el uso constructivo de los emparejamientos bilineales, creando de esta manera el área de criptografía basada en emparejamientos.

Las características de los emparamentas bilineales los hacen esencialmente atractivos para extender y completar las propiedades de la criptografía basada en curvas elípticas, pues los emparejamientos sirven como un puente entre la eficiencia de los grupos cíclicos aditivos formados por los puntos en una curva elíptica, y las características de los grupos cíclicos multiplicativos.

La gran diversidad en protocolos criptográficos basados en emparejamientos bilineales,

Objetivo	Descripción
Privacidad o Confidencialidad	Procurar mantener la información en secreto salvo para las entidades autorizadas para ver dicha información.
Integridad	Garantizar que la información no sea alterada por medios no autorizados o desconocidos.
No repudio	Prevenir el rechazo de compromisos o acciones previas. El no repudio del emisor previene la negación del envío de información. El no repudio del receptor evita que se pueda negar la recepción de un mensaje.
Identificación	Corroborar la identidad de cada entidad participante.
Autenticación	Corroborar la fuente de la información.
Autorización	Traspasar a otra entidad permiso oficial para realizar alguna acción.
Validación	Aportar medios que permitan autorizar en tiempo real el uso o manipulación de recursos.
Control de acceso	Restringir el acceso a recursos de entidades con los privilegios adecuados.
Certificación	Respaldar la información mediante una entidad de confianza.
Registro de tiempo	Almacenar el tiempo de creación o existencia de la información.
Atestiguar	Verificar la creación o existencia de la información por una entidad diferente a la entidad que creó dicha información.
Recibo	Avisar la recepción de alguna información.
Confirmación	Avisar que los servicios han sido proporcionados.
Propietario	Proporcionar un medio por el que se entrega a una entidad los derechos legales para usar o transferir algún recurso.
Anonimato	Esconder y proteger la identidad de una entidad que participa en algún proceso.
Revocación	Retirar la certificación o autorización.

Tabla 4.1: Objetivos generales de los protocolos.

ocasiona que el análisis de ellos se convierta en una tarea demasiado exhaustiva. Dentro de [Dutta et al., 2004] se brinda una explicación general del funcionamiento de los protocolos basados en emparejamientos más relevantes al momento de su publicación. Aunque la publicación tiene varios años, es válida ya que en ella se explican las primitivas básicas que componen un protocolo basado en emparejamientos bilineales.

Los protocolos basados en emparejamientos pueden dividirse de manera general en dos tipos:

1. Aquellos que la única construcción conocida requiere necesariamente el uso de emparejamientos.

   Por ejemplo, la criptografía basada en la identidad [Boneh and Franklin, 2001] y las firmas agregadas no triviales [Boneh et al., 2003].

2. Aquellos que aunque pueden construirse sin necesidad de utilizar emparejamientos bilineales, la utilización de estos bloques permiten una funcionalidad mejorada. Por ejemplo, el acuerdo llaves de Joux [Joux, 2004], y la búsqueda de llave publica para el cifrado de datos.

Con base en su funcionamiento los protocolos basados en emparejamientos cumplen diversos objetivos, entre los principales podemos destacar:

- *Cifrado de datos*. Es el proceso por el que una información legible es transformada, mediante un algoritmo, en información ilegible.

- *Firma digital*. Es el método que permite asociar la identidad de una entidad a un documento como autor del mismo. Por mencionar algún tipo tenemos firmas cortas, firmas a ciegas, multi-firmado y firmado agregado.

- *El acuerdo de llaves*. Es la manera en que las entidades participantes se ponen de acuerdo en el valor de una información secreta, donde dos o más participantes requieren establecer un secreto en común.

- *La criptografía de umbral*. Es útil para eliminar las fallas en la centralización y proporciona resistencia a la corrupción por parte del adversario de un cierto porcentaje de las entidades que integran un sistema distribuido.

A partir del protocolo de firma corta de Boneh y Franklin [Boneh and Franklin, 2001] se han creado gran cantidad de protocolos. La mayoría de estos protocolos están basados en los protocolos de firma corta, el de establecimiento de clave tripartito en una ronda y el de cifrado basado en la identidad. Para los esquemas de voto electrónico, las firmas a ciegas juegan un rol importante en la propuesta de nuevos esquemas. Además, han surgido nuevas propuestas para el cifrado de datos, como lo es, el cifrado basado en atributos.

En lo que resta del capítulo se presenta la descripción de estos protocolos. Buscamos destacar los detalles que permitieron dar una propuesta de un lenguaje para la descripción de protocolos, la cual se detalla en el capítulo 5 de este documento.

4.4.1 Protocolo de una ronda Diffie-Hellman tripartito

Este protocolo fue propuesto por Joux [Joux, 2004], y hace uso del emparejamiento $\hat{e}(g, g)$, donde $g \in \mathbb{G}_1$. Es una propuesta análoga al protocolo de Diffie-Helman que involucra la participación de tres participantes A, B y C, y se requiere de un solo intercambio de mensajes para construir el secreto en común $K_{A,B,C}$.

El protocolo comienza con la selección de un punto generador P que existe en la curva elíptica E, posteriormente las entidades A, B y C seleccionan un número aleatorio a, b y $c \in \mathbb{F}_p$ respectivamente. Individualmente cada entidad debe realizar el cómputo $P_A = [a]P$, $P_B = [b]P$ y $P_C = [c]P$ según corresponda. Después, al mismo tiempo A envía el mensaje P_A a B y C, B envía P_B a A y C, y la entidad C envían P_C a B y A. Una vez recibido el mensaje, cada entidad realiza el computo $F(a, P_B, PC)$, $F(b, P_A, P_C)$ y $F(c, PA, PB)$, donde la función $F()$ corresponde a

$$F(x, P, Q) = \hat{e}(P, Q)^x$$

donde $x \in \mathbb{F}_p$, $P, Q \in E(\mathbb{F}_p)$, y $\hat{e}()$ es el computo de un emparejamiento bilineal.

En este caso, el secreto en común es la salida de la función $F()$ dado que $\hat{e}(P_b, P_c)^a =$

$\hat{e}(P_a, P_c)^b = \hat{e}(P_a, P_b)^c = \hat{e}([P, P)^{abc}$. En la figura 4.1 se puede apreciar de manera gráfica el funcionamiento del protocolo, donde Alicia, Beto y Carlos representan las entidades A, B y C mencionadas anteriormente.

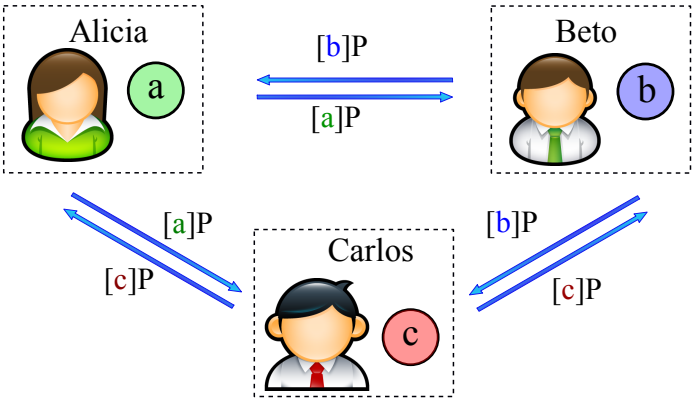


Figura 4.1: Protocolo de una ronda Diffie-Hellman tripartito.

4.4.2 Firma Corta

Este protocolo de firma lo propuso Boneh, Lynn y Shacham [Boneh et al., 2001], utiliza el emparejamiento $\hat{e}(g, g)$, donde $g \in \mathbb{G}_1$, y la función que proyecta cadenas alfanuméricas arbitrarias a un punto H_1. El protocolo se compone de tres secciones: la generación de la llave, el proceso de firmado y por último el proceso de verificado de la firma (véase la figura 4.2).

1. *Generación de llave*: La clave privada de Alicia denotada como a es seleccionada aleatoriamente del conjunto $[2, n-2]$, mientras que la llave pública es el elemento $A = [a]P$.

2. *Firma*: La firma de un mensaje $m \in 0, 1^*$ se denota como $S = [a]M$, donde $M = H(m)$.

3. *Verificación*: Para verificar la firma, cualquier participante que posea la llave pública de Alicia debe calcular $M = H(m)$, y determinar si $\hat{e}(M, A) \stackrel{?}{=} \hat{e}(S, P)$ se cumple.

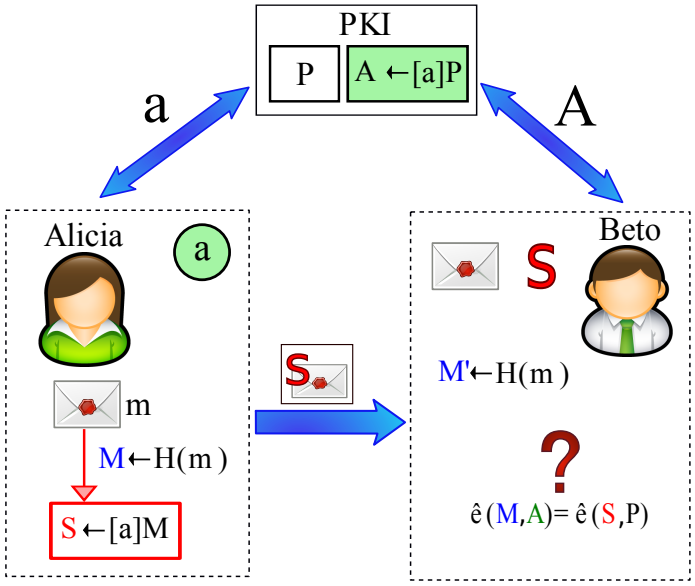


Figura 4.2: Protocolo de firma corta.

4.4.3 Firma a ciegas de Boldyreva

Las firmas digitales son usadas para autenticar al usuario que firma el mensaje. En algunas aplicaciones es necesario que el signatario desconozca el mensaje, por tanto, una tercera entidad solicita la firma. Este tipo de firmas son denominadas firmas a ciegas y son muy útiles en aplicaciones como votaciones electrónicas y monederos electrónicos.

Boldyreva [Boldyreva, 2002] propuso una firma a ciegas basada en emparejamientos bilineales. Sean r un número primo, $P \in \mathbb{G}_1$ un generador , dos grupos aditivos $\mathbb{G}_1$ y $\mathbb{G}_2$, un grupo multiplicativo $\mathbb{G}_T$ todos los grupos de orden r, la función picadillo H_1 y el emparejamiento bilineal $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. El proceso de firmado se compone de 5 secciones las cuales se describen a continuación:

- *Generación de llaves*: Se genera una llave privada seleccionando aleatoriamente $d \in \mathbb{Z}_r^*$. La llave pública se crea computando $Q = [d]P$

- *Ocultamiento*: Se desea ocultar el mensaje m. Primero se computa el picadillo $M = H_1(m)$. Aleatoriamiente se selecciona el factor de opacidad $b \in \mathbb{Z}_r^*$. Finalmente el mensaje se oculta

calculando $\hat{M} = [b]M$.

- *Firmado*: La firma del mensaje oculto se obtiene computando $\hat{S} = [d]\hat{M}$.
- *Descubrimiento*: Para describir la firma se calcula $S = b^{-1}\hat{S}$.
- *Verificación*: La verificación se realiza comparando los emparejamientos $\hat{e}(P, S)$ y $\hat{e}(Q, H_1(m))$, si el resultado es igual la firma es válida, de otra forma se rechaza.

4.4.4 Firma agregada

Boneh et al. proponen en 2003 [Boneh et al., 2003] un protocolo para crear una firma única a partir de la agregación de otras firmas digitales. Dado n firmas correspondientes a n distintos mensajes $M_i \in \{0,1\}^*$, pertenecientes a i entidades, donde $1 \leq i \leq n$. Una firma individual es denotada por $\sigma_i \in \mathbb{G}_1$. Se requiere de dos generadores $g_1 \in \mathbb{G}_1$ y $g_2 \in \mathbb{G}_2$ y la función picadillo H_1 (véase la sección 3.6).

El protocolo se compone por cinco algoritmos: generación de llaves, firmado, verificación, agregación y verificación de la agregación. La descripción de protocolo queda de la siguiente manera:

- *Generación de llaves*: Para cada usuario se elige un entero aleatorio $x \in \mathbb{Z}_p$ y calcula $v \leftarrow g_2^x$. La clave pública del usuario es $v \in \mathbb{G}_2$, y la llave secreta es $x \in \mathbb{Z}_p$.
- *Firmado*: Para cada usuario en particular, dada la llave secreta x y un mensaje m, se realiza el picadillo $h \leftarrow H_1(M)$, donde $h \in G_1$. La firma se obtiene calculando $\sigma = h^x$, donde $h \in \mathbb{G}_1$
- *Verificación*: Dada la llave pública v de un usuario, un mensaje M, y una firma σ. Se computa $h \leftarrow H_1(M)$, y la firma σ se acepta si $\hat{e}(\sigma, g2) = \hat{e}(h, v)$ se cumple, de lo contrario es rechazada.

- *Agregación*: Para agregar usuarios tal que $U \subseteq \mathbb{U}$, se le asigna a cada usuario un índice i, desde 1 hasta $k = |U|$. Cada usuario $u_i \in U$ genera una firma $\sigma_i \in \mathbb{G}_1$ sobre un mensaje $M_i \in \{0,1\}^*$. Los mensajes M_i deben ser todos diferentes. Para crear la firma agregada se calcula $\Sigma \leftarrow_{i=1}^{k} \sigma_i$, donde $\Sigma \in \mathbb{G}_1$.

- *Verificación de la agregación*: Dada una firma agregada $\Sigma \in \mathbb{G}_1$ que corresponde a un subconjunto de usuarios U indexados, cada usuario con su mensaje $M_i \in \{0,1\}$, y su clave pública $v_i \in \mathbb{G}_2$. Para verificar la firma Σ se realiza lo siguiente:

  1. Se verifica que todos los mensajes M_i sean diferentes, de otra manera se rechaza.

  2. Calcular $h_i \leftarrow H_1(M_i)$, donde $1 \leq i \leq k = |U|$. Se acepta si $\hat{e}(\Sigma, g2) = \prod_{i=1}^{k} \hat{e}(h_i, v_i)$ se cumple, de otra manera se rechaza.

4.4.5 Cifrado basado en la Identidad

En los esquemas de llave pública, si una entidad A necesita enviar un mensaje seguro a otra B, el cifrado se realiza con la clave pública de B, el problema que se presenta es que la entidad A no puede tener la certeza de que la llave pública de B o si un impostor creó una llave utilizando los datos de B.

Una opción para solucionar el problema anterior consiste en usar autoridades certificadores, las cuales son las responsables de crear certificado para las llaves públicas de las entidades involucradas. Aunque usar certificados parecer ser una solución simple, aunque existen dificultades para administrarlos.

Boneh y Franklin en 2001 [Boneh and Franklin, 2001] propusieron el primer esquema de cifrado basado en la identidad. Anteriormente Shamir [Shamir, 1985] propuso de manera teórica este tipo de cifrado, pero el esquema de Boneh y Franklin fue el primero en ser práctico para su implementación.

En este caso el intercambio de mensajes se realiza entre las entidades Alicia y Beto, identificadas con A y B respectivamente. Alicia desea enviar un mensaje cifrado a Beto utilizando una cadena alfanúmerica que identifique a Beto (por ejemplo, un correo electrónico o un identificador). Beto recibe el mensaje y lo descifra utilizando la llave privada proporcionada por una tercera entidad denominada Servidor de Llaves. El esquema se divide en cuatro fases o secciones: la *Inicialización*, la *Generación de Llaves*, el *Cifrado* y el *Descrifrado*, las cuales se describen a continuación:

- *Inicialización*: El algoritmo procede de la siguiente manera:

  1. Se escoge un emparejamiento bilineal de la forma $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ y el generador $P \in \mathbb{G}_1$.

  2. Aleatoriamente se genera la llave maestra $t \in \mathbb{Z}_q^*$, donde q es un primo grande. Se crea la llave pública del servidor $P_{pub} = [s]P$.

  3. Se escogen las funciones picadillo H_1 (véase la sección 3.6) y $H_2 : \mathbb{G}_2 \to \{0,1\}^n$, donde n es la longitud del mensaje $m = \{0,1\}^n$.

- *Cifrado*: Alicia toma la llave pública P_{pub} del servidor de llaves y el mensaje M, el cifrado se realiza de la siguiente manera:

  1. Para que Beto cifre un mensaje m a Alicia, él calcula $Q_A = H_1(ID_A) \in \mathbb{G}_1$.

  2. Se escoge arbitrariamente un entero $r \in [2, n-2]$, y calcula $R = [r]P$.

  3. El cifrado C es la dupla $C = \langle R, c \rangle$, donde $c = m \oplus H_2(\hat{e}(Q_A, P_{pub})^r)$.

- *Generación de llaves*: Alicia solicita su clave privada del servidos de llaves. El servidor genera un identificador $ID_A \in \{0,1\}^n$ usando la función picadillo H_1 de la siguiente manera: $d_A = [t]H_1(ID_A)$, y es entregado a Alicia de manera segura.

- *Descifrado*: Alicia recibe el cifrado C, y obtiene el mensaje m descifrado usando su llave privada d_A computando $m = c \oplus H_2(\hat{e}(d_A, R))$

La representación a bloques de este esquema puede observarse en la figura 4.3.

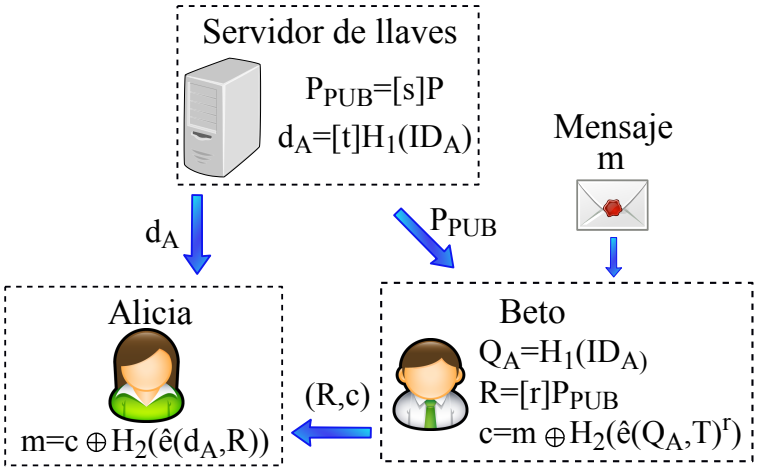


Figura 4.3: Cifrado basado en la identidad.

4.4.6 Cifrado basado en atributos

Sahai y Waters presentaron una nueva forma de cifrar en [Bethencourt et al., 2007], donde la persona que da los datos para ser cifrados establece un predicado o fórmula, que describe la manera en que él desea compartir los datos y cómo otro usuario puede obtener la llave secreta al tener un conjunto de atributos válidos.

Este protocolo permite compartir un secreto, y verlo cuando el usuario cuente con un conjunto de privilegios mínimos requeridos por una política de acceso previamente establecida. Esto sirve cuando se desea que usuarios de un mismo nivel tengan acceso a un recurso compartido.

Para este trabajo de investigación, la implementación que se realizó es basada en [Sánchez Ramírez, 2012], dentro de este trabajo, se detalla la implementación de una biblioteca criptográfica sobre emparejamientos en un dispositivo móvil. Para nuestro caso, utilizamos los

algoritmos presentados en dicha tesis. Para la política de acceso necesaria para este esquema, se utiliza un *esquema lineal de secreto compartido* (esquema LSSS por sus siglas en ingles), presentado por Liu y Cao [Karchmer and Wigderson, 1993].

De manera general, el cifrado basado en atributos se divide en cuatro algoritmos o secciones, pero se utiliza una quinta etapa para la *delegación* de atributos, debido a que en algunos sistemas puede ser necesario que un usuario necesite proporcionar acceso a otro, como es en el caso de algunos sistemas distribuidos.

- *Inicialización*: Este algoritmo toma como entrada el párametro de seguridad λ y el universo de atributos U. A partir del páramentro de seguridad se definen los grupos $\mathbb{G}_1$ y $\mathbb{G}_2$ de 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: $PK = P, Q, \hat{e}(Q,P)^{\alpha}, [a]P, H_1, \ldots, H_U$. La autoridad establece $MSK = [\alpha]P$ como la llave secreta maestra.

- *Cifrado*: El algoritmo de cifrado toma como entrada los parámetros públicos PK y el mensaje M a ser cifrado. Además, toma como entrada una estructura LSSS(M,ρ), donde la función ρ asocia las filas de M con los atributos, donde M es una matriz de $\ell \times n$. El algoritmo selecciona un vector aleatorio $v = (s, y_1, \ldots, y_n) \in \mathbb{Z}_r^n$, 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 $r_i, \ldots, r_\ell \in \mathbb{Z}_r$.

  Entonces el cifrado se publica como:

$$CT = \begin{cases} C = M \cdot \hat{e}(Q,P)^{\alpha s}, C' = [s]Q, \\ (C_1 = \lambda([a]P) - [r_1]H_{\rho(1)}, D_1 = [r_1]Q), \ldots, (C_\ell = \lambda_\ell([a]P) - [r_\ell]H_{\rho(\ell)}, D_1 = [r_\ell]Q) \end{cases}$$

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

- *Generación de llaves*: Este algoritmo toma como entrada la llave maestra $MSK = [\alpha]P$ y un conjunto S de atributos. Primero se selecciona aleatoriamente un número $t \in \mathbb{Z}_r$, entonces se crea la llave privada como:

$$SK = \{K = [\alpha]P + [t]([a]P), L = [t]Q, \forall x \in SK_x = [t]H_x\}.$$

- *Descrifrado*: 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 de atributos 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 $\sum_{i \in I} \omega_i \lambda_i = \delta \in \mathbb{Z}_r$.

  Entonces para realizar el descifrado el algoritmo primero calcula:

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

  Finalmente toma este valor y lo multiplica por C obteniendo así el mensaje M.

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

$$SK' = \{K' = K + [s]([a]P), L = L + [s]Q, \forall x \in S' K'_x = K_x + [s]H_x\}$$

5

Compilador COMPROC

COMPROC, (**COM**pilador para **PRO**trocolos **C**riptográficos) es un compilador criptográfico que tiene la finalidad de generar código en lenguaje Magma y lenguaje C/C++, de la implementación de protocolos criptográficos basados en emparejamientos bilineales. LENDPROC, (**LEN**guaje para **D**escripción de **PRO**tocolos **C**riptográficos) es nuestra propuesta de un lenguaje para descripción de protocolos, el cual será utilizado por nuestro compilador. En este capítulo mostraremos el análisis realizado para generar dicha propuesta. Además, mencionamos las herramientas utilizadas para crear COMPROC, mostramos la arquitectura del mismo, y su funcionamiento.

5.1 Herramientas del desarrollo

5.1.1 Magma

Magma [Bosma et al., 1997] es un paquete de software diseñado para cálculos algebraicos, teoría de números, geometría algebraica y combinatoria algebraica.

el software proporciona un entorno matemáticamente riguroso para definir y trabajar con estructuras tales como grupos, anillos, cuerpos, módulos, álgebra, esquemas, curvas, grafos y muchos otros. Es distribuido por el Grupo de Álgebra Computacional de la Universidad de Sydney. Actualmente cuenta con funciones para trabajar con curvas elípticas e incluso computar los emparejamientos *Weil*, *Tate* y *Ate*. Aunque *Magma* tiene un gran número funciones implementadas para su uso, también brinda soporte para hacer construcciones propias.

5.1.2 ANTLR

ANother Tool for Language Recognition [Parr and Quong,], es una herramienta que opera sobre lenguajes, proporcionando un marco para construir reconocedores, intérpretes, compiladores y traductores de lenguajes a partir de las descripciones gramaticales de los mismos.

ANTLR cae dentro de la categoría de meta-programas, por ser un programa que escribe otros programas. Partiendo de la descripción formal de la gramática de un lenguaje, ANTLR genera un programa que determina si una sentencia o palabra pertenece a dicho lenguaje (reconocedor), utilizando algoritmos LL(∗)[1] de análisis. Si a dicha gramática se le añaden acciones escritas en un lenguaje de programación, el reconocedor se transforma en un traductor o intérprete.

Además, ANTLR proporciona facilidades para la creación de estructuras intermedias de

[1] El *analizador sintactico LL*, es un analizador sintáctico descendente, por un conjunto de gramática libre de contexto. En éste analizador las entradas son de izquierda a derecha, y construcciones de derivaciones por la izquierda de una sentencia o enunciado

análisis, como son los *ASTs - Abstract Syntax Tree*[2], para recorrer dichas estructuras, y provee mecanismos para recuperarse automáticamente de errores y realizar reportes de los mismos. ANTLR es un proyecto bajo licencia BSD, viniendo con todo el código fuente disponible, y preparado para su instalación bajo plataformas Linux, Windows y Mac OS-X.

ANTLR es capaz de generar un analizador léxico, sintáctico o semántico en varios lenguajes (Java,C/C++ y C# en su versión 3.2) a partir de unos archivos escritos en un lenguaje propio. Dicho lenguaje es básicamente una serie de reglas BNF y un conjunto de construcciones auxiliares. En nuestro caso, COMPROC se exportó a lenguaje C/C++.

Los analizadores generados realizan un análisis sintáctico descendente por la izquierda denomidando *análisis LL(k)*. Éste tipo de análisis construye un árbol sintáctico de la raíz hacia las hojas (del símbolo inicial de la gramática hacia los símbolos terminales) y lee los identificadores léxicos de izquierda a derecha construyendo las derivaciones a la izquierda primero.

5.2 Lenguaje para descripción de protocolos

Uno de los requerimientos más importantes para un compilador es el lenguaje con el cual se escribirán los programas de entrada, ya que la correcta ejecución de un programa dependerá de las estructuras secuenciales, las estructuras de control, los ciclos, los tipos de datos creados, incluso de la notación que se utiliza en los comentarios.

Al momento de realizar este trabajo de tesis, no se encontró algún lenguaje para describir protocolos criptográficos dentro de la literatura criptográfica,. Por tal motivo, en este apartado se detallará la estructura del lenguaje definido para nuestro compilador. Además, mostramos el análisis hecho para elaborar dicha propuesta.

[2]Un árbol AST, es una representación de árbol de la estructura sintáctica abstracta de código fuente escrito en un lenguaje de programación

5.2.1 Descripción de protocolos

Dentro de la literatura relacionada con el diseño y definición de protocolos criptográficos, existen diversas formas para poder describir el diseño de un protocolo. La manera en que se presente dicho diseño depende exclusivamente del diseñador del protocolo, en ocasiones, un mismo diseño presenta diferentes implementaciones, debido a la interpretación que se entienda de la descripción.

Analizando las diferentes formas para especificar un protocolo, podemos observar similitudes en la descripción de estos. Gracias a estas similitudes, podemos clasificar las descripciones en dos diferentes tipos: la descripción *literal* y la descripción *estándar*.

5.2.1.1. Descripción literal

Este tipo de descripción es la que podemos observar en la mayoría de los diseños revisados, básicamente, y como su nombre lo indica, es una descripción textual de un diseño. Las descripciones que se muestran en la sección 4.4 se presentan utilizando esta forma. Las características principales de este tipo de descripción son:

- Por medio del lenguaje natural se expresan las interacciones entre las entidades que se ven involucradas en el protocolo.
- El orden en que se deben ejecutar los cálculos se indica en forma textual.
- La indicación del tipo de cada variable o entidad se realiza mediante el uso del operador de relación '$\in$'.
- Para el operador de asignación se utiliza el carácter '$\leftarrow$'.
- Las entradas y requerimientos para cada algoritmo se indican mediante el uso de frases como: "dado el entero...", "se requiere de la llave..." o similares.

- Las salidas de los algoritmos se indican mediante el uso de las frases: “se envía la llave...”,
- La comparación entre variables y resultado de cálculos se indica con el carácter ’=’, o con la frases: “se compara si...” o “se verifica sí...”.
- Las funciones usadas dentro de la descripción se detallan al inicio del documento, o en ocasiones, en una sección especifica del protocolo.
- En algunos casos, todas las variables, entidades y datos se declaran en la sección de “Inicialización”, si es que esta existe, de no ser así, se realiza en las diferentes secciones donde se utilizan.

Podemos decir que la descripción literal es extensa en detalles, ya que explica a fondo todas las interacciones entre entidades, el tipo de variables usadas y cálculos que se presentan dentro del protoclo. La tarea de realizar el proceso de traducción a un lenguaje de programación recae en el implementador del protocolo. Esto puede convertirse en un problema, ya que dependiendo de la complejidad del diseño, de las funciones usadas, el tipo de emparejamiento empleado o por restricciones del sistema, puede ocasionar que el implementador omita o modifique partes de la propuesta original, ocasionando en un caso extremo, que la implementación no cumpla con los servicios de seguridad que originalmente se abarcaban con el diseño original.

5.2.1.2. Descripción estándar

Aunque es menos común que la descripción literal, la descripción estándar utiliza un lenguaje que tiene cierta similitud con pseudocódigo. Este tipo de descripción trata de abstraer el contenido de la descripción literal, utilizando un lenguaje más limitado, dicha descripción posee las siguientes características:

- La descripción del protocolo es más fácil de leer, ya que las sentencias pueden leerse de manera secuencial.

- La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí, por ejemplo: el caso de las comparaciones o el tipo de dato en la entrada de cada sección.
- El tipo de dato se indica mediante el carácter '$\in$'.
- El operador de asignación es indicado con el carácter '$\leftarrow$'.
- La entrada y salida de datos para cada sección se indica al inicio de la misma.
- La comparación entre variables y resultado de cálculos es indicado con el uso del carácter compuesto '$\stackrel{?}{=}$'.
- La selección aleatoria de un elemento se indica con el carácter compuesto '$\stackrel{\$}{\leftarrow}$'
- La declaración de elementos nuevos se realiza al momento, es decir, cuando aparece por primera vez en la descripción.

Retomando los protocolos presentados en la sección 4.4, presentamos la descripción estándar de los protocolos de firma corta y Diffie-Hellman tripartito en los algoritmos 5.2 y 5.1 respectivamente.

Algoritmo 5.1 Protocolo de una ronda Diffie-Hellman tripartito

- Generación de llaves

  Entrada: $P \in \mathbb{G}_1$

  Salida: a, P_a, b, P_b, c, P_c

  1: $A : a \stackrel{\$}{\leftarrow} \mathbb{Z}_q, P_a = [a]P$

  2: $B : b \stackrel{\$}{\leftarrow} \mathbb{Z}_q, P_b = [b]P$

  3: $C : c \stackrel{\$}{\leftarrow} \mathbb{Z}_q, P_c = [c]P$

- Intercambio de datos

  Entrada: P_a, P_b, P_c

  1: $A : P_a \rightarrow B, C$

  2: $B : P_b \rightarrow A, C$

  3: $C : P_c \rightarrow A, B$

- Secreto compartido

  Entrada: P_a, P_b, P_c

  Salida: K_{ABC}

  1: $K_A = \hat{e}(P_b, P_c)^a$

  2: $K_B = \hat{e}(P_a, P_c)^b$

  3: $K_C = \hat{e}(P_a, P_b)^c$

  4: $K_{ABC} = K_A = K_B = K_C = \hat{e}(P, P)^{abc}$

Algoritmo 5.2 Protocolo de firma corta

- Generación de llaves

  Entrada: $P \in \mathbb{G}_1$,
  Salida: a, A
  1: $K_a : a \xleftarrow{\$} \mathbb{Z}_q^*$
  2: $PK_a : A \leftarrow [a]P$

- Firmado

  Entrada: $m \in \{0,1\}^*$
  Salida: S
  1: $M \leftarrow H_1(m)$
  2: $S \leftarrow [a]M$

- Verificación

  Entrada: S
  1: $M' \leftarrow H_1(m)$
  2: $\hat{e}(M', A) \stackrel{?}{=} \hat{e}(S, P)$

5.2.2 Lenguaje LENPROC

Actualmente los lenguajes de programación de alto nivel permiten declarar métodos (subrutinas) y variables o atributos, prácticamente en cualquier parte del programa principal, siguiendo las reglas sintácticas definidas de cada uno de éstos lenguajes; por ejemplo: en lenguaje C, se pueden declarar métodos y atributos sin importar el orden en el que estén situados con respecto a otras subrutinas.

A diferencia de los lenguajes actuales, LENPROC presenta rigidez y poca flexibilidad en cuanto al orden y estructura que debe tener la descripción de un protocolo, generalmente, se debe a que en la literatura primero se muestran los elementos que pertenecen al protocolo, como: entidades y variables. para después mostrar la definición de cada uno de sus secciones; por ejemplo: generación de llaves, cifrado o generación de firma, por decir algunas.

5.2.3 Estructura general del lenguaje LENPROC

Aunque dentro de la literatura no existe algún lenguaje formal para poder describir protocolos criptográficos, en este trabajo de tesis se brinda una propuesta para dicho propósito.

Utilizando la propuesta del lenguaje, en su estructura general, la descripción de un protocolo se compone de dos secciones: el encabezado del protocolo y el cuerpo del protocolo. Dicha

estructura puede ser observada en la figura 5.1.

```
/*Inicia Encabezado*/
PROTOCOL nombre_del_protocolo
/*Termina Encabezado*/

BEGIN
/*Inicia Cuerpo de la Descripción*/
declaración de variables
DESCRIPTION{
          SECTION nombre_seccion_1{
                    sentencia 1
                    sentencia 2
                    ..
          }
          SECTION nombre_seccion_1{ ... }
}
/*Termina Cuerpo de la Descripción*/
END
```

Figura 5.1: Estructura general de una descripción

Las secciones mencionadas anteriormente se describen enseguida:

1. **Encabezado**: El encabezado del protocolo está compuesto por la palabra reservada `PROTOCOL`, seguida del identificador para el *nombre del protocolo*.

   a) *Nombre del protocolo*: Es un una cadena alfanumérica que sirve para identificar y nombrar el protocolo que se describe.

2. **Cuerpo**: Dentro del cuerpo se localiza la descripción del protocolo, aquí se describirán las diferentes etapas que componen el protocolo, por ejemplo: la generación de llaves o el cifrado de datos. Además, el cuerpo cuenta con una sección para agregar todos los elementos que interactúan en el protocolo, es indispensable declarar el tipo y nombre de cada elemento que aparecerá en la descripción, dicha declaración se realiza dentro de la sección llamada *declaración de variables*.

a) *Sección para declaración de variables*: Sirve para crear un identificador para aquellos elementos o variables que intervienen dentro de la especificación del protocolo. Además, durante el proceso de compilado, la declaración de variables sirve para verificar errores semánticos. También, se verifica la correctitud en los operandos y parámetros que son utilizados por las diversas funciones y primitivas criptográficas.

b) *Sección para descripción del protocolo*: Esta sección se compone por las diferentes secciones que forman al protocolo criptográfico. Dentro de estas secciones se indican todas las sentencias que son necesarias para el protocolo. La escritura de sentencias se realiza utilizando los operadores, las funciones, las estructuras de control y las primitivas criptográficas que son válidas dentro del lenguaje; por ejemplo: adiciones, operaciones lógicas, la generación de números aleatorios, y la multiplicación escalar, por decir algunas. Dentro de cada sección, la precedencia de ejecución de las sentencias, será indicada mediante el orden en que aparecen en el código.

En la descripción 5.1 las palabras resaltadas en negro son palabras reservadas dentro del lenguaje. Además, en el mismo lenguaje se cuenta con palabras reservadas que sirven para trabajar con un entorno de emparejamientos bilineales, por ejemplo: los valores de p y r, los identificadores para las extensiones de campo y los identificadores para las curvas elípticas usadas. Una explicación de estos elementos y otros detalles pueden ser consultados en [Beuchat et al., 2010].

5.2.3.1. Sección para declaración de variables

En la mayoría de lenguajes de programación se pueden declarar variables durante cualquier parte del código fuente, en esta propuesta de lenguaje, se requiere que la declaración se realice previamente, con la finalidad de poder generar el entorno de la implementación en lenguaje Magma o lenguaje C/C++.

La declaración de variables puede verse ilustrada en la figura 5.2. El identificador de la variable es una cadena de caracteres alfanuméricos y los tipos son palabras reservadas dentro del lenguaje. En el cuadro 5.1 se muestran los principales tipos de datos usados dentro del lenguaje.

```
tipo_variable identificador_variable_1,
identificador_variable_2,...;
tipo_variable identificador_variable_1[largo_arreglo];
```

Figura 5.2: Declaración de Variables.

En la tabla 5.1 el punto $(x, y) \in \mathbb{G}_1$, el punto $(x', y') \in \mathbb{G}_2$, y el elemento en $\mathbb{G}_T \in \mathbb{F}_{p^{12}}$ donde $\mathbb{G}_1, \mathbb{G}_2$ y $\mathbb{G}_T$ están definidos en el apéndice A

Tipo	**Descripción**	**Palabra reservada**	**Ejemplo**
Cadena alfanumérica	Cadena de caracteres de longitud variable comienza con " y termina en "	tString	"Algo", "Esto es una cadena"
Punto en $\mathbb{G}1$	Un punto de una Curva Elíptica en el grupo $\mathbb{G}1$	tPointG1	(x, y)
Punto en $\mathbb{G}2$	Un punto de una Curva Elíptica en el grupo $\mathbb{G}2$	tPointG2	(x', y')
Elemento en $\mathbb{G}_T$	Un punto de una Curva Elíptica en el grupo $\mathbb{G}_T$	tPointGT	
Escalar	Es un número entero de n-bits	tScalar	$(125689)_{10}$, $(1256875245300)_{10}$

Tabla 5.1: Tipos de datos

5.2.3.2. Sección para la descripción del protocolo

Los pasos o sentencias a seguir en el protocolo a ser descrito, quedarán ubicados dentro de esta sección. El inicio de la la sección comienza con la palabra `DESCRIPTION` quedando entre llaves las secciones o etapas que conforman dicha descripción. En esta sección pueden ser descritas

las diferentes etapas que conforman el protocolo, por ejemplo, el protocolo de firma corta BLS [Boneh et al., 2001] está conformado por tres etapas principales: la sección de generación de llaves, la sección del firmado y la sección para la verificación de la firma.

Al inicio de cada sección se encuentra la palabra **SECTION**, seguido por el nombre para la sección. Las sentencias que se ejecutan en cada sección del protocolo quedarán encerradas entre llaves. En la figura 5.3 podemos observar la descripción para la sección de generación de llaves del protocolo BLS. Podemos utilizar cuantas secciones se requieran dentro de una descripción.

```
SECTION Generacion_Llave{
x := Random(1,r_p);
P := Generator(r_E);
X := [x]P;
}
```

Figura 5.3: Sección de generación de llaves (protocolo BLS)

Las sentencias especifican y controlan el flujo de la implementación. Se cuenta con diferentes tipos de sentencias, en el cuadro 5.2 se muestran algunos de estos tipos:

Sentencia	Ejemplo	Descripción

Expresión	`2 + x;`
	`X := [x]P;`
	`M := H1(‘‘mensaje’’);`
		Es una secuencia de operadores, operandos, elementos de puntuación y palabras clave, que especifican un cálculo.

Compuestas	`if (primero) then`
	`x:=1;`
	`else`
	`x:=2;`
	`end if;`
		También denominadas bloques, se utilizan en aquellas situaciones en que la sintaxis espera una sentencia especificación, pero se requiere usar varias

Iteración	`for i := 0 to 10 do`
	`x := x + 1;`
	`end for;`
		Permiten repetir un conjunto de sentencias ejecutando un bucle

Tabla 5.2: Principales tipos de sentencias

5.3 Arquitectura de COMPROC

En ésta sección se describen las diferentes etapas que componen el proceso de compilación, explicando las tareas que se realizan para obtener una traducción correcta de la especificación de un protocolo.

De forma general, un compilador criptográfico tiene la intención de generar el código de un programa que contiene determinadas funciones criptográficas, este código puede estar escrito en cualquier lenguaje de programación. COMPROC genera código para lenguaje Magma y Lenguaje C/C++. Además, es posible invocar al intérprete de Magma o al compilador GCC para ejecutar el código de la implementación. En la figura 5.4 se puede apreciar la arquitectura general del funcionamiento de COMPROC.

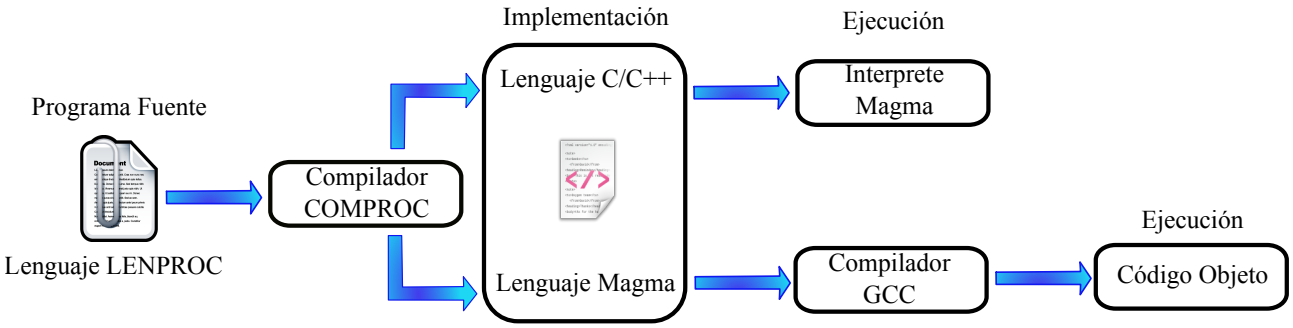


Figura 5.4: Arquitectura de COMPROC.

El proceso de compilación interno de COMPROC está dividido de la misma forma que un compilador típico, sus partes pueden ser agrupadas en dos secciones generales: el análisis del programa fuente y la síntesis del código de salida.

5.3.1 Análisis

El objetivo general en esta etapa consiste en realizar la comprobación de la correctitud del programa fuente. Este análisis se divide en tres diferentes secciones:

1. **Análisis Léxico**: Consiste en descomponer el programa en componentes léxicos. En general, se busca identificar tres componentes: identificadores, cadenas de caracteres y valores numéricos. Además, mientras se realiza este proceso en la sección de declaración de variables, se obtienen dos tablas que sirven de soporte para las siguientes tareas.

   La primera tabla, llamada *tabla de símbolos*, sirve para la verificación de errores durante el análisis sintáctico y semántico. La segunda tabla a obtener será llamada *tabla de tipos*, la cual está compuesta con los identificadores de todas las variables declaradas, junto con su tipo de dato. Ésta tabla cobra relevancia, ya que con ayuda de ella, se realiza la primer generación de código intermedio. Dicha generación consiste en formar un preámbulo con la declaración de las variables en lenguaje Magma o lenguaje C/C++, según se halla indicado al comienzo del proceso de compilación, además sirve para verificar en cada sentencia que el tipo de operandos utilizados corresponda con el tipo de operación especificado.

   Las principales expresiones regulares que se comprueban durante esta sección se pueden ver reflejadas en la tabla 5.3.

Componente Léxico	Expresión Regular	Ejemplo
Identificador	('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*	"algo","A_l_G_o","varN1"
Cadenas de caracteres	' " '('\u0020'..'\u007E')+ ' " '	"String o lo que sea"
Números	('0'..'9')+	"502","005","22013"

Tabla 5.3: Principales expresiones regulares válidas.

   De igual manera, se revisa si aparecen palabras reservas propias del lenguaje LENPROC, Magma o C/C++. Por ejemplo: no es válido utilizar la palabra reservada "*namespace*", para declarar una variable dentro de una descripción, ya que dicha palabra pertenece al lenguaje C/C++. El apéndice B contiene una lista completa de las palabras reservadas.

2. **Análisis Sintáctico**: Es la fase del analizador que se encarga de verificar que el código de entrada corresponda a la gramática propuesta. En caso de que la verificación de el

programa de entrada sea válida, se suministra el árbol sintáctico que lo reconoce.

Además, durante el análisis se accede a la tabla de símbolos, para hacer parte del trabajo del analizador semántico; también, se genera código intermedio conforme se analiza cada una de las sentencias de código (haciendo una primera traducción de éstas a sus respectivas salidas en lenguaje Magma o lenguaje C/C++). Por ultimo, se identifican los diversos errores que se producen, informando de ellos cuando son encontrados.

La sintaxis completa del lenguaje LENPROC se puede revisar en el apéndice C, la cual está descrita en sentencias BNF.

3. **Análisis Semántico**: Ésta etapa del análisis utiliza como entrada el árbol sintáctico detectado para comprobar restricciones con el tipo de datos y otras limitaciones semánticas, preparándose para la generación de código. El objetivo principal es evitar los "absurdos" (por ejemplo evitar la multiplicación entre cadenas alfanuméricas y punto de una curva) dentro de la propuesta de nuestro lenguaje.

   Esta etapa usa como herramienta la tabla de símbolos, con la finalidad de realizar dos verificaciones: la primera consiste en verificar el tipo de operadores y operandos en las sentencias, la segunda verificación consiste en revisar el orden en que se indican las operaciones y funciones, es decir, le da una precedencia a todos los operadores que se utilizan. Además se verifican otros puntos, tales como: que todas las variables han sido previamente declaradas y la verificación de los rangos de los *arrays* estén dentro de un rango valido.

5.3.2 Síntesis

Una vez analizado el programa de entrada, es necesario generar código para la máquina objetivo. Cabe aclarar que un compilador tradicional tiene como entrada el código de un programa en algún lenguaje de alto nivel y a su salida se obtienen código maquina o ejecutable.

En el caso de este trabajo de tesis, estrictamente hablando se está realizando una traducción de código, ya que se parte de un código fuente de la especificación de un protocolo (propuesto en esta tesis), y se entrega el código de la implementación en lenguaje C/C++ o lenguaje Magma, que posteriormente será compilado y ejecutado por el compilador o intérprete que corresponda a cada lenguaje. En la figura 5.5 se puede ver un diagrama a bloques de cómo interactúan todas las etapas del compilador.

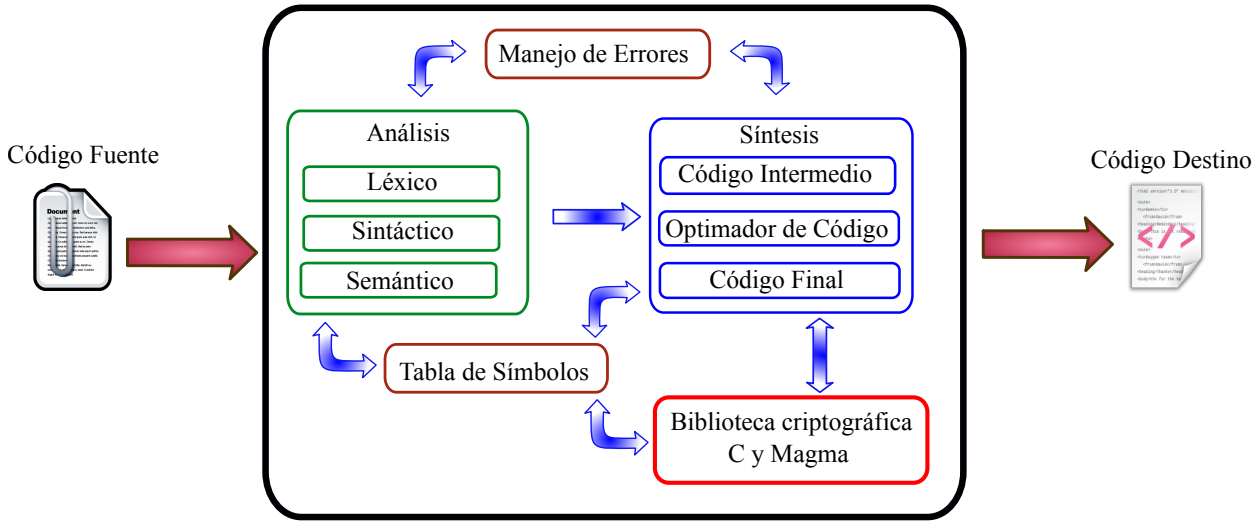


Figura 5.5: Arquitectura interna de COMPROC

El objetivo de la síntesis consiste en generar la salida expresada en el lenguaje destino. Esta generación de código está dividida en tres faces:

- **Generación de código intermedio**: Consiste en generar un código preliminar de la versión final, en este caso, el proceso se divide en generación de las tres secciones del código final: la primera sección contienen los diversos preámbulos que se forman con la información obtenida de la identificación de las variables identificadas en el análisis, la segunda sección está compuesta con el código de la primitivas criptográficas utilizadas, mientras que la tercera sección está compuesta por la traducción de la especificación del protocolo.

- **Optimización de código**: En esta fase se busca que el código generado cuente con la mayor eficiencia posible para su futura ejecución. Por ejemplo, se busca eliminar sentencias repetidas, reducir el número de operadores de una sentencia sin que ésta altere el resultado de la misma. En general, se busca que la implementación sea lo más eficientemente posible sin que se modifique la especificación original del protocolo.

- **Generación final de código**: En nuestro caso, al tratarse de una traducción, se toman las diferentes partes de código obtenidas en las etapas anteriores y se agrupan en una implementación final, la cual puede ser ejecutada posteriormente con su respectivo compilador.

  Al final de este proceso se agregan funciones que ayudan a la presentación de los resultados del protocolo, para el caso caso particular de nuestro traductor, se interpreta la implementación lenguaje Magma, o se compila el código obtenido en lenguaje C.

5.4 Generación automática de código

La generación de código data desde la existencia de los primeros compiladores, hasta la aparición de los primeros generadores de código comerciales u orientados a usuarios finales; la generación automática de código era exclusividad de programas compiladores especializados. En esta sección, se describen las referencias consultadas con respecto a esta tarea, ya que se retoman las principales ideas para el funcionamiento de nuestra herramienta.

Con respecto a la implementación de esquemas criptográficos, en [Dominguez Perez and Scott, 2009] los autores presentan una herramienta para la generación semiautomática de código en lenguaje C de la implementación de emparejamientos *Tate*, *ate*, *R-ate* y óptimo. El código generado posteriormente puede ser utilizado en la construcción de protocolos criptográficos.

Cuando se trabaja con RSA, la implementación puede ser utilizada para trabajar en

diferentes niveles de seguridad con diferentes parámetros, sin necesidad de modificar la implementación, buscando que la implementación sea eficiente. En el caso de las construcciones basadas en emparejamientos bilineales se requiere un trabajo más complejo para poder adaptar la implementación a diferentes niveles de seguridad, ya depende de diversos factores; tales como: la familia de curvas con las que se este trabajando, o la selección de parámetros relacionados con el emparejamiento. En este caso, la generación de código parte de selección de parámetros de entrada, que nos permiten establecer diferentes niveles seguridad, otorgando una construcción eficiente trabajando con curvas *amigables* con los emparejamientos. El compilador muestra su mejor desempeño utilizando curvas con grado de encajamiento $k = 2^i \cdot 3^j$, donde $i > 0$ y $j \geq 0$.

Sin abundar en detalles sobre la herramienta propia, el documento hace mención de todas las funciones involucradas en la generación del código del emparejamiento, como son: el ciclo de Miller, la exponenciación final, la generación de cadenas de adición y el picadillo a $\mathbb{G}_2$.

El código generado utiliza la biblioteca *MIRACL* [Scott, 2003] para la representación de los valores de precisión variable, además de utilizar *Magma* en algunas partes durante la ejecución. Una descripción detallada del método propuesto para la generación puede observase en [Dominguez Perez, 2011].

En [Pozza et al., 2004], se cuenta con una propuesta para llevar al cabo la generación semiautomática de código Java a partir de la descripción de un protocolo criptográfico, dicha descripción esta basada en el lenguaje formal del cálculo *spi*.

El cálculo *spi* es una extensión de cálculo *pi*, el cual está diseñado para describir y analizar la seguridad en los protocolos. Si solamente se utiliza el cálculo *pi* para describir un protocolo criptográfico, la descripción se limita en ciertos aspectos. Ya que el cálculo *pi* no cuenta con una forma lo suficientemente robusta que ayude a describir primitivas criptográficas en toda su extensión. Una explicación más detallada de este problema se tiene en [Abadi and Gordon, 1998].

El resultado obtenido de ese trabajo, es una herramienta de software que lleva el nombre

de *Spi2Java*, la cual, al momento de la elaboración del artículo, es capaz de obtener la implementación en código Java de protocolos de seguridad ya estandarizados. Además de obtener el código de la implementación, *Spi2Java* incluye un analizador léxico y sintáctico de protocolos, junto con el proceso de análisis de seguridad del protocolo.

El programa *Spi2Java* está compuesto de forma general de dos módulos. El primero está formado por dos secciones, la primer sección contiene los tipos de términos (*Term Type*) utilizados durante el proceso de obtención del código de salida; la segunda sección es la encargada de verificar la descripción hecha del protocolo (*Description Checker*), verificando errores léxicos y sintácticos de dicha descripción. Ambas secciones se encargan de llenar el vacío de información entre la especificación del protocolo y, la aplicación del mismo (refiriéndose a los tipos de datos utilizados). En particular, este bloque comprueba automáticamente si las variables que se utilizan son consistentes dentro del proceso descrito del protocolo, y si esta verificación es positiva, les asigna un espacio dentro de la implementación en Java.

El segundo módulo es el encargado de la generación del código Java. El funcionamiento de este módulo consiste en que, a partir de una especificación dada, el generador de código escribe la implementación del protocolo en lenguaje Java. Además, el generador de código también produce un esqueleto de la aplicación, que contiene algunos archivos de clases útiles para construir una aplicación cliente/servidor. El diagrama de la figura 5.6 muestra el funcionamiento del proceso de la generación de código.

El archivo de entrada, contiene la descripción del protocolo usando el lenguaje *Spi*. En la primera etapa del proceso, no realiza un análisis semántico, esta parte trata de ser cubierta por la sección *Term type*. En cada nuevo archivo, se deben describir los tipos de datos que se utilizarán, esta es una diferencia notable, ya que en un compilador típico existen tipos de datos pre-establecidos. Con *Spi2Java*, cada nueva descripción requiere especificar las entidades involucradas, indicando factores como: qué tipo de operaciones pueden hacer, con qué entidades se pueden comunicar o la vulnerabilidad que cada entidad presenta. Dejando una fuerte carga

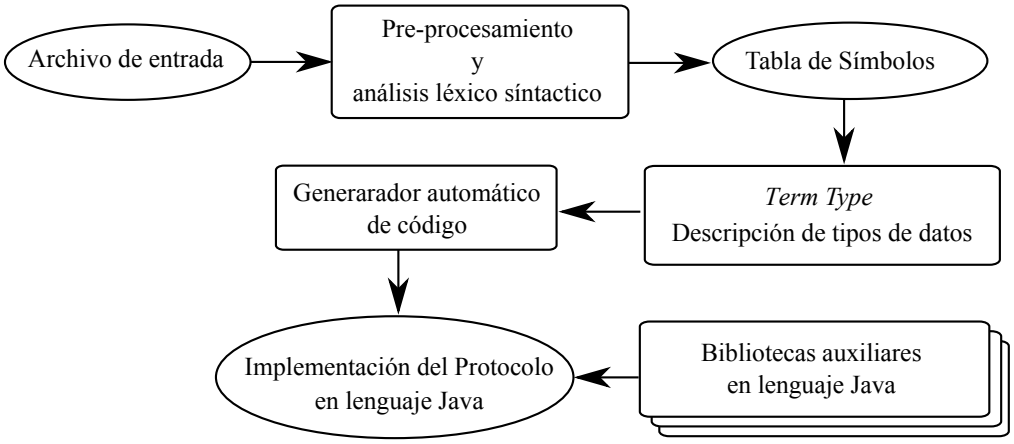


Figura 5.6: Generación de código, usando *Spi2Java*

de trabajo para el usuario al momento de iniciar la descripción de un protocolo.

En nuestro trabajo de tesis, COMPROC tiene diferentes etapas para generar código intermedio. Cuando se requiere de una implementación en lenguaje C/C++, COMPROC genera tres versiones de código intermedio; las primeras dos versión de código intermedio son generadas por nuestro compilador, estas contienen una primer traducción de las sentencias encontradas en el código fuente, además, incluyen el código que permite utilizar la biblioteca criptográfica junto con la declaración de variables. La tercera versión será generada por el compilador GCC, esta última versión solo se creara sí se desea ejecutar la implementación de un protocolo descrito.

En cuanto a lenguaje Magma, siempre se obtendrán las primeras dos versiones de código, la tercera versión no se genera por que el interprete de Magma ejecuta cada instrucción al momento en que es leída del archivo de entrada. En la figura 5.7 se muestra el proceso que sigue COMPROC para generar el código de la implementación.

5.4.1 Descripción del protocolo de firma corta BLS

El primer requerimiento para compilar un un protocolo, es contar con el código de la descripción, por tal motivo, en la figura 5.8 se muestra la descripción completa del protocolo de firma corta BLS. Dentro de la descripción se puede notar el uso de diversas primitivas

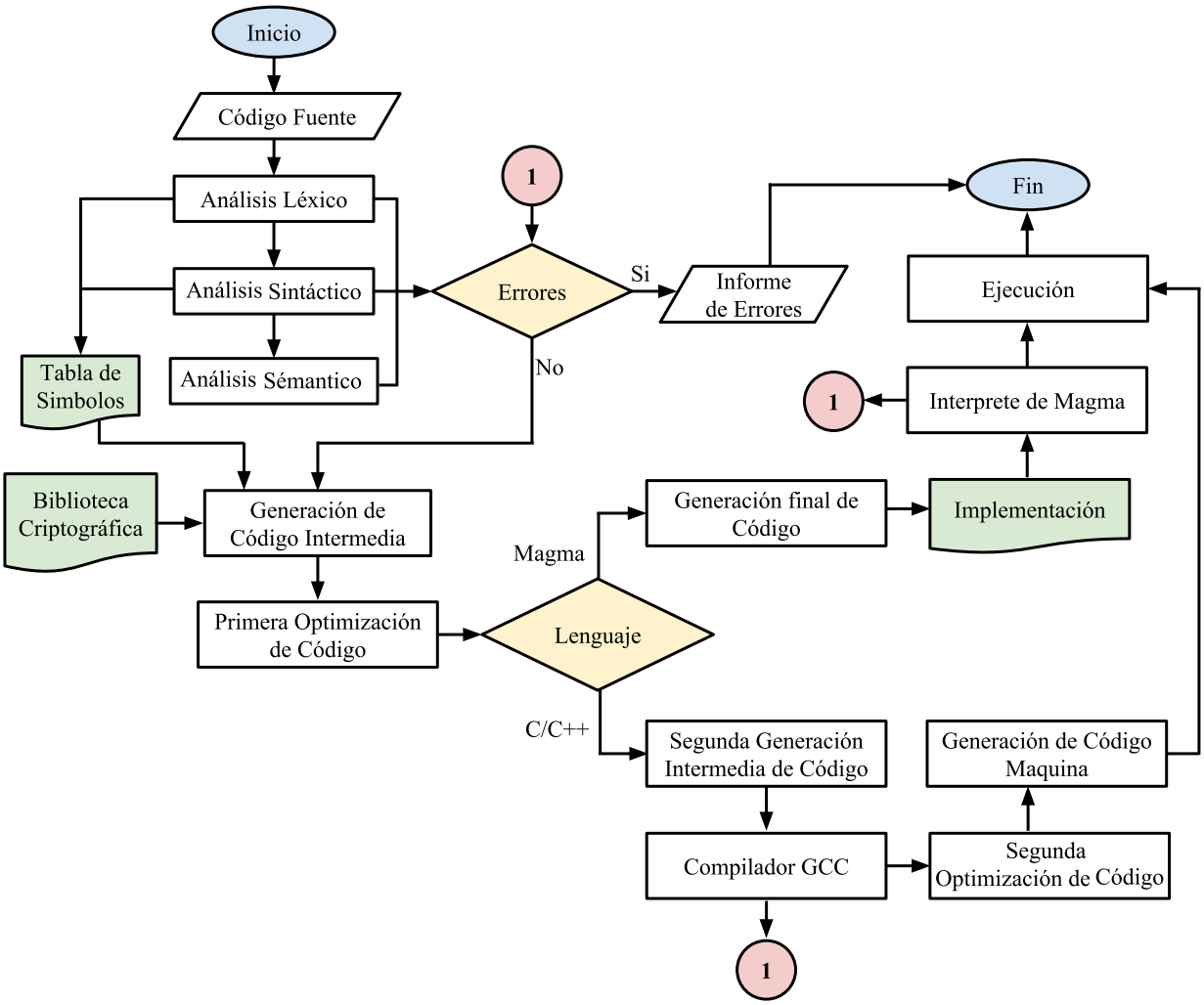


Figura 5.7: Generación de código, usando LENPROC

criptográficas, como son: multiplicación escalar, generación de números aleatorios, generación de elementos generadores, función picadillo y el emparejamiento bilineal.

```
PROTOCOL BLS{
  tString m,mp;
  tPointG1 P,A;
  tPointG2 M,Mp,S;
  tNumber a;

  DESCRIPTION{
    SECTION GeneracionLlaves{
      a := Random(1,r_p); //Llave privada
      P := Generator(r_E);
      A := [x]P; //Llave publica
    }
    SECTION Firma{
      m :="Hola Beto";
      M := H2(m);
      S :=[a]M; //Firmado
    }
    SECTION Verificacion{
      mp :="Hola Rogelio";
      Mp := H2(mp);
      if(E(P,S)==E(A,Mp)) then
        Print("Firma Valida");
      else
        Print("Firma Invalida");
      fi;
      Preview(E(P,S));
      Preview(E(A,Mp));
    }}}
```

Figura 5.8: Descripción del protocolo BLS, utilizando el lenguaje LENPROC

5.4.2 Descripción del protocolo de firma a ciegas

En la figura 5.9 se puede apreciar la descripción del protocolo de firma a ciegas de Bolydereva. En este caso, aparece la sección *Revision*, la cual no está presente en el protocolo original, de

esta forma respetamos la secciones originales del diseño facilitando la lectura del protocolo.

```
PROTOCOL BlindSign_Boldyreva
BEGIN
  /*HEAD*/
  tString m,mp;
  tPointG1 P,X;
  tPointG2 Mp,S,Sp,HashM;
  tNumber x,r,rInv;
  /*BODY*/
  DESCRIPTION
  {
    SECTION Setup{
      P:=Generator(r_E);//Generador
      x:=Random(1,r_r);//Secret Key
      X:=[x]P;//Public key
    }
    SECTION Blinding{
      m := "Test";
      r := Random(1,r_r);
      HashM := H2(m);
      Mp:= [r]HashM;
    }
    SECTION Sign{
      Sp:= [x]Mp;//Signing
    }
    SECTION Unblinding{
      rInv:=Inverse(r,r_r);
      S:=[rInv]Sp;
    }
    SECTION Verificacion{
      mp:="Color Test ";
      if (E(X,H2(mp))==E(P,S)) then
        Print("Valid Signature");
      else
        Print("Invalid Signatur");
      fi;
    }
    SECTION Revision{
      /*Submission of data*/
      Preview(E(X,H2(mp)));
      Preview(E(P,S));
    }
  }
END
```

Figura 5.9: Descripción del protocolo de firma a ciegas, LENPROC

5.4.3 Características de las descripciones

A continuación enlistamos algunas características que se presentan en las descripciones de las secciones 5.4.1 y 5.4.2:

- El uso de primitivas criptográficas en su mayoría se realiza mediante llamados a funciones,

por ejemplo, el emparejamiento bilineal está definido por el llamado a la función $E()$.

- La única primitiva que utiliza una función para su llamado es la multiplicación escalar, la cual se llama mediante el uso de la expresión $[x]P$, en este caso, x será el escalar y P un punto en una curva elíptica. El uso de los corchetes es exclusivo para la multiplicación escalar, por tal motivo, la agrupación, la precedencia y la definición de un argumento se indicará solo con el uso de paréntesis.

- Las expresiones en color verde son considerados comentarios dentro de la descripción y de forma general no tienen relevancia para la implementación, por lo cual son ignorados durante el proceso de compilación.

- Las palabras en color azul son reservadas dentro del lenguaje LENPROC (véase apéndice B).

- Un último detalle que cobra importancia, ya que afecta el tiempo de cómputo de la implementación, se presenta al momento de usar la función *Preview()*. En los dos casos se calcula el emparejamiento bilineal dentro del llamado de función, el mismo cálculo se realizo dentro de la comparación para validar la firma. Por tal motivo, el uso de variables para almacenar resultados parciales ayuda al desempeño de la implementación.

6

Conclusiones y trabajo futuro

En este trabajo de tesis se desarrolló el compilador criptográfico COMPROC, el cual fue desarrollado sobre lenguaje C/C++. COMPROC tiene el objetivo principal proporcionar el código fuente de la implementación de un protocolo criptográfico en lenguaje Magma y lenguaje C/C++, pero además, haciendo uso de herramientas externas, COMPROC es capaz de ejecutar dicha implementación.

6.1 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:

- Todo compilador requiere de un lenguaje de programación, el cual sea capaz de abstraer los elementos y operaciones que expresan procesos que pueden ser llevados a cabo por las computadoras. En este caso, la propuesta que realizamos es un lenguaje para la descripción

de protocolos criptográficos basados en emparejamientos bilineales. LENPROC trata de abstraer todos los elementos involucrados en el diseño de protocolos, como son: las funciones o primitivas criptográficas, las operaciones entre ellas, las entidades involucradas y los elementos o tipo de datos utilizados en cada etapa del protocolo.

- La definición del lenguaje presentó un reto interesante, primeramente, la definición de las palabras reservadas y estructuras, las cuales debían ser representativas en el área,y segundo, se buscó que la manera de utilizarlas fuera sencilla y explícita, la intención es que la tarea de describir un protocolo con el uso de LENPROC fuera simple pero clara a la vez.

- Uno de los objetivos principales de cualquier compilador, es la optimización del código generado. COMPROC es un compilador que utiliza un intérprete y un compilador externo, con el uso de estas herramientas complementamos la tarea de optimización de código.

- El uso de una biblioteca criptográfica eficiente garantiza que la implementación sea eficiente. Para el caso de la traducción a lenguaje C/C++, se usó una biblioteca criptográfica externa, la cual reporta una buena eficiencia con respecto a tres funciones involucradas: el emparejamiento bilineal, la multiplicación escalar de puntos y la proyección de una cadena a un punto. Estas funciones son las base de varios sistemas criptográficos basados en emparejamientos.

- Con respecto a la biblioteca en lenguaje Magma, se utilizaron los mismos algoritmos que en la biblioteca de lenguaje C/C++, el objetivo era tener una eficiencia homogénea entre ambas bibliotecas.

- Los protocolos presentados en el capítulo 4 fueron usados como casos de estudio para verificar la correctitud de la traducción y la eficiencia de la implementación, pero el verdadero alcance y utilidad del modelo propuesto no se podrá estimar hasta realizar

la descripción de un gran número de protocolos.

- Usando cierto tipo de emparejamientos, se pueden obtener diferentes versiones de la implementación de un protocolo. Para el desarrollo de este trabajo se utilizó el emparejamiento asimétrico $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, también llamado emparejamiento de tipo tres. Se consideró el uso de curvas BN con 126 bits de seguridad. Como se mencionó en la sección A.11, 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 tamaño del subgrupo de las curvas elípticas. En este trabajo se tomó $z = -(2^{62}+2^{55}+1)$. La selección es de este parámetro es importante ya que se mejora la eficiencia en diferentes operaciones de la biblioteca.

- 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: ω-NAF, *comb*, GLV y GS. Los cuales reducen el número de sumas y doblados de puntos con respecto al método binario descrito en la sección 3.1.

- Para finalizar, podemos concluir que para realizar un compilador no es necesario iniciar desde cero, si no que se pueden tomar un conjunto de herramientas e integrarlas de cierto modo que cada una cumpla una función especifica dentro del compilador, para así poder lograr el objetivo final que es obtener el programa ejecutable del diseño de un protocolo. Al igual que con cualquier lenguaje de programación, una mala descripción requerirá mayor tiempo para su ejecución.

6.2 Trabajo Futuro

- Día a día surgen nuevas bibliotecas criptográficas las cuales buscan ser más eficientes, la estructura de COMPROC permite adaptar otras bibliotecas criptográficas. Por lo cual

queda pendiente la búsqueda de otras bibliotecas disponibles. Dichas bibliotecas deben contener las primitivas utilizadas en los esquemas basados en emparejamientos.

- Buscar nuevos métodos de optimización de código, el objetivo sería buscar que la carga para las herramientas de compilación externas sea menor, mejorando de esta manera el proceso de compilación.

- Hasta el momento COMPROC solo trabaja sobre esquemas basados en emparejamientos bilineales, por tal motivo, se tendría que extender la gramática del lenguaje LENPROC y que de esta manera se pudiese trabajar con otro tipo de criptografía, como RSA o criptografía de Curvas Elípticas.

- Crear una interfaz visual para brindar un entorno más agradable (tecnologías *touch*) para los usuarios.

- Implementar en su totalidad una biblioteca criptográfica que se adapte en su totalidad a COMPROC, ya que la adaptabilidad cien porciento funcional de bibliotecas externas puede ser una tarea complicada para futuros usuarios de nuestra herramienta.

- Realizar más pruebas de correctitud del compilador, se tiene un portafolio con protocolos representativos (véase el cápitulo 4), los cuales presentan una implementación correcta, pero se busca ampliar este número con la finalidad de mostrar el verdadero potencial de la herramienta.

A

Conceptos Básicos

La principal primitiva criptográfica usada en los protocolos analizados durante el desarrollo de éste trabajo es el emparejamiento bilineal. Por tal motivo, en este apéndice se revisan las estructuras algebraicas como los grupos abelianos, anillos, campos finitos, entre otros, a través de cuales se construye dicha primitiva. Además, Se describen algunas definiciones de teoría de números y álgebra de acuerdo con [Shoup, 2005, Cohen et al., 2010, Castañeda, 2011, Hankerson et al., 2004].

A.1 Grupo

Definición A.1 (Función indicatriz de Euler). *Sea n un entero positivo, la función $\varphi(n)$ de Euler se define como*

$$\varphi(n) = \#\{k \in \mathbb{N} : k \leq n, mcd(k, n) = 1\}$$

Es decir la cantidad de enteros positivos menores o iguales a n que son primos relativos con n.

Un grupo es un objeto matemático abstracto denotado como $(\mathbb{G}, \star)$, donde $\mathbb{G}$ es un conjunto de elementos y $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$, es una operación binaria entre los elementos del conjunto, que satisfacen los siguientes axiomas:

I. La operación $\star$ es *cerrada* sobre los elementos del conjunto $\mathbb{G}$, es decir, para todo $a, b \in \mathbb{G}$, satisface $a \star b \in \mathbb{G}$.

II. Existe un *elemento identidad* $e \in \mathbb{G}$, tal que, $a \star e = e \star a = a$, donde $a \in \mathbb{G}$. El elemento identidad del grupo $\mathbb{G}$ necesariamente es único.

III. La operación $\star$ es *asociativa* sobre los elementos del conjunto, es decir, para todo $a, b, c \in \mathbb{G}$, cumple $(a \star b) \star c = a \star (b \star c)$.

IV. Para cualquier elemento $a \in \mathbb{G}$ existe un *elemento inverso* $a^{-1} \in \mathbb{G}$, tal que $a \star a^{-1} = e$.

El grupo es llamado *Abeliano* si cumple con la propiedad de *conmutatividad*, es decir, que para todo $a, b \in \mathbb{G}$ se satisface $a \star b = b \star a$.

A continuación se definen algunos conceptos relacionados con los grupos:

Definición A.2 (Orden del Grupo). *El orden de un grupo $(\mathbb{G}, \star)$ está definido como el número de elementos en el conjunto $\mathbb{G}$. Los grupos pueden tener orden finito o infinito.*

Definición A.3 (Orden de un elemento). *Sea $(\mathbb{G}, \star)$ un grupo, y $g \in \mathbb{G}$, denotamos la operación $\star^m(g)$, que consiste en aplicar m veces consecutivas el operador $\star$ sobre el elemento g consigo mismo, donde $g \in \mathbb{G}$ y $m \in \mathbb{Z}^+$. Entonces se dice que el orden de g es el menor entero positivo s, tal que $\star^s(g) = e$.*

Definición A.4 (Generador del grupo). *Dado $(\mathbb{G}, \star)$ se dice que $g \in \mathbb{G}$ es un elemento generador del grupo, si para cualquier $h \in \mathbb{G}$ existe $i \in \mathbb{Z}^+$, tal que $h = \star^i(g)$. Al grupo generado por g se le denota como $\langle g \rangle$.*

Definición A.5 (Grupo Cíclico). *Un grupo* $\mathbb{G} = (\star, e)$ *es cíclico, si* $\mathbb{G} = \langle g \rangle$ *para algún generador que* $g \in \mathbb{G}$.

Todo subgrupo de un grupo cíclico $\mathbb{G}$ es también cíclico, es decir, si el orden de $\mathbb{G}$ es n entonces para cada divisor d de n, $\mathbb{G}$ contiene exactamente un subgrupo cíclico de orden d.

El número de elementos generadores en un grupo cíclico finito $G = (\star, e)$ de orden n, está definido como $\varphi(n)$, donde $\varphi()$ denota la función indicatriz de Euler. Por lo tanto, si $\mathbb{G}$ es de orden primo p, entonces $\mathbb{G}$ tiene $\varphi(p) = p - 1$ generadores.

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

Notación aditiva.

Un grupo $\mathbb{G}$ es escrito en forma aditiva, cuando se utiliza el operador "+" para representar la operación de grupo, y el "0" para representar el elemento identidad, quedando: $(\mathbb{G}, +, 0)$. El elemento inverso de cualquier elemento $a \in \mathbb{G}$ es denotado como $-a$. Para la aplicación de m veces el operador $+$ sobre un elemento $b \in \mathbb{G}$ se denota como mb, donde $m \in \mathbb{Z}^+$.

Notación multiplicativa.

Un grupo $\mathbb{G}$ es escrito en forma multiplicativa, cuando la operación de grupo es representada por "$\times$", y el elemento identidad con "1", quedando: $(\mathbb{G}, \times, 1)$. El elemento inverso de cualquier elemento $a \in \mathbb{G}$ es denotado como a^{-1}. Para la aplicación de m veces el operador $\times$ sobre un elemento $b \in \mathbb{G}$ se denota como b^m, donde $m \in \mathbb{Z}^+$.

A.1.1 Subgrupo

Sea $\mathbb{G} = (\mathbb{G}, \star)$ un grupo y sea $\mathbb{H} \subset \mathbb{G}$. Se dice que $\mathbb{H}$ es un subgrupo de $\mathbb{G}$ si cumple:

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

A.1.2 Clase lateral

Sea $\mathbb{G} = (\star, e)$ un grupo abeliano y $\mathbb{H} = (\star, e)$ un subgrupo de $\mathbb{G}$. Para todo $a, b \in \mathbb{G}$ se escribe $a \equiv b (\text{mod } H)$, si $a \star \bar{b} \in H$, donde $\bar{b}$ es el inverso de b. A la expresión "$\equiv$ (mod $\mathbb{H}$)" se le conoce como *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 $[a]_{\mathbb{H}} = a \star \mathbb{H} = \{a \star h | h \in \mathbb{H}\}$, es decir:

$$x \in [a]_{\mathbb{H}} \iff x \equiv a (\text{mod } \mathbb{H})$$

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* $\mathbb{H}$.

A.2 Anillo

Un anillo R es un conjunto conformado por dos operaciones binarias: "$+$" y "$\cdot$", y satisface las siguientes características:

I. R es un grupo conmutativo respeto a "$+$", es decir, para todo $a, b \in R$ cumplen $a+b = b+a$

II. Existe una identidad multiplicativa en R, es decir, existe un elemento $1_R \in R$, tal que, $1_R \cdot a = a \cdot 1_R = a$, para todo $a \in R$.

III. La operación "$\cdot$" es cerrada y asociativa sobre R, es decir, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

IV. La opreación "$\cdot$" es distributiva sobre "$+$", es decir, para todo $a, b, c \in R$ cumplen: $a \cdot (b + c) = a \cdot b + a \cdot c$ y $(b + c) \cdot a = b \cdot a + c \cdot a$.

V. La multiplicación es conmutativa. Para todo $a, b \in R$, $a \cdot b = b \cdot a$.

Ejemplo A.1. *El conjunto $\mathbb{Z}$ de números enteros, junto con la adición y la multiplicación son un anillo. El conjunto $\mathbb{Z}[x]$ de polinomios con coeficientes en $\mathbb{Z}$, junto con la adición y multiplicación de polinomios son un anillo.*

A.3 Campo

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

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

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

III. El producto se distribuye a ambos lados de la suma, es decir, $(a + b) \times c = a \times b + b \times c$ para todo $a, b, c \in \mathbb{F}$.

Ejemplo A.2. *El conjunto de números racionales $\mathbb{Q}$, junto con la ley adición y la multiplicación son un campo. El conjunto cociente $\mathbb{Z}/\mathbb{Z}_p$, con la adición y multiplicación de enteros, también son un campo para cualquier número primo p.*

Definición A.6 (Característica de un campo). *Dado el campo $\mathbb{F}$ y $n \in \mathbb{Z}^+$, se dice que n es la característica de $\mathbb{F}$, si n es el menor entero positivo tal que $n \times 1 = \sum_{i=0}^{n-1} = 0$, en otras palabras, es el número mínimo de veces que hay que operar la unidad para obtener el elemento identidad. En caso de que no exista tal entero n, se dice que $\mathbb{F}$ es de característica 0.*

Definición A.7 (Campo finito). *Un campo $\mathbb{F}$ es finito, si su número de elementos es finito. Los campos finitos también son conocidos como campos de Galois.*

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 $\mathbb{F}_p = 0, 1, 2, \ldots, p-2, p-1$ define un campo finito $(\mathbb{F}_p, +, \cdot, 0, 1)$ bajo las operaciones suma y multiplicación módulo p.

A.3.1 Extensión de un campo finito

Definición A.8 (Polinomio irreducible). *Sea $f(z)$ un polinomio de orden $m \leq 2$, donde $m \in \mathbb{N}$, se dice que $f(z)$ es irreducible, si no puede factorizarse como el producto de polinomios de grado menor a m.*

El conjunto de polinomios en la variable z con coeficientes en $\mathbb{F}_p$, está denotado por $\mathbb{F}_p[z]$. Dado un número $n \in \mathbb{N}$, el conjunto finito compuesto por los polinomios en $\mathbb{F}_p[z]$ de grado menor a $n-1$, es decir,

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

forman un "campo finito de característica p", denotado por $\mathbb{F}_{p^n}$ bajo las operaciones de adición y multiplicación módulo $f(z)$, donde $f(z)$ es un polinomio irreducible de grado n. De manera general $\mathbb{F}_{p^n}$ se puede representar como:

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

Definición A.9 (Cerradura algebraica de un campo finito $\mathbb{F}_p$). *Sea p un número primo, la cerradura algebraica del campo finito $\mathbb{F}_p$, denotada por $\overline{\mathbb{F}}_p$, es el conjunto infinito de todas sus extensiones, es decir,*

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

A.4 Torres de Campo

Se conoce como torres de campo a las construcciones formadas por campos finitos, donde cada campo finito representa la extensión de los inferiores. Por ejemplo, sea $\mathbb{F}_{p^m} = \mathbb{F}_p[z]/f(z)$ una extensión de campo de $\mathbb{F}_p$, está 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 $k|m$.

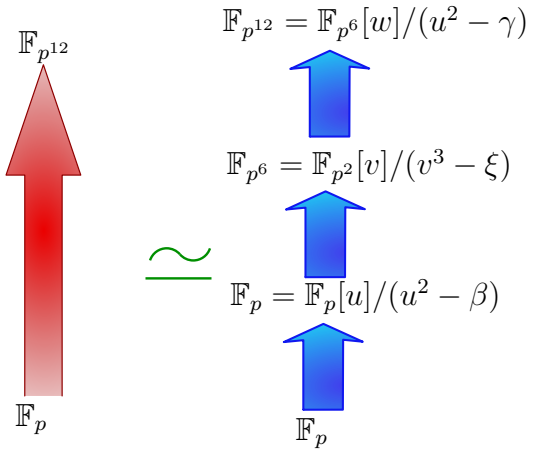


Figura A.1: Torres de Campo

Los niveles que conforman las torres de campo depende de la extensión que se requiera. No existe una manera única de construir una torre, sin embargo, la manera en que es construida puede proporcionar mayor eficiencia en las operaciones. Particularmente, se dice que un campo finito $\mathbb{F}_{p^m}$ es *amable* con los emparejamientos si se usan extensiones cuadráticas y cúbicas para la construcción de la torre, logrando un alto nivel de eficiencia en cada extensión.

Por motivos de implementación, en este trabajo de tesis se realizó la construcción de la torre de campo $\mathbb{F}_{p^{12}}$ ilustrada en la figura A.1. En [Aranha et al., 2010] se describen las fórmulas usadas en este trabajo para el computo de emparejamientos.

A.5 Grupo ciclotómico

Definición A.10 (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*

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

El conjunto de las raíces n-ésimas de la unidad es denotado como μ_n, y forma un grupo cíclico

$$\mu_n = (\mu_n, \cdot, 1).$$

Definición A.11 (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 $\mu_n = \langle z^* \rangle$. Sea $\varphi(\cdot)$ la función indicatriz de Euler, μ_n tiene φ raíces primitivas.*

Definición A.12 (Polinomio ciclotómico). *Se define el n-ésimo polinomio ciclotómico $\Phi_n(z)$ de grado $\varphi(n)$ de la siguiente manera:*

$$\Phi_n(z) = \prod_{l=0}^{\varphi(n)-1} (z - z_l^*)$$

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

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

Definición A.13 (Grupo ciclotómico). *Sea p un número primo y $\mathbb{F}_{p^n}^*$ el grupo multiplicativo de un campo finito de característica p, el n-ésimo grupo ciclotómico $\mathbb{G}_{\Phi_n(p)}$ es un subgrupo de $\mathbb{F}_{p^n}^*$ definido por:*

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

A.6 Morfismos

Un *morfismo* es un proyección entre dos estructuras matemáticas. Existen diferentes tipos de morfismos:

- *Monomorfismos*: Un monomorfismo de X a Y está denotado como $f : X \to Y$. Para todos los morfismos $g_1, g_2 : Z \to X$, se cumple que $f \circ g_1 = f \circ g_2$.
- *Isomorfismo*: Se dice que $f : X \to Y$ es un isomorfismo, si existe un morfismo $g : Y \to X$.
- *Endomorfismo*: Es un morfismo de un objeto matemático a sí mismo, es decir, $f : X \to X$.
- *Automorfismo*: Un automorfismo es un endomorfismo que también es un isomorfismo.

En el ámbito del álgebra los morfismos se conocen como homomorfismos.

A.7 Eigenespacio

El concepto de *eigenespacio* es utilizado en la definición de emparejamientos. por lo tanto, a continuación se muestra una definición general e informal de dicho concepto.

Los vectores propios, autovectores o *eigenvectores* de un operador lineal son los vectores no nulos que, cuando son transformados por el operador, dan lugar a la multiplicación de sí mismos por un escalar λ, el cual es llamado valor propio o *eigenvalor*. El *eigenespacio*-λ de un operador lineal, es el conjunto de eigenvectores con valor propio λ.

A.8 Curvas Elípticas

Una curva elíptica E sobre un campo $\mathcal{K}$, se define como la gráfica correspondiente con la ecuación de Weierstrass [Hankerson et al., 2004] (en coordenadas afines)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{A.1}$$

donde las constantes $a_1, a_2, a_3, a_4, a_6 \in \mathcal{K}$. Se escribe $E/\mathcal{K}$ para enfatizar que E esta definida sobre $\mathcal{K}$. Cabe mencionar que si E esta definida sobre $\mathcal{K}$, entonces E esta definida también sobre cualquier extensión de $\mathcal{K}$.

Si la característica del campo es distinta de 2 y de 3, usando transformaciones lineales de las variables la ecuación de la curva se puede expresar como la ecuación reducida de Weierstrass:

$$E' : y^2 = x^3 + ax + b \tag{A.2}$$

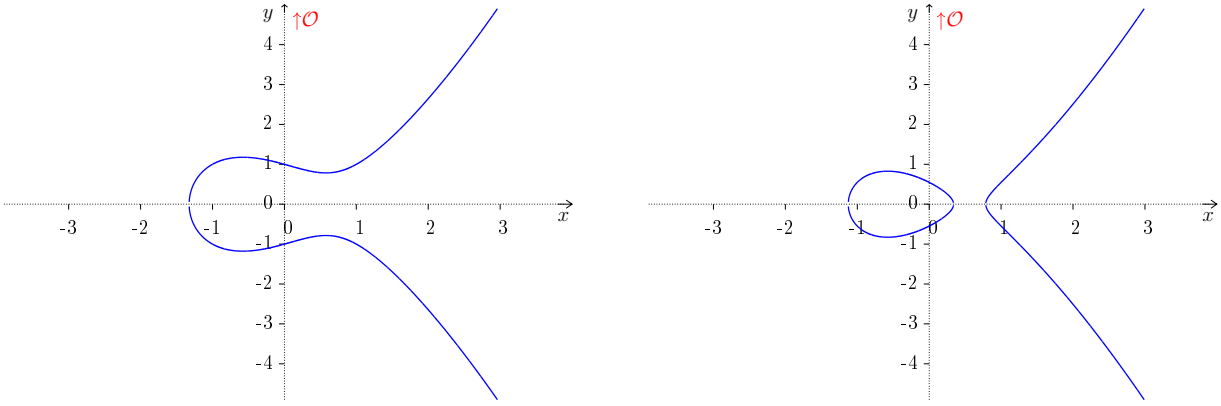
donde a y b son constantes en $\mathcal{K}$ y para este caso serán enteros módulo un primo grande. En la figura A.2 podemos ver las curvas $E_1 : y^2 = x^3 - x + 1$ y $E_2 : y^2 = x^3 - x$ definidas sobre $\mathbb{R}$.

Definición A.14 (Punto al infinito). *El punto al infinito correspondiente a (∞, ∞), es llamado punto al infinito y se denota con $\mathcal{O}$. Dicho punto se encuentra situado en el extremo inferior y superior de las ordenadas, de manera que cualquier línea vertical interseca a $\mathcal{O}$.*

Sea $E/\mathcal{K}$ la curva elíptica definida con la ecuación A.2 y $\mathcal{K}'$ cualquier extensión de $\mathcal{K}$ el conjunto de los $\mathcal{K}$-puntos racionales se denota como:

$$E(\mathcal{K}) = \{(x, y) \in \mathcal{K}' \times \mathcal{K}' : y^2 - x^3 - ax - b = 0\} \cup \{\mathcal{O}\} \tag{A.3}$$

donde los $\mathcal{K}'$-puntos racionales en E son los puntos (x, y) que satisfacen la ecuación de la curva



(a) $E_1 : y^2 = x^3 - x + 1$ (b) $E_2 : y^2 = x^3 - x$

Figura A.2: Curvas elípticas sobre $\mathbb{R}$

y cuyas coordenadas x y y pertenecen a $\mathcal{K}'$. El punto al infinito $\mathcal{O}$ es considerado como $\mathcal{K}'$-punto racional para todas las extensiones de $\mathcal{K}'$ de $\mathcal{K}$.

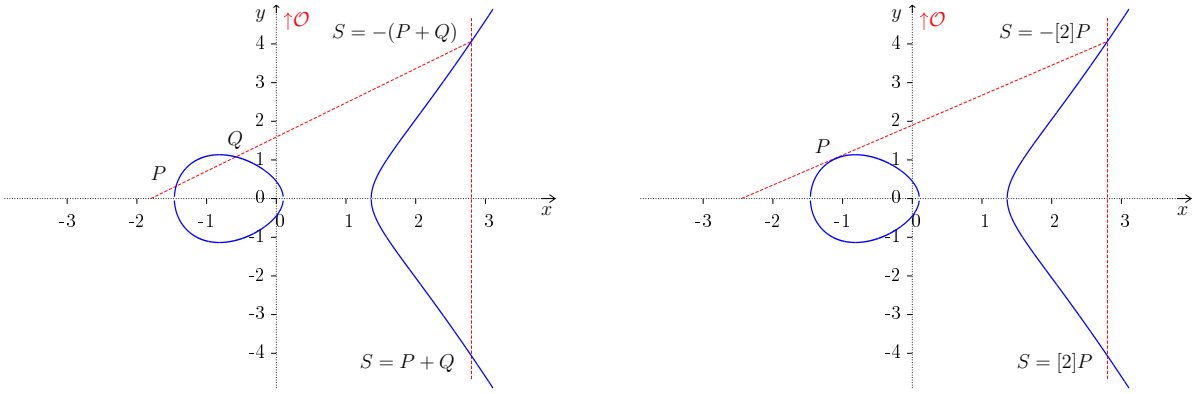
A.9 Ley de Grupo

Dada la curva elíptica $E/\mathcal{K}$, el conjunto de $\mathcal{K}$-puntos racionales forma un grupo abeliano de notación aditiva cuyo elemento identidad corresponde al punto en el infinito $\mathcal{O}$. La notación $E(\mathcal{K})$ es comúnmente usada para denotar a dicho grupo.

Suma de puntos.

Una manera de explicar esta operación es mediante su interpretación geométrica. Dada la curva elíptica $E/\mathcal{K}$ y los puntos $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, donde $P, Q \in E(\mathcal{K})$, la suma de dos puntos sobre una curva elíptica consiste en considerar la recta secante r (véase figura A.3(a)) que pasa por los dos puntos P y Q y se calcula el tercer punto de intersección R de la recta r con la curva. El punto $P + Q$ es el punto intersección de la curva con la recta que pasa por R y el punto en el infinito, es decir, la recta que pasa por R y es paralela al eje de las ordenadas.

En el caso que los puntos P y Q coincidan ($P = Q$) se toma la tangente a la curva en el punto y se realiza la suma de la manera descrita anteriormente, a este proceso se le conoce como el doblado de un punto (véase figura A.3(b)).



(a) Suma (b) Doblado

Figura A.3: Operaciones en una curva elíptica en los reales $\mathbb{R}$

Dada la curva elíptica $E/\mathcal{K}$, donde, $E : y^2 = x^3 + ax + b$ y los puntos $P, Q \in E(\mathcal{K})$, omitiremos la interpretación geométrica para el resto del presente trabajo de investigación, quedando el cálculo de $P + Q \in E(\mathcal{K})$ de la siguiente manera:

I. *Elemento identidad*: $P + \mathcal{O} = \mathcal{O} + P = P$.

II. *Negativos*: Dado $P = (x, y)$, entonces $(x, y) + (x, -y) = \mathcal{O}$. El punto $(x, -y)$ se denota como $-P$ y se denomina como negativo de P o el inverso de P.

III. *Suma de puntos*: Sea $P_1 = (x_1, y_1), P_2(x_2, y_2)$, donde $P, Q \in E(\mathcal{K})$ y $P \neq \pm Q$, entonces $P + Q = (x_3, y_3)$ tal que:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \qquad \text{y} \qquad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \tag{A.4}$$

IV. *Doblado de puntos*: Sea $P = (x_1, y_1)$, donde $a \in \mathbb{Z}$, $P \in E(\mathcal{K})$ y $P \neq -P$, entonces $[2]P = (x_3, y_3)$ tal que:

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - x_1 - x_2 \qquad \text{y} \qquad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \tag{A.5}$$

Podemos deducir las ecuaciones de la suma y el doblado de puntos partiendo de la ecuación de la recta $\ell_{P,Q}$ definida como:

$$y = m(x - x_P) + y_P, \text{ donde } m = \frac{y_Q - y_P}{y_Q - x_Q}, \tag{A.6}$$

se calcula la intersección de $\ell_{P,Q}$ en $E/\mathcal{K}$, igualando la ecuación A.6 con la ecuación de la curva elíptica, es decir:

$$(m(x - x_P) + y_P)^2 = x^3 + ax + b. \tag{A.7}$$

Desarrollando la ecuación A.7 obtenemos:

$$x^3 - m^2x^2 + (a + 2x_Pm^2 - 2y_Pm)x + (b - x_P^2m^2 + 2x_Py_Pm - y_P^2) = 0. \tag{A.8}$$

Para un polinomio de grado 3 se satisface la siguiente igualdad:

$$(x - a)(x - b)(x - c) = x^3 - (a + b + c)x^2 + (ac + ab + bc)x + abc \tag{A.9}$$

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

En este caso conocemos dos de las raíces, las cuales corresponden a los dos primeros puntos, entonces el problema consiste en encontrar la tercera raíz del polinomio. A partir de las ecuaciones A.8 y A.9 que $m^2 = (x_1 + x_2 + x_3)$. Si despejamos el punto x_3 y sustituimos en la ecuación A.7 obtenemos el punto S'. Finalmente si reflejamos este punto, obtenemos las ecuaciones mostradas en la ley de grupo.

Para el caso del doblado de puntos tenemos que $P = Q$, por lo que la pendiente m de la ecuación de la recta tangente $\ell_{P,P}$ se calcula a través de la derivada de la ecuación de la curva elíptica evaluada en el punto P.

A.10 Curvas elípticas sobre campos finitos

Si tenemos que E está definida sobre un campo finito $\mathbb{F}_q$, entonces las puntos en E serán los pares de coordenadas (x, y), tales que $x, y \in \mathbb{F}_q$ satisfagan la ecuación de la curva elíptica. Además tal curva será denotada como $E(\mathbb{F}_q)$. A continuación se describen algunas propiedades del grupo $E(\mathbb{F}_q)$, las cuales son importantes dentro del contexto de la criptografía.

A.10.1 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 el 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], que dice:

Teorema A.1 (Hasse). *Sea E una curva elíptica definida sobre el campo $\mathbb{F}_q$, entonces*

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

Alternativamente, se puede escribir $\#E(\mathbb{F}_q) = q + 1 - t$, 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 $\#E(\mathbb{F}_q) \approx q$.

A.10.2 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 $[k]P = \mathcal{O}$.

La operación $[k]P$ es conocida como *multiplicación escalar de punto* la cual se detalla en la sección 3.1. Es importante destacar que el orden de un punto siempre divide al orden de la curva, en virtud del teorema de Lagrange.

A.10.3 Puntos de torsión

Dada una curva elíptica $E(\mathbb{F}_p)$, sea $E(\overline{\mathbb{F}}_p)$ la cerradura algebraica (definición A.9) de $\mathbb{F}_p$ y $r \in \mathbb{Z}^+$. Se define el conjunto de puntos de torsión r de $E(\overline{\mathbb{F}}_p)$, denotado como $E(\overline{\mathbb{F}}_p[r])$, como el conjunto de puntos en $E(\overline{\mathbb{F}}_p)$ cuyo orden es r:

$$E(\overline{\mathbb{F}}_p)[r] = \{P \in \overline{\mathbb{F}}_p)[r] \mid [r]P = \mathcal{O}\} \tag{A.10}$$

Es importante destacar que para propósitos criptográficos r se toma como un factor primo de $E(\overline{\mathbb{F}}_p)$ grande.

A.10.4 Grado de encajamiento

Para dos números primos p y r, dado un campo finito $\mathbb{F}_p$, considérese una curva elíptica $E/\mathbb{F}_p$ tal que $\#E(\mathbb{F}_p) = h \cdot r$, donde $h \in \mathbb{Z}^+$. Sea k un entero positivo, se dice que k es el grado de encajamiento de $E/\mathbb{F}_p$ con respecto a p y r, si k es el menor entero positivo tal que:

$$r|p^k - 1 \tag{A.11}$$

Sea $\Phi_k(\cdot)$ el k-ésimo polinomio ciclotómico, por definición se cumple que $\Phi_k(p)|p^k - 1$ y por lo tanto $r|\Phi_k(p)$. Dado que $p \equiv t - 1 \bmod r$, donde t es la traza de E sobre $\mathbb{F}_p$, alternativamente el grado de encajamiento puede ser definido como el menor entero positivo k, tal que:

$$r|\Phi_k(t-1). \tag{A.12}$$

A.10.5 Curva enlazada

Definición A.15 (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 A.16 (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, entonces 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:

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

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

A.10.6 Endomorfismo de Frobenius

Definición A.17 (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:

$$\pi^2(Q) - t\pi(Q) + pQ = \mathcal{O}. \tag{A.13}$$

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

$$\sigma(u) = (u - 1)(u - p) \bmod r. \tag{A.14}$$

Por tanto existen dos conjuntos en $E(\mathbb{F}_{p^k})[r]$ definidos como $\{P \in E(\mathbb{F}_{p^k}) | \pi(P) = P\}$ y $\{Q \in E(\mathbb{F}_{p^k}) | \pi(Q) = pQ\}$ [Barreto et al., 2004]. 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 $(x, y) = (x^p, y^p)$.

A.11 Curvas amables con los emparejamientos

En [Freeman et al., 2010] encontramos la definición formal de una curva amable con los emparejamientos.

Definición A.18. *Sea E una curva elíptica ordinaria definida sobre un campo finito $\mathbb{F}_p$. Se dice que E es una curva amable con los emparejamientos [Freeman et al., 2010] si satisface las siguiente condiciones:*

I. Existe un número r tal que $r \geq \sqrt{p}$ y $r | \#E(\mathbb{F}_p)$.

II. El grado de encajamiento k de la curva $E/\mathbb{F}_p$ con respecto a r es menor que $log_2(r)/8$.

Las curvas de este tipo de 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 computa los enteros p, r y t, los cuales deben satisfacer las siguientes características:

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:

$$4p - t^2 = Df^2$$

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

Curvas Barreto-Naehrig.

Las curvas Barreto-Naehrig(BN) [Barreto and Naehrig, 2005] se definen por la ecuación $E : y^2 = x^3 + b$, donde $b = 0$. Las curvas BN definen curvas de orden primo r, con 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.

En esta familia, la curva $E/\mathbb{F}_p : y^2 = x^3 + b$ es isomorfa a la curva $E' = \mathbb{F}_{p^2} : Y^2 = X^3 + b/\xi$, 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.

A.12 Funciones racionales de la curva elíptica

Dada una curva elíptica E definida sobre un campo finito $\mathbb{F}_q$, donde $q = p^n$ y n $\mathbb{Z}^+$, sea $\overline{\mathbb{F}}_q$ la cerradura algebraica de $\mathbb{F}_q$, se dice que $f(x, y)$ es una función racional en $E/\mathbb{F}_q$, si existe un punto $P = (x_P, y_P) \in E(\overline{\mathbb{F}}_q)$, tal que $f(x_P, y_p) \neq \infty$. El conjunto de funciones racionales en $E/\mathbb{F}_q$ está denotado por $\overline{\mathbb{F}}_q(E)$ y para todo $f \in \overline{\mathbb{F}}_q(E)$ se cumple que $f(P)$ es un elemento en el conjunto $\{\overline{\mathbb{F}}_q \cup \infty\}$ [Washington, 2008].

Sean P un punto en la curva elíptica $E/\mathbb{F}_q$ y una función racional de $f \in \overline{\mathbb{F}}_q(E)$, se dice que f tiene un *cero* en P si $f(P) = 0$. De igual 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 infinito.

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

$$f(P) = (u(P))^m . g(P)$$

donde $m \in \mathbb{Z}$, $u(P) = 0$ y $g(P) \neq \{0, \infty\}$. Por lo que si $m > 0$ entonces f tiene un cero en P y si $m < 0$ entonces f tiene un polo en P. El número m es el orden de f en P denotado como:

$$\mathrm{ord}_P(f) = m$$

A.13 Divisores

Sea $E/\mathbb{F}_q$ una curva elíptica, a cada punto $p \in E(\overline{\mathbb{F}}_q)$ se le asigna el símbolo formal $[P]$. Un divisor $\mathcal{D}$ sobre $E/\mathbb{F}_q$, es una combinación lineal finita de dichos símbolos con coeficientes en $\mathbb{Z}$ [Washington, 2008].

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

Entonces, un divisor es un elemento del grupo abeliano generado por los símbolos $[P]$. El grupo de divisores se denota como $\mathrm{Div}(E)$. Los operadores que definen a un divisor son el *soporte*, el *grado* y la *suma*, los cuales se defines de la siguiente manera:

- Soporte:
$$\mathrm{supp}(\mathcal{D}) = \{P \in E | a_j \neq 0\}$$

- Grado:
$$\deg(\mathcal{D}) = \sum_j a_j \in \mathbb{Z}$$

- Suma:
$$\mathrm{sum}(\mathcal{D}) = \sum_j a_j P_j \in E \in (\overline{\mathbb{F}}_p)$$

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 $\mathrm{Div}(E)$, el cual se denota como $\mathrm{Div}^0(E)$.

Dividores principales

Dada una curva elíptica $E/\mathbb{F}_q$ con $\deg(\mathcal{D}) = 0$ y $\mathrm{sum}(\mathcal{D}) = \mathcal{O}$, es llamado *principal* si existe

una función racional $f \in \overline{\mathbb{F}}_q(E)$, tal que $\mathcal{D}$ =div(f), donde

$$\text{div}(f) = \sum_{P_j \in E} \text{ord}_{P_j}(f)[P_j].$$

Las funciones racionales $\ell_{P,P}$ y $\ell_{P,Q}$ (véase la sección A.9), las cuales denotan a la recta tangente a E en el punto P y a la recta secante a E en los puntos P y Q, respectivamente, los divisores principales correspondientes a dichas funciones son:

$$\begin{aligned} \text{div}(\ell_{P,P}) &= 2[P] + [-2P] - 3[\mathcal{O}] \\ \text{div}(\ell_{P,Q}) &= [P] + [Q] - [-(P+Q)] - 3[\mathcal{O}]. \end{aligned}$$

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

- div$(f \cdot g)$ = div(f)+ div(g).
- div(f/g) = div(f)− div(g).
- La función f es una constante, sí y sólo si div$(f) = 0$.

Una función racional f puede ser evaluada en un divisor $\mathcal{D} = \sum_j a_j[P_j]$, a través de la siguiente fórmula:

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

de tal manera que para todo $n \in \mathbb{Z}$

$$f(\mathcal{D})^n = f(n\mathcal{D}),$$

donde $n\mathbb{D} = \sum_j n \cdot a_j[P_j]$.

B

Palabras reservadas

Las palabras reservadas son identificadores predefinidos reservados que tienen significados especiales y no se pueden utilizar como identificadores de una descripción de un protocolo. Las palabras reservadas de otros lenguajes serán igualmente reservadas para nuestra propuesta. Este apéndice enlista todas ellas.

B.1 Lenguaje C

auto	const	double	float	int	short	struct	unsigned
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static	union	while

B.2 Lenguaje C++

asm	class	friend	or	static_cast	typeid
and	compl	inline	or_eq	template	typename
and_eq	const_cast	mutable	operator	this	using
bitand	delete	namespace	private	throw	virtual
bitor	dynamic_cast	new	public	true	wchar_t
bool	explicit	not	protected	try	xor_eq
catch	false	not_eq	reinterpret_cast		xor

B.3 Lenguaje Magma

_	cmpeq	error	if	ne	repeat	true
adj	cmpne	eval	iload	not	require	try
and	continue	exists	import	notadj	requirege	until
assert	declare	exit	in	notin	requirerange	vprint
assert2	default	false	intrinsic	notsubset	restore	vprintf
assert3	delete	for	is	or	return	vtime
assigned	diff	forall	join	print	save	when
break	div	forward	le	printf	sdiff	where
by	do	fprintf	load	procedure	select	while
case	elif	freeze	local	quit	subset	xor
cat	else	function	lt	random	then	
catch	end	ge	meet	read	time	
clear	eq	gt	mod	readi	to	

B.4 Lenguaje LENPROC

tString	tPointG1	tPointG2	tPointGT	tScalar
r_u	r_Fp	r_bp	r_bp6	r_EextK
r_p	r_b	r_Etwist	r_ExtSix	r_np
r_r	r_E	r_xi	r_Fq12	
r_t	r_Fq2	r_Fq6	r_bp12	
PROTOCOL	VARIABLE	DESCRIPTION	SECTION	BEGIN
END	for	to	do	end_for
if	then	else	fi	Print
Preview	E	H	H1	H2
Random	Generator	Inverse	Summation	MultiPairing
Xor	And	Or	Print	

C

Gramática de LENPROC

C.1 Estructura gramatical

C.1.1 Cuerpo del Protocolo

protocolo :
    protocol_definicion EOF;

protocol_definicion :
    PROTOCOL protocol_nombre BEGIN variable_seccion_definicion description_seccion_definicion END;

protocol_nombre :
    identificador_definicion;

C.1.2 Sección para declaración de variables

variable_seccion_definicion :
    variable_declaracion+
    | WS;

```
variable_nombre :
        variable_un_nombre
      | variable_varios_nombres ;


variable_declaracion :
        variable_tipo variable_nombre PC;


variable_varios_nombres :
        variable_un_nombre (COMA variable_un_nombre)+;


variable_un_nombre :
        identificador_definicion
      | identificador_definicion_array;


variable_tipo :
        Tipo_NUMERO
      | Tipo_CADENA
      | Tipo_PUNTOG1
      | Tipo_PUNTOG2
      | Tipo_PUNTOGt
      | Tipo_PUNTO ;
```

C.1.3 Sección para descripción del protocolo

```
description_seccion_definicion :
        DESCRIPTION LLI section_definicion+ LLD ;


section_definicion :
        SECTION section_nombre LLI section_pasos LLD ;


section_nombre :
        identificador_definicion ;


section_pasos :
        statement_definicion+;
```

C.2 Sentencias

statement_definicion :
    statement PC ;

statement :
    expresion_statement
    | assigment_statement
    | if_statement_definicion
    | for_expresion_definicion ;

expresion_statement :
    expresion_definicion ;

assigment_statement :
    assigment_expresion_definicion ;

solo_expresion_definicion :
    expresion_definicion ;

assigment_expresion_definicion :
    primary_expression operador_asignacion assigment_expresion ;

assigment_variable :
    unary_expression ;

assigment_expresion :
    expresion_definicion ;

operador_asignacion :
    OP_ASIGNACION ;

for_expresion_definicion :
    FOR PI for_condicion_desde PD TO for_condicion_hasta DO statement_definicion+ END_FOR ;

for_condicion_desde :
    identificador_definicion OP_ASIGNACION unary_expression ;

for_condicion_hasta :
unary_expression ;

funcion_expression_definicion :
print_definicion
| preview_definicion
| summation_definicion
| primitiva_criptrografica ;

print_definicion :
print_funcion parentesis_izq unary_expression parentesis_der ;

print_funcion :
OP_F_PRINT ;

preview_definicion:
preview_funcion parentesis_izq unary_expression parentesis_der ;

preview_funcion :
OP_F_PREVIEW ;

if_statement_definicion :
IF PI if_condicion PD THEN statement_definicion+ (ELSE statement_definicion+)? FI ;

if_condicion :
statement;

C.3 Defincián de las expresiones

expresion_definicion :
and_logico_definicion ;

and_logico_definicion :
and_operando (and_operador and_operando)* ;

```
and_operando :
        xor_logico_definicion ;

xor_logico_definicion :
        or_logico_operando (or_operador or_logico_operando)* ;

or_logico_operando :
        or_comparacion_definicion ;

or_comparacion_definicion :
                    or_comparacion_operando          (or_comparacion_operador
or_comparacion_operando)? ;

or_comparacion_operando :
        and_comparacion_definicion ;

and_comparacion_definicion :
                  and_comparacion_operando          (and_comparacion_operador
and_comparacion_operando)? ;

and_comparacion_operando :
        equivalencia_expresion_relacion ;

equivalencia_expresion_relacion :
        equivalencia_operando (equivalencia_operador equivalencia_operando)? ;

equivalencia_operando :
        relacion_expresion_definicion ;

relacion_expresion_definicion :
        relacion_operando (relacion_operador relacion_operando)? ;

relacion_operando :
        cocatenacion_expresion_definicion ;

cocatenacion_expresion_definicion :
        cocatenacion_operando (cocatenacion_operador cocatenacion_operando)* ;
```

```
cocatenacion_operando :
        additive_expresion_definicion ;

additive_expresion_definicion :
        additive_operando (additive_operador additive_operando)* ;

additive_operando :
        multiplicative_expresion_definicion ;

multiplicative_expresion_definicion :
                    multiplicative_multiplicando              (multiplicative_operador
multiplicative_multiplicando)* ;

multiplicative_multiplicando :
        exponentiation_expresion_definicion ;

exponentiation_expresion_definicion :
                     exponentiation_operando              (exponentiation_operador
exponentiation_operando)* ;

exponentiation_operando :
        unary_expression ;

unary_expression :
        primary_expression
      | funcion_expression_definicion
      | parentesis_izq! expresion_definicion parentesis_der! ;

primary_expression :
        operando_identificador
      | operando_numero
      | operando_string ;
```

C.4 Operadores

```
additive_operador :
        OP_A_ADD
```

```
        | OP_A_LESS ;

multiplicative_operador :
        OP_A_MUL
        | OP_A_DIV ;

exponentiation_operador :
        OP_A_EXP ;

xor_operador :
        OP_L_XOR ;

and_operador :
        OP_L_AND ;

or_operador :
        OP_L_OR ;

or_comparacion_operador :
        OP_C_OR ;

and_comparacion_operador :
        OP_C_AND ;

relacion_operador :
        OP_C_LT
        | OP_C_GT
        | OP_C_LE
        | OP_C_GE ;

equivalencia_operador :
        OP_C_EQ
        | OP_C_NE ;

cocatenacion_operador :
        OP_F_CONCATENA ;
```

C.5 Primitivas criptográficas

primitiva_criptrografica :
pairing_definicion
| multipairing_definicion
| scalarMultiplication_definicion
| hash_definicion
| mapToPointG1_definicion
| mapToPointG2_definicion
| randomNumber_definicion
| inverse_definicion
| generatorElement_definicion ;

pairing_definicion :
pairing_funcion parentesis_izq pairing_parametro_G1 separador pairing_parametro_G2 parentesis_der ;

multipairing_definicion :
multipairing_funcion parentesis_izq pairing_parametro_G1 separador pairing_parametro_G2 separador multipairing_desde separador multipairing_hasta parentesis_der ;

scalarMultiplication_definicion :
CI scalarMultiplication_parametro_escalar CD scalarMultiplication_parametro_punto ;

hash_definicion :
hash_funcion parentesis_izq hash_parametro parentesis_der ;

mapToPointG1_definicion :
mapToPointG1_funcion parentesis_izq mapToPointG1_parametro parentesis_der ;

mapToPointG2_definicion :
mapToPointG2_funcion parentesis_izq mapToPointG2_parametro parentesis_der ;

randomNumber_definicion :
randomNumber_funcion parentesis_izq randomNumber_rangoInferior

separador randomNumber_rangoSuperior parentesis_der ;

generatorElement_definicion :
generatorElement_funcion parentesis_izq generatorElement_parametro parentesis_der ;

summation_definicion :
OP_F_SUMMATION PI summation_elemento separador summation_superior separador summation_inferior PD ;

summation_elemento :
statement ;

summation_superior :
unary_expression ;

summation_inferior :
unary_expression ;

pairing_funcion :
OP_P_PAIRING ;

multipairing_funcion :
OP_P_MULTIPAIRING ;

pairing_parametro_G1 :
expresion_definicion ;

pairing_parametro_G2 :
expresion_definicion ;

scalarMultiplication_parametro_escalar :
expresion_definicion ;

scalarMultiplication_parametro_punto :
unary_expression ;

```
hash_parametro :
        expresion_definicion ;

hash_funcion :
        OP_P_HASH ;

mapToPointG1_parametro :
        expresion_definicion ;

mapToPointG1_funcion :
        OP_P_HASH_G1 ;

mapToPointG2_parametro :
        expresion_definicion ;

mapToPointG2_funcion :
        OP_P_HASH_G2 ;

randomNumber_funcion :
        OP_P_RANDOM ;

randomNumber_rangoInferior :
        expresion_definicion ;

randomNumber_rangoSuperior :
        expresion_definicion ;

inverse_definicion :
         inverse_funcion parentesis_izq inverse_parametro_elemento separador
inverse_parametro_modulo parentesis_der ;

inverse_funcion :
        OP_F_INVERSE ;

inverse_parametro_elemento :
        unary_expression ;
```

```
inverse_parametro_modulo :
        unary_expression ;

generatorElement_parametro :
        unary_expression ;

generatorElement_funcion :
        OP_P_GENERATOR ;

identificador_definicion_array :
        identificador_definicion CI identificador_definicion CD
      | identificador_definicion CI number_definicion CD ;

operando_identificador :
        identificador_definicion
      | identificador_definicion_array
      | palabra_reservada_definicion ;

operando_identificador_array :
        identificador_definicion_array ;

operando_string :
        string_definicion ;

operando_numero :
        number_definicion ;

string_definicion :
        STRING ;

identificador_definicion :
        ID ;

number_definicion :
        INT ;

palabra_reservada_definicion :
        Reserved_u
      | Reserved_p
```

```
        | Reserved_r
        | Reserved_t
        | Reserved_Fp
        | Reserved_b
        | Reserved_E
        | Reserved_Fq2
        | Reserved_bp
        | Reserved_Etwist
        | Reserved_xi
        | Reserved_Fq6
        | Reserved_bp6
        | Reserved_ExtSix
        | Reserved_Fq12
        | Reserved_bp12
        | Reserved_EextK
        | Reserved_np ;

parentesis_izq :
        PI ;

parentesis_der :
        PD ;

separador :
        COMA ;
```

C.6 Elementos léxicos

```
ID :
        (ALPHA)(ALPHA|DIGIT|'_')*;

INT :
        DIGIT+;

STRING :
        ' " '(ALPHA|DIGIT|'\ '|'\t')*' " ';

ALPHA :
        ('a'..'z'|'A'..'Z');
```

```
DIGIT :
        ('0'..'9');

COMMENT:
        '\\' . '\n'
      | '/*' (.)* '*/' ;

WS :
        ( ' ' ; '\t' ; '\r' ; '\n');

PI :
        '(';

PD :
        ')';
```

Bibliografía

[Abadi and Gordon, 1998] Abadi, M. and Gordon, A. D. (1998). A Calculus for Cryptographic Protocols, The Spi Calculus. *Research Report 149*.

[Aho et al., 1998] Aho, A., Sethi, R., Ullman, J., Suárez, P., and López, P. (1998). *Compiladores: Principios, Técnicas y Herramientas*. Addison-Wesley Iberoamericana.

[Aranha et al., 2010] Aranha, D. F., Karabina, K., Longa, P., Gebotys, C. H., and López, J. (2010). Faster explicit formulas for computing pairings over ordinary curves. Cryptology ePrint Archive, Report 2010/526. http://eprint.iacr.org/.

[Barreto et al., 2004] Barreto, P., Lynn, B., and Scott, M. (2004). On the Selection of Pairing-Friendly Groups. In Matsui, M. and Zuccherato, R., editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer Berlin Heidelberg.

[Barreto et al., 2002] Barreto, P. S. L. M., Kim, H. Y., Lynn, B., and Scott, M. (2002). Efficient Algorithms for Pairing-Based Cryptosystems. *IACR Cryptology ePrint Archive*, 2002:8.

[Barreto and Naehrig, 2005] Barreto, P. S. L. M. and Naehrig, M. (2005). Pairing-Friendly Elliptic Curves of Prime Order. In Preneel, B. and Tavares, S. E., editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer.

[Bethencourt et al., 2007] Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-Policy Attribute-Based Encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA. IEEE Computer Society.

[Beuchat et al., 2010] Beuchat, J., González-Díaz, J., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., and Teruya, T. (2010). High-Speed Software Implementation of the Optimal

ate Pairing over Barreto–Naehrig curves. *Pairing-Based Cryptography-Pairing 2010*, pages 21–39.

[Boehm, 1986] Boehm, B. (1986). A Spiral Model of Software Development and Enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24.

[Boldyreva, 2002] Boldyreva, A. (2002). Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In Desmedt, Y., editor, *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin Heidelberg.

[Boneh and Franklin, 2001] Boneh, D. and Franklin, M. (2001). Identity-Based Encryption from the Weil Pairing. In Kilian, J., editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin Heidelberg.

[Boneh et al., 2003] Boneh, D., Gentry, C., Lynn, B., and Shacham, H. (2003). Aggregate and verifiably encrypted signatures from bilinear maps. In Biham, E., editor, *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer Berlin Heidelberg.

[Boneh et al., 2001] Boneh, D., Lynn, B., and Shacham, H. (2001). Short signatures from the Weil pairing. In Colin Boyd, editor, ASIACRYPT, volume 2248 of Lecture Notes in Computer Science, pages 514–532. Springer.

[Bosma et al., 1997] Bosma, W., Cannon, J., and Playoust, C. (1997). The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265. Computational algebra and number theory (London, 1993).

[Castañeda, 2011] Castañeda, L. F. (2011). Estudio y Análisis de Emparejamientos Bilineales Definidos sobre Curvas Ordinarias con Alto Nivel de Seguridad. Master's thesis, Department of Computer Science, Cinvestav, México.

[Choie et al., 2005] Choie, Y., Jeong, E., and Lee, E. (2005). Efficient Identity-Based Authenticated Key Agreement Protocol from Pairings. *Applied Mathematics and Computation*, 162(1):179–188.

[Cohen et al., 2010] Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., and Vercauteren, F. (2010). *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Mathematics and Its Applications. Taylor & Francis.

[Daemen and Rijmen, 2002] Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer.

[Dominguez Perez, 2011] Dominguez Perez, L. J. (2011). *Developing an Automatic Generation Tool for Cryptographic Pairing Functions*. PhD thesis, Dublin City University.

[Dominguez Perez and Scott, 2009] Dominguez Perez, L. J. and Scott, M. (2009). Automatic generation of optimised cryptographic pairing functions. SPEED-CC 2009 Workshop memories. http://www.hyperelliptic.org/SPEED/record09.pdf.

[Dutta et al., 2004] Dutta, R., Barua, R., and Sarkar, P. (2004). Pairing-Based Cryptographic Protocols: A Survey. In *In Cryptology ePrint Archive, Report 2004/064*.

[Freeman et al., 2010] Freeman, D., Scott, M., and Teske, E. (2010). A Taxonomy of Pairing-Friendly Elliptic Curves. *J. Cryptology*, 23(2):224–280.

[Galbraith and Scott, 2008] Galbraith, S. D. and Scott, M. (2008). Exponentiation in Pairing-Friendly Groups Using Homomorphisms. In *Pairing*, Lecture Notes in Computer Science, pages 211–224. Springer.

[Gallant et al., 2001] Gallant, R. P., Lambert, R. J., and Vanstone, S. A. (2001). Faster point multiplication on elliptic curves with efficient endomorphisms. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 190–200, London, UK, UK. Springer-Verlag.

[Goldwasser and Kilian, 1999] Goldwasser, S. and Kilian, J. (1999). Primality testing using elliptic curves. *J. ACM*, 46(4):450–472.

[Hankerson et al., 2004] Hankerson, D., Menezes, A. J., and Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secasus, New Jersey, USA.

[Hess et al., 2006] Hess, F., Smart, N., and Vercauteren, F. (2006). The Eta Pairing Revisited. *Information Theory, IEEE Transactions on*, 52(10):4595–4602.

[Joux, 2004] Joux, A. (2004). A One Round Protocol for Tripartite Diffie-Hellman. *J. Cryptology*, 17(4):263–276.

[Karchmer and Wigderson, 1993] Karchmer, M. and Wigderson, A. (1993). On Span Programs. In *In Proc. of the 8th IEEE Structure in Complexity Theory*, pages 102–111. IEEE Computer Society Press.

[Menezes et al., 1991] Menezes, A., Vanstone, S., and Okamoto, T. (1991). Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 80–89, New York, NY, USA. ACM.

[Menezes et al., 1996] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[Miller, 2004] Miller, V. S. (2004). The Weil Pairing, and Its Efficient Calculation. *J. Cryptol.*, 17(4):235–261.

[Mitsunari et al., 2002] Mitsunari, S., Sakai, R., and Kasahara, M. (2002). A New Traitor Tracing. *EICE Trans Fundam Electron Commun Comput Sci (Inst Electron Inf Commun Eng)*, E85-A:481–484.

[NIST, 2008] NIST (2008). Federal Information Processing Standards Publications (FIPS Pubs). http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[Paar and Pelzl, 2010] Paar, C. and Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag New York Inc.

[Parr and Quong,] Parr, T. J. and Quong, R. W. ANTLR: A Predicated-LL(k) Parser Generator,.

[Pozza et al., 2004] Pozza, D., Sisto, R., and Durante, L. (2004). Spi2java: Automatic Cryptographic Protocol Java Code Generation from Spi Calculus. *Proceedings of the International Conference on Advanced Information Networking and Applications*, pages 400–405.

[Sakai et al., 2000] Sakai, R., Ohgishi, K., and Kasahara, M. (2000). Cryptosystems Based on Pairing. Symposium on Cryptography and Information Security (SCIS2000), pages 26–28. Okinawa, Japan.

[Scott, 2003] Scott, M. (2003). MIRACL, Multiprecision Integer and Rational Arithmetic C/C++ Library. SIAM J. Comput. 32(3):586–615, 2003.

[Shamir, 1985] Shamir, A. (1985). Identity-Based Cryptosystems and Signature Schemes. In Blakley, G. and Chaum, D., editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Berlin Heidelberg.

[Shoup, 2005] Shoup, V. (2005). *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, New York, NY, USA.

[Stallings, 2002] Stallings, W. (2002). *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition.

[Sánchez Ramírez, 2012] Sánchez Ramírez, A. H. (2012). Implementación de la criptografía basada en atributos en un dispositivo móvil. Master's thesis, CINVESTAV, Depatamento de Computación.

[Trappe and Washington, 2006] Trappe, W. and Washington, L. (2006). *Introduction to Cryptography: With Coding Theory*. Pearson Prentice Hall.

[Vercauteren, 2008] Vercauteren, F. (2008). Optimal pairings. *IACR Cryptology ePrint Archive*, 2008:96.

[Vieites, 2006] Vieites, Á. (2006). *Enciclopedia de la Seguridad Informática*. RA-MA S.A. Editorial y Publicaciones.

[Wang et al., 2009] Wang, M., Wei, P., Zhang, H., and Zheng, Y. (2009). Optimal Pairing Revisited. Report 2009/564.

[Washington, 2008] Washington, L. C. (2008). *Elliptic Curves: Number Theory and Cryptography, Second Edition*. Chapman & Hall/CRC, 2da. edition.

[Weil, 1948] Weil, A. (1948). *Variétés abéliennes et courbes algébriques*. Paris : Hermann.