Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Computación

Implementación eficiente de prueba de suavidad para polinomios

Tesis que presenta

Isaac Andrés Canales Martínez

para obtener el grado de

Maestro en Ciencias en Computación

Director de la tesis:

Dr. Francisco José Rambó Rodríguez Henríquez

México, D.F. Diciembre, 2015

Agradecimientos

Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, particularmente al Departamento de Computación. Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico durante estos dos años de estudios.

Resumen

Un problema se considera difícil de resolver si no se conoce ningún algoritmo que lo solucione con complejidad polinomial en tiempo. Por ejemplo, dado un grupo cíclico con orden suficientemente grande, la exponenciación es un cálculo sencillo, mientras que su operación inversa, el logaritmo discreto, se cree que es un cálculo difícil. Una función de un solo sentido (*one-way function* en inglés) es una función fácil de calcular pero difícil de invertir, y el logaritmo discreto se clasifica como una función de este tipo. Las funciones de un solo sentido se utilizan en criptografía de llave pública; el protocolo Diffie-Hellman de intercambio de llaves y el criptosistema ElGamal, son ejemplos del uso del logaritmo discreto.

Actualmente, el algoritmo estándar para calcular el logaritmo discreto, es el algoritmo de cálculo de índices. La complejidad de este algoritmo es sub-exponencial en tiempo y consta de cuatro fases: (i) base de factores, (ii) generación de relaciones, (iii) álgebra lineal y (iv) descenso. Desde que Leonard Adleman propuso la primer versión del algoritmo en 1979, se han presentado diversas mejoras en las fases para reducir su complejidad total. La fase de descenso se ejecuta por pasos, y la operación más costosa en cada paso es determinar si un polinomio es *suave* respecto a un grado establecido. Para lo anterior, se tiene la opción de factorizar al polinomio en cuestión, o la de calcular una prueba de suavidad. La segunda estrategia resulta ser más eficiente en términos computacionales.

Este trabajo está enfocado en la prueba de suavidad para polinomios, con el obejtivo principal de realizar una implementación eficiente de dicha prueba. Para ello, se escogió un campo finito que contiene un subgrupo multiplicativo de interés criptográfico, y se calculó el logaritmo discreto en este grupo. Específicamente, la implementación de la prueba de suavidad se utilizó en los pasos de descenso por fracciones continuas y descenso clásico. Actualmente, no se han reportado resultados del cálculo del logaritmo discreto en el campo elegido, y nuestro propósito es establecer una referencia en tiempo del esfuerzo requerido para este cálculo.

Abstract

A problem is considered to be difficult if there is no known algorithm that solves it with polynomial time complexity. For example, given a cyclic group whose order is sufficiently large, exponentiation is a rather simple computation, whereas its inverse operation, i.e. the discrete logarithm, is believed to be a difficult one. A *one-way function* is a function that is easy to compute, but difficult to invert, and the discrete logarithm is classified as a function of this kind. One-way functions are widely used within public key cryptography; the Diffie-Hellman key exchange protocol and the ElGamal cryptosystem, are examples of the usage of the discrete logarithm.

Nowadays, the standard algorithm to compute the discrete logarithm is the index-calculus algorithm. This algorithm has sub-exponential time complexity and consists of four phases: (i) factor base, (ii) generation of relations, (iii) linear algebra and (iv) descent. Since Leonard Adleman proposed the first version of this algorithm in 1979, there have been many improvements in its phases, aiming to reduce the total complexity of the whole algorithm. The descent phase is performed in steps, and the most complex operation in these steps is to determine whether a polynomial is *smooth* with respect to a given degree. For this, one can either factorize the polynomial or perform a smoothness test on it. The latter approach is more efficient from a computational perspective.

This work focuses on the smoothness test for polynomials, and the main objective is to develop an efficient implementation of this test. We chose a finite field that contains a multiplicative subgroup of cryptographic interest, and we computed the discrete logarithm in that group. Specifically, the smoothness test implementation was used in the continued-fraction and classical steps of the descent phase. At the time of writing this manuscript, there are no reported results on the computation of the discrete logarithm for the chosen field, and our purpose is to establish a time reference of the effort required for this computation.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

CAPÍTULO 1

Introducción

Desde hace muchos años ha existido la necesidad de intercambiar información, en algunas ocasiones esta información es muy importante para quienes la envían y reciben, y es de suma importancia garantizar que nadie más tenga acceso a ella. Un problema sustancial es que los canales de comunicación y medios de almacenamiento son, en su mayoría, inseguros, ya que un adversario (una entidad distinta al remitente y destinatario) puede de alguna manera acceder a la información, y en algunos casos incluso modificarla o destruirla.

La *seguridad* es un concepto que no tiene una definición concreta. Esto se debe a que las características que hacen segura a una aplicación, no necesariamente son las mismas características que hacen segura a otra. Por ejemplo, en una aplicación A, la seguridad puede significar que la información solo sea accesible para ciertos usuarios, mientras que para una aplicación B, la seguridad significa garantizar que un usuario es realmente quien dice ser. Sin embargo, la seguridad se puede definir en términos de los siguientes *servicios de seguridad*:

- *Confidencialidad*. Garantizar que la información solo puede ser accedida y manipulada por las entidades autorizadas.

- *Integridad*. Asegurar que la información no sea modificada por alguna entidad no autorizada.
- *Autenticación*. Garantizar que una entidad es quien dice ser.
- *No Repudio*. Evitar que una entidad niegue alguna acción o compromiso efectuado.
- *Disponibilidad*. Asegurar que la información este disponible en todo momento a un nivel adecuado de desempeño.

La *criptografía* provee herramientas para proporcionar seguridad cuando se transmite información a través de un canal de comunicación inseguro. Inicialmente, la criptografía fue empleada en el contexto militar y diplomático. Actualmente, debido al gran desarrollo tecnológico en los últimos años, existe un gran número de aplicaciones que hacen uso de la criptografía. Entre las aplicaciones más importanes se pueden mencionar el cifrado de datos, firmas digitales, dinero electrónico y voto electrónico.

En sus inicios, la criptografía se utilizaba con el propósito de establecer comunicación confidencial. Para esto, las entidades que se comunican establecen un secreto compartido, denominado *llave*. La información que se desea enviar se denomina *mensaje en claro*, el cual se *cifra* con la llave, es decir, se obtiene un *mensaje cifrado* que contiene de manera incomprensible la misma información que el mensaje en claro. El mensaje cifrado es enviado a través del canal inseguro, por lo tanto un adversario no es capaz de comprender su contenido. El destinatario utiliza la llave para *descifrar* el mensaje y obtener el mensaje en claro. El esquema anterior se denomina *criptografía simétrica* o *criptografía de llave secreta*. Un inconveniente de la criptografía simétrica, es que las entidades deben acordar la llave antes de intercambiar mensajes, y en la práctica esto puede ser complicado.

En 1976, Whitfield Diffie y Martin Hellman [13] presentan la *criptografía asimétrica* o *criptografía de llave pública*. En este esquema, cada entidad cuenta con un par de llaves, una *llave privada* y una *llave pública*. La idea principal es que las llaves públicas de cada entidad sean, como su nombre lo indica, públicas para todos, pero cada entidad debe mantener en secreto su llave privada. Para establecer comunicación, el remitente cifra el mensaje utilizando la llave pública del destinatario, y éste descifra el mensaje utilizando su llave privada. Con este esquema, no hay necesidad de acordar un secreto compartido para comenzar a intercambiar información.

Un problema se considera difícil de resolver, si no se conoce ningún algoritmo que lo solucione con complejidad polinomial en tiempo [12]. La seguridad de los sistemas criptográficos actuales, está basada en la dificultad de resolver algún problema matemático. Por ejemplo, el criptosistema RSA [36] basa su seguridad en la dificultad de factorizar números enteros grandes. El criptosistema ElGamal [14] y el protocolo Diffie-Hellman

de intercambio de llaves [13], basan su seguridad en la dificultad de calcular el logaritmo discreto.

Particularmente, Diffie y Hellman mencionan que la seguridad de su protocolo de intercambio de llaves, depende directamente de la dificultad de calcular logaritmos discretos en estructuras matemáticas denominadas *campos finitos*. También mencionan que su protocolo se volvería inseguro si se llegara a encontrar un algoritmo que solucione este problema con complejidad logarítmica respecto al tamaño del campo [13]. A partir de esto, se han propuesto diversas técnicas y algoritmos para resolver el problema del logaritmo discreto.

Actualmente, el *algoritmo de cálculo de índices* es el algoritmo más eficiente para resolver el logaritmo discreto en campos finitos. Este algoritmo tiene complejidad *subexponencial* en tiempo y consta de cuatro fases: (i) base de factores, (ii) generación de relaciones, (iii) álgebra lineal y (iv) descenso. En 1979, Leonard Adleman presentó la primera versión del algoritmo [3], y desde entonces se han propuesto diversas modificaciones en sus fases, con el objetivo de disminuir la complejidad total.

Particularmente, la fase de descenso se ejecuta por pasos, y la operación más costosa en cada paso es determinar si un *polinomio* es suave respecto a un grado establecido. Para lo anterior, se tiene la opción de *factorizar* al polinomio en cuestión, o la de calcular una *prueba de suavidad*. La segunda estrategia resulta ser más eficiente en términos computacionales.

Objetivos

El objetivo principal de este trabajo es realizar una implementación eficiente de la prueba de suavidad para polinomios. Se desea calcular el logaritmo discreto en un campo finito de interés criptográfico, donde no haya resultados reportados de dicho cálculo, ya que se busca establecer una referencia en tiempo del esfuerzo requerido. Para realizar el cálculo anterior, la implementación de la prueba de suavidad se utilizará en la fase de descenso del algoritmo de cálculo de índices. Los objetivos particulares se mencionan a continuación:

- Analizar distintas estrategias para realizar aritmética en campos finitos y anillos de polinomios.
- Implementar la aritmética en el campo finito de interés y su respectivo anillo de polinomios, para establecer una referencia del tiempo de ejecución de dicha aritmética.
- Implementar la prueba de suavidad para polinomios, y establecer una referencia del tiempo de ejecución para polinomios sobre el campo de interés.

- Implementar los primeros pasos de la fase de descenso del algoritmo de cálculo de índices, y establecer una referencia del tiempo requerido para el campo de interés.

Organización del documento

En el capítulo 2, se presenta el problema del logaritmo discreto y los conceptos matemáticos básicos para estudiarlo y comprenderlo. Asimismo, se presenta de manera general el algoritmo de cálculo de índices. El material de este capítulo está basado principalmente de [38], [33], [32] y [31].

Los polinomios definidos sobre campos finitos son los objetos matemáticos centrales en este trabajo. Realizar aritmética con estos polinomios implica realizar aritmética con sus coeficientes, es decir, elementos del campo finito sobre el cual se definen. En el capítulo 3, primero se presenta la aritmética polinomial, y posteriormente se presenta la aritmética para campos finitos. El contenido de este capítulo se basa principalmente de [40], [38] y [33].

La prueba de suavidad se utiliza en el algoritmo de cálculo de índices para determinar si un polinomio es *suave* con respecto a un grado establecido. La factorización es otra opción para determinar la suavidad de un polinomio, sin embargo, resulta ser más costosa en términos computacionales. En el capítulo 4 se presenta un breve análisis del costo computacional de ambas alternativas.

El campo $\mathbb{F}_{3^{6\cdot 509}}$ es un campo de interés criptográfico [2] del cual no se han reportado resultados del cálculo del logaritmo discreto. En el capítulo 5, se muestra la versión del algoritmo de cálculo de índices, el análisis y los parámetros de diseño para calcular el logaritmo discreto en $\mathbb{F}_{3^{6\cdot 509}}$.

Los detalles de la implementación de la prueba de suavidad se presentan en el capítulo 6. Esta implementación se utiliza para calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$ y se realizó en el lenguaje de programación C. En este capítulo se detalla la aritmética en el campo base, anillo de polinomios y los pasos de la fase de descenso. Finalmente, en el capítulo 7 se presentan las conclusiones y el trabajo futuro.

CAPÍTULO 2

Problema del logaritmo discreto

En teoría elemental de números existe una colección de problemas que se cree que son difíciles de resolver de manera eficiente, es decir, no se conoce ningún algoritmo de tiempo polinomial que los solucione, y son estos los problemas en los que la criptografía descansa para definir primitivas y protocolos encargados de ofrecer seguridad en diversas aplicaciones. Ejemplos de dichos problemas son la factorización entera[1] y el cálculo del logaritmo discreto en grupos cíclicos; este último problema es el tema a tratar a lo largo de este capítulo.

Para estudiar y comprender el problema del logaritmo discreto, primero se presentan las definiciones y características de estructuras matemáticas tales como grupo, campo finito y anillo de polinomios, lo cual se realiza en la sección 2.1. En la sección 2.2 se define formalmente el problema del logaritmo discreto, y la explicación de por qué es usado en criptografía de llave pública se presenta en la sección 2.3. Finalmente, en la sección 2.4 se presenta el algoritmo de cálculo de índices, que es el algoritmo estándar para resolver el problema del logaritmo discreto.

[1] El problema de la factorización entera es aquel que dado un número entero n, se determinen sus factores primos. Por ejemplo, $2015 = 5 \times 13 \times 31$.

2.1. Contexto teórico

En esta sección se presentan los conceptos matemáticos básicos para estudiar y comprender el problema del logaritmo discreto. Referencias generales para esta sección son [38], [33] y [32].

2.1.1. Grupos

Definición 2.1.1 (Operación binaria)**.** Una *operación binaria* $\star$ sobre un conjunto G es una función $\star : G \times G \to G$. Se dice que una operación binaria es *asociativa* si para cualesquiera $a, b, c \in G$ se cumple que $a\star(b\star c) = (a\star b)\star c$. Se dice que una operación binaria es *conmutativa* si para cualesquiera $a, b \in G$ se cumple que $a \star b = b \star a$.

Ejemplo 2.1.1. Dado el conjunto $\mathbb{Z}$ de los números enteros:

- La suma aritmética es una operación binaria asociativa y conmutativa.
- La resta aritmética es una operación binaria no asociativa y no conmutativa. $\lozenge$

Definición 2.1.2 (Grupo)**.** Un *grupo* es una estructura matemática formada por un conjunto no vacío G, denominado *conjunto subyacente*, y una operación binaria $\star$ sobre G, denominada *ley de grupo*. La tupla $(G, \star)$ debe satisfacer:

- **Cerradura.** Para todos elementos $a, b \in G, a \star b \in G$ (esto se cumple por la definición de operación binaria).
- **Asociatividad.** Para todos elementos $a, b, c \in G, (a \star b) \star c = a \star (b \star c)$.
- **Existencia de un elemento neutro.** Existe un único elemento $e \in G$, tal que para todo elemento $a \in G$, se tiene que $a \star e = e \star a = a$.
- **Existencia de inversos.** Para cada elemento $a \in G$, existe un elemento $a' \in G$, tal que $a \star a' = a' \star a = e$.

Definición 2.1.3 (Grupo abeliano)**.** Un grupo $(G, \star)$ se denomina *grupo abeliano* si su operación binaria $\star$ es conmutativa, es decir, para todos elementos $a, b \in G, a\star b = b\star a$.

Ejemplo 2.1.2. El conjunto $\mathbb{Z}$ de los números enteros con la suma aritmética forman un grupo, ya que:

- La suma de dos números enteros genera otro número entero.
- Para cualesquiera números enteros a, b, c se tiene que $a + (b + c) = (a + b) + c$.

- Para cualquier número entero a se tiene que $a + 0 = 0 + a = a$, por lo tanto 0 es el elemento neutro.
- Para cualquier número entero a existe un entero b tal que $a + b = b + a = 0$.
- Además, debido a que la suma aritmética es conmutativa, el anterior es un grupo abeliano. $\Diamond$

Definición 2.1.4 (Orden del grupo)**.** El *orden de un grupo* $(G, \star)$ es el número de elementos en el conjunto G. Los grupos pueden tener orden finito o infinito.

Definición 2.1.5 (Subgrupo)**.** Sean $(G, \star)$ un grupo y H un subconjunto no vacío de G, se dice que $(H, \star)$ es un *subgrupo* de $(G, \star)$, si H con la operación $\star$ es a su vez un grupo, es decir, si H es cerrado y asociativo bajo $\star$, existe un elemento neutro y existen inversos.

Ejemplo 2.1.3. El conjunto $G = \{1, -1, i, -i\}$ de números complejos con la multiplicación forman el grupo $(G, *)$ de orden 4. El subconjunto $G' = \{1, -1\}$ de G forma el subgrupo $(G', *)$ de orden 2. $\Diamond$

Definición 2.1.6 (Subgrupo generado)**.** Sean $(G, \star)$ un grupo y S un subconjunto de G, el *subgrupo generado* por S, denotado como $\langle S \rangle$, es el mínimo subgrupo de $(G, \star)$ que contiene a S.

Notación. Dado que un grupo es una tupla $(G, \star)$, esta notación se abreviará diciendo simplemente que G es un grupo.

Definición 2.1.7 (Orden de un elemento)**.** El *orden de un elemento* $a \in G$ es el orden del subgrupo $\langle \{a\} \rangle$, el cual será escrito sólamente como $\langle a \rangle$. También se puede definir como el menor entero positivo m, tal que el elemento neutro se obtiene aplicando la operación $\star$ a m copias del elemento a. Un elemento puede tener orden finito o infinito.

Definición 2.1.8 (Grupo cíclico)**.** Se dice que un grupo G es un *grupo cíclico* si existe un elemento $g \in G$ tal que $G = \langle g \rangle$. El elemento g se denomina *generador*.

Ejemplo 2.1.4. El grupo $G = (\{1, -1, i, -i\}, *)$ es un grupo cíclico con generador i, debido a que $G = \langle i \rangle$. $\Diamond$

Se puede llamar *grupo aditivo* a un grupo donde la ley de grupo se denomina adición (o suma) y se denota como $+$, el elemento neutro se denota como 0, el inverso de un elemento a es $-a$, y la aplicación de la operación $+$ a m copias de un elemento a se denota como ma.

Se puede llamar *grupo multiplicativo* a un grupo donde la ley de grupo se denomina multiplicación (o producto) y se denota como $\cdot$ o en ocasiones se omite, por ejemplo $a \cdot b$ o ab, el elemento neutro se denota como 1, el inverso de un elemento a es a^{-1}, y la aplicación de la operación $\cdot$ a m copias de un elemento a se denota como a^m.

2.1.2. Anillos

Definición 2.1.9 (Anillo)**.** Un *anillo* es una estructura matemática formada por un conjunto R, y dos operaciones binarias $+$ y $\cdot$. La tupla $(R, +, \cdot)$ debe satisfacer:

- $(R, +)$ es un grupo abeliano, siendo 0 el elemento neutro.
- $(R \setminus \{0\}, \cdot)$ es un *monoide* (es decir, es cerrado y asociativo bajo $\cdot$, y existe un elemento neutro), siendo 1 el elemento neutro.
- Distribución de la multiplicación sobre la suma, es decir, para todos elementos $a, b, c \in R$, se tiene que $a \cdot (b + c) = a \cdot b + a \cdot c$ y $(a + b) \cdot c = a \cdot c + b \cdot c$.

Un anillo se denomina *conmutativo* si la multiplicación es conmutativa. Nótese que la suma es conmutativa por definición.

Ejemplo 2.1.5. El conjunto $\mathbb{Z}$ de los números enteros con la suma y multiplicación, forman un anillo, siendo 0 y 1 los elementos neutro aditivo y neutro multiplicativo, respectivamente. $\Diamond$

Definición 2.1.10 (Sub-anillo)**.** Sean $(R, +, \cdot)$ un anillo y J un subconjunto no vacío de R, se dice que $(J, +, \cdot)$ es un *sub-anillo* de $(R, +, \cdot)$, si J con las operaciones $+$ y $\cdot$ es a su vez un anillo.

Notación. Dado que un anillo es una tupla $(R, +, \cdot)$, esta notación se abreviará diciendo simplemente que R es un anillo.

Definición 2.1.11 (Ideal)**.** Sean R un anillo y J un sub-anillo de R, J es un *ideal* de R si para todos $a \in J$ y $r \in R$ se tiene que $ar \in J$ y $ra \in J$.

Definición 2.1.12 (Ideal principal)**.** Sea R un anillo conmutativo, el ideal más pequeño que contien a un elemento $a \in R$ es el ideal $(a) = \{ba : b \in R\}$. Sea J un ideal de R, J es un *ideal principal* si existe un elemento $a \in R$ tal que $J = (a)$ y J se denomina el ideal principal generado por a.

Ejemplo 2.1.6. El subconjunto $P = \{\dots, -4, -2, 0, 2, 4, \dots\}$ de los números enteros es un sub-anillo de $\mathbb{Z}$ y es un ideal, ya que cualquier número entero multiplicado por un número par, es a su vez un número par. Además, P es un ideal principal, ya que $P = (2)$. $\Diamond$

Definición 2.1.13 (Clase de residuos)**.** Sea R un anillo y J un ideal de R, la *clase de residuos* de un elemento $a \in R$ módulo J, denotada como $[a]$, consiste de todos los elementos de R que tiene la forma $a+c$ para algún elemento $c \in J$. Dados los elementos $a, b \in R$, se dice que a y b son *congruentes* módulo J, denotado como $a \equiv b \pmod{J}$, si ambos están en la misma clase de residuos módulo J.

2.1.3. Campos

Definición 2.1.14 (Campo)**.** Un *campo* es una estructura matemática formada por un conjunto $\mathbb{F}$, y dos operaciones binarias $+$ y $\cdot$. La tupla $(\mathbb{F}, +, \cdot)$ debe satisfacer:

- $(\mathbb{F}, +)$ es un grupo abeliano, siendo 0 el elemento neutro.
- $(\mathbb{F} \setminus \{0\}, \cdot)$ es un grupo abeliano, siendo 1 el elemento neutro.
- Distribución de la multiplicación sobre la suma, es decir, para todos elementos $a, b, c \in \mathbb{F}$, se tiene que $a \cdot (b + c) = a \cdot b + a \cdot c$ y $(a + b) \cdot c = a \cdot c + b \cdot c$.

Ejemplo 2.1.7. El conjunto $\mathbb{Q}$ de los números racionales con la suma y multiplicación, forman un campo, siendo 0 y 1 los elementos neutro aditivo y neutro multiplicativo, respectivamente. $\Diamond$

Nótese que un campo es un caso particular de un anillo, en donde todos los elementos del conjunto subyacente, excepto el 0, tienen inversos multiplicativos y la multiplicación es conmutativa. En el resto de la sección se presentan definiciones en el contexto de campos, sin embargo, éstas pueden generalizarse para anillos (por ejemplo, ver capítulo 1 sección 2 de [33]).

Notación. Dado que un campo es una tupla $(\mathbb{F}, +, \cdot)$, esta notación se abreviará diciendo simplemente que $\mathbb{F}$ es un campo. Para referirse al grupo abeliano aditivo de un campo $\mathbb{F}$, se utilizará $\mathbb{F}^+$, y para referirse al grupo abeliano multiplicativo, se utilizará $\mathbb{F}^*$.

Definición 2.1.15 (Característica de un campo)**.** Dado un campo $\mathbb{F}$, su *característica*, denotada como $\text{char}(\mathbb{F})$, es el mínimo número n, tal que el elemento neutro aditivo se obtiene tras aplicar la suma a n copias del elemento neutro multiplicativo. El campo tiene característica cero si nunca se obtiene el elemento neutro aditivo. Si $\text{char}(\mathbb{F}) \neq 0$, entonces existe un número primo p tal que $\text{char}(\mathbb{F}) = p$.

Un campo $\mathbb{F}$ es *infinito* si $\text{char}(\mathbb{F}) = 0$. Un campo $\mathbb{F}$ es *finito* si $\text{char}(\mathbb{F}) = p$, con $p \neq 0$, y por lo tanto tiene un número finito de elementos; en general, un campo finito contiene $q = p^m$ elementos, $m \geq 1$, y se denota como $\mathbb{F}_q$.

Ejemplo 2.1.8. Dado el conjunto $\mathbb{F}_5 = \{0, 1, 2, 3, 4\}$, se tiene que:

- $\mathbb{F}_5$ con la suma módulo 5 es un grupo abeliano.
- $\mathbb{F}_5^* = \mathbb{F}_5 \setminus \{0\}$ con la multiplicación módulo 5 es un grupo abeliano.
- La multiplicación se distribuye sobre la suma.

Entonces, $\mathbb{F}_5$ con la suma y multiplicación módulo 5 forman un campo finito y su caracterísitca es 5. ◊

Definición 2.1.16 (Extensión de campo)**.** Sea L un campo, un *subcampo* K de L es un subconjunto del conjunto subyacente de L, el cual también forma un campo con respecto a las operaciones binarias de L. El campo L se denomina *extensión de campo* de K, y se denota como L/K (leyéndose L sobre K). Dada una extensión de campo L/K, L puede ser visto como un espacio vectorial sobre K; la dimensión de dicho espacio vectorial se denomina *grado de la extensión* y se denota como $[L : K]$.

Una manera de generar extensiones de campo, es "añadiendo" nuevos elementos al campo base K, por ejemplo, $L = K(\alpha)$, es decir, L es el campo que contiene todas las expresiones racionales formadas usando un nuevo elemento α y los elementos de K.

Ejemplo 2.1.9. El campo $\mathbb{C}$ de los números complejos es una extensión del campo $\mathbb{R}$ de los números reales. Se tiene que $\mathbb{C} = \mathbb{R}(i)$, es decir, un número complejo c se expresa como $a + bi$, donde a y b son números reales y el elemento i es la unidad imaginaria que satisface $i^2 = -1$, claramente $i \notin \mathbb{R}$. En este caso se tiene que $[\mathbb{C} : \mathbb{R}] = 2$. ◊

2.1.4. Anillo de polinomios

En esta sección, la definición de anillo de polinomios se hará sobre campos finitos, sin embargo, un anillo de polinomios se pueden definir de manera general sobre un anillo (por ejemplo, ver capítulo 1 sección 3 de [33]).

Definición 2.1.17 (Anillo de polinomios)**.** El *anillo de polinomios* en la variable X sobre un campo finito $\mathbb{F}$, denotado como $\mathbb{F}[X]$, es el conjunto cuyos elementos, denominados *polinomios*, son sumas finitas de potencias de la variable X, es decir:

$$\sum_{i=0}^{n} a_i X^i = a_0 + a_1 X + a_2 X^2 + \cdots + a_{n-1} X^{n-1} + a_n X^n$$

donde los *coeficientes* a_i son elementos de $\mathbb{F}$. El coeficiente a_n se denomina *coeficiente principal* y a_0 se denomina *término constante*.

El *polinomio cero*, denotado como 0, es aquel cuyos coeficientes son todos cero. Los *polinomios constantes* son aquellos cuyos coeficientes son todos cero excepto el término constante.

El anillo de polinomios $\mathbb{F}[X]$ con la *suma y multiplicación de polinomios* forman un anillo conmutativo; en el capítulo 3 se definen estas operaciones. El polinomio 0 y el polinomio constante 1 son los elementos neutro aditivo y neutro multiplicativo de $\mathbb{F}[X]$, respectivamente.

Definición 2.1.18 (Polinomio mónico)**.** Un polinomio $p \in \mathbb{F}[X]$ se denomina *mónico* cuando su coeficiente principal es 1. Si p no es mónico, éste puede expresarse como el producto de un polinomio mónico y una *unidad*, es decir, un elemento de $\mathbb{F}$.

Definición 2.1.19 (Grado de un polinomio)**.** El *grado de un polinomio* $p \in \mathbb{F}[X]$, denotado como $\text{grad}(p)$, es la máxima potencia de la variable X cuyo coeficiente es distinto de cero. Si $p = 0$, entonces $\text{grad}(p) = -\infty$.

Dados los polinomios $f, g \in \mathbb{F}[X]$, si existe algún polinomio $h \in \mathbb{F}[X]$ tal que $f = gh$, se dice que g *divide* a f y se denota como $g \mid f$. También se puede decir que g es un *divisor* de f, que f es *divisible por* g, o que f es un *múltiplo* de g. En caso contrario, se dice que g *no divide a* f y se denota como $g \nmid f$.

Ejemplo 2.1.10. Dado el campo $\mathbb{F}_5$, algunos elementos de $\mathbb{F}_5[X]$ son:

$$\begin{aligned} a &= 2X^5 + 4X^4 + 4X^3 + X^2 + 4X + 2, \\ b &= X^3 + 2X^2 + 4X + 2, \\ c &= 2X^2 + 1, \end{aligned}$$

con $\text{grad}(a) = 5$, $\text{grad}(b) = 3$ y $\text{grad}(c) = 2$. El polinomio b es mónico, mientras que a y c no lo son, pero pueden expresarse como:

$$\begin{aligned} a &= 2 \cdot (X^5 + 2X^4 + 2X^3 + 3X^2 + 2X + 1), \\ c &= 2 \cdot (X^2 + 3). \end{aligned}$$

Finalmente, a es divisible por b y c, ya que $a = bc$. $\Diamond$

Definición 2.1.20 (Polinomio irreducible)**.** Un polinomio no constante $p \in \mathbb{F}[X]$ es un *polinomio irreducible* si éste no es divisible por ningún otro polinomio $a \in \mathbb{F}[X]$ tal que $\text{grad}(p) > \text{grad}(a) \geq 1$.

Ejemplo 2.1.11. Los polinomios $a, b, c \in \mathbb{F}_5[X]$, con:

$$a = X^2 + 4X + 2, \quad b = X^3 + 3X + 3, \quad c = X^4 + 4X^2 + 4X + 2,$$

son irreducibles, ya que no son divisibles por ningún polinomio no constante de grado menor a ellos. $\Diamond$

Definición 2.1.21 (Anillo cociente)**.** Dado un anillo de polinomios $\mathbb{F}[X]$ y un polinomio $P \in \mathbb{F}[X]$, el *anillo cociente* $\mathbb{F}[X]/(P)$ se define como el conjunto que contiene a las clases de residuos módulo P.

Si $\mathbb{F}_q$ es un campo finito con q elementos y P es un polinomio irreducible de grado n en $\mathbb{F}_q[X]$, entonces $\mathbb{F}_q[X]/(P)$ forma un campo finito con q^n elementos. Dado lo anterior, el campo $\mathbb{F}_{q^n}$ es isomorfo a $\mathbb{F}_q[X]/(P)$ para cualquier polinomio irreducible P de grado n en $\mathbb{F}_q[X]$, y suele escribirse como:

$$\mathbb{F}_{q^n} \cong \mathbb{F}[X]/(P).$$

Ejemplo 2.1.12. Dado el campo $\mathbb{F}_5$ y el polinomio irreducible $P = X^2 + 4X + 2$, se tiene que:

$$\mathbb{F}_{5^2} \cong \mathbb{F}_5[X]/(X^2 + 4X + 2),$$

y entonces $\mathbb{F}_{5^2}$ es el campo finito de orden $5^2 = 25$ conformado por los polinomios de $\mathbb{F}_5[X]$ módulo $X^2 + 4X + 2$, es decir:

$$\mathbb{F}_{5^2} = \left\{ \begin{array}{rrrrr} 0, & 1, & 2, & 3, & 4, \\ X, & X{+}1, & X{+}2, & X{+}3, & X{+}4, \\ 2X, & 2X{+}1, & 2X{+}2, & 2X{+}3, & 2X{+}4, \\ 3X, & 3X{+}1, & 3X{+}2, & 3X{+}3, & 3X{+}4, \\ 4X, & 4X{+}1, & 4X{+}2, & 4X{+}3, & 4X{+}4 \end{array} \right\}$$

El campo $\mathbb{F}_{5^2}$ es una extensión de grado 2 del campo $\mathbb{F}_5$. $\Diamond$

2.2. Definición del problema del logaritmo discreto

Cuando se trabaja con números reales, la dificultad de realizar una exponenciación, por ejemplo $a = b^x$, no es significativamente menor a la de calcular su función inversa, es decir, el cálculo del logaritmo $x = \log_b a$. Sin embargo, cuando se trabaja en campos finitos, una exponenciación $a = b^x$ es un cálculo relativamente sencillo, pero surge la pregunta de cómo calcular el logaritmo x de un elemento a. La pregunta anterior se denomina problema del logaritmo discreto. La palabra discreto se utiliza para diferenciar el contexto de un grupo cíclico del contexto continuo (como los número reales).

Definición 2.2.1 (Problema del logaritmo discreto)**.** Sean G un grupo cíclico de orden primo r y $a, b \in G$, donde a es una potencia de b, el *problema del logaritmo discreto* es determinar el número entero $x \in [1, r-1]$ tal que $a = b^x$.

Ejemplo 2.2.1. Dado el campo finito $\mathbb{F}_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ y seleccionando el generador $b = 2$, el logaritmo discreto de los elementos de $\mathbb{F}_{11}^*$ es:

$$\begin{aligned} \log_2 1 &= 10 & \log_2 6 &= 9 \\ \log_2 2 &= 1 & \log_2 7 &= 7 \\ \log_2 3 &= 8 & \log_2 8 &= 3 \\ \log_2 4 &= 2 & \log_2 9 &= 6 \\ \log_2 5 &= 4 & \log_2 10 &= 5 \end{aligned}$$ ◊

2.3. Uso en Criptografía

Pueden existir distintas maneras de resolver un problema dado, algunas soluciones pueden ser sencillas, mientras que otras pueden ser complejas. Si se conoce una solución sencilla, se puede decir que el problema es fácil de resolver, sin embargo, si la o las soluciones conocidas son complejas, el problema suele categorizarse como difícil de resolver. Se definirá a un *algoritmo* como una serie de pasos finitos para dar solución a un problema. Por lo tanto, un problema es fácil de resolver si se conoce algún algoritmo sencillo que lo solucione, y por sencillo se entenderá a un algoritmo con complejidad polinomial [12] en su tiempo de ejecución[2]. Asimismo, un problema es difícil de resolver si no se conoce ningún algoritmo que lo resuelva con complejidad polinomial en su tiempo de ejecución.

Las *funciones de un solo sentido*, *funciones unidireccionales* o *one-way functions* (en inglés) son funciones fáciles de calcular pero difíciles de invertir. Actualmente no hay una demostración de la existencia de este tipo de funciones, solamente se conjetura sobre su existencia. Este tipo de funciones son utilizadas en criptografía de llave pública o asimétrica.

Una exponenciación en un grupo cíclico es una operación fácil de realizar, mientras que su operación inversa, calcular el logaritmo discreto, se cree que es difícil de resolver. Debido a lo anterior, el problema del logaritmo discreto es considerado como una función de un solo sentido, y por lo tanto, es utilizado en criptografía de llave pública.

[2]También se puede considerar la complejidad en espacio, sin embargo, aquí solamente se considerará el tiempo de ejecución.

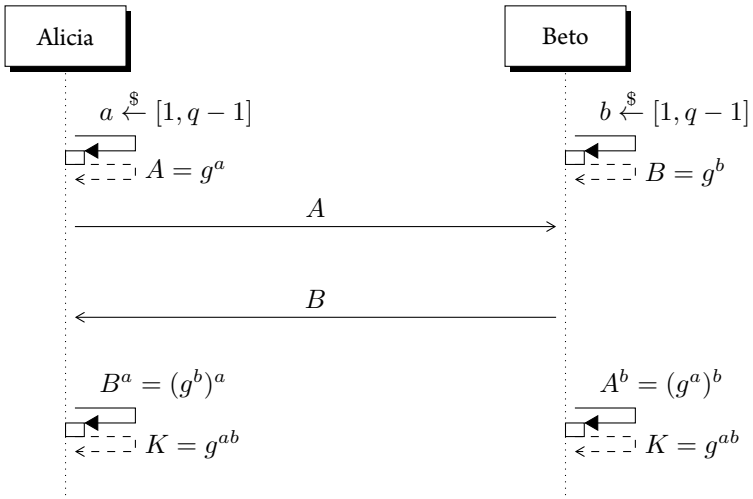


Figura 2.1: Protocolo Diffie-Hellman de intercambio de llaves. El valor q, el campo finito $\mathbb{F}_q$ y un generador g de $\mathbb{F}_q^*$ son información pública.

Protocolo Diffie-Hellman de intercambio de llaves

Uno de los problemas principales en la criptografía de llave privada o simétrica, es acordar la llave con la que se realizarán las operaciones de cifrado y descifrado. En su famoso artículo de 1976, Whitfield Diffie y Martin Hellman [13] fueron los primeros en proponer una solución para el problema anterior, cuya seguridad descansa en el problema del logaritmo discreto. Este protocolo permite a dos entidades establecer un secreto compartido utilizando un canal inseguro; este secreto se puede utilizar como llave para establecer comunicación segura en un sistema criptográfico de llave secreta.

Supongamos que dos usuarios Alicia y Beto desean acordar una llave utilizando el protocolo Diffie-Hellman. Para poder llevar a cabo lo anterior, se establece un valor q, un campo finito $\mathbb{F}_q$ y un generador g de $\mathbb{F}_q^*$. En el modelo de seguridad se considera que esta información es conocida por todas las partes, incluso para su adversario Eva.

En la figura 2.1 se ilustra el funcionamiento del protocolo Diffie-Hellman de intercambio de llaves. Primero, Alicia selecciona un número aleatorio a entre 1 y $q-1$, calcula $A = g^a$ y envía A a Beto . Asimismo, Beto selecciona un número aleatorio b entre 1 y $q-1$, calcula $B = g^b$, y envía B a Alicia. Finalmente, la llave será $K = g^{ab}$, ya que Alicia puede calcular $K = B^a = (g^b)^a = g^{ab}$, y Beto puede calcular $K = A^b = (g^a)^b = g^{ab}$.

Mientras se realiza el proceso anterior, Eva monitorea el canal inseguro de comunicación y conoce los valores $A = g^a$ y $B = g^b$, sin embargo no puede calcular el valor $K = g^{ab}$, ya que debería calcular $A^{\log_g B}$ o $B^{\log_g A}$, lo cual es difícil de realizar.

Lo anterior se denomina *suposición Diffie-Hellman*, es decir, la hipótesis de que conociendo solamente los valores g^a y g^b, es computacionalmente infactible calcular g^{ab}. Es claro que si es posible calcular logaritmos discretos, entonces la suposición Diffie-Hellman no sería cierta y el protocolo Diffie-Hellman no sería seguro.

Criptosistema ElGamal

El criptosistema ElGamal es un criptosistema de llave pública, basado en el protocolo Diffie-Hellman, propuesto por Taher Elgamal in 1985 [14]. La seguridad de este criptosistema descansa en la dificultad del cálculo del logaritmo discreto.

Supongamos que dos usuarios Alicia y Beto desean comunicarse de manera segura utilizando el criptosistema ElGamal. Para poder llevar a cabo lo anterior, se establece un valor q, un campo finito $\mathbb{F}_q$ y un generador g de $\mathbb{F}_q^*$. En el modelo de seguridad se considera que esta información es conocida por todas las partes, incluso para su adversario Eva.

En la figura 2.2 se ilustra el funcionamiento del criptosistema ElGamal. Inicialmente, Alicia selecciona un número aleatorio a entre 1 y $q-1$, y ejecuta el algoritmo de generación de llaves, obteniendo $\text{Prv}_A = a$ como su llave privada y $\text{Pub}_A = g^a$ como su llave pública. Para enviar un mensaje a Alicia, Beto selecciona un número aleatorio b entre 1 y $q-1$, calcula g^b, cifra el mensaje m mediante $m(\text{Pub}_A)^b = m(g^a)^b = mg^{ab}$, y envía (g^b, mg^{ab}) a Alicia. Para descifrar lo que recibió de Beto, Alicia calcula $(g^b)^{-a} = g^{-ab}$, y el mensaje original se obtiene mediante $m = mg^{ab}g^{-ab}$.

Mientras se realiza el proceso anterior, Eva monitorea el canal inseguro de comunicación y conoce los valores g^b, mg^{ab} y la llave pública g^a de Alicia, sin embargo no puede calcular el valor g^{ab}, ya que al igual que el protocolo Diffie-Hellman, existe la suposición de que conociendo solamente los valores g^a y g^b, es computacionalmente infactible calcular g^{ab}, a menos de que exista una manera eficiente de calcular el logaritmo discreto.

2.4. Algoritmo de cálculo de índices

Dado un problema, el tiempo requerido para que el mejor algoritmo encuentre una solución, depende de la instancia del problema. Regularmente mientras mas grande es la instancia, mayor será el tiempo que tomará encontrar una solución. Por ejemplo, dado el problema de factorización entera, el tiempo de ejecución para calcular los factores primos de un número de 128 bits, es menor al tiempo requerido para calcular los factores primos de un número de 2048 bits. Debido a lo anterior, cuando se analiza un algoritmo, su tiempo de ejecución está determinado en función del tamaño de la entrada.

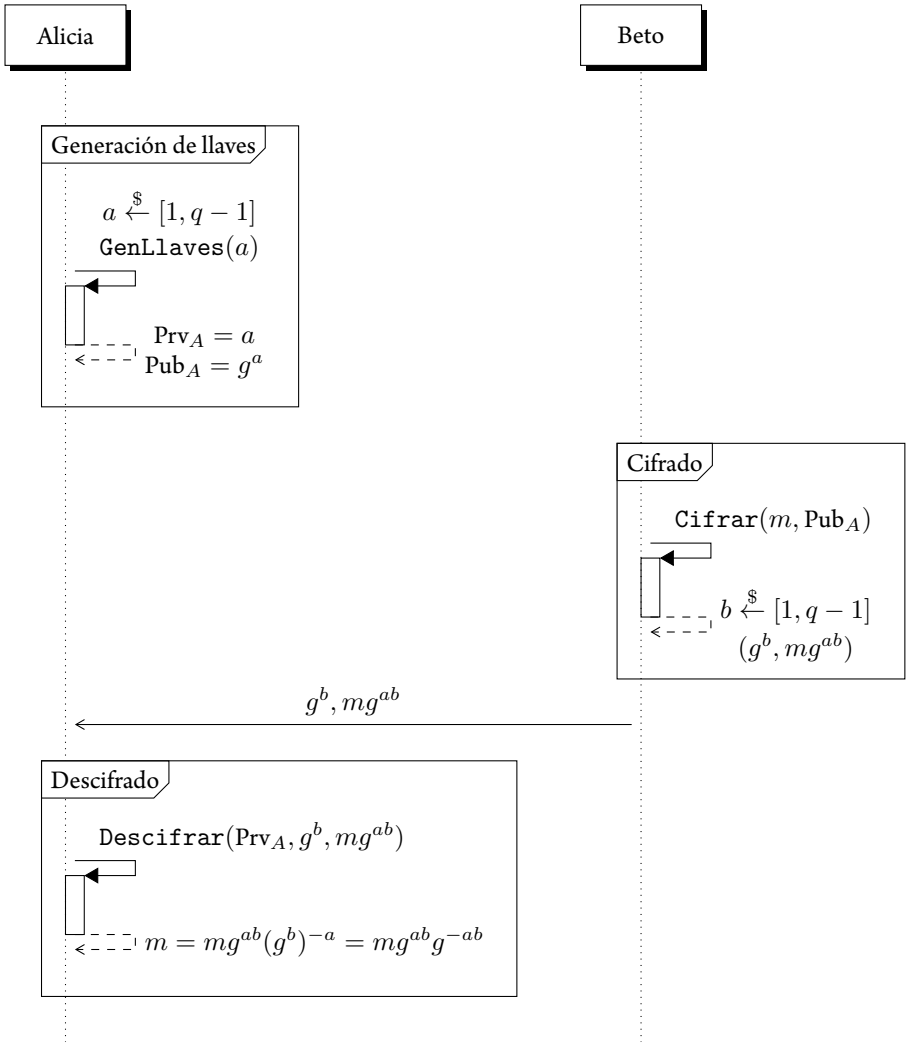


Figura 2.2: Criptosistema ElGamal. El valor q, el campo finito $\mathbb{F}_q$ y un generador g de $\mathbb{F}_q^*$ son información pública.

Para el problema del logaritmo discreto en un campo $\mathbb{F}_q$, el tamaño de la entrada está dado por $O(\log q)$ bits. Un algoritmo de complejidad *polinomial* es aquel cuyo tiempo de ejecución está acotado por un polinomio sobre el tamaño de la entrada $(\log q)^c$, donde c es una constante. Un algoritmo de complejidad *exponencial* es aquel cuyo tiempo de ejecución es de la forma q^c, donde c es una constante. Finalmente, un algoritmo de complejidad *subexponencial* es aquel cuyo tiempo de ejecución es de la forma:

$$L_q[\alpha, c] = e^{c(\log q)^{\alpha}(\log\log q)^{1-\alpha}}$$

donde $0 < \alpha < 1$ y c es una constante. Nótese que si $\alpha = 0$ el algoritmo tiene complejidad polinomial, y si $\alpha = 1$ el algoritmo tiene complejidad exponencial.

El *algoritmo de cálculo de índices* es actualmente el algoritmo más eficiente para calcular logaritmos discretos. El algoritmo se ejecuta por fases, las cuales se explican brevemente a continuación:

- **Base de factores**. Se selecciona, de acuerdo a algún criterio, un conjunto de b elementos del campo de interés, los cuales se denominarán *base de factores*; la base de factores está conformada regularmente por un conjunto de elementos pequeños del campo, es decir, polinomios de grado chico, tales como el conjunto de todos los polinomios lineales.

- **Generación de relaciones**. Esta fase consiste en buscar un conjunto de b relaciones linealmente independientes entre una potencia de un generador g del campo y los elementos de la base de factores; cada relación es vista como una ecuación que formará parte de un sistema de ecuaciones de b incógnitas, las cuales representan los logaritmos discretos de los elementos de la base de factores.

- **Álgebra lineal**. En esta fase se encuentra la solución al sistema de ecuaciones lineales obtenido previamente.

- **Descenso**. En la fase final, se expresa al elemento h, al cual se desea calcular el logaritmo discreto, en términos de los elementos de la base de factores.

Ejemplo 2.4.1. Dado el campo $\mathbb{F}_{83}$, se escoge un generador $g = 2$ y el reto $h = 31$. Se desea calcular $\log_g(h)$ utilizando el algoritmo de cálculo de índices:

- **Base de factores**. Se establece que la base de factores será $B = \{2, 3, 5, 7\}$.

- **Generación de relaciones**. De manera aleatoria se obtienen las relaciones:

$$\begin{array}{lll} 2^1 = 2 & 2^7 = 45 = 3^2 \cdot 5 & 2^{14} = 33 = 3 \cdot 11 \\ 2^{12} = 29 & 2^8 = 7 & 2^{13} = 58 = 2 \cdot 29 \\ 2^{15} = 66 = 2 \cdot 3 \cdot 11 & 2^{17} = 15 = 3 \cdot 5 & \end{array}$$

y se seleccionan aquellas que contienen a los elementos de la base de factores:

$$2^1 = 2, 2^7 = 3^2 \cdot 5, 2^8 = 7 \text{ y } 2^{17} = 3 \cdot 5.$$

Estas relaciones forman un sistema de 4 ecuaciones con 4 incógnitas.

- **Álgebra lineal**. Se encuentra la solución al sistema de ecuaciones anterior:

$$\begin{array}{c} 2 \quad 3 \quad 5 \quad 7 \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 7 \\ 8 \\ 17 \end{pmatrix}$$

y entonces $\log_2(2) = 1, \log_2(3) = 72, \log_2(5) = 27$ y $\log_2(7) = 8$.

- **Descenso**. Se tiene que $h^2 = (31)^2 = 48 = 2^4 \cdot 3 \Rightarrow 2\log_2(h) = 4 + 72 \Rightarrow \log_2(h) = 38$. ◊

Adleman en 1979 [3] propone el algoritmo de cálculo de índices para resolver el problema del logaritmo discreto en $\mathbb{F}_p$, donde p es un número primo, y la complejidad del algoritmo es subexponencial $L_p[\frac{1}{2}, 2]$. En 1982, Hellman y Reyneri [22] adaptaron el algoritmo de Adleman para calcular logaritmos discretos en $\mathbb{F}_{p^m}$, donde p es un número primo, $m \in \mathbb{N}$, poniendo especial atención al caso $\mathbb{F}_{2^m}$, y la complejidad de este algoritmo es $L_{2^m}[\frac{1}{2}, 1.414]$. En 1984, Coppersmith [10] presentó una versión del algoritmo con complejidad $L_{2^m}[\frac{1}{3}, 1.526]$ para calcular logaritmos discretos en $\mathbb{F}_{2^m}$.

Las modificaciones que se realizan al algoritmo de cálculo de índices original tienen como objetivo mejorar los tiempos de ejecución de una o varias fases, disminuyendo así la complejidad total del algoritmo. En los últimos años se han implementado diversas técnicas para calcular el logaritmo discreto en campos finitos que tienen la forma $\mathbb{F}_Q$ con característica pequeña $(\text{char}(\mathbb{F}_Q) = 2, 3)$, donde $Q = q^n$, q es una potencia de un número primo y $n \in \mathbb{N}$. Algunos resultados importantes se muestran a continuación:

- Joux y Lercier 2006 [26]: $L_Q[\frac{1}{3}, 1.442]$.
- Joux 2012 [25]: $L_Q[\frac{1}{3}, 0.961]$.
- Gologlu et. al 2013 [17]: $L_Q[\frac{1}{3}, 0.961]$ para algunos campos y entre $L_Q[\frac{1}{3}, 0.763]$ y $L_Q[\frac{1}{3}, 0.794]$, para otros.
- Joux 2013 [24]: $L_Q[\frac{1}{4} + o(1), c]$ (para algún c indeterminado).

CAPÍTULO 3

De la aritmética de polinomios

En la sección 2.1.4 se definió que el conjunto de polinomios sobre un campo finito, con la suma y multiplicación polinomial, forman un anillo denominado anillo de polinomios. Allí también se mencionó la idea de divisibilidad y se definió el anillo cociente. En la sección 3.1 de este capítulo, se definen formalmente las operaciones suma, multiplicación y división polinomial. Posteriormente, se define el máximo común divisor de polinomios y se muestra la manera de calcularlo. La sección finaliza detallando la aritmética en el anillo cociente.

Las operaciones polinomiales anteriores requieren de operaciones aritméticas entre los coeficientes de los polinomios. Las operaciones entre coeficientes se realizan en el campo sobre el cual está definido el anillo de polinomios. La sección 3.2 está dedicada a la aritmética en campos finitos. Primero se presentan dos maneras de representar los elementos de un campo, y posteriormente se muestran alternativas para realizar aritmética con las distintas representaciones.

3.1. Aritmética en anillos de polinomios

En esta sección se muestra la aritmética de polinomios sobre un campo finito $\mathbb{F}_q$ y sus correspondientes algoritmos. Se inicia con aritmética básica, continuando con el máximo común divisor de polinomios, y se finaliza con la aritmética en el anillo cociente. Para la exposición de esta sección, se supondrá que se tiene la aritmética del campo $\mathbb{F}_q$ sobre el cual se define el anillo de polinomios. El costo de los algoritmos se expresa en número de operaciones en $\mathbb{F}_q$, y a menos que se indique lo contrario, se supondrá que el costo de cualquier operación en $\mathbb{F}_q$ es $O(1)$.

Sean $f = \sum_{i=0}^{n} f_i X^i$ y $g = \sum_{i=0}^{n} g_i X^i$ polinomios en $\mathbb{F}_q[X]$ ambos con el mismo grado n, entonces f y g se consideran *iguales* si y solo si $f_i = g_i$ para $i \in \{0, \ldots, n\}$. Además se tiene que $-f = \sum_{i=0}^{n} -f_i X^i$.

Un polinomio $f \in \mathbb{F}_q[X]$ se puede representar como un vector $(f_n, \ldots, f_0)$ de $n+1$ elementos, donde cada f_i del vector es el mismo f_i de la representación como suma de potencias de X.

3.1.1. Aritmética básica

Suma y resta

Sean $f = \sum_{i=0}^{n} f_i X^i$ y $g = \sum_{i=0}^{n} g_i X^i$ polinomios en $\mathbb{F}_q[X]$. Entonces, la *suma* $h = f + g$ y *resta* $h' = f - g$ están definidas como:

$$h = \sum_{i=0}^{n} h_i X^i = \sum_{i=0}^{n} (f_i + g_i) X^i \quad \text{y} \quad h' = \sum_{i=0}^{n} h'_i X^i = \sum_{i=0}^{n} (f_i - g_i) X^i.$$

Nótese que la resta puede ser definida como una suma, es decir, $h' = f + (-g)$, entonces ambas operaciones serán referidas como suma. Sin pérdida de generalidad se supondrá que $f, g \in \mathbb{F}_q[X]$ tienen el mismo grado n. El algoritmo 3.1 corresponde a la suma polinomial y su costo es $O(n)$ operaciones en $\mathbb{F}_q$ [40].

Ejemplo 3.1.1. Sean

$$f = 4X^4 + 3X^2 + 2X + 2 \quad \text{y} \quad g = 2X^4 + 3X^3 + 3X^2 + X + 1$$

polinomios en $\mathbb{F}_5[X]$, la suma y la resta son:

$$\begin{array}{rr} + & \begin{array}{r} 4X^4 \quad +3X^2 +2X +2 \\ 2X^4 +3X^3 +3X^2 + X +1 \\ \hline X^4 +3X^3 + X^2 +3X +3 \end{array} \end{array} \qquad \begin{array}{rr} - & \begin{array}{r} 4X^4 \quad +3X^2 +2X +2 \\ 2X^4 +3X^3 +3X^2 + X +1 \\ \hline 2X^4 +2X^3 \quad + X +1 \end{array} \end{array} \quad \Diamond$$

Algoritmo 3.1 `SumaPolinomial` [40]

Entrada:

$f = (f_n, \ldots, f_0) \in \mathbb{F}_q[X]$, $g = (g_n, \ldots, g_0) \in \mathbb{F}_q[X]$

Salida:

$h = (h_n, \ldots, h_0) \in \mathbb{F}_q[X]$

1: **para** $i \leftarrow 0, \ldots, n$ **hacer**
2: $\quad h_i \leftarrow f_i + h_i$
3: **fin para**
4: **regresar** $(h_n, \ldots, h_0)$

Multiplicación

Sean $f = \sum_{i=0}^{n} f_i X^i$ y $g = \sum_{j=0}^{m} g_j X^j$ polinomios en $\mathbb{F}_q[X]$, la *multiplicación* $h = f \cdot g$ está definida como:

$$h = \sum_{k=0}^{n+m} h_k X^k, \quad \text{con} \quad h_k = \sum_{\substack{0 \le i \le n \\ 0 \le j \le m \\ i+j=k}} f_i g_j.$$

La multiplicación polinomial se puede realizar usando el *método de la escuela* (*schoolbook method* en inglés). Este método se denomina así, debido a que los cálculos se realizan de la misma manera como se aprende a multiplicar en los primeros años escolares. Sin pérdida de generalidad se supondrá que $f, g \in \mathbb{F}_q[X]$ tienen el mismo grado n. El algoritmo 3.2 corresponde a la multiplicación polinomial con el método de la escuela y su costo es $O(n^2)$ operaciones en $\mathbb{F}_q$ [40].

Ejemplo 3.1.2. Sean

$$f = 3X^3 + 2X^2 + X + 2 \quad \text{y} \quad g = 2X^3 + 3X^2 + X + 1$$

polinomios en $\mathbb{F}_5[X]$, la multiplicación utilizando el método de escuela es:

$$\begin{array}{r}
3X^3 + 2X^2 + X + 2 \\
\times \quad 2X^3 + 3X^2 + X + 1 \\
\hline
3X^3 + 2X^2 + X + 2 \\
3X^4 + 2X^3 + X^2 + 2X \\
4X^5 + X^4 + 3X^3 + X^2 \\
X^6 + 4X^5 + 2X^4 + 4X^3 \\
\hline
X^6 + 3X^5 + X^4 + 2X^3 + 4X^2 + 3X + 2
\end{array}$$

$\Diamond$

Algoritmo 3.2 `MultPolinomialEscuela` [40]

Entrada:

$f = (f_n, \ldots, f_0) \in \mathbb{F}_q[X], g = (g_n, \ldots, g_0) \in \mathbb{F}_q[X]$

Salida:

$h = (h_{2n}, \ldots, h_0) \in \mathbb{F}_q[X]$

1: **para** $i \leftarrow 0, \ldots, 2n$ **hacer**
2: $h_i \leftarrow 0$
3: **fin para**
4: **para** $i \leftarrow 0, \ldots, n$ **hacer**
5: **para** $j \leftarrow 0, \ldots, n$ **hacer**
6: $h_{i+j} \leftarrow h_{i+j} + a_i b_j$
7: **fin para**
8: **fin para**
9: **regresar** $(h_{2n}, \ldots, h_0)$

Otra alternativa para realizar la multiplicación polinomial es el método de Karatsuba [29]. Sin pérdida de generalidad, se supondrá que $f, g \in \mathbb{F}_q[X]$ tienen el mismo grado n. Sean $m = \lceil (n+1)/2 \rceil$, $f = \sum_{i=0}^{n} f_i X^i$ y $g = \sum_{j=0}^{n} g_j X^j$, estos polinomios se reescriben de la siguiente manera:

$$f = f_A X^m + f_B \quad \text{y} \quad g = g_A X^m + g_B,$$

donde $f_A, f_B, g_A, g_B \in \mathbb{F}_q[X]$ y tienen grado a lo más $m-1$. De lo anterior se tiene que:

$$f \cdot g = f_A g_A X^{2m} + (f_A g_B + f_B g_A) X^m + f_B g_B,$$

y se utilizan cuatro multiplicaciones de polinomios de grado $m-1$. Sin embargo, el cálculo anterior se puede realizar como sigue:

$$\begin{aligned} t_0 &= f_B g_B, \\ t_2 &= f_A g_A, \\ t_1 &= (f_A + f_B)(g_A + g_B) - t_0 - t_2 = f_A g_B + f_B g_A, \end{aligned}$$

y entonces $f \cdot g = t_2 X^{2m} + t_1 X^m + t_0$. Este método utilizan tres multiplicaciones de polinomios de grado $m-1$ y algunas sumas adicionales. El algoritmo 3.3 corresponde a la multiplicación polinomial con el método de Karatsuba y su costo es $O(n^{\log_2 3})$ operaciones en $\mathbb{F}_q$ [40].

Algoritmo 3.3 `MultPolinomialKaratsuba` [40]

Entrada:

$f = (f_n, \ldots, f_0) \in \mathbb{F}_q[X], g = (g_n, \ldots, g_0) \in \mathbb{F}_q[X]$

Salida:

$h = (h_{2n}, \ldots, h_0) \in \mathbb{F}_q[X]$

1: **si** $n = 0$
2: **regresar** $f \cdot g$ /* $f \cdot g \in \mathbb{F}_q$ */
3: **fin si**
4: $m \leftarrow \lceil (n+1)/2 \rceil$
5: $f_A, f_B \leftarrow (f_n, \ldots, f_m), (f_{m-1}, \ldots, f_0)$
6: $g_A, g_B \leftarrow (g_n, \ldots, g_m), (g_{m-1}, \ldots, g_0)$
7: $t_0 \leftarrow$`MultPolinomialKaratsuba(`f_B, g_B`)`
8: $t_2 \leftarrow$`MultPolinomialKaratsuba(`f_A, g_A`)`
9: $t_1 \leftarrow$`MultPolinomialKaratsuba(`$f_A + f_B, g_A + g_B$`)`
10: **regresar** $(t_2 X^{2m} + (t_1 - t_0 - t_2)X^m + t_0)$

Ejemplo 3.1.3. Sean $f, g \in \mathbb{F}_5[X]$ como en el ejemplo 3.1.2, se tiene que:

$$\begin{aligned} f &= f_A X^2 + f_B, \quad \text{donde} \quad f_A = 3X + 2, f_B = X + 2, \text{ y} \\ g &= g_A X^2 + g_B, \quad \text{donde} \quad g_A = 2X + 3, g_B = X + 1. \end{aligned}$$

La multiplicación utilizando el método de Karatsuba es:

$$\begin{aligned} t_0 &= (X+2)(X+1) \\ &= X^2 + 3X + 2, \\ t_2 &= (3X+2)(2X+3) \\ &= X^2 + 3X + 1, \\ t_1 &= (3X + 2 + X + 2)(2X + 3 + X + 1) - t_0 - t_2 \\ &= 2X + 3, \end{aligned}$$

y entonces:

$$\begin{aligned} f \cdot g &= (X^2 + 3X + 1)X^4 + (2X + 3)X^2 + (X^2 + 3X + 2) \\ &= X^6 + 3X^5 + X^4 + 2X^3 + 4X^2 + 3X + 2. \end{aligned}$$

Nótese que $f \cdot g$ con el método de Karatsuba es igual al resultado obtenido en el ejemplo 3.1.2 utilizando el método de escuela. $\Diamond$

División

Teorema 3.1.1 (División de polinomios [33, Teorema 1.52]). *Sea $g \neq 0$ un polinomio en $\mathbb{F}_q[X]$, entonces para cualquier $f \in \mathbb{F}_q[X]$, existen únicos polinomios $q, r \in \mathbb{F}_q$ tal que $f = qg + r$ con $\mathrm{grad}(r) < \mathrm{grad}(g)$.*

Los polinomios q y r a los que se refiere el teorema 3.1.1 se denominan *cociente* y *residuo* respectivamente. El operador "$/$" denota la operación que calcula el cociente, es decir, $q = f/g$. El operador "$\mathrm{mód}$" denota la operación que calcula el residuo, es decir, $r = f \mod g$, y se dice que r es el *residuo de f módulo g*.

En el algoritmo 3.4 se muestra la manera de calcular la división polinomial. En la línea 4 se calcula un inverso multiplicativo en $\mathbb{F}_q$. Las operaciones en las líneas 7 y 8 se realizan a lo más $n - m + 1$ veces; en la línea 7 se utiliza una multiplicación en $\mathbb{F}_q$, y en la línea 8, calcular $q_i \cdot g$ utiliza m multiplicaciones y m sumas en $\mathbb{F}_q$, mientras que multiplicar por X^i es realizar un "corrimiento" de i posiciones a la izquierda. En total se utilizan a lo más $(2m+1)(n-m+1)$ operaciones en $\mathbb{F}_q$ y el cálculo inicial del inverso multiplicativo. Como se menciona en [40], si $n < 2m$, entonces el algoritmo 3.4 utiliza a lo más $2m^2 + O(m)$ operaciones en $\mathbb{F}_q$ y su costo es $O(m^2)$.

Algoritmo 3.4 `DivPolinomial` [40]

Entrada:
$f = (f_n, \ldots, f_0) \in \mathbb{F}_q[X], g = (g_m, \ldots, g_0) \in \mathbb{F}_q[X]$ con $g \neq 0$
Salida:
$q, r \in \mathbb{F}_q[X]$ tal que $f = qg + r$ con $\mathrm{grad}(r) < \mathrm{grad}(g)$

1: **si** $n < m$
2: **regresar** $0, f$ /* $q = 0, r = f$ */
3: **fin si**
4: $r \leftarrow f, u \leftarrow g_m^{-1}$
5: **para** $i \leftarrow n - m, \ldots, 0$ **hacer**
6: **si** $\mathrm{grad}(r) = m + i$
7: $q_i \leftarrow \mathrm{cp}(r) \cdot u$ /* $\mathrm{cp}(r)$ `denota el coeficiente principal de` r */
8: $r \leftarrow r - q_i X^i \cdot g$
9: **si no**
10: $q_i \leftarrow 0$
11: **fin si**
12: **fin para**
13: **regresar** q, r

Ejemplo 3.1.4. Sean

$$f = 4X^4 + X^3 + 4X^2 + 2X + 3 \quad \text{y} \quad g = X^2 + 3X + 1$$

polinomios en $\mathbb{F}_5[X]$, la división es:

$$\begin{array}{rl|l}
 & & 4X^2\ +4X\ +3 \\
\cline{2-3}
X^2+3X+1 & 4X^4\ +\ X^3\ +4X^2\ +2X\ +3 & \\
 & \underline{-(4X^4\ +2X^3\ +4X^2)} & \\
 & 4X^3\ +0X^2\ +2X & \\
 & \underline{-(4X^3\ +2X^2\ +4X)} & \\
 & 3X^2\ +3X\ +3 & \\
 & \underline{-(3X^2\ +4X\ +3)} & \\
 & 4X &
\end{array}$$

Entonces $q = 4X^2 + 4X + 3$, $r = 4X$ y $f = qg + r$. ◊

3.1.2. Máximo común divisor

Definición 3.1.2 (Máximo común divisor)**.** Dados los polinomios $f, g \in \mathbb{F}_q[X]$, se denomina *divisor común* de f y g a un polinomio $d \in \mathbb{F}_q[X]$ tal que $d \mid f$ y $d \mid g$. El polinomio d se denomina *máximo común divisor* de f y g, denotado como $\text{mcd}(f, g)$, si d es mónico y todos los demás divisores comunes de f y g dividen a d.

Teorema 3.1.3 ([33, Teorema 1.55])**.** *Sean $f_1, \ldots, f_n \in \mathbb{F}_q[X]$, con al menos un f_i distinto de cero, existe un único polinomio mónico $d \in \mathbb{F}_q[X]$, tal que (i) d divide a cada f_i y (ii) cualquier polinomio $c \in \mathbb{F}_q[X]$ que divide a algún f_i, divide a d. Además, este polinomio d puede ser expresado como:*

$$d = b_1 f_1 + \cdots + b_n f_n, \quad \textit{donde} \quad b_1, \ldots, b_n \in \mathbb{F}_q[X].$$

El polinomio mónico d al que se refiere el teorema 3.1.3 es el máximo común divisor de $f_1, \ldots, f_n \in \mathbb{F}_q[X]$, se establece que es único y que puede ser expresado en términos de $f_1, \ldots, f_n$. Si $\text{mcd}(f_1, \ldots, f_n) = 1$, se dice que estos polinomios son *primos relativos*, y se dice que son *primos relativos a pares* si $\text{mcd}(f_i, f_j) = 1$ para $1 \leq i < j \leq n$.

Dados $f, g \in \mathbb{F}_q[X]$, su máximo común divisor puede calcularse por medio del *algoritmo de Euclides*. Sin pérdida de generalidad, se supondrá que $\text{grad}(f) \geq \text{grad}(g)$ y

$g \neq 0$. El algoritmo de Euclides realiza divisiones sucesivas que, por el teorema 3.1.1, se pueden expresar de la siguiente manera:

$$\begin{aligned}
r_0 &= f, \\
r_1 &= g, \\
r_0 &= q_1 r_1 + r_2 && 0 \leq \text{grad}(r_2) < \text{grad}(r_1), \\
&\vdots && \vdots \\
r_{k-2} &= q_{k-1} r_{k-1} + r_k && 0 \leq \text{grad}(r_k) < \text{grad}(r_{k-1}), \\
r_{k-1} &= q_k r_k && r_{k+1} = 0,
\end{aligned}$$

y $\text{mcd}(f, g) = c^{-1} r_k$, donde c es el coeficiente principal de r_k [38, Teorema 17.2]; multiplicar por c^{-1} asegura que $\text{mcd}(f, g)$ es mónico. El algoritmo 3.5 corresponde al algoritmo de Euclides y su costo es $O(nm)$ operaciones en $\mathbb{F}_q$, donde $n = \text{grad}(f)$ y $m = \text{grad}(g)$ [38].

Algoritmo 3.5 `Euclides` [38]

Entrada:
$f, g \in \mathbb{F}_q[X]$, tal que $\text{grad}(f) \geq \text{grad}(g)$ y $g \neq 0$
Salida:
$\text{mcd}(f, g) \in \mathbb{F}_q[X]$
1: $r \leftarrow f, r' \leftarrow g$
2: **mientras** $r' \neq 0$ **hacer**
3: $r'' \leftarrow r \mod r'$
4: $(r, r') \leftarrow (r', r'')$
5: **fin mientras**
6: $c \leftarrow \text{cp}(r)$ /* $\text{cp}(r)$ `denota el coeficiente principal de` r */
7: **regresar** $c^{-1} r$

Ejemplo 3.1.5. Sean

$$f = 3X^6+3X^5+2X^4+4X^3+4X^2+3X+3 \quad \text{y} \quad g = X^4+2X^3+3X^2+3X+2$$

polinomios en $\mathbb{F}_5[X]$, el algoritmo de Euclides aplicado a f y g resulta en:

i	Cálculos del algoritmo de Euclides
0	$q_0 = -$ $r_0 = 3X^6 + 3X^5 + 2X^4 + 4X^3 + 4X^2 + 3X + 3$
	$\vdots$

$$\vdots$$

i	Cálculos del algoritmo de Euclides
1	$q_1 = 3X^2 + 2X + 4$ $r_1 = X^4 + 2X^3 + 3X^2 + 3X + 2$
2	$q_2 = X + 2$ $r_2 = X^3 + 2X$
3	$q_3 = X + 1$ $r_3 = X^2 + 4X + 2$
4	$q_4 = X + 1$ $r_4 = X + 3$
5	$q_5 = 4X + 2$ $r_5 = 4$
6	$q_6 = -$ $r_6 = 0$

De la tabla anterior se obtiene $c = \text{cp}(r_5) = 4$, entonces $\text{mcd}(f, g) = c^{-1}r_5 = 1$ y f es primo relativo de g. $\Diamond$

Ahora, dados $f, g \in \mathbb{F}_q[X]$ y sea $d = \text{mcd}(f, g)$, por el teorema 3.1.3 se sabe que existen $s, t \in \mathbb{F}_q[X]$ tal que $d = fs + gt$. Los polinomios d, s y t pueden calcularse por medio del *algoritmo extendido de Euclides*. Sin pérdida de generalidad, se supondrá que $\text{grad}(f) \geq \text{grad}(g)$ y $g \neq 0$. Para este algoritmo, se obtienen los valores $r_0, \ldots, r_{k+1}$ como en el algoritmo de Euclides, y además se calcula:

$$\begin{array}{ll} s_0 = 1 & t_0 = 0 \\ s_1 = 0 & t_1 = 1 \\ \vdots & \vdots \\ s_{i+1} = s_{i-1} - s_i q_i & t_{i+1} = t_{i-1} - t_i q_i, \quad i = \{1, \ldots, k\}. \end{array}$$

En cada iteración se cumple que $fs_i + gt_i = r_i$, y particularmente $fs_k + gt_k = r_k = c \cdot \text{mcd}(f, g)$, donde c es el coeficiente principal de r_k [38, Teorema 17.4]. De lo anterior, $\text{mcd}(f, g) = c^{-1}r_k$, $s = c^{-1}s_k$ y $t = c^{-1}t_k$ cumpliéndose que $fs + gt = \text{mcd}(f, g)$. El algoritmo 3.6 corresponde al algoritmo extendido de Euclides y su costo es $O(nm)$ operaciones en $\mathbb{F}_q$, donde $n = \text{grad}(f)$ y $m = \text{grad}(g)$ [38].

Ejemplo 3.1.6. Sean $f, g \in \mathbb{F}_5[X]$ como en el ejemplo 3.1.5, el algoritmo extendido de Euclides aplicado a estos polinomios resulta en:

Algoritmo 3.6 `EuclidesExtendido` [38]

Entrada:

$f, g \in \mathbb{F}_q[X]$, tal que grad$(f) \geq$ grad(g) y $g \neq 0$

Salida:

mcd$(f, g), s, t \in \mathbb{F}_q[X]$ tal que $fs + gt =$ mcd(f, g)

1: $r \leftarrow f, r' \leftarrow g$
2: $s \leftarrow 1, s' \leftarrow 0$
3: $t \leftarrow 0, t' \leftarrow 1$
4: **mientras** $r' \neq 0$ **hacer**
5: $q, r'' \leftarrow$`DivPolinomial(`r, r'`)`
6: $(r, s, t, r', s', t') \leftarrow (r', s', t', r'', s - s'q, t - t'q)$
7: **fin mientras**
8: $c \leftarrow$ cp(r) /* cp(r) `denota el coeficiente principal de` r */
9: **regresar** $c^{-1}r, c^{-1}s, c^{-1}t$

i	Cálculos del algoritmo extendido de Euclides
0	$q_0 = -$ $r_0 = 3X^6 + 3X^5 + 2X^4 + 4X^3 + 4X^2 + 3X + 3$ $s_0 = 1$ $t_0 = 0$
1	$q_1 = 3X^2 + 2X + 4$ $r_1 = X^4 + 2X^3 + 3X^2 + 3X + 2$ $s_1 = 0$ $t_1 = 1$
2	$q_2 = X + 2$ $r_2 = X^3 + 2X$ $s_2 = 1$ $t_2 = 2X^2 + 3X + 1$
3	$q_3 = X + 1$ $r_3 = X^2 + 4X + 2$ $s_3 = 4X + 3$ $t_3 = 3X^3 + 3X^2 + 3X + 4$
4	$q_4 = X + 1$ $r_4 = X + 3$ $s_4 = X^2 + 3X + 3$
	$\vdots$

$$\vdots$$

i	Cálculos del algoritmo extendido de Euclides
	$t_4 = 2X^4 + 4X^3 + X^2 + X + 2$
5	$q_5 = 4X + 2$ $r_5 = 4$ $s_5 = 4X^3 + X^2 + 3X$ $t_5 = 3X^5 + 4X^4 + 3X^3 + X^2 + 2$
6	$q_6 = -$ $r_6 = 0$ $s_6 = 4X^4 + 3X^3 + 2X^2 + 2X + 3$ $t_6 = 3X^6 + 3X^5 + 2X^4 + 4X^3 + 4X^2 + 3X + 3$

De la tabla anterior se obtiene $c = \text{cp}(r_5) = 4$, entonces $\text{mcd}(f, g) = c^{-1}r_5 = 1$, $s = c^{-1}s_5 = X^3 + 4X^2 + 2X$ y $t = c^{-1}t_5 = 2X^5 + X^4 + 2X^3 + 4X^2 + 3$. Nótese que se cumple $fs + gt = \text{mcd}(f, g)$. $\Diamond$

3.1.3. Aritmética en el anillo cociente

Dado $P \in \mathbb{F}_q[X]$ de grado ℓ, el anillo cociente $\mathbb{F}_q[X]/(P)$ es el conjunto de las clases de residuos módulo P, donde (P) es el ideal generado por P. La clase de residuos módulo P de un polinomio $a \in \mathbb{F}_q[X]$ se define como $[a] = \{a + b : b \in (P)\}$ (véase sección 2.1.2). Para cada $\alpha \in \mathbb{F}_q[X]/(P)$ existe un único polinomio $A \in \mathbb{F}_q[X]$ tal que $\text{grad}(A) < \ell$, denominado *representante canónico de* α [38, Sección 17.1].

Por simplicidad, se dirá que $\mathbb{F}_q[X]/(P)$ es el conjunto que contiene los representantes canónicos de las clases de residuos. Entonces, si $A \in \mathbb{F}_q[X]$ es el representante canónico de $\alpha \in \mathbb{F}_q[X]/(P)$, se dirá que $A \in \mathbb{F}_q[X]/(P)$. Por lo anterior, al realizar aritmética en $\mathbb{F}_q[X]/(P)$, los polinomios de grado $\geq \ell$ deben ser *reducidos módulo* P, es decir, se debe calcular su residuo módulo P.

Al igual que en la sección 3.1, los polinomios en $\mathbb{F}_q[X]/(P)$ se pueden representar como un vector de ℓ coeficientes.

Reducción

Sea $f \in \mathbb{F}_q[X]/(P)$ de grado $n \geq \ell$, la reducción de f módulo P se puede realizar utilizando la división polinomial, es decir, $q, r = \texttt{DivPolinomial}(f, P)$ y r es el resultado buscado, ya que $r = f \mod P$. En este caso, la reducción utiliza una división polinomial y su costo es $O(\ell^2)$ operaciones en $\mathbb{F}_q$.

Ahora, sin pérdida de generalidad se supondrá que P es mónico, entonces:

$$P = X^{\ell} + \sum_{i=0}^{\ell-1} p_i X^i \Rightarrow P' = X^{\ell} = \sum_{i=0}^{\ell-1} -p_i X^i,$$

y $P' = X^{\ell} \bmod P$. Otra alternativa para realizar la reducción de f módulo P, es utilizar el valor de P' como en el algoritmo 3.7.

Algoritmo 3.7 `Reduccion`

Entrada:

$f = (f_n, \ldots, f_0) \in \mathbb{F}_q[X]$, $P = (p_\ell, \ldots, p_0) \in \mathbb{F}_q[X]$ con $n \geq \ell$

Salida:

$r \in \mathbb{F}_q[X]$, el residuo de f módulo P

1: $P' \leftarrow (-p_{\ell-1}, \ldots, -p_0)$
2: $Q \leftarrow P'$ /* $Q = X^{\ell} \bmod P$ */
3: $r \leftarrow (f_{\ell-1}, \ldots, f_0) + f_\ell \cdot Q$
4: **para** $i \leftarrow \ell + 1, \ldots, n$ **hacer**
5: $Q \leftarrow Q \cdot X$ /* Q `tiene grado` ℓ */
6: $Q \leftarrow (q_{\ell-1}, \ldots, q_0) + q_\ell \cdot P'$ /* `reducir` q_ℓ `y entonces` $Q = X^i \bmod P$ */
7: $r \leftarrow r + f_i \cdot Q$
8: **fin para**
9: **regresar** r

El cálculo en la línea 3 utiliza ℓ multiplicaciones y ℓ sumas en $\mathbb{F}_q$. Las operaciones en las líneas 5, 6 y 7 se realizan $n - \ell$ veces; en la línea 5, multiplicar por X es realizar un "corrimiento" de una posición a la izquierda, las líneas 6 y 7 utilizan ℓ multiplicaciones y ℓ sumas en $\mathbb{F}_q$ cada una. En total se utilizan $2\ell + (4\ell)(n - \ell)$ operaciones en $\mathbb{F}_q$. Utilizando el mismo argumento que en la división polinomial, si $n < 2\ell$, el costo del algoritmo 3.7 es $O(\ell^2)$ operaciones en $\mathbb{F}_q$.

Ejemplo 3.1.7. Sean

$$P = X^3 + 3X + 3 \quad \text{y} \quad f = X^5 + 4X^4 + 3X^3 + 3X^2 + 2X + 1$$

polinomios en $\mathbb{F}_5[X]$, utilizando `DivPolinomial(`f, P`)`, se obtiene:

$$q = X^2 + 4X \quad \text{y} \quad r = 3X^2 + 1.$$

Ahora, se tiene que $P' = 2X + 2$, y entonces la reducción de f módulo P se puede calcular como sigue:

i	Reducción
$-$	$Q = 2X + 2$ $r = (3X^2 + 2X + 1) + 3Q$ $= 3X^2 + 3X + 2$
4	$Q = (2X + 2)X = (2X^2 + 2X) + 0P'$ $= 2X^2 + 2X$ $r = (3X^2 + 3X + 2) + 4Q$ $= X^2 + X + 2$
5	$Q = (2X^2 + 2X)X = (2X^2) + 2P'$ $= 2X^2 + 4X + 4$ $r = (X^2 + X + 2) + Q$ $= 3X^2 + 1$

Nótese que los resultados obtenidos por los algoritmos de división polinomial y de reducción, son iguales. ◊

Aritmética básica

Sean $f, g \in \mathbb{F}_q[X]/(P)$ de grado n, la suma se realiza de la misma manera que en la sección 3.1.1. Para esta operación no es necesario reducir módulo P, ya que $f + g$ es un polinomio de grado n, y entonces el costo de la suma es $O(n)$ operaciones en $\mathbb{F}_q$.

Sean $f, g \in \mathbb{F}_q[X]/(P)$ de grado n, la multiplicación se realiza de la misma manera que en la sección 3.1.1. Para esta operación se tienen dos escenarios:

a. Si $2n < \ell$, no es necesario reducir módulo P, y el costo de la multiplicación es $O(n^2)$ o $O(n^{\log_2 3})$ operaciones en $\mathbb{F}_q$, dependiendo del método empleado.

b. Si $2n \geq \ell$, el resultado se debe reducir módulo P, y el costo de la multiplicación es $O(\ell^2)$ operaciones en $\mathbb{F}_q$.

Cálculo de inverso multiplicativo

Sea $f \in \mathbb{F}_q[X]/(P)$ de grado n, utilizando el algoritmo extendido de Euclides con f y P, se obtienen los polinomios $d, s, t \in \mathbb{F}_q[X]$ tal que $d = \text{mcd}(f, P)$ y $fs + Pt = d$. Si $d \neq 1$, f no es primo relativo de P y por lo tanto no tiene inverso multiplicativo módulo P. Si $d = 1$, entonces s es el inverso multiplicativo de f módulo P. El costo del algoritmo extendido de Euclides con f y P es $O(\ell n)$ operaciones en $\mathbb{F}_q$, y por lo tanto, calcular el inverso multiplicativo tiene el mismo costo $O(\ell n)$ [38, Teorema 17.6].

Exponenciación

Sean $f \in \mathbb{F}_q[X]/(P)$ de grado n y $e > 0$, se puede calcular $f^e \in \mathbb{F}_q[X]/(P)$ utilizando el método de *exponenciación binaria*, que se muestra en el algoritmo 3.8. Sea $k = \log_2 e$, el algoritmo 3.8 realiza k cuadrados (multiplicaciones) y a lo más k multiplicaciones en $\mathbb{F}_q[X]/(P)$, entonces su costo es $O(n^2 \log e)$ operaciones en $\mathbb{F}_q$ [38].

Algoritmo 3.8 `ExponenciacionBinaria` [38]

Entrada:
$f \in \mathbb{F}_q[X]/(P)$
$e \in \mathbb{N}$
Salida:
$f^e \in \mathbb{F}_q[X](P)$
1: calcular la representación binaria $(b_k, \ldots, b_0)$ de e
2: $a \leftarrow f$
3: **para** $i \leftarrow k-1, \ldots, 0$ **hacer**
4: $a \leftarrow a^2 \bmod P$
5: **si** $b_i = 1$
6: $a \leftarrow af \bmod P$
7: **fin si**
8: **fin para**
9: **regresar** a

3.2. Aritmética en campos finitos

Un campo finito $\mathbb{F}_q$ es un *campo primo* si contiene $q = p$ elementos, donde p es un número primo. Si $\mathbb{F}_q$ no es un campo primo, entonces es una extensión de grado $n > 1$ de un campo primo $\mathbb{F}_p$, y contiene $q = p^n$ elementos. En esta sección p representará un número primo y q representará la potencia de un número primo.

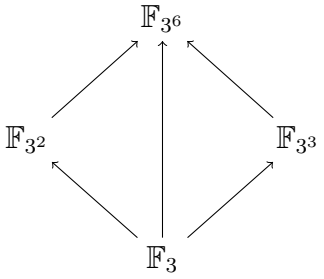
Sea $\mathbb{F}_q$ una extensión de grado m de un campo finito L, entonces los elementos de $\mathbb{F}_q$ pueden representarse como polinomios de grado $< m$ con coeficientes en L (véase sección 2.1.4). Asimismo, el campo L puede ser una extensión de grado ℓ de un campo finito K, y los elementos de L pueden representarse como polinomios de grado $< \ell$ con coeficientes en K. El subcampo de $\mathbb{F}_q$ sobre el cual se pueden comenzar a generar las extensiones anteriores, es el campo primo $\mathbb{F}_p$, donde p es la característica de $\mathbb{F}_q$, y esto se ilustra en la figura 3.1.

$$
\begin{array}{ll}
\mathbb{F}_q & [\mathbb{F}_q : L] = m, \quad \mathbb{F}_q = \{\gamma = \sum_{i=0}^{m-1} \gamma_i X^i : \gamma_i \in L\} \\
\uparrow & \\
L & [L : K] = \ell, \quad L = \{\beta = \sum_{i=0}^{\ell-1} \beta_i W^i : \beta_i \in K\} \\
\uparrow & \\
K & \vdots \\
\uparrow & \\
\vdots & \\
\uparrow & \\
J & [J : \mathbb{F}_p] = j, \quad J = \{\alpha = \sum_{i=0}^{j-1} \alpha_i U^i : \alpha_i \in \mathbb{F}_p\} \\
\uparrow & \\
\mathbb{F}_p & \mathbb{F}_p = \{0, 1, \ldots, p-1\}
\end{array}
$$

Figura 3.1: Extensiones para generar un campo finito. $\mathbb{F}_q$ es una extensión de un campo finito L, que a su vez es una extensión de otro campo finito K. Partiendo del campo primo $\mathbb{F}_p$, donde $p = \text{char}(\mathbb{F}_q)$, se pueden construir extensiones hasta obtener el campo $\mathbb{F}_q$.

Sea $q = p^n$, el campo $\mathbb{F}_q$ puede generarse de diversas maneras. La manera directa es generar una extensión de $\mathbb{F}_p$ de grado n. Otra manera es (i) seleccionar un número $a > 1$ tal que $a \mid n$, (ii) generar una extensión J de $\mathbb{F}_p$ de grado a, y (iii) $\mathbb{F}_q$ se genera como una extensión de J de grado $b = n/a$, es decir, primero se genera $J = \mathbb{F}_{p^a}$ y después $\mathbb{F}_q = \mathbb{F}_{(p^a)^b} = \mathbb{F}_{p^n}$. También es posible generar $J = \mathbb{F}_{p^b}$ y después $\mathbb{F}_q = \mathbb{F}_{(p^b)^a} = \mathbb{F}_{p^n}$. En términos generales, $\mathbb{F}_q$ se puede obtener generando extensiones, tal que el grado de cada extensión con respecto a su campo base, divide a n.

Ejemplo 3.2.1. El campo $\mathbb{F}_{3^6}$ se puede generar de las siguientes maneras:



$\Diamond$

Cuando un campo finito se genera a través de varias extensiones, como $\mathbb{F}_3 \to \mathbb{F}_{3^2} \to \mathbb{F}_{3^6}$ o $\mathbb{F}_3 \to \mathbb{F}_{3^3} \to \mathbb{F}_{3^6}$ en el ejemplo 3.2.1, se denomina *torre de campos* [4]. Existen técnicas específicas para realizar aritmética de manera eficiente con torres de campos, como las que se presentan en [4], [9], [37], [21] y [6].

En esta sección se discute sobre la aritmética en campos finitos con característica pequeña (2 o 3), y que además son una extensión pequeña de un campo primo (por ejemplo $n \leq 6$). Primero se presentan dos maneras de representar los elementos del campo, y posteriormente se muestran alternativas para realizar aritmética de acuerdo a la representación que se utilice. Las torres de campos y su aritmética están fuera del alcance de esta sección.

3.2.1. Representación polinomial

Sean $\mathbb{F}_p$ un campo primo y $\mathbb{F}_q$ una extensión de grado n de $\mathbb{F}_p$, los elementos de $\mathbb{F}_q$ son los polinomios de grado $< n$ con coeficientes en $\mathbb{F}_p$, debido a que $\mathbb{F}_q \cong \mathbb{F}_p[X]/(I)$ para algún polinomio irreducible $I \in \mathbb{F}_p[X]$ de grado n.

La representación polinomial consiste en expresar los elementos de $\mathbb{F}_q$ como un vector de elementos, es decir:

$$a = \sum_{i=0}^{n-1} a_i X^i \in \mathbb{F}_q \mapsto (a_{n-1}, \ldots, a_0).$$

Dado que $\mathbb{F}_q$ es isomorfo a un anillo cociente $\mathbb{F}_p[X]/(I)$, la aritmética en $\mathbb{F}_q$ es la aritmética polinomial módulo I (véase sección 3.1.3).

La aritmética polinomial módulo I no es siempre la alternativa más eficiente. Por ejemplo, cuando $\mathbb{F}_q$ tiene característica pequeña y es una extensión pequeña de un campo primo, la aritmética en $\mathbb{F}_q$ puede realizarse de forma más eficiente utilizando tablas de pre-cómputo. Esta alternativa se presenta en la sección 3.2.3.

3.2.2. Representación exponencial

Sea $g \in \mathbb{F}_q$ un generador de $\mathbb{F}_q^*$, cada elemento $a \in \mathbb{F}_q^*$ es una potencia de g, es decir, $a = g^i$ con $i \in \{0, \ldots, q-2\}$. Nótese que el elemento $0 \in \mathbb{F}_q$ no se puede representar como una potencia de g. Los elementos de $\mathbb{F}_q$ se representan de manera exponencial de acuerdo a la función $\psi : \mathbb{F}_q \to \{0, \ldots, q-2\} \cup \{\infty\}$, como sigue:

$$\psi(a) = \begin{cases} \infty & \text{si } a = 0, \\ i & \text{si } a = g^i. \end{cases}$$

La representación exponencial facilita el cálculo de las operaciones en $\mathbb{F}_q^*$. Para $\mathbb{F}_q^+$ es posible calcular el inverso aditivo, sin embargo, la suma no se puede realizar de manera directa. Sean $a = g^m, b = g^n \in \mathbb{F}_q$ y $e > 0$, las operaciones aritméticas, excepto la suma, se calculan con aritmética de enteros módulo $q-1$:

- **Multiplicación**. $a \cdot b = g^m \cdot g^n = g^{(m+n) \bmod q-1}$.
- **Inverso aditivo**. $-a = -1 \cdot g^m = g^{\alpha} g^m = g^{(\alpha+m) \bmod q-1}$, donde $\alpha = \frac{q-1}{2}$.
- **Inverso multiplicativo**. $a^{-1} = (g^m)^{-1} = g^{(-m) \bmod q-1}$.
- **Exponenciación**. $a^e = (g^m)^e = g^{(me) \bmod q-1}$.

En las operaciones anteriores se debe tratar como un caso especial cuando $a = 0$ o $b = 0$. En el cálculo del inverso aditivo, $\alpha = (q-1)/2$ por lo siguiente: g tiene orden $q-1$ por ser un generador, es decir, $g^{q-1} = 1 \Rightarrow \left(g^{(q-1)/2}\right)^2 = 1 \Rightarrow g^{(q-1)/2} = \pm 1 \Rightarrow g^{(q-1)/2} = -1$. Para realizar la suma, se tienen las siguientes opciones:

a. Pre-calcular una tabla para traducir entre representación exponencial y polinomial. Los sumandos se traducen a representación polinomial, la suma se realiza como una suma de polinomios y el resultado se traduce a representación exponencial.

b. Utilizar una tabla de pre-cómputo para la suma, como se explica en la sección 3.2.3.

c. Utilizar el logaritmo de Zech, como se explica en la sección 3.2.4.

En vez de utilizar las operaciones módulo $q-1$ anteriores, la aritmética en $\mathbb{F}_q$ con representación exponencial puede realizarse utilizando tablas de pre-cómputo, como se explica en la sección 3.2.3.

Ejemplo 3.2.2. Dado el campo $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$ y el generador $g = 3$, la representación exponencial es:

$$\begin{array}{llll} g^0 = 1 & g^1 = 3 & g^2 = 2 & g^3 = 6 \\ g^4 = 4 & g^5 = 5 & \infty = 0. & \end{array}$$

Dados $g^5, g^4 \in \mathbb{F}_7$ y $e = 10$, las operaciones aritméticas con g^5 y g^4 pueden calcularse como sigue:

- $g^5 \cdot g^4 = g^{(5+4) \bmod 6} = g^3$.
- $-g^5 = g^{(3+5) \bmod 6} = g^2$ y $-g^4 = g^{(3+4) \bmod 6} = g$.
- $(g^5)^{-1} = g^{-5 \bmod 6} = g$ y $(g^4)^{-1} = g^{-4 \bmod 6} = g^2$.
- $(g^5)^e = g^{50 \bmod 6} = g^2$ y $(g^4)^e = g^{40 \bmod 6} = g^4$.
- Traduciendo representaciones, $g^5 + g^4 = (5+4) \bmod 7 = 2 = g^2$. $\Diamond$

Ejemplo 3.2.3. Sea $\mathbb{F}_{3^2} \cong \mathbb{F}_3[X]/(I)$ con el polinomio irreducible $I = X^2 + X + 2$, entonces:

$$\mathbb{F}_{3^2} = \left\{ \begin{array}{ccc} 0, & 1, & 2 \\ X, & X{+}1, & X{+}2 \\ 2X, & 2X{+}1, & 2X{+}2 \end{array} \right\}.$$

Dados $\mathbb{F}_{3^2}$ y el generador $g = X$, la representación exponencial es:

$$\begin{array}{lll} g^0 = 1 & g^1 = X & g^2 = 2X + 1 \\ g^3 = 2X + 2 & g^4 = 2 & g^5 = 2X \\ g^6 = X + 2 & g^7 = X + 1 & \infty = 0. \end{array}$$

Dados $g^3, g^7 \in \mathbb{F}_{3^2}$ y $e = 10$, las operaciones aritméticas con g^3 y g^7 pueden calcularse como sigue:

- $g^3 \cdot g^7 = g^{(3+7) \bmod 8} = g^2$.
- $-g^3 = g^{(4+3) \bmod 8} = g^7$ y $-g^7 = g^{(4+7) \bmod 8} = g^3$.
- $(g^3)^{-1} = g^{-3 \bmod 8} = g^5$ y $(g^7)^{-1} = g^{-7 \bmod 8} = g$.
- $(g^3)^e = g^{30 \bmod 8} = g^6$ y $(g^7)^e = g^{70 \bmod 8} = g^6$.
- Traduciendo representaciones, $g^3 + g^7 = ((2X{+}2) + (X{+}1)) = 0 = \infty$. $\Diamond$

3.2.3. Pre-cómputo

La aritmética en un campo finito que es una extensión pequeña (por ejemplo ≤ 6) de un campo primo puede realizarse utilizando tablas de pre-cómputo. Estas tablas contienen el resultado de las operaciones aritméticas aplicadas entre todos los elementos del campo, y "calcular" el resultado de alguna operación, consiste en hacer una búsqueda en la tabla de la operación correspondiente. Este método se puede utilizar tanto con representación polinomial como con representación exponencial.

Ejemplo 3.2.4. Sea $\mathbb{F}_{3^2}$ como en el ejemplo 3.2.3, la tabla de suma para la representación

exponencial es la siguiente:

$+$	∞	g^0	g^1	g^2	g^3	g^4	g^5	g^6	g^7
∞	∞	g^0	g^1	g^2	g^3	g^4	g^5	g^6	g^7
g^0	g^0	g^4	g^7	g^3	g^5	∞	g^2	g^1	g^6
g^1	g^1	g^7	g^5	g^0	g^4	g^6	∞	g^3	g^2
g^2	g^2	g^3	g^0	g^6	g^1	g^5	g^7	∞	g^4
g^3	g^3	g^5	g^4	g^1	g^7	g^2	g^6	g^0	∞
g^4	g^4	∞	g^6	g^5	g^2	g^0	g^3	g^7	g^1
g^5	g^5	g^2	∞	g^7	g^6	g^3	g^1	g^4	g^0
g^6	g^6	g^1	g^3	∞	g^0	g^7	g^4	g^2	g^5
g^7	g^7	g^6	g^2	g^4	∞	g^1	g^0	g^5	g^3

y la tabla de suma para la representación polinomial es la siguiente:

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

$\Diamond$

Dado un campo $\mathbb{F}_q$, las tablas de suma y multiplicación requieren espacio para almacenar q^2 elementos de $\mathbb{F}_q$, cada una. Las tablas de inversos aditivos e inversos multiplicativos requieren espacio para almacenar q elementos de $\mathbb{F}_q$, cada una. Ambas representaciones tienen los mismos requerimientos de espacio. La exponenciación se puede realizar utilizando el método binario (algoritmo 3.8), donde las operaciones requeridas se calculan con la tabla de multiplicación.

3.2.4. Logaritmo de Zech

El logaritmo de Zech (también conocido como logaritmo de Jacobi, por [23]) es un método para calcular la suma en campos finitos donde los elementos se representan como potencias de un generador [33], [34]. La idea de este método es calcular la suma utilizando una multiplicación y una tabla de pre-cómputo.

Sea $\mathbb{F}_q$ un campo finito cuyos elementos se representan de manera exponencial con base g, entonces $g^{Z(m)} = 1 + g^m$, donde Z es una función en los enteros módulo $q-1$. Sean $a = g^m, b = g^n \in \mathbb{F}_q$, la suma y resta de estos elementos se puede calcular de la siguiente manera:

$$a + b = g^m + g^n = g^m g^{Z(n-m)} = g^m(1 + g^{(n-m) \bmod q-1}),$$
$$a - b = g^m + (-g^n) = g^m g^{Z(\alpha+n-m)} = g^m(1 + g^{(\alpha+n-m) \bmod q-1}),$$

donde $\alpha = (q-1)/2$ (véase sección 3.2.2). Nótese que $g^{Z(\alpha)} = (1 + g^\alpha) = 0 = \infty$. Al igual que con las demás operaciones con representación exponencial, se debe tratar como un caso especial cuando $a = 0$ o $b = 0$.

Todos los valores de $g^{Z(i)}$, con $i \in \{0, \ldots, q-2\}$, más la representación de 0 se almacenan en una tabla de pre-cómputo, entonces se requiere espacio para almacenar q elementos de $\mathbb{F}_q$.

Ejemplo 3.2.5. Sea $\mathbb{F}_{3^2}$ como en el ejemplo 3.2.3, la tabla del logaritmo de Zech es la siguiente:

$g^{Z(0)}$	$g^{Z(1)}$	$g^{Z(2)}$	$g^{Z(3)}$	$g^{Z(4)}$	$g^{Z(5)}$	$g^{Z(6)}$	$g^{Z(7)}$	∞
g^4	g^7	g^3	g^5	∞	g^2	g^1	g^6	0

Tomando el elemento g, se muestra el cálculo de la suma con los demás elementos:

$$g + g^0 = gg^{Z(7)} = gg^6 = g^7, \qquad g + g^1 = gg^{Z(0)} = gg^4 = g^5,$$
$$g + g^2 = gg^{Z(1)} = gg^7 = g^0, \qquad g + g^3 = gg^{Z(2)} = gg^3 = g^4,$$
$$g + g^4 = gg^{Z(3)} = gg^5 = g^6, \qquad g + g^5 = gg^{Z(4)} = g\infty = \infty,$$
$$g + g^6 = gg^{Z(5)} = gg^2 = g^3, \qquad g + g^7 = gg^{Z(6)} = gg^1 = g^2,$$
$$g + \infty = g.$$

$\Diamond$

CAPÍTULO 4

De la suavidad de polinomios

En [3], Leonard Adleman presentó un algoritmo para calcular el logaritmo discreto en un campo $\mathbb{F}_p$, con p un número primo, y allí define cuando un número natural es *suave* con respecto a un límite dado. Adleman menciona que Ronald Rivest[1] sugirió el uso del término suave. En [22], Hellman y Reyneri generalizaron el algoritmo de Adleman para $\mathbb{F}_q$, con $q = p^m$, definiendo también la suavidad en los polinomios.

Las definiciones de suavidad para números naturales y polinomios sobre campos finitos se presentan en la sección 4.1, allí también se muestran algunos resultados conocidos respecto a polinomios suaves. La *factorización* y *prueba de suavidad* son dos alternativas para determinar si un polinomio es suave o no respecto a un límite dado, y se presentan en las secciones 4.2 y 4.3, respectivamente. Al final del capítulo se hace una breve discusión en relación al uso de una u otra alternativa al calcular el logaritmo discreto.

[1] Ronald Rivest es un matemático y criptógrafo estadounidense, conocido principalmente por proponer el criptosistema RSA junto con Adi Shamir y Leonard Adleman.

4.1. Preliminares

El *Teorema fundamental de la aritmética* afirma que cualquier número entero positivo mayor que 1 puede ser expresado de forma única como un producto de números primos, y este producto es único salvo en el orden de los factores [16].

Definición 4.1.1 (Suavidad de un número entero)**.** Dados $a = \prod_i p_i^{e_i} \in \mathbb{N}$ y $m \geq 1$, a es *m-suave* si para todo $p_i, p_i \leq m$, donde p_i son los factores primos de a con multiplicidad $e_i \geq 1$.

Ejemplo 4.1.1. Dados los números $2015, 27000$ y las potencias de 2:

- El número $2015 = 5 \cdot 13 \cdot 31$ es 31-suave.
- El número $27000 = 2^3 \cdot 3^3 \cdot 5^3$ es 5-suave.
- Todas las potencias de 2 son 2-suaves. ◊

Nótese que si p_k es el factor primo más grande de un número a, entonces a es m-suave para cualquier $m \geq p_k$.

De manera análoga a los números enteros, un polinomio puede expresarse de forma única como un producto de factores:

Teorema 4.1.2 (Factorización única en anillos de polinomios [33, Teorema 1.59])**.** *Dado un campo finito $\mathbb{F}_q$, cualquier polinomio $f \in \mathbb{F}_q[X]$ de grado mayor que cero puede ser expresado de forma:*

$$f = a f_1^{e_1} \dots f_k^{e_k},$$

donde $a \in \mathbb{F}_q$, $f_1, \dots, f_k$ son polinomios mónicos irreducibles en $\mathbb{F}_q[X]$ distintos a pares, y $e_1, \dots, e_k$ son enteros positivos. Esta factorización es única salvo en el orden de los factores.

Definición 4.1.3 (Suavidad de un polinomio)**.** Dados $f = a \prod_i f_i^{e_i} \in \mathbb{F}_q[X]$ y $m \geq 1$, f es *m-suave* si para todo $f_i, \text{grad}(f_i) \leq m$, donde $a \in \mathbb{F}_q$ y f_i son los factores irreducibles de f con multiplicidad $e_i \geq 1$.

Ejemplo 4.1.2. Dados $f, g, h \in \mathbb{F}_5[X]$, con $f = X^5$, $g = X^{10} + 2X^9 + X^7 + 2X^6$ y $h = 3X^3 + X^2 + 3X + 1$:

- $f = (X)^5$ es 1-suave
- $g = (X^2 + 4X + 1) \cdot (X + 2) \cdot (X + 1) \cdot (X)^6$ es 2-suave.
- $h = 3 \cdot (X + 2)^2 \cdot (X + 3)$ es 1-suave ◊

Nótese que si f_k es el factor irreducible con grado mayor de un polinomio f, entonces f es m-suave para cualquier $m \geq \text{grad}(f_k)$.

Como se mencionó en la sección 2.1.4, un polinomio siempre se puede expresar en términos de un polinomio mónico y una unidad, por lo que la exposición en el resto del capítulo se hará suponiendo que se trabaja con polinomios mónicos.

A continuación se presentan algunos resultados conocidos de los polinomios suaves sobre un campo finito [2], [15]. El número de polinomios mónicos de grado n sobre un campo $\mathbb{F}_q$ es q^n. El número de polinomios mónicos irreducibles de grado n sobre $\mathbb{F}_q$ es:

$$I_q(n) = \frac{1}{n} \sum_{d|n} \mu(n/d) q^d,$$

donde μ es la función de Möbius:

$$\mu(a) = \begin{cases} 0 & \text{si } a \text{ tiene uno o más factores primos repetidos,} \\ 1 & \text{si } a = 1, \\ (-1)^k & \text{si } a \text{ es el producto de } k \text{ distintos factores primos.} \end{cases}$$

La función generadora para polinomios mónicos m-suaves en $\mathbb{F}_q[X]$, está dada por:

$$F(u, z) = \prod_{\ell=1}^{m} \left(1 + \frac{uz^\ell}{1 - z^\ell}\right)^{I_q(\ell)},$$

donde u indica el número de factores irreducibles distintos y z indica el grado del polinomio. El número de polinomios mónicos m-suaves de grado n en $\mathbb{F}_q[X]$ que tienen exactamente k factores mónicos irreducibles distintos, es:

$$N_q(m, n, k) = [u^k z^n] F(u, z),$$

donde $[\cdot]$ denota el operador coeficiente. De lo anterior, el número de polinomios mónicos m-suaves de grado n en $\mathbb{F}_q[X]$ es:

$$N_q(m, n) = [z^n] F(1, z).$$

Finalmente, el número promedio de factores mónicos irreducibles distintos entre todos los polinomios mónicos m-suaves de grado n en $\mathbb{F}_q[X]$, es:

$$A_q(m, n) = \frac{[z^n] \left(\frac{\partial F}{\partial u}\big|_{u=1}\right)}{N_q(m, n)}.$$

Dados q, m y n, los valores de $N_q(m, n)$ y $A_q(m, n)$ se puede obtener de manera eficiente utilizando un sistema algebraico computacional, con el cual se calculan los primeros $n + 1$ términos de la expansión de series de Taylor de $F(1, z)$ y después se extrae el coeficiente de z^n [2].

4.2. Factorización

Sea $\mathbb{F}_q$ un campo finito, el problema de *factorización* consiste en expresar un polinomio mónico $f \in \mathbb{F}_q[X]$ de grado n como $f = f_1^{e_1} \dots f_k^{e_k}$, donde cada factor f_i es un polinomio mónico irreducible con multiplicidad $e_i \geq 1$, y $f_1, \dots, f_k$ son primos relativos a pares.

Definición 4.2.1 (Polinomio libre de cuadrados)**.** Un polinomio $f \in \mathbb{F}_q[X]$ es *libre de cuadrados* si para cualquier $g \in \mathbb{F}_q[X]$ con grad$(g) \geq 1$, g^2 no divide a h.

El proceso general de factorización se realiza en fases:

1. Primero, f se reemplaza por un polinomio *libre de cuadrados*. Para esta fase se tienen dos alternativas:

   a) *Eliminación de factores repetidos* (EFR). Se obtiene un polinomio libre de cuadrados que contiene todos los factores ireducibles de f, pero con multiplicidad uno.

   b) *Descomposición libre de cuadrados* (DLC). Se obtiene el conjunto de polinomios $g_1, \dots, g_\ell$, tal que $f = g_1 g_2^2 \dots g_\ell^\ell$, donde $g_\ell \neq 1$ y cada g_i es mónico, libre de cuadrados, primo relativo con los demás $g_j, j \neq i$, y $g_i = \prod_{e_j = i} f_j$, es decir, cada g_i es el producto de los factores irreducibles que dividen a f cuando están elevados a la potencia i.

   En [15] se demuestra que la diferencia en el costo total de la factorización es mínima si se utiliza EFR o DLC. Aquí se utilizará EFR.

2. *Factorización de grado distinto* (FGD). En la segunda fase, un polinomio libre de cuadrados se separa en polinomios cuyos factores irreducibles tienen todos el mismo grado.

3. *Factorización de grado igual* (FGI). En la última fase, se separa completamente los polinomios cuyos factores irreducibles tienen todos el mismo grado.

El costo de factorizar un polinomio mónico $f \in \mathbb{F}_q[X]$ de grado n se mide en número de operaciones en $\mathbb{F}_q$. Se supondrá que el costo de cualquier operación en $\mathbb{F}_q$ es $O(1)$. Cuando se realizan operaciones con polinomios de grado $\leq n$, el costo de una suma es $O(n)$, mientras que el costo de una multiplicación, división y cálculo del máximo común divisor es $O(n^2)$.

El algoritmo 4.1 muestra el proceso general descrito previamente. Los costos de EFR, FGD y FGI son $O(n^2)$, $O(n^3 \log q)$ y $O(n^2 \log q)$ operaciones en $\mathbb{F}_q$, respectivamente. El costo total del algoritmo 4.1 es $O(n^3 \log q)$ operaciones en $\mathbb{F}_q$ [15].

Algoritmo 4.1 `Factorizacion` [15]

Entrada:

$f \in \mathbb{F}_q[X]$, polinomio mónico de grado n

Salida:

$f_1, \ldots, f_k \in \mathbb{F}_q[X]$, factores irreducibles de f

1: $a \leftarrow$ `EFR(`f`)`
2: $b_1, \ldots, b_\ell \leftarrow$ `FGD(`a`)`
3: $F \leftarrow 1$
4: **para** $i \leftarrow 1$ **hasta** ℓ **hacer**
5: $\quad F \leftarrow F \cup$ `FGI(`b_i`,` i`)`
6: **fin para**
7: **regresar** $F \cup$ `Factorizacion(`f/a`)`

4.2.1. Eliminación de factores repetidos

La primera fase para factorizar un polinomio mónico $f = f_1^{e_1} \ldots f_k^{e_k} \in \mathbb{F}_q[X]$ de grado n, consiste en obtener un polinomio libre de cuadrados que contiene a todos los factores irreducibles de f, pero con multiplicidad uno, es decir:

$$f \mapsto a = \prod_{i=1}^{k} f_i.$$

El algoritmo 4.2 es un algoritmo determinista que muestra el proceso para eliminar los factores repetidos, su correcto funcionamiento se basa en la siguiente propiedad: El máximo común divisor de un polinomio $f \in \mathbb{F}_q[X]$ y su derivada f', extrae todos los factores repetidos de f [33, Sección 4.1]. El costo del algoritmo 4.2 es igual al costo de calcular el máximo común divisor entre un polinomio de grado n y otro de grado a lo más n, entonces el costo de EFR es $O(n^2)$ operaciones en $\mathbb{F}_q$ [15].

4.2.2. Factorización de grado distinto

En esta fase, un polinomio libre de cuadrados a se separa en polinomios cuyos factores irreducibles tienen todos el mismo grado, es decir:

$$a \mapsto b_1 b_2 \ldots b_\ell \quad \text{tal que } b_i = \prod_{\text{grad}(f_j)=i} f_j.$$

El algoritmo 4.3 es un algoritmo determinista que muestra el proceso para realizar la factorización de grado distinto, su correcto funcionamiento se basa en la siguiente pro-

Algoritmo 4.2 EFR [15]

Entrada:

$f \in \mathbb{F}_q[X]$, polinomio mónico de grado n

Salida:

$a \in \mathbb{F}_q[X]$, polinomio libre de cuadrados

1: $g \leftarrow \text{mcd}(f, f')$
2: $a \leftarrow f/g$
3: $k \leftarrow \text{mcd}(g, a)$
4: **mientras** $k \neq 1$ **hacer**
5: $g \leftarrow g/k$
6: $k \leftarrow \text{mcd}(g, a)$
7: **fin mientras**
8: **si** $g \neq 1$
9: $a \leftarrow a * \texttt{EFR}(g^{1/p})$
10: **fin si**
11: **regresar** a

piedad: Para $i \geq 1$, el polinomio $X^{q^i} - X \in \mathbb{F}_q[X]$ es el producto de todos los polinomios mónicos irreducibles en $\mathbb{F}_q[X]$ cuyo grado divide a i [33, Teorema 3.20]. El costo del algoritmo 4.3 es $O(n^3 \log q)$ operaciones en $\mathbb{F}_q$ [15].

4.2.3. Factorización de grado igual

Después de las fases anteriores, solamente resta procesar un conjunto de polinomios mónicos libres de cuadrados, tal que cada polinomio está compuesto por factores que tienen todos el mismo grado. En esta fase, los polinomios que tienen esta característica se separan completamente en sus factores irreducibles, es decir:

$$b_i \mapsto b_{i,1} b_{i,2} \ldots b_{i,\ell} \quad \text{tal que grad}(b_{i,j}) = i.$$

El algoritmo 4.4 es un algoritmo probabilista que muestra el proceso para realizar la factorización de grado igual, su correcto funcionamiento se basa en la siguiente propiedad: Sea $b = b_1 b_2 \ldots b_\ell \in \mathbb{F}_q[X]$, donde todos los b_j son polinomios irreducibles de grado i y primos relativos a pares, se tiene que:

$$\mathbb{F}_q[X]/(b) \cong \mathbb{F}_q[X]/(b_1) \times \cdots \times \mathbb{F}_q[X]/(b_\ell),$$

y entonces un elemento $h \in \mathbb{F}_q[X]/(b)$ está asociado a una ℓ-tupla $(h_1, \ldots, h_\ell)$, donde cada h_j es un elemento de $\mathbb{F}_q[X]/(b_j)$ [38, Sección 16.4]. El costo del algoritmo 4.4 es $O(n^2 \log q)$ operaciones en $\mathbb{F}_q$ [15].

Algoritmo 4.3 `FGD` [15]

Entrada:

$a \in \mathbb{F}_q[X]$, polinomio libre de cuadrados

Salida:

$b_1, b_2, \ldots, b_n \in \mathbb{F}_q[X]$, tal que b_i es el producto de factores irreducibles de grado i

1: $d \leftarrow \text{grad}(a)$
2: $g \leftarrow a$
3: $h \leftarrow X$
4: **para** $i \leftarrow 1, \ldots, d$ **hacer**
5: $\quad h \leftarrow h^q \bmod g$
6: $\quad b_i \leftarrow \text{mcd}(h - X, g)$
7: $\quad g \leftarrow g/b_i$
8: $\quad$ **si** $b_i \neq 1$
9: $\quad\quad h \leftarrow h \bmod g$
10: $\quad$ **fin si**
11: **fin para**
12: **regresar** $b_1, b_2, \ldots, b_n$

Algoritmo 4.4 `FGI` [15]

Entrada:

$b \in \mathbb{F}_q[X]$, polinomio cuyos factores irreducibles tienen todos grado i
i, grado de los factores irreducibles de b

Salida:

$b_1, b_2, \ldots, b_\ell \in \mathbb{F}_q[X]$, tal que cada b_j tiene grado i

1: **si** $\text{grad}(b) \leq i$
2: $\quad$ **regresar** b
3: **fin si**
4: $h \leftarrow$ `PolinomioAleatorio(`$\text{grad}(b) - 1$`)`
5: $a \leftarrow h^{\frac{q^i - 1}{2}} - 1 \bmod b$
6: $d \leftarrow \text{mcd}(a, b)$
7: **regresar** `FGI(`d, i`),FGI(`$b/d, i$`)`

4.2.4. Estado del arte

Berlekamp [5] presentó el primer algoritmo probabilista de tiempo polinomial para resolver el problema de factorización. La solución a este problema se obtiene calculando el espacio nulo de una matriz definida sobre $\mathbb{F}_q$. El algoritmo de Berlekamp utiliza $O(n^\omega + n^2 \log n \log q)$ operaciones en $\mathbb{F}_q$, siendo $O(n^\omega)$ el número de operaciones requerido para multiplicar dos matrices de $n \times n$ (se supondrá que $2 < \omega \leq 3$).

Cantor y Zassenhaus [8] presentaron un algoritmo de factorización que se ejecuta en dos fases: la primera fase es la factorización de grado distinto y la segunda fase es la factorización de grado igual; se supone que el polinomio a factorizar es libre de cuadrados. El algoritmo de Cantor-Zassenhaus utiliza $O(n^3 \log q)$ operaciones en $\mathbb{F}_q$.

El algoritmo de Berlekamp es más rápido que el de Cantor-Zassenhaus cuando q es más grande. La desventaja del algoritmo de Berlekamp es que requiere almacenamiento para $O(n^2)$ elementos de $\mathbb{F}_q$, mientras que el de Cantor-Zassenhaus solamente para $O(n)$ elementos de $\mathbb{F}_q$.

El proceso general de factorización que se muestra en el algoritmo 4.1 está basado en el algoritmo de Cantor-Zassenhaus, y tiene un costo de $O(n^3 \log q)$ operaciones en $\mathbb{F}_q$.

En [28], Kaltofen y Shoup presentan el primer algoritmo sub-cuadrático de factorización, aunque no resulta ser práctico debido a las técnicas de multiplicación de matrices que utiliza [41]. Shoup presenta en [39] una implementación del algoritmo anterior utilizando $O(n^{2.5} + n^{1+o(1)} \log q)$ operaciones en $\mathbb{F}_q$.

Actualmente, los algoritmos de factorización más eficientes son los de von zur Gathen, Kaltofen y Shoup. El algoritmo de von zur Gathen y Shoup [42] utiliza $O((n^2 + n \log q) \cdot (\log n)^2 \log \log n)$ operaciones en $\mathbb{F}_q$. El algoritmo de Kaltofen y Shoup [28] utiliza $O(n^{1.815} \log q)$ operaciones en $\mathbb{F}_q$ para realizar la factorización de grado distinto.

En [41], von zur Gahten y Panario muestran un extenso estudio de algoritmos para factorizar polinomios sobre campos finitos. Allí se mencionan y comparan de manera asintótica diversos algoritmos, también se presentan resultados de algunas implementaciones que resultan ser muy eficientes.

4.3. Prueba de suavidad

Teorema 4.3.1 (Prueba de suavidad [10], [2]). *Sea $\mathbb{F}_q$ un campo finito, se puede determinar si un polinomio mónico $f \in \mathbb{F}_q[X]$ de grado n es m-suave calculando:*

$$w = f' \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X) \bmod f,$$

y revisando si $w = 0$, donde f' es la derivada formal de f. Si $w \neq 0$, con certeza f no es m-suave. Si $w = 0$, f se puede declarar m-suave, ya que puede ocurrir que $w = 0$ cuando f no es m-suave.

Demostración. Se sabe que los factores de $X^{q^i} - X$ son todos los polinomios mónicos irreducibles cuyo grado divide a i [33, Teorema 3.20]. Sea $P = \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X)$, iterando i desde $\lceil m/2 \rceil$ hasta m, los factores de P son todos los polinomios mónicos irreducibles de grado $\leq m$. Los factores de f' son todos los factores irreducibles de f, pero con su multiplicidad reducida en 1. De lo anterior, $f'P$ contiene a todos los factores irreducibles de f de grado $\leq m$ con la misma multiplicidad que tienen en f.

Primero se supondrá que f es m-suave. Si $w = 0$, significa que $f'P$ es múltiplo de f, y entonces todos los factores irreducibles de f aparecen en $f'P$ con la misma multiplicidad, dado que esto se cumple solamente para los factores irreducibles de grado $\leq m$, la prueba genera una respuesta correcta. Debido a lo anterior, nunca puede ocurrir que $w \neq 0$ cuando f es m-suave.

Ahora se supondrá que f no es m-suave. Si $w \neq 0$, significa que $f'P$ no es múltiplo de f, y entonces hay al menos un factor irreducible f_j de f que no está en $f'P$, por lo tanto f_j tiene grado $> m$ y la prueba genera una respuesta correcta.

Ahora se comentará acerca de la falibilidad de la prueba. Suponiendo que $w = 0$ y $f = \prod_i f_i^{e_i}$ no es m-suave, se define:

$$I_1 = \{i \mid \text{grad}(f_i) \leq m\}, \quad f_{I_1} = \prod_{i \in I_1} f_i^{e_i},$$
$$I_2 = \{i \mid \text{grad}(f_i) > m\}, \quad f_{I_2} = \prod_{i \in I_2} f_i^{e_i},$$

por lo tanto $f = f_{I_1} f_{I_2}$, $f' = f_{I_1} f'_{I_2} + f'_{I_1} f_{I_2}$ y $w = (f_{I_1} f'_{I_2} P + f'_{I_1} f_{I_2} P) \bmod f$. Dado que f_{I_1} es m-suave, $f'_{I_1} P \equiv 0 \pmod{f_{I_1}}$, y multiplicando ambos lados por f_{I_2} se obtiene $f'_{I_1} f_{I_2} P \equiv 0 \pmod{f}$. Entonces, $w = 0 \Rightarrow f_{I_1} f'_{I_2} P \equiv 0 \pmod{f}$. Dado que $f_{I_1} P$ y f_{I_2} no tienen factores comunes, $f'_{I_2} \equiv 0 \pmod{f_{I_2}}$ y entonces $f'_{I_2} = 0$ ya que $\text{grad}(f'_{I_2}) < \text{grad}(f_{I_2})$. Por lo tanto, para tener $w = 0$ cuando f no es m-suave, se debe cumplir que $f'_{I_2} = 0$, y esto ocurre cuando la multiplicidad de todos los factores irreducibles de f con grado $> m$ es un múltiplo de la característica de $\mathbb{F}_q$. □

Dada la condición que un polinomio debe satisfacer para que la prueba de suavidad falle, el resultado de esta prueba es correcta con probabilidad extremadamente alta.

El cómputo de los valores X^{q^i} para $i \in \{\lceil m/2 \rceil, \ldots, m\}$, es el cálculo más costoso de la prueba de suavidad. En la actualidad, el procedimiento de Granger et al. [18] es el más eficiente para realizar el cálculo de dicha prueba, como se presenta a continuación.

Dado $f \in \mathbb{F}_q[X]$ de grado n y sea R el anillo cociente $\mathbb{F}_q[X]/(f)$, se denotará como $[a(X)]$ a una clase de residuos en R. Primero se explica como calcular alguna potencia $[X^{p^{rs}}]$, donde p es la característica de $\mathbb{F}_q$, y posteriormente se explica como calcular las potencias $[X^{q^i}]$.

1. **Cálculo de potencias $[X^{p^{rs}}]$.** Primero se calcula $[X^{p^r}], [X^{2p^r}], \ldots, [X^{(n-1)p^r}]$ multiplicando consecutivamente por $[X]$. Con estos valores se puede calcular una potencia p^r en R, es decir, aplicar $\phi : R \to R, \alpha \mapsto \alpha^{p^r}$ como sigue:

$$\left[\sum_{i=0}^{n-1} a_i X^i\right]^{p^r} = \sum_{i=0}^{n-1} a_i^{p^r} \left[X^{ip^r}\right].$$

   Nótese que todos los $[X^{ip^r}]$ fueron pre-computados. Para calcular alguna potencia $[X^{p^{rs}}]$, el mapa ϕ se aplica repetidamente, es decir, $[X^{p^{ri}}] = \phi^{i-1}([X^{p^r}])$ para $i \in \{2, \ldots, s\}$.

2. **Cálculo de potencias $[X^{q^i}]$.** Sea $q = p^{rs}$, aplicando repetidamente el mapa ϕ se calcula $[X^{p^{rj}}]$ para $j \in \{2, \ldots, sm\}$, y por lo tanto se obtiene $[X^{q^i}] = [X^{p^{rsi}}]$ para $i \in \{1, \ldots, m\}$.

En [18], Granger et al. reportan que este procedimiento utiliza $n^2(p^r + sm)$ multiplicaciones en $\mathbb{F}_q$. Para realizar una comparación justa con la factorización, aquí se calcula el número de operaciones en $\mathbb{F}_q$ que utiliza el procedimiento de Granger et al., siguiedo el mismo análisis con el que determinaron el número de multiplicaciones en $\mathbb{F}_q$.

Para calcular $[X^{p^r}], \ldots, [X^{(n-1)p^r}]$, cada multiplicación consecutiva por $[X]$ es vista como un corrimiento. Se requiere de $(n-1)(p^r-1)$ corrimientos, cada uno utilizando n multiplicaciones y n sumas en $\mathbb{F}_q$, en total poco menos de $2n^2p^r$ operaciones en $\mathbb{F}_q$.

El mapa ϕ requiere n exponenciaciones en $\mathbb{F}_q$, n multiplicaciones escalares (es decir, el proceso de multiplicar un elemento de $\mathbb{F}_q$ por un polinomio de grado $< n$) y n^2 sumas en $\mathbb{F}_q$, es decir, el mapa ϕ requiere $2n^2 + n$ operaciones en $\mathbb{F}_q$. Para calcular las potencias $[X^{q^i}]$, el mapa ϕ se aplica $sm - 1$ veces, en total poco menos de $2n^2sm + nsm$ operaciones en $\mathbb{F}_q$.

Al igual que en la sección 4.2, el costo de multiplicar y dividir dos polinomios de grado n es $O(n^2)$ operaciones en $\mathbb{F}_q$, por lo que una multiplicación en R (es decir, multiplicar dos polinomios de grado $< n$ módulo f) tiene un costo de $O(2n^2)$. El producto $f' \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X) \bmod f$ se calcula con $\lceil m/2 \rceil$ multiplicaciones en R, en total $O(n^2m)$ operaciones en $\mathbb{F}_q$. Finalmente, la prueba de suavidad se calcula con $2n^2p^r + 2n^2sm + nsm + O(n^2m) = O(n^2(p^r + sm))$ operaciones en $\mathbb{F}_q$.

Factorización o prueba de suavidad

Supóngase que dado un polinomio $f \in \mathbb{F}_q[X]$ se pregunta si f es m-suave, para algún valor $m \geq 1$. Nótese que no se requiere conocer la suavidad exacta de f, simplemente se desea saber si f es m-suave o no. Para solucionar este problema se tienen dos opciones. La primera opción es factorizar a f y posteriormente verificar si estos factores tienen grado menor o igual que m. La segunda opción es calcular la prueba de suavidad a f. El problema de la primera estrategia es que la factorización tiene un costo mayor comparado con el de la prueba de suavidad, mientras que la desventaja de la prueba de suavidad es que ésta puede fallar (dando falsos positivos, mas no falsos negativos), sin embargo, esto ocurre con probabilidad extremadamente baja.

Como se mencionó en la sección 2.4, el ojetivo de la fase de descenso (del algoritmo de cálculo de índices) es expresar un polinomio h en términos de aquellos que conforman la base de factores. La fase de descenso, que se discutirá en el siguiente capítulo, se ejecuta a través de una combinación de diversos algoritmos y estrategias, y determinar si un polinomio es suave con respecto a un valor m, es una "operación" muy utilizada, de hecho es la condición que determina el fin de su ejecución; además, el costo del descenso depende directamente del costo de esta operación. Por lo anterior, es de suma importancia optimizar al máximo la eficiencia con la que se determina si un polinomio es m-suave.

En el siguiente capítulo se explica con detalle la fase de descenso, la cual utiliza la prueba de suavidad debido a que su costo es menor comparado con el costo de la factorización. Al final de cada etapa del descenso se realiza una factorización para continuar con la siguiente etapa, por lo que se pueden identificar los polinomios que llegaran a ser declarados m-suaves y no lo sean (con probabilidad muy baja).

CAPÍTULO 5

Cálculo del logaritmo discreto

En la sección 2.4 se presentó el algorítmo de cálculo de índices, explicando brevemente las fases que lo componen. Este algoritmo tiene complejidad subexponencial en su tiempo de ejecución, y desde que Adleman lo planteó en 1979 [3], se han propuesto mejoras en sus fases con el objetivo de disminuir la complejidad total.

En [1], Adj et al. muestran una estimación del costo de calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$, utilizando el algoritmo de complejidad subexponencial $L[\frac{1}{4}+o(1)]$ propuesto por Joux en 2013 [24]. La motivación de realizar el cálculo en el campo $\mathbb{F}_{3^{6\cdot 509}}$ se presenta en [2]. Debido a la estimación favorable obtenida por Adj et al., se realizó una implementación de la prueba de suavidad, la cual se utiliza en la fase de descenso del cálculo mencionado anteriormente.

En la sección 5.1 se describe el algoritmo empleado para calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$. El algoritmo no se explica de manera detallada, solamente se expone la idea general de su funcionamiento. En las secciones 5.2 y 5.3 se explica el descenso de fracciones continuas y descenso clásico, respectivamente. Estos pasos de la fase de descenso hacen uso de la implementación desarrollada en este trabajo. Finalmente, los valores específicos para el cálculo del logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$, se muestran en la sección 5.4.

5.1. Vista general del algoritmo

El algoritmo empleado para calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$ es el presentado por Joux [24], con el cómputo de la base de factores de Joux y Pierrot [27].

Definiciones

Sea p un número primo pequeño (por ejemplo 2 o 3), se define $q = p^l$ con $l \geq 1$ y $n \leq q + 2$. Sean $N = q^n - 1$ y r un factor primo de N, se desea calcular el logaritmo discreto en el subgrupo de $\mathbb{F}_{q^n}^*$ de orden r. Lo anterior significa que dados un elemento generador g de $\mathbb{F}_{q^n}^*$ y un elemento h de $\mathbb{F}_{q^n}^*$ de orden r, se desea calcular $\log_g h$.

El número de polinomios mónicos en $\mathbb{F}_q[X]$ de grado d que son m suaves se denota como $N_q(m, d)$, el número promedio de factores mónicos irreducibles distintos entre todos los polinomios m-suaves se denota como $A_q(m, d)$ y $S_q(m, d)$ representa el costo de realizar una prueba de m-suavidad a un polinomio en $\mathbb{F}_q[X]$ de grado d. La manera de calcular $N_q(m, d)$ y $A_q(m, d)$ se presenta en la sección 4.1, y el costo de $S_q(m, d)$ se muestra en la sección 4.3.

Configuración inicial

Seleccionar dos polinomios $h_0, h_1 \in \mathbb{F}_q[X]$, con $\text{grad}(h_0) = 1$ y $\text{grad}(h_1) = 2$, tal que $h_1 X^q - h_0$ tenga un factor irreducible I_X de grado n, y entonces:

$$X^q \equiv \frac{h_0}{h_1} \pmod{I_X} \tag{5.1}$$

El campo $\mathbb{F}_{q^n}$ se representa como:

$$\mathbb{F}_{q^n} = \mathbb{F}_q[X]/(I_X)$$

y los elementos de $\mathbb{F}_{q^n}$ se representan como polinomios en $\mathbb{F}_q[X]$ con grado a lo más $n - 1$.

Base de factores, generación de relaciones y álgebra lineal

Debido a que el algoritmo de Joux y Pierrot [27] disminuye la complejidad de calcular logaritmos de polinomios de grado menor o igual que 4, se establece que la base de factores esté conformada por estos polinomios.

El procedimiento de generación de relaciones para los polinomios de grado 1, 2, 3 y 4, se explica en las secciones 4.2 de [24], 3.1, 3.2 y 3.3 de [27], respectivamente. Las relaciones obtenidas son vistas como ecuaciones que forman parte de un sistema de ecuaciones

lineales, cuya solución representa el logaritmo discreto de los polinomios que forman las relaciones.

El algoritmo de Wiedemann para resolver sistemas de ecuaciones dispersos [44], [11] se utiliza para encontrar los logaritmos de los polinomios en la base de factores. Primero se deben calcular los logaritmos de los polinomios lineales y cuadráticos, ya que éstos son necesarios para calcular los logaritmos de los polinomios cúbicos. Finalmente, con los logaritmos anteriores, se calculan los logaritmos de los polinomios de grado 4.

Descenso

En esta fase se busca expresar al polinomio h como un producto de polinomios de grado menor que grad(h). Estos nuevos polinomios de grado menor se expresan como el producto de otros de grado aún menor. Este proceso se repite hasta que el producto consiste solamente de aquellos polinomios en la base de factores. Dado que se conocen los logaritmos de estos polinomios, se puede calcular el logaritmo de h . Esta fase se ejecuta en distintos pasos:

- Descenso por fracciones continuas. Este paso se explica en la sección 5.2.
- Descenso clásico. Este paso se explica en la sección 5.3.
- Descenso por bases de Gröbner. Este paso se explica en las secciones 5.3 de [24], 3.7 de [2] y 3.3 de [19].

5.2. Descenso por fracciones continuas

En esta sección se explica el descenso por fracciones continuas. Esta fase comienza con el polinomio h. Dado que grad$(h) \leq n-1$, se supondrá que grad$(h) = n-1$.

Primero, se calcula h' multiplicando el polinomio h por una potencia aleatoria de g:

$$h' = h \cdot g^i, i \in [0, q^n - 2] \tag{5.2}$$

Después, utilizando el algoritmo extendido de Euclides se obtiene:

$$w_1 = w_2 \cdot h' + u \cdot I_X$$

y por lo tanto:

$$h' \equiv \frac{w_1}{w_2} \pmod{I_X}$$

donde $w_1, w_2, u \in \mathbb{F}_q[X]$ con grad(w_1), grad$(w_2) \approx n/2$. Suponiendo que n es impar, entonces grad$(w_1) =$ grad$(w_2) = (n-1)/2$. Este proceso se repite hasta que w_1 y w_2 sean m-suaves, para algún valor $m < (n-1)/2$.

Obteniendo la factorización $\prod_\alpha w_{1,\alpha}^{e_{1,\alpha}}$ de w_1 y $\prod_\beta w_{2,\beta}^{e_{2,\beta}}$ de w_2 se tiene que:

$$\log_g w_1 = \sum_\alpha e_{1,\alpha} \log_g w_{1,\alpha}$$
$$\log_g w_2 = \sum_\beta e_{2,\beta} \log_g w_{2,\beta}$$

y por la ecuación (5.2) el logaritmo de h se puede expresar en función de los logaritmos de polinomios de grado a lo más m:

$$\log_g h = \log_g h' - i = \log_g w_1 - \log w_2 - i \text{ mód } r$$

Se supondrá que todos los polinomios usados para calcular $\log_g h$ tienen grado m. Para expresar el logaritmo de cada uno de estos polinomios de grado m, en función de logaritmos de polinomios de grado menor, se utiliza el descenso clásico que se describe en la sección 5.3.

La probabilidad de que un polinomio de grado $(n-1)/2$ sea m-suave es:

$$\frac{N_q(m,(n-1)/2)}{q^{(n-1)/2}}.$$

El número esperado de pruebas de suavidad a realizar hasta que un polinomio aleatorio de grado $(n-1)/2$ sea m-suave, está dado por el inverso de la probabilidad anterior. En esta fase se busca que dos polinomios de grado $(n-1)/2$ sean m-suaves, por lo que el costo esperado de este descenso es:

$$\left(\frac{q^{(n-1)/2}}{N_q(m,(n-1)/2)}\right)^2 \cdot S_q(m,(n-1)/2) \tag{5.3}$$

El número esperado de factores irreducibles distintos de w_1 y w_2, es

$$2A_q(m,(n-1)/2) \tag{5.4}$$

5.3. Descenso clásico

En esta sección se explica el descenso clásico. Esta fase comienza con un polinomio Q de grado D.

Previamente se definieron los polinomios h_0 y h_1, y se estableció que $q = p^l$. Sea $s \in [0, l]$, para un polinomio $P \in \mathbb{F}_q[X]$, $\overline{P}$ denota el polinomio obtenido de elevar cada coeficiente de P a la potencia p^{l-s}. Sea $R \in \mathbb{F}_q[X, Y]$:

$$\begin{aligned} R(X, X^{p^s})^{p^{l-s}} &= \overline{R}(X^{p^{l-s}}, X^q) \quad \text{por ec. (5.1)} \\ &\equiv \overline{R}\left(X^{p^{l-s}}, \frac{h_0}{h_1}\right) \quad (\text{mód } I_X) \end{aligned}$$

Sea ϵ el grado de R con respecto a la variable Y, denotado por $\text{grad}_Y(R)$, se define:

$$\begin{aligned} R_1 &= R(X, X^{p^s}) \\ R_2 &= h_1^\epsilon \overline{R}\left(X^{p^{l-s}}, \frac{h_0}{h_1}\right) \end{aligned}$$

y entonces:

$$R_2 \equiv h_1^\epsilon R_1^{p^{l-s}} \quad (\text{mód } I_X) \tag{5.5}$$

Para poder expresar el logaritmo de Q en función de los logaritmos de polinomios de grado menor, se necesita de un polinomio R tal que (i) $Q \mid R_1$, (ii) $\text{grad}(R_1/Q)$ y $\text{grad}(R_2)$ estén balanceados, y (iii) R_1/Q y R_2 sean m-suaves para algún valor $m < D$. Debido a (i) y (iii), $R_1 = R_1' Q$, para algún $R_1' \in \mathbb{F}_q[X]$ m-suave, y por la ecuación (5.5):

$$R_2 \equiv h_1^\epsilon {R_1'}^{p^{l-s}} Q^{p^{l-s}} \quad (\text{mód } I_X) \tag{5.6}$$

Obteniendo la factorización $\prod_\alpha r_{1,\alpha}^{e_{1,\alpha}}$ de R_1' y $\prod_\beta r_{2,\beta}^{e_{2,\beta}}$ de R_2 se tiene que:

$$\begin{aligned} \log_g R_1' &= \sum_\alpha e_{1,\alpha} \log_g r_{1,\alpha} \\ \log_g R_2 &= \sum_\beta e_{2,\beta} \log_g r_{2,\beta} \end{aligned}$$

y por la ecuación (5.6) el logaritmo de Q se puede expresar en función de los logaritmos de polinomios de grado a lo más m:

$$\log_g Q = (\log_g R_2 - \epsilon \log_g h_1 - p^{l-s} \log_g R_1')p^{-(l-s)} \text{ mód } r$$

Se supondrá que todos los polinomios usados para calcular $\log_g Q$ tienen grado m. Para expresar el logaritmo de cada uno de estos polinomios de grado m, en función de logaritmos de polinomios de grado menor, se debe decidir entre utilizar el descenso por bases de Gröbner o nuevamente el descenso clásico. La decisión depende del valor de m,

ya que como se explica en [24], mientras más pequeño sea el grado de Q, la complejidad del descenso clásico aumenta.

Sea $\text{grad}_Y(R) = 1$, entonces R se representa como $R(X,Y) = w_1(X) - w_2(X)Y$, donde $w_1, w_2 \in \mathbb{F}_q[X]$. Para generar un polinomio R que satisfaga (i) y (ii) se debe encontrar una base $\{(u_1,u_2),(v_1,v_2)\}$ de la retícula:

$$L_Q = \left\{(w_1,w_2) \in \mathbb{F}_q[X] \times \mathbb{F}_q[X] : Q \mid \left(w_1(X) - w_2(X)X^{p^s}\right)\right\}$$

donde $u_1, u_2, v_1, v_2 \in \mathbb{F}_q[X]$ con $\text{grad}(u_1), \text{grad}(u_2), \text{grad}(v_1), \text{grad}(v_2) \approx D/2$, y se define:

$$(w_1, w_2) = (Au_1 + Bv_1, Au_2 + Bv_2)$$

donde $B \in \mathbb{F}_q[X]$ es mónico de grado b y $A \in \mathbb{F}_q[X]$ de grado $a = b - 1$. El número de puntos en la retícula es q^{2b}.

Se tiene que $\text{grad}(w_1), \text{grad}(w_2) \approx b + D/2$, por lo tanto $t_1 = \text{grad}(R_1) \approx (b + D/2) + p^s$ y $t_2 = \text{grad}(R_2) \approx 2 + (b + D/2)p^{l-s}$. Para asegurar que en la retícula existen suficientes puntos para generar un polinomio R tal que R_1/Q y R_2 son m-suaves, los parámetros s y b deben seleccionarse tal que:

$$q^{2b} \gg \frac{q^{t_1 - D}}{N_q(m, t_1 - D)} \cdot \frac{q^{t_2}}{N_q(m, t_2)} \tag{5.7}$$

es decir, el número de puntos en la retícula debe ser mayor al número esperado de pruebas de suavidad a realizar, que está dado por el inverso de la probabilidad de que un polinomio de grado $t_1 - D$ y un polinomio de grado t_2, sean m suaves.

La base de L_Q se calcula utilizando el algoritmo extendido de Euclides con X^{p^s} y Q, obteniendo:

$$u_1 = u_2 X^{p^s} + uQ$$
$$v_1 = v_2 X^{p^s} + vQ$$

donde $u, v \in \mathbb{F}_q[X]$. Si $D = 2d$, es decir, es un número par, entonces:

$$\text{grad}(u_1) = d - 1, \text{grad}(u_2) = d, \text{grad}(v_1) = d \text{ y } \text{grad}(v_2) = d - 1$$

mientras que si $D = 2d + 1$, es decir, es un número impar, se tiene que:

$$\text{grad}(u_1) = d, \text{grad}(u_2) = d, \text{grad}(v_1) = d + 1 \text{ y } \text{grad}(v_2) = d - 1$$

El cálculo de la base de la retícula L_Q se realiza una vez, por lo que su costo puede ser ignorado, y entonces el costo esperado de este descenso es:

$$\frac{q^{t_1 - D}}{N_q(m, t_1 - D)} \cdot \frac{q^{t_2}}{N_q(m, t_2)} \cdot \text{mín}(S_q(m, t_1 - D), S_q(m, t_2)) \tag{5.8}$$

El número esperado de factores irreducibles distintos de R_1/Q y R_2, es

$$A_q(m, t_1 - D) + A_q(m, t_2) \tag{5.9}$$

5.4. Configuración para el cálculo en $\mathbb{F}_{3^{6\cdot 509}}$

En esta sección se presentan los valores de los parámetros de diseño para calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$, y el análisis de costo utilizando el algoritmo descrito en la sección 5.1.

Sean $q = 3^6$, $n = 509$ y $N = 3^{6\cdot 509} - 1$, se desea calcular el logaritmo discreto en el subgrupo de orden r de $\mathbb{F}^*_{3^{6\cdot 509}}$, donde $r = (3^{509} - 3^{255} + 1)/7$ es un número primo de 804 bits.

Los campos $\mathbb{F}_{3^6}$ y $\mathbb{F}_{3^{6\cdot 509}}$ se representan como:

$$\begin{aligned} \mathbb{F}_{3^6} &= \mathbb{F}_3[u]/(u^6 + 2u^4 + u^2 + 2u + 2) \quad \text{y} \\ \mathbb{F}_{3^{6\cdot 509}} &= \mathbb{F}_{3^6}[X]/(I_X), \end{aligned}$$

donde $I_X \in \mathbb{F}_{3^6}[X]$ es el factor irreducible de grado 509 de $h_1(X)X^q - h_0(X)$ con:

$$\begin{aligned} h_0(X) &= u^{316}X + u^{135} \in \mathbb{F}_{3^6}[X], \\ h_1(X) &= X^2 + u^{424}X \in \mathbb{F}_{3^6}[X], \end{aligned}$$

y se escogió el siguiente generador de $\mathbb{F}^*_{3^{6\cdot 509}}$:

$$g = X + u^2.$$

El objetivo es calcular $\log_g h$ para un elemento $h \in \mathbb{F}^*_{3^{6\cdot 509}}$ de orden r. Para generar el reto h, primero se calculó:

$$h' = \sum_{i=0}^{508} \left(u^{\lfloor \pi \cdot (3^6)^{i+1} \rfloor \bmod 3^6} \right) X^i$$

donde la precisión de π es de 2000 posiciones decimales, y después $h = (h')^{N/r}$. El reto h tiene grado 508 y se calcula de esta manera para evitar suspicacia y escepticismo en la legitimidad del cálculo del logaritmo discreto.

Dado que se decidió utilizar el algoritmo de Joux y Pierrot [27], la base de factores contiene los polinomios cuyo grado es ≤ 4.

La fase de descenso se ilustra en la figura 5.1 y se ejecuta de la siguiente manera:

- Descenso por fracciones continuas (descenso de polinomios de grado 254 a 40).
- Descenso clásico (descenso de polinomios de grado 40 a 21).
- Descenso clásico (descenso de polinomios de grado 21 a 15).
- Descenso por bases de Gröbner (descenso de polinomios de grado 15 a 4).

Descenso por fracciones continuas

En este paso de la fase de descenso, primero el polinomio h de grado 508 se expresa en términos de dos polinomios w_1 y w_2 ambos de grado 254. Después, se busca expresar a w_1 y w_2 como un producto de polinomios de grado a lo más 40.

De acuerdo a la ecuación (5.3), el costo esperado de esta fase de descenso es:

$$\left(\frac{(3^6)^{254}}{N_{3^6}(40,254)}\right)^2 \cdot S_{3^6}(40,254) \approx 2^{33.763860} \cdot S_{3^6}(40,254) \tag{5.10}$$

donde el valor $S_{3^6}(40,254)$ de la implementación, se presenta en la sección 6.4. Por la ecuación (5.4), el número esperado de factores irreducibles, es:

$$2A_{3^6}(40,254) = 2(12.4452) \approx 25.$$

No todos los factores Q_i de los polinomios w_1 y w_2 serán de grado 40. Los factores Q_i tal que $40 \geq \text{grad}(Q_i) \geq 22$, pasarán al primer descenso clásico, los factores Q_i tal que $21 \geq \text{grad}(Q_i) \geq 16$, pasarán directamente al segundo descenso clásico, y los factores Q_i tal que $15 \geq \text{grad}(Q_i) \geq 5$, pasarán directamente al descenso por bases de Gröbner. Aquellos Q_i tal que $4 \geq \text{grad}(Q_i) \geq 1$, pertenecen a la base de factores y sus logaritmos son conocidos.

Descenso clásico

De la sección 5.3 se tiene que:

$$R_1 = R(X, X^{p^s}) = w_1(X) - w_2(X)X^{p^s}$$
$$R_2 = h_1\overline{R}\left(X^{p^{l-s}}, \frac{h_0}{h_1}\right) = h_1\overline{w_1}\left(X^{p^{l-s}}\right) - h_0\overline{w_2}\left(X^{p^{l-s}}\right)$$

donde $h_0, h_1 \in \mathbb{F}_q[X]$ con $\text{grad}(h_0) = 1$, $\text{grad}(h_1) = 2$, por lo tanto:

$$t_1 = \text{grad}(R_1) = \text{máx}(\text{grad}(w_1), \text{grad}(w_2) + p^s)$$
$$t_2 = \text{grad}(R_2) = \text{máx}(2 + \text{grad}(w_1) \cdot p^{l-s}, 1 + \text{grad}(w_2) \cdot p^{l-s})$$

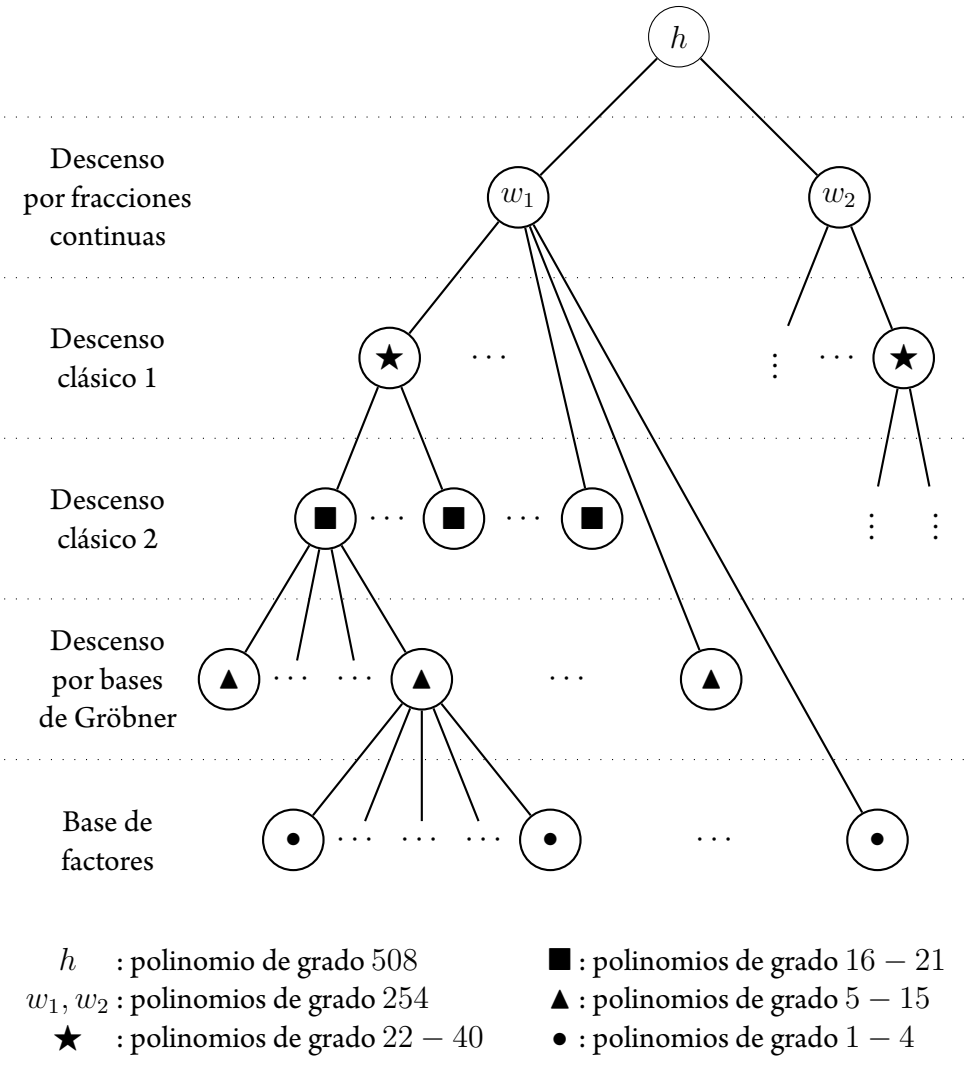


Figura 5.1: Pasos de la fase de descenso para el cálculo del logaritmo discreto en $\mathbb{F}_{3^{6\cdot 509}}$.

Primer descenso clásico

En el primer paso del descenso clásico se busca expresar un polinomio Q de grado $D \in \{22, \ldots, 40\}$, como un producto de polinomios de grado a lo más 21. Aquí se utilizan los parámetros $s = 4$ y $b = 2$, entonces grad$(B) = 2$ y grad$(A) = 1$, haciendo que el número de puntos en la retícula L_Q sea $(3^6)^4$. En la tabla 5.1 se muestran los distintos valores posibles para t_1 y t_2, los cuales siempre satisfacen la ecuacuión (5.7). De todos los posibles valores de t_1 y t_2, el costo mayor de la prueba de suavidad se presenta cuando $D = 40$.

De acuerdo a la ecuacuión (5.8), el costo esperado del primer descenso clásico es:

$$\frac{(3^6)^{62}}{N_{3^6}(21,62)} \cdot \frac{(3^6)^{200}}{N_{3^6}(21,200)} \cdot \text{mín}(S_{3^6}(21,62), S_{3^6}(21,200)) \approx 2^{35.640400} \cdot S_{3^6}(21,62) \tag{5.11}$$

donde los valores de $S_{3^6}(21,62)$ y $S_{3^6}(21,200)$ de la implementación, se presentan en la sección 6.4. Por la ecuación (5.9), el número esperado de factores irreducibles por cada polinomio Q, es:

$$A_{3^6}(21,62) + A_{3^6}(21,200) = 6.9265 + 15.9078 \approx 23,$$

entonces se espera tener aproximadamente $25 \times 23 = 575$ polinomios irreducibles al final del primer descenso clásico.

No todos los factores P_i, usados para expresar a cada polinomio Q, serán de grado 21. Los factores P_i tal que $21 \geq \text{grad}(P_i) \geq 16$, pasarán al segundo descenso clásico, y los factores P_i tal que $15 \geq \text{grad}(P_i) \geq 5$, pasarán directamente al descenso por bases de Gröbner. Aquellos P_i tal que $4 \geq \text{grad}(P_i) \geq 1$, pertenecen a la base de factores y sus logaritmos son conocidos.

Segundo descenso clásico

En el segundo paso del descenso clásico se busca expresar un polinomio Q de grado $D \in \{16, \ldots, 21\}$, como un producto de polinomios de grado a lo más 15. Aquí hay dos casos para los parámetros s y b:

1. Cuando $D \in [21, 19]$, se utiliza $s = 4$ y $b = 2$, entonces grad$(B) = 2$ y grad$(A) = 1$, haciendo que el número de puntos en L_Q sea $(3^6)^4$.

2. Cuando $D \in [18, 16]$, se utiliza $s = 4$ y $b = 1$, entonces grad$(B) = 1$, sin embargo, el polinomio A se mantiene de grado 1, por lo que el número de puntos en L_Q es $(3^6)^3$ y no $(3^6)^2$.

D	t_1	$t_1 - D$	t_2	$\lvert L_Q \rvert$	$\frac{(3^6)^{t_1-D}}{N_{3^6}(21,t_1-D)} \cdot \frac{(3^6)^{t_2}}{N_{3^6}(21,t_2)}$
40	102	62	200		$2^{35.640400}$
39	101	62	200		$2^{35.640400}$
38	101	63	191		$2^{33.634932}$
37	100	63	191		$2^{33.634932}$
36	100	64	182		$2^{31.667226}$
35	99	64	182		$2^{31.667226}$
34	99	65	173		$2^{29.738783}$
33	98	65	173		$2^{29.738783}$
32	98	66	164		$2^{27.851745}$
31	97	66	164	$(3^6)^4 \approx 2^{38.039100}$	$2^{27.851745}$
30	97	67	155		$2^{26.008500}$
29	96	67	155		$2^{26.008500}$
28	96	68	146		$2^{24.211662}$
27	95	68	146		$2^{24.211662}$
26	95	69	137		$2^{22.464257}$
25	94	69	137		$2^{22.464257}$
24	94	70	128		$2^{20.769508}$
23	93	70	128		$2^{20.769508}$
22	93	71	119		$2^{19.131466}$

Tabla 5.1: Valores para el primer paso del descenso clásico. En la primera, segunda y cuarta columnas se muestran los posibles valores para los grados D, t_1 y t_2 de los polinomios Q, R_1 y R_2 respectivamente. Las pruebas de suavidad se aplican a polinomios de grado $t_1 - D$ (tercera columna) y t_2. Nótese que el número esperado de pruebas de suavidad a realizar, mostrado en la sexta columna, siempre es menor que el número de puntos de la retícula L_Q en la quinta columna, cumpliéndose la ecuacuión (5.7) para todos los posibles valores.

D	t_1	$t_1 - D$	t_2	$\lvert L_Q \rvert$	$\frac{(3^6)^{t_1-D}}{N_{3^6}(15,t_1-D)} \cdot \frac{(3^6)^{t_2}}{N_{3^6}(15,t_2)}$
21	92	71	119	$(3^6)^4 \approx 2^{38.039100}$	$2^{33.441673}$
20	92	72	110		$2^{30.936996}$
19	91	72	110		$2^{30.936996}$
18	91	73	92	$(3^6)^3 \approx 2^{28.529325}$	$2^{25.954221}$
17	90	73	92		$2^{25.954221}$
16	90	74	83		$2^{23.744950}$

Tabla 5.2: Valores para el segundo paso del descenso clásico. En la primera, segunda y cuarta columnas se muestran los posibles valores para los grados D, t_1 y t_2 de los polinomios Q, R_1 y R_2 respectivamente. Las pruebas de suavidad se aplican a polinomios de grado $t_1 - D$ (tercera columna) y t_2. Nótese que el número esperado de pruebas de suavidad a realizar, mostrado en la sexta columna, siempre es menor que el número de puntos de la retícula L_Q en la quinta columna, cumpliéndose la ecuacuión (5.7) para todos los posibles valores.

En la tabla 5.2 se muestran los distintos valores posibles para t_1 y t_2, los cuales siempre satisfacen la ecuacuión (5.7) en ambos casos. De todos los posibles valores de t_1 y t_2, el costo mayor de la prueba de suavidad se presenta cuando $D = 21$.

De acuerdo a la ecuacuión (5.8), el costo esperado del segundo descenso clásico es:

$$\frac{(3^6)^{71}}{N_{3^6}(15,71)} \cdot \frac{(3^6)^{119}}{N_{3^6}(15,119)} \cdot \text{mín}(S_{3^6}(15,71), S_{3^6}(15,119)) \approx 2^{33.441673} \cdot S_{3^6}(15,71) \tag{5.12}$$

donde los valores de $S_{3^6}(15,71)$ y $S_{3^6}(15,119)$ de la implementación se presentan en la sección 6.4. Por la ecuación (5.9), el número esperado de factores irreducibles por cada polinomio Q, es:

$$A_{3^6}(15,71) + A_{3^6}(15,119) = 9.0837 + 13.3809 \approx 23,$$

entonces se espera tener aproximadamente $575 \times 23 = 13\,225$ polinomios irreducibles al final del segundo descenso clásico.

No todos los factores P_i, usados para expresar a cada polinomios Q, serán de grado 15. Los factores P_i tal que $15 \geq \text{grad}(P_i) \geq 5$, pasarán al descenso por bases de Gröbner. Aquellos P_i tal que $4 \geq \text{grad}(P_i) \geq 1$, pertenecen a la base de factores y sus logaritmos son conocidos.

CAPÍTULO 6

Implementación y resultados

En [1], Adj et al. muestran que es factible calcular el logaritmo discreto en los campos $\mathbb{F}_{3^{6\cdot 137}}$ y $\mathbb{F}_{3^{6\cdot 163}}$ utilizando el sistema algebraico computacional Magma [7]. Los autores concluyen que a pesar de haber obtenido resultados en poco tiempo, la implementación no es óptima para esos campos.

Como se menciona al inicio del capitulo 5, se realizó una implementación de la prueba de suavidad, la cual se utiliza en el descenso por fracciones continuas y descenso clásico para calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 509}}$. Dado que no hay resultados de referencia, esta implementación se considerará eficiente si su tiempo de ejecución es menor al tiempo requerido por la factorización de Magma.

En este capítulo se presentan los detalles de la implementación. Primero, en la sección 6.1 se presenta brevemente la manera de medir el tiempo de ejecución de un fragmento de código, ya que a lo largo de este capítulo se muestran los tiempos de ejecución de distintas operaciones. En las secciones 6.2 y 6.3 se muestra la implementación de la aritmética en el campo base y en el anillo de polinomios, respectivamente. La sección 6.4 presenta la implementación de la prueba de suavidad utilizada por los descensos de fracciones continuas y clásico. Finalmente, los resultados del cálculo de estos descensos se muestran en la sección 6.5.

6.1. Medición del tiempo de ejecución

El tiempo de ejecución de algún programa o fragmento de código se puede medir en segundos (o fracciones de segundo tales como milisegundo, microsegundo, etc.) o en ciclos de reloj. Para realizar una medición en segundos, se debe obtener el tiempo en el que el programa inicia su ejecución, el tiempo de finalización, y sustraer estas cantidades; este procedimiento es en general sencillo de realizar, ya que diversos lenguajes de programación proporcionan rutinas para obtener estos tiempos. Sin embargo, medir el tiempo de ejecución en ciclos de reloj es una tarea compleja, ya que se necesita código especial que no todos los lenguajes de programación proporcionan.

En [35], se presenta un método para medir los ciclos de reloj requeridos en la ejecución de algún fragmento de código en lenguaje C [30]. Este método supone que se trabaja con procesadores Intel, sistema operativo Linux y el compilador GCC. Para medir el tiempo, se crea un módulo del *kernel* de Linux, y con la instrucción `RDTSCP` de lenguaje ensamblador, se obtiene el número de ciclos de reloj necesarios para ejecutar el código de interés. El módulo del *kernel* realiza diversas ejecuciones de este código para calcular datos estadísticos de los tiempos obtenidos, tales como media, mediana y varianza, y con base a estos indicadores, poder determinar correctamente su tiempo de ejecución.

El método presentado en [35] se utiliza para medir el tiempo de ejecución de las operaciones aritméticas del campo $\mathbb{F}_{3^6}$, operaciones aritméticas del anillo de polinomios $\mathbb{F}_{3^6}[X]$ y la prueba de suavidad. Para obtener los datos estadísticos, se realizaron $100\,000$ ejecuciones de las operaciones aritméticas en $\mathbb{F}_{3^6}$, $100\,000$ ejecuciones de las operaciones aritméticas en $\mathbb{F}_{3^6}[X]$, y $20\,000$ ejecuciones de la prueba de suavidad. Las mediciones se realizaron en una máquina con procesador Intel i7-3520M a 2.9 GHz, sistema operativo openSUSE 13.2 con *kernel* 3.16.7-24 y GCC 4.8.3.

6.2. Aritmética en el campo base

Las operaciones implementadas en el campo $\mathbb{F}_{3^6}$ son inverso aditivo, suma, resta, multiplicación, inverso multiplicativo y exponenciación. Es de gran importancia que el tiempo de ejecución de cada operación se reduzca lo más posible, ya que la eficiencia de la aritmética en el anillo de polinomios sobre $\mathbb{F}_{3^6}$, depende directamente de la eficiencia de estas operaciones.

Representación de elementos

En la sección 5.4 se estableció que $\mathbb{F}_{3^6} = \mathbb{F}_3[u]/(u^6 + 2u^4 + u^2 + 2u + 2)$, y entonces un elemento $a \in \mathbb{F}_{3^6}$ es visto como $a = \sum_{i=0}^{5} a_i u^i$, donde $a_i \in \mathbb{F}_3 = \{0, 1, 2\}$. Con

esta construcción se tiene que u es un generador de $\mathbb{F}_{3^6}^*$, entonces cualquier elemento $a \in \mathbb{F}_{3^6}^*$ puede expresarse como una potencia de u. Los elementos de $\mathbb{F}_{3^6}$ se representan de manera exponencial de acuerdo a la función $\psi : \mathbb{F}_{3^6} \to \{0, \ldots, 728\}$, como sigue:

$$\psi(a) = \begin{cases} 0 & \text{si } a = 0, \\ i & \text{si } a = u^i, \text{ donde } i \in \{1, \ldots, 728\}. \end{cases}$$

Según la sección 3.2.2, la imágen de ψ para $\mathbb{F}_{3^6}$ sería $\{0, \ldots, 727\} \cup \{\infty\}$, sin embargo, en esta implementación la imágen es $\{0, \ldots, 728\}$, debido a que $0 \in \mathbb{F}_{3^6}$ se representa como 0, y $q^0 \in \mathbb{F}_{3^6}$ se representa como 728, ya que $q^0 = q^{728}$. Utilizando la representación exponencial, un elemento de $\mathbb{F}_{3^6}$ se almacena en una variable de tipo `uint16_t` (entero sin signo de 16 bits).

Operaciones aritméticas

Inverso aditivo. El cálculo del inverso aditivo se realiza utilizando una tabla de precómputo (véase sección 3.2.3); este pre-cómputo se realizó con el sistema algebraico Magma. Esta estrategia requiere almacenamiento para 3^6 elementos en $\mathbb{F}_{3^6}$, es decir, $1\,458$ bytes. Sean $a \in \mathbb{F}_{3^6}$ y `tbl_invad` la tabla de pre-cómputo, entonces $-a =$ `tbl_invad`$[a]$.

Suma. La suma se calcula utilizando una tabla de precómputo (véase sección 3.2.3); este pre-cómputo se realizó con el sistema algebraico Magma. Esta estrategia requiere almacenamiento para $(3^6)^2$ elementos en $\mathbb{F}_{3^6}$, es decir, $1\,062\,882$ bytes. Sean $a, b \in \mathbb{F}_{3^6}$ y `tbl_suma` la tabla de pre-cómputo, entonces $a + b =$ `tbl_suma`$[a][b]$.

Resta. La resta se calcula utilizando las tablas de precómputo del inverso aditivo y suma. Primero se calcula un inverso aditivo y posteriormente se calcula una suma. Sean $a, b \in \mathbb{F}_{3^6}$, entonces $a - b =$ `tbl_suma`$[a][$`tbl_invad`$[b]]$.

Multiplicación. La multiplicación se calcula utilizando aritmética entera módulo 728, como se explica en la sección 3.2.2. Sean $a, b \in \mathbb{F}_{3^6}$ con $a = u^m$ y $b = u^n$, entonces $a \cdot b = u^{(m+n) \bmod 728}$. Si $(m + n) \bmod 728 = 0$, entonces $a \cdot b = u^{728}$.

Inverso multiplicativo. El cálculo del inverso multiplicativo se realiza utilizando aritmética entera módulo 728, como se explica en la sección 3.2.2. Sea $a = u^m \in \mathbb{F}_{3^6}$, entonces $a^{-1} = u^{-m \bmod 728}$. Si $-m \bmod 728 = 0$, entonces $a^{-1} = u^{728}$.

Exponenciación. La exponenciación se calcula utilizando aritmética entera módulo 728, como se explica en la sección 3.2.2. Sean $a = u^m \in \mathbb{F}_{3^6}$ y $e > 0$, entonces $a^e = u^{(me) \bmod 728}$. Si $(me) \bmod 728 = 0$, entonces $a^e = u^{728}$.

	Tiempo de ejecución (en ciclos)		
	Pre-cómputo	**Arit.** $\mathrm{mód}728$	**Log. de Zech**
Inv. aditivo	5	5	-
Suma	14	-	26
Resta	(1) 17 (2) 20	-	30
Multiplicación	14	5	-
Inv. multiplicativo	5	5	-

Tabla 6.1: Tiempos de ejecución de las operaciones aritméticas en $\mathbb{F}_{3^6}$. Los tiempos están expresados en ciclos de reloj. Las celdas en gris representan la manera en la que se implementaron las operaciones. La versión (1) de la resta utiliza tabla de pre-cómputo para calcular el inverso aditivo, mientras que la opción (2) utiliza aritmética $\mathrm{mód}728$.

Resultados del tiempo de ejecución

En la tabla 6.1 se muestra el tiempo de ejecución de las operaciones aritméticas en el campo $\mathbb{F}_{3^6}$, utilizando las diferentes alternativas para la representación polinomial (véase sección 3.2). La primera columna muestra los tiempos utilizando tablas de pre-cómputo, la segunda columna muestra los tiempos utilizando aritmética módulo 728, excepto para la suma y la resta, los tiempos empleando logaritmo de Zech para estas operaciones se muestra en la tercera columna. Las celdas en gris representan la manera en la que se implementaron las operaciones aritméticas en $\mathbb{F}_{3^6}$.

Como se mencionó previamente, la resta se calcula con un inverso aditivo y la tabla de pre-cómputo de la suma. Para calcular dicho inverso aditivo, la opción (1) de la resta muestra el tiempo utilizando la tabla de pre-cómputo, mientras que la opción (2) muestra el tiempo utilizando aritmética $\mathrm{mód}728$. Se decidió utilizar la tabla de pre-cómputo para calcular el inverso aditivo, debido a que la opción (1) presenta mayor rapidez para calcular la resta. En el caso del inverso multiplicativo, se eligió la aritmética $\mathrm{mód}728$ porque tiene la ventaja de no requerir espacio para almacenar una tabla adicional.

6.3. Aritmética en el anillo de polinomios

Las operaciones implementadas en el anillo de polinomios $\mathbb{F}_{3^6}[X]$ son suma, resta, multiplicación, división y máximo común divisor. Todas estas operaciones, salvo la multiplicación y el máximo común divisor, se implementaron siguiendo los algoritmos de la sección 3.1. Un polinomio en $\mathbb{F}_{3^6}[X]$ se almacena en un arreglo de tipo `uint16_t`. Las operaciones entre los coeficientes se realizan como se describe en la sección 6.2.

Operaciones aritméticas

Suma/Resta. La suma y resta se calculan utilizando el algoritmo de suma polinomial (véase algoritmo 3.1).

División. La división se calcula utilizando el algoritmo de división polinomial (véase algoritmo 3.4).

Multiplicación. La multiplicación se calcula utilizando una combinación del método de Karatsuba (algoritmo 3.3) y el método de la escuela (algoritmo 3.2).

Para multiplicar polinomios en $\mathbb{F}_{3^6}[X]$ con n coeficientes, el número de sumas y multiplicaciones en $\mathbb{F}_{3^6}$ requerido por el método de la escuela, es:

$$S_E = (n-1)^2 \quad \text{y} \tag{6.1}$$
$$M_E = n^2, \tag{6.2}$$

respectivamente [43], mientras que el número de sumas y multiplicaciones en $\mathbb{F}_{3^6}$ que requiere el método de Karatsuba, es:

$$S_K = 6n^{\log_2 3} - 8n + 2 \quad \text{y} \tag{6.3}$$
$$M_K = n^{\log_2 3}, \tag{6.4}$$

respectivamente [43]. Dado que los tiempos de ejecución de la suma y multiplicación en $\mathbb{F}_{3^6}$ son 14 y 5 ciclos de reloj respectivamente (véase tabla 6.1), se establecerá como unidad de tiempo una multiplicación en $\mathbb{F}_{3^6}$, y entonces una suma es equivalente a 2.8 multiplicaciones en $\mathbb{F}_{3^6}$.

Para multiplicar polinomios con n coeficientes, el método de Karatsuba realiza multiplicaciones de polinomios con $n/2$ coeficientes. Las multiplicaciones anteriores se calculan multiplicando polinomios con $n/2^2$ coeficientes, que a su vez se calculan multiplicando polinomios con $n/2^3$ coeficientes, y así sucesivamente, hasta multiplicar polinomios con sólamente un coeficiente. Para determinar la manera más eficiente de calcular la multiplicación, se analizó el costo del método de Karatsuba utilizando el método de la escuela para multiplicar polinomios con $n/2^i$ coeficientes, para los diferentes valores de $i \in \{0, \ldots, \lceil \log_2 n \rceil\}$.

El descenso por fracciones continuas es la fase donde se requiere multiplicar polinomios con el mayor número de coeficientes: se multiplican polinomios con 255 coeficientes (véase sección 5.4). Por lo anterior, se debe determinar la combinación más eficiente para multiplicar polinomios con 255 coeficientes. En la tabla 6.2 se muestran los costos de las distintas combinaciones. Para interpretar correctamente la tabla 6.2, se presenta el ejemplo 6.3.1.

i	Método de la escuela				Método de Karatsuba			Tiempo total
	$n_E = \frac{256}{2^i}$	# sumas	# mult.	Tiempo	$n_K = \frac{256}{n_E}$	# sumas	# mult.	
0	256	65 025	65 536	247 606.0	1	0	1	247 606
1	128	16 129	16 384	61 545.2	2	4	3	186 070
2	64	3 969	4 096	15 209.2	4	24	9	141 184
3	32	961	1 024	3 714.8	8	100	27	109 260
4	16	225	256	886.0	16	360	81	87 894
5	8	49	64	201.2	32	1 204	243	75 862
6	4	9	16	41.2	64	3 864	729	73 312
7	2	1	4	6.8	128	12 100	2 187	82 632
8	1	0	1	1.0	256	37 320	6 561	111 057

Tabla 6.2: Análisis de las alternativas para multiplicar polinomios con 256 coeficientes. La unidad de tiempo es multiplicaciones en $\mathbb{F}_{3^6}$. La columna 2 muestra las alternativas del número de coeficientes que se pueden multiplicar con el método de la escuela. Las columnas 3 y 4 muestran el número de sumas y multiplicaciones requeridas por el método de la escuela para n_E coeficientes; en este caso, el tiempo de una suma es 2.8 y el de una multiplicación es 1. Las columnas 7 y 8 muestran el número de sumas y multiplicaciones requeridas por el método de Karatsuba para n_K coeficientes; en este caso, el tiempo de una suma es $2.8 \times n_E$ y el de una multiplicación es el tiempo de la columna 5. El tiempo total para calcular la multiplicación es el tiempo requerido por el método de Karatsuba, y se muestra en la columna 9; mientras menor sea el tiempo total, la combinación de los métodos resulta más eficiente.

Ejemplo 6.3.1. Cuando $i = 5$, se utiliza el método de la escuela para multiplicar polinomios con $n_E = 8$ coeficientes. Entonces, se utiliza el método de Karatsuba para multiplicar polinomios con $n_K = 32$ coeficientes, ya que el polinomio original se puede considerar como un polinomio de 32 coeficientes, donde cada coeficiente es a su vez un polinomio con 8 coeficientes.

Por las ecuaciones (6.1) y (6.2), cada multiplicación calculada con el método de la escuela requiere 49 sumas y 64 multiplicaciones en $\mathbb{F}_q$, en total $(49 \times 2.8) + 64 = 201.2$ unidades de tiempo.

Por las ecuaciones (6.3) y (6.4), la multiplicación utilizando el método de Karatsuba requiere $1\,204$ sumas y 243 multiplicaciones de polinomios con 8 coeficientes. El tiempo requerido por cada suma es 2.8×8 unidades de tiempo y por cada multiplicación es 201.2 unidades de tiempo, en total $(1\,204 \times 2.8 \times 8) + (243 \times 201.2) = 75\,862$ unidades de tiempo. ◊

La tabla 6.2 indica que la manera más eficiente de calcular la multiplicación es utilizando el método de escuela para 4 coeficientes y el método de Karatsuba para 64 coeficientes. Para validar esta afirmación, se realizó la implementación de todas las alternativas

i	$n_E = \frac{256}{2^i}$	$n_K = \frac{256}{n_E}$	**Tiempo de ejecución** (en miles de ciclos)
0	256	1	571
1	128	2	447
2	64	4	375
3	32	8	323
4	16	16	288
5	8	32	248
6	4	64	261
7	2	128	382
8	1	256	733

Tabla 6.3: Tiempos de ejecución de las alternativas para multiplicar polinomios con 256 coeficientes. Los tiempos están expresados en miles de ciclos de reloj.

mostradas en la tabla 6.2, y los tiempos de ejecución obtenidos se muestran en la tabla 6.3. La implementación muestra que la multiplicación polinomial se calcula de manera más eficiente utilizando el método de Karatsuba para polinomios con 32 coeficientes, y el método de la escuela para 8 coeficientes. El algoritmo 6.1 muestra la manera en la que se implementó la multiplicación.

Máximo común divisor. El máximo común divisor se calcula utilizando una versión modificada del algoritmo extendido de Euclides (algoritmo 3.6). Esta versión modificada se muestra en el algoritmo 6.2.

Esta operación se utiliza en los descensos por fracciones continuas y clásico (véanse secciones 5.2 y 5.3, respectivamente), en ambos casos se busca que el mcd obtenido tenga un grado específico. Recuérdese que el algoritmo original con los polinomios f y g, calcula en cada iteración los valores r_i, s_i y t_i tal que $r_i = fs_i + gt_i$ (ver sección 3.1.2). La diferencia del algoritmo 6.2 respecto al algoritmo 3.6 original, es que la versión modificada detiene su ejecución cuando r_i tiene el grado deseado.

Resultados del tiempo de ejecución

Las pruebas de suavidad en los descensos por fracciones continuas y clásico, realizan operaciones aritméticas entre polinomios con diversos grados (véanse tablas 5.1 y 5.2 en la sección 5.4). Los tiempos de ejecución de las operaciones aritméticas en el anillo $\mathbb{F}_{3^6}[X]$, para polinomios con estos grados, se muestran en la tabla 6.4. Los tiempos de la suma, resta y multiplicación, fueron medidos realizando cada operación con dos polinomios del mismo grado. En el caso de la división, el grado del divisor es el grado que se muestra

Algoritmo 6.1 `MultPolinomial`

Entrada:
$f = (f_n, \ldots, f_0) \in \mathbb{F}_q[X], g = (g_n, \ldots, g_0) \in \mathbb{F}_q[X]$
Salida:
$h = (h_{2n}, \ldots, h_0) \in \mathbb{F}_q[X]$
1: **si** $n \leq 8$
2: **regresar** `MultPolinomialEscuela`(f, g) /* `algoritmo 3.3` */
3: **fin si**
4: $m \leftarrow \lceil (n+1)/2 \rceil$
5: $f_A, f_B \leftarrow (f_n, \ldots, f_m), (f_{m-1}, \ldots, f_0)$
6: $g_A, g_B \leftarrow (g_n, \ldots, g_m), (g_{m-1}, \ldots, g_0)$
7: $t_0 \leftarrow$`MultPolinomial`(f_B, g_B)
8: $t_2 \leftarrow$`MultPolinomial`(f_A, g_A)
9: $t_1 \leftarrow$`MultPolinomial`$(f_A + f_B, g_A + g_B)$
10: **regresar** $(t_2 X^{2m} + (t_1 - t_0 - t_2)X^m + t_0)$

Algoritmo 6.2 `EuclidesExtendido`

Entrada:
$f, g \in \mathbb{F}_q[X]$, tal que $\text{grad}(f) \geq \text{grad}(g)$ y $g \neq 0$
$d > 0$
Salida:
$r, s, t \in \mathbb{F}_q[X]$ tal que $r = fs + gt$ y $\text{grad}(r) \approx d$
1: $r \leftarrow f, r' \leftarrow g$
2: $s \leftarrow 1, s' \leftarrow 0$
3: $t \leftarrow 0, t' \leftarrow 1$
4: **mientras** $r' \neq 0$ **hacer**
5: $q, r'' \leftarrow$`DivPolinomial`(r, r')
6: $(r, s, t, r', s', t') \leftarrow (r', s', t', r'', s - s'q, t - t'q)$
7: **si** $\text{grad}(r) \leq d$
8: $c \leftarrow \text{cp}(r)$ /* $\text{cp}(r)$ `denota el coeficiente principal de` r */
9: **regresar** $c^{-1}r, c^{-1}s, c^{-1}t$
10: **fin si**
11: **fin mientras**
12: $c \leftarrow \text{cp}(r)$ /* $\text{cp}(r)$ `denota el coeficiente principal de` r */
13: **regresar** $c^{-1}r, c^{-1}s, c^{-1}t$

en la primera columna de la tabla 6.4, mientras que el grado del dividendo es el doble del grado del divisor, por ejemplo, para la primera fila se reporta el tiempo requerido para dividir un polinomio de grado 508 por uno de grado 254.

6.4. Prueba de suavidad

La prueba de suavidad se calcula utilizando el método de Granger et al. [18] que se describe en la sección 4.3.

Para calcular el logaritmo discreto en $\mathbb{F}_{3^{6\cdot 509}}$, el descenso por fracciones continuas realiza pruebas de suavidad a polinomios con grado 254. Los pasos del descenso clásico realizan pruebas de suavidad a diversos polinomios, y el grado de éstos se muestran en las tablas 5.1 y 5.2. En la tabla 6.5 se muestra el tiempo de ejecución de la prueba de suavidad para los polinomios anteriores.

Dado que no hay resultados de referencia, la implementación de la prueba de suavidad se considerará eficiente si su tiempo de ejecución es menor al tiempo requerido por la factorización de Magma. Este sistema algebraico computacional no provee mecanismos para medir el tiempo de ejecución en ciclos de reloj, por lo tanto la comparación se realizó midiendo el tiempo en milisegundos. Se utlizó la versión 2.20-2 de Magma. Los resultados obtenidos se muestran en la tabla 6.6; nótese que en todos los casos, el tiempo de ejecución de la prueba de suavidad es menor que la factorización de Magma.

Tiempo de ejecución esperado de los descensos

Como se muestra en la sección 5.4, el tiempo de ejecución de los descensos por fracciones continuas y clásico dependen directamente del tiempo de ejecución de la prueba de suavidad. Para el caso del descenso por fracciones continuas, su tiempo está dado por la ecuación (5.10), el valor de $S_{3^6}(40, 254)$ es 265.70 millones de ciclos de reloj, entonces el tiempo de ejecución esperado es:

$$2^{33.763860} \cdot (265.70 \times 10^6) \approx 10^{18.588326} \text{ ciclos de reloj.}$$

El tiempo de ejecución del primer descenso clásico está dado por la ecuación (5.11), y el valor de $S_{3^6}(21, 62)$ es 3.60 millones de ciclos de reloj. De acuerdo a la sección 5.4, se espera obtener 25 factores irreducibles del descenso por fracciones continuas, entonces el tiempo esperado del primer paso de descenso clásico, es:

$$2^{35.640400} \cdot (3.60 \times 10^6) \cdot 25 \approx 10^{18.683071} \text{ ciclos de reloj.}$$

La ecuación (5.12) muestra el tiempo de ejecución del segundo descenso clásico, y el valor de $S_{3^6}(15, 71)$ es 3.72 millones de ciclos de reloj. De acuerdo a la sección 5.4, se

Grado	**Tiempo de ejecución** (en ciclos)			
	Suma	Resta	Multiplicación	División
254	1 100	1 200	248 000	670 400
200	850	990	181 100	409 400
191	825	940	149 400	379 400
182	790	900	146 700	339 200
173	750	850	140 430	307 200
164	720	820	132 900	275 800
155	680	765	114 300	247 150
146	640	730	110 550	219 100
137	600	695	102 900	193 800
128	565	650	88 500	169 300
119	535	605	77 450	147 100
110	500	560	62 730	128 600
92	420	485	48 640	89 070
83	400	440	44 500	73 000
74	355	395	36 550	58 900
73	350	380	36 100	57 100
72	345	375	35 600	55 600
71	340	370	34 410	54 100
70	335	365	34 330	52 700
69	330	360	33 900	51 200
68	325	355	33 600	49 800
67	320	350	32 700	48 500
66	315	345	32 320	47 400
65	310	340	31 250	45 700
64	305	335	30 000	44 400
63	300	330	26 600	43 600
62	295	325	26 500	42 300

Tabla 6.4: Tiempos de ejecución de las operaciones aritméticas en $\mathbb{F}_{3^6}[X]$. Los tiempos están expresados en ciclos de reloj. Los descensos por fracciones continuas y clásico realizan operaciones entre polinomios cuyos grados se muestran en la primera columna.

Descenso	m (suavidad)	**Grado**	**Tiempo de ejecución** (en millones de ciclos)
Frac. continuas	40	254	265.70
Clásico (paso 1)	21	200	35.63
		191	32.70
		182	29.51
		173	26.79
		164	24.14
		155	21.62
		146	19.22
		137	17.02
		128	14.75
		119	12.79
		71	4.71
		70	4.59
		69	4.46
		68	4.34
		67	4.22
		66	4.08
		65	3.96
		64	3.80
		63	3.70
		62	3.60
Clásico (paso 2)	15	119	10.14
		110	8.65
		92	6.13
		83	5.03
		74	4.02
		73	3.92
		72	3.81
		71	3.72

Tabla 6.5: Tiempos de ejecución de las pruebas de suavidad requeridas en el descenso por fracciones continuas y descenso clásico. Los tiempos están expresados en millones de ciclos de reloj.

Descenso	**Grado**	**Tiempo de ejecución** (en milisegundos)	
		Prueba de suavidad	**Fact. Magma**
Frac. continuas	254	29.0986	141.3260
Clásico (paso 1)	200	10.6662	72.2620
	191	9.6892	62.8740
	182	8.8332	53.3860
	173	8.0016	46.5880
	164	7.1851	41.4840
	155	6.4297	36.0580
	146	5.7323	31.0600
	137	5.0665	27.3780
	128	4.3788	20.1640
	119	3.8145	16.0980
	71	1.3987	3.9060
	70	1.3623	3.8020
	69	1.3270	3.6880
	68	1.2895	3.6440
	67	1.2534	3.5160
	66	1.2162	3.3940
	65	1.1785	3.0400
	64	1.1314	2.9600
	63	1.1065	2.9880
	62	1.0764	2.8840
Clásico (paso 2)	119	2.9452	16.0980
	110	2.5101	13.3120
	92	1.7828	8.3640
	83	1.4623	6.1040
	74	1.1779	4.7100
	73	1.1453	4.3180
	72	1.1080	4.2760
	71	1.0812	3.9060

Tabla 6.6: Comparación de los tiempos de ejecución de la prueba de suavidad y la factorización de Magma. Los tiempos están expresados en milisegundos. Los tiempos de ejecución representan el tiempo promedio de $5\,000$ ejecuciones.

Descenso	**Frec. procesador** (en GHz)	**Tiempo de ejecución** (en años)
Frac. continuas	2.0	61.44
	2.2	55.85
	2.4	51.20
	2.6	47.26
	2.8	43.88
	3.0	40.96
	3.2	38.40
Clásico (paso 1)	2.0	76.42
	2.2	69.47
	2.4	63.68
	2.6	58.78
	2.8	54.58
	3.0	50.94
	3.2	47.76
Clásico (paso 2)	2.0	395.65
	2.2	359.68
	2.4	329.71
	2.6	304.35
	2.8	282.61
	3.0	263.77
	3.2	247.28

Tabla 6.7: Tiempos de ejecución esperados del descenso por fracciones continuas y descenso clásico. Los tiempos están expresados en años.

espera obtener 575 factores irreducibles del primer descenso clásico, entonces el tiempo esperado del segundo paso de descenso clásico, es::

$$2^{33.441673} \cdot (3.72 \times 10^6) \cdot 575 \approx 10^{19.397157} \text{ ciclos de reloj.}$$

Suponiendo que los descensos se ejecutan sólamente en una máquina, la tabla 6.7 muestra el tiempo de ejecución esperado de cada descenso, utilizando diferentes frecuencias de procesador. Los cómputos anteriores son factibles de realizar si se tiene acceso a algunos cientos de núcleos de procesamiento.

Frec. procesador (en GHz)	**Núcleos disponibles**	**Núcleos utilizados**		
		Frac. cont.	**Clásico 1**	**Clásico 2**
1.90	32	0	32	32
2.00	32	32	32	32
2.30	88	0	88	88
2.60	108	108	108	108
3.00	4	4	4	4
3.20	48	48	48	48
3.40	72	72	72	72
3.50	6	6	6	6
Total	390	270	390	390

Tabla 6.8: Núcleos de procesamiento utilizados en el descenso por fracciones continuas y descenso clásico.

6.5. Resultados

Las ejecuciones del descenso por fracciones continuas y descenso clásico se realizaron en diversas máquinas. El número total de núcleos de procesamiento utilizados fue 270 para el descenso por fracciones continuas y 390 para los dos descensos clásicos. En la tabla 6.8 se muestra la frecuencia de procesamiento de estos núcleos.

Los descensos se ejecutaron en paralelo sobre todos los núcleos utilizados, y cuando uno de ellos encontraba el resultado, los demás terminaban su ejecución. En esta sección se reportan los siguientes tiempos de ejecución:

- **Tiempo real.** Es el tiempo requerido por el núcleo que encontró el resultado.
- **Tiempo total.** Es el tiempo requerido por todos los núcleos hasta que se encontró el resultado.

Descenso por fracciones continuas

De acuerdo a la tabla 6.8, la frecuencia de procesamiento promedio para el descenso por fracciones continuas es de 2.87 GHz.

La tabla 6.7 indica que el tiempo esperado de este cálculo, es poco menos de 43.88 años. El tiempo total para encontrar el resultado fue 51 años, mientras que su tiempo real fue 71.45 dias. El número de factores irreducibles obtenido y sus respectivos grados se muestran en la tabla 6.9.

Polinomio	Grados de los factores			
	$40-22$	$21-16$	$15-5$	$4-1$
w_1	6	1	3	1
w_2	6	2	2	1
Total	12	3	5	2

Tabla 6.9: Número de factores irreducibles obtenidos del descenso por fracciones continuas.

Primer descenso clásico

De acuerdo a la tabla 6.8, la frecuencia de procesamiento promedio para el primer descenso clásico es de 2.66 GHz.

La tabla 6.7 indica que el tiempo esperado de este cálculo, considerando que el descenso se aplica a 25 polinomios, es aproximadamente 58.78 años. Sin embargo, este descenso se aplica sólamente a 12 polinomios, y entonces el tiempo esperado es:

$$2^{35.640400} \cdot (3.60 \times 10^6) \cdot 12 \approx 10^{18.364313} \text{ ciclos de reloj,}$$

que es equivalente a 27.58 años. El tiempo total para encontrar el resultado fue 9.85 años, mientras que su tiempo real fue 22.32 dias. El número de factores irreducibles obtenidos y sus respectivos grados se muestran en la tabla 6.10; las columnas 2 a 4 indican el número de factores irreducibles obtenidos tanto de R_1/Q y R_2 (véase sección 5.3).

Segundo descenso clásico

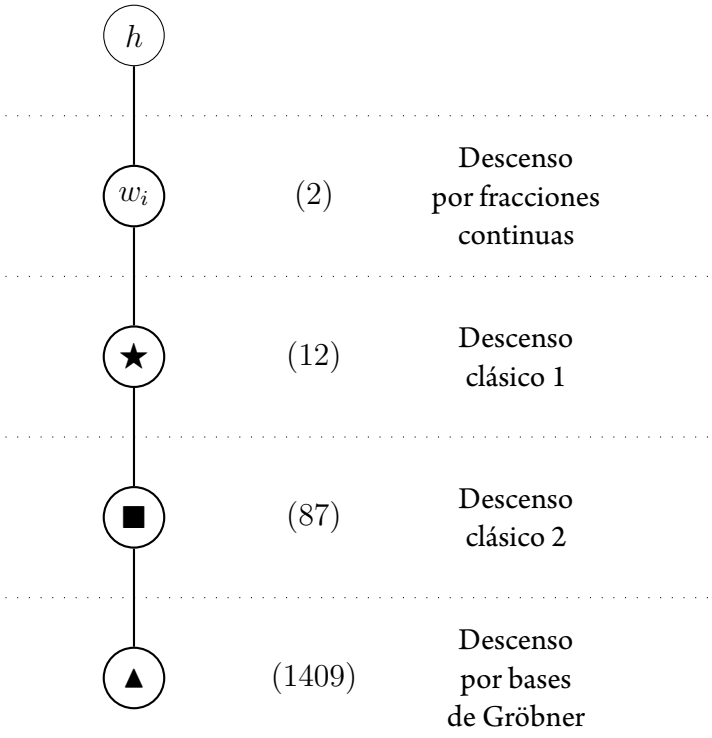
De acuerdo a la tabla 6.8, la frecuencia de procesamiento promedio para el segundo descenso clásico es de 2.66 GHz.

La tabla 6.7 indica que el tiempo esperado de este cálculo, considerando que el descenso se aplica a 575 polinomios, es aproximadamente 304.35 años. Sin embargo, este descenso se aplica sólamente a 87 polinomios (3 del descenso por fracciones continuas y 84 del primer descenso clásico), y entonces el tiempo esperado es:

$$2^{33.441673} \cdot (3.72 \times 10^6) \cdot 87 \approx 10^{18.577008} \text{ ciclos de reloj,}$$

que es equivalente a 45.01 años. El tiempo total para encontrar el resultado fue 10.09 años, mientras que su tiempo real fue 56.20 dias.

El número total de polinomios obtenidos hasta este descenso, incluyendo el descenso por fracciones continuas y el primer descenso clásico, se muestran en la figura 6.1.



h : polinomio de grado 508
w_i : polinomios de grado 254
★ : polinomios de grado $22 - 40$
■ : polinomios de grado $16 - 21$
▲ : polinomios de grado $5 - 15$

Figura 6.1: Polinomios obtenidos de ejecutar los descensos por fracciones continuas y clásico. Los números en paréntesis indican el número de polinomios procesados en cada fase de descenso.

Polinomio	Grados de los factores		
	$21-16$	$15-5$	$4-1$
Q_1	5	10	5
Q_2	7	5	7
Q_3	6	12	6
Q_4	7	11	1
Q_5	10	6	5
Q_6	7	11	6
Q_7	6	9	6
Q_8	6	10	3
Q_9	6	9	6
Q_{10}	6	8	8
Q_{11}	9	9	4
Q_{12}	9	9	5
Total	84	109	62

Tabla 6.10: Número de factores irreducibles obtenidos del primer descenso clásico.

Código fuente de la implementación

El código fuente de la implementación que se presenta en este capítulo, se encuentra disponible en:

`http://computacion.cs.cinvestav.mx/~icanales/`

La implementación de la aritmética en el campo $\mathbb{F}_{3^6}$, la aritmética en el anillo $\mathbb{F}_{3^6}[X]$ y la prueba de suavidad, se encuentra en la carpeta *main/c/arith* y contiene los siguientes archivos:

- *fp6.h*. Archivo de cabecera que contiene la definición de las funciones para realizar aritmética en $\mathbb{F}_{3^6}$, $\mathbb{F}_{3^6}[X]$ y la prueba de suavidad.
- *fp6-impl-exprep-negtable.h*. Archivo de cabecera que contiene la tabla de pre-cómputo del inverso aditivo en $\mathbb{F}_{3^6}$.
- *fp6-impl-exprep-addtable.h*. Archivo de cabecera que contiene la tabla de pre-cómputo de la suma en $\mathbb{F}_{3^6}$.
- *fp6-impl.h*. Archivo de cabecera que contiene la aritmética en $\mathbb{F}_{3^6}$ y $\mathbb{F}_{3^6}[X]$.

- *fp6.c*. Archivo C que implementa las funciones definidas en el archivo *fp6.h*.

La implementación del descenso por fracciones continuas y descenso clásico, se encuentra en la carpeta *main/c* y contiene los siguientes archivos:

- *contfracdesc.c*. Archivo C que contiene la implementación del descenso por fracciones continuas como se describe en la sección 5.2.
- *clasdesc.c*. Archivo C que contiene la implementación del descenso clásico como se describe en la sección 5.3.

El código fuente del módulo del *kernel* de Linux para realizar las mediciones del tiempo de ejecución, se encuentra en la carpeta *bench/c/kernel-module/arith*. Esta carpeta contiene los mismos archivos mencionados al inicio de esta sección, sin embargo, se encuentra adicionalmente los archivos:

- *stat.h*. Archivo de cabecera que contiene la definición de las funciones para calcular los datos estadísticos mencionados en la sección 6.1.
- *stat.c*. Archivo C que implementa las funciones definidas en el archivo *stat.h*.
- *bench.c*. Archivo C del módulo del *kernel* de Linux. Este archivo contiene el código que se presenta en [35].

CAPÍTULO 7

Conclusiones y trabajo futuro

Conclusiones

El objetivo principal de este trabajo era realizar una implementación eficiente de la prueba de suavidad para polinomios, y utilizarla en la fase de descenso del cálculo del logaritmo discreto. En este trabajo se escogió el campo $\mathbb{F}_{3^{6\cdot 509}}$ debido (i) a su interés criptográfico [2] y (ii) a que actualmente no existen resultados del tiempo requerido para calcular el logaritmo discreto. De manera general se concluye que el resultado de este trabajo cumple con el objetivo anterior, ya que la implementación de la prueba de suavidad resultó ser más eficiente en tiempo comparada con nuestra referencia, es decir, la factorización de Magma. Asimismo, se obtuvieron satisfactoriamente los resultados del descenso por fracciones continuas y descenso clásico. Al momento de escribir esta tesis, el cálculo del logaritmo discreto se encuentra en el último paso de la fase de descenso, y el resultado final será reportado cuando termine su ejecución.

En el capítulo 3 se estudió la aritmética en campos finitos. Se analizaron las alternativas para representar los elementos y las estrategias para realizar aritmética en campos con caracterśistica pequeña (2 o 3), y que además son una extensión pequeña (por ejemplo

≤ 6) de un campo primo. Particularmente para nuestra implementación, el tiempo de ejecución óptimo se obtuvo con la representación polinomial, utilizando tablas de precómputo para calcular la suma e inverso aditivo.

La tabla 6.1 muestra los tiempos de nuestra implementación para el campo $\mathbb{F}_{3^6}$. Previo a este trabajo no existían resultados reportados del tiempo requerido para realizar aritmética en el campo anterior. Los tiempos de la tabla 6.1 pueden ser utilizados para comparar resultados ya reportados, por ejemplo en los campos $\mathbb{F}_{3^{6\cdot 137}}$ y $\mathbb{F}_{3^{6\cdot 163}}$[1], y como referencia para estimar tiempos en otros campos de interés criptográfico, por ejemplo $\mathbb{F}_{3^{6\cdot 1429}}$ [1].

La aritmética en anillos de polinomios también fue tema de estudio en el capítulo 3. Se analizó la manera en la que se realizan las operaciones aritméticas básicas, el cálculo del máximo común divisor y la aritmética en el anillo cociente. Las operaciones anteriores, excepto la multiplicación, se implementaron utilizando algoritmos ya conocidos en la literatura (véase por ejemplo [40], [38]). Como se muestra en [43], una implementación óptima de la multiplicación se puede conseguir combinando el método de Karatsuba y el método de la escuela. Particularmente para nuestra implementación, la multiplicación se calcula con el método de Karatsuba utilizando el método de la escuela para multiplicar polinomios con a lo más 8 coeficientes (véase tabla 6.3); con esta estrategia se obtuvieron los tiempos de ejecución óptimos para multiplicar polinomios.

La tabla 6.4 muestra los tiempos de nuestra implementación para el anillo de polinomios sobre el campo $\mathbb{F}_{3^6}$. Previo a este trabajo no existían resultados reportados del tiempo requerido para realizar aritmética en el anillo de polinomios anterior.

La factorización y la prueba de suavidad son dos alternativas para determinar si un polinomio es suave o no respecto a un límite dado. El problema de la factorización es que su costo computacional es mayor comparado con el de la prueba de suavidad, mientras que la desventaja de la prueba de suavidad es que ésta puede fallar (dando falsos positivos, mas no falsos negativos), sin embargo, esto ocurre con probabilidad extremadamente baja. Por lo anterior, la prueba de suavidad se utiliza en la fase de descenso del algoritmo para calcular el logaritmo discreto.

La implementación de la prueba de suavidad se realizó siguiendo el método de Granger et al. [18], ya que en la actualidad, es el método más eficiente para calcular dicha prueba. Dado que no existen resultados reportados para la implementación de esta prueba, se decidió realizar una comparación de nuestra implementación con la factorización de Magma. Como se muestra en la tabla 6.6, el tiempo de ejecución de nuestra implementación resultó ser siempre menor al tiempo de la factorización de Magma. Por lo anterior, el uso de nuestra implementación repercutió directamente en el tiempo de ejecución de los descensos por fracciones continuas y clásico, ya que si de huber utilizado la estrate-

gia de emplear Magma como en [1], el tiempo real y total de estos descensos hubiera sido mayor. La tabla 6.5 muestra los tiempos de nuestra implementación de la prueba de suavidad para polinomios sobre el campo $\mathbb{F}_{3^6}$.

El tiempo de ejecución esperado del descenso por fracciones continuas era 43.88 años; el tiempo total para encontrar el resultado fue 51 años, mientras que su tiempo real fue 71.45 días. El tiempo esperado del primer descenso clásico era 27.58 años; el resultado se obtuvo en tiempo total de 9.85 años, y tiempo real de 22.32 días. Finalmente, el tiempo de ejecución esperado del segundo descenso clásico era 45.01 años; el tiempo total para encontrar el resultado fue 10.09 años, mientras que su tiempo real fue 56.20 días.

Trabajo futuro

A pesar de que los objetivos de este trabajo se cumplieron de manera satisfactoria, aún queda trabajo por realizar para complementar los resultados obtenidos, y se menciona a continuación:

- Nuestra implementación es eficiente en tiempo de ejecución, sin embargo, el código actual puede optimizarse utilizando lenguaje ensamblador.
- Realizar una implementación eficiente de la factorización para polinomios sobre el anillo $\mathbb{F}_{3^6}[X]$, y comparar su tiempo de ejecución con el tiempo de nuestra implementación de la prueba de suavidad.
- Utilizar el método de potencias de 2 presentado por Granger et al. en [20] para realizar los descensos por fracciones continuas y clásico. Utilizando este método se reduciría el tiempo de los descensos anteriores, sin embargo, el número de polinomios que se generan en cada paso de la fase de descenso, aumenta significativamente comparado con la estrategia utilizada en este trabajo.
- Realizar el cálculo del logaritmo discreto en una extensión de grado mayor. Por ejemplo, en [1] se presenta un análisis del tiempo esperado para calcular el logaritmo discreto en el campo $\mathbb{F}_{3^{6\cdot 1429}}$.

APÉNDICE A

Script de Magma para calcular el reto h

```
// Parametros
p := 3;
l := 6;
q := p^l;
n := 509;
N := q^n - 1;

r := (3^509 - 3^255 + 1) div 7;

// Definicion de los campos F_(3^6) y F_(3^(6*509))
F3 := FiniteField(3);
P3<U> := PolynomialRing(F3);
Fq<u> := ext<F3|U^6 + 2*U^4 + U^2 + 2*U + 2>;
Pq<X> := PolynomialRing(Fq);

// Definicion de h0 y h1
h0 := u^316*X + u^135;
```

```
h1 := X^2 + u^424*X;

// Definicion del campo F_(3^(6*509))
poly := h1*X^q - h0;
Ix := Max(Factorization(poly))[1];
Fqn<x> := ext<Fq|Ix>;

// Generador de F_(3^(6*509))
gen := X + u^2;

// Calculo del reto h
Re := RealField(2000);
pival := Pi(Re);

hp := 0;
for i := 0 to n-1 do
hp := hp + u^(Floor(pival*q^(i+1)) mod q)*(x^i);
end for;

h := hp^(N div r);
h_poly := elt<Pq|ElementToSequence(h)>;
```

Bibliografía

[1] G. Adj, A. Menezes, T. Oliveira, and F. Rodríguez-Henríquez, "Computing Discrete Logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ using Magma," in *Arithmetic of Finite Fields*, ser. Lecture Notes in Computer Science, C. K. Koç, S. Mesnager, and E. Savaş, Eds., vol. 9061, Springer International Publishing, 2015, pp. 3–22.

[2] ——, "Weakness of $\mathbb{F}_{3^{6\cdot 509}}$ for Discrete Logarithm Cryptography," in *Pairing - Based Cryptography – Pairing 2013*, ser. Lecture Notes in Computer Science, Z. Cao and F. Zhang, Eds., vol. 8365, Springer International Publishing, 2014, pp. 20–44.

[3] L. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," in *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, Oct. 1979, pp. 55–60.

[4] S. Baktir and B. Sunar, "Optimal Tower Fields," *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 1231–1243, Oct. 2004.

[5] E. R. Berlekamp, "Factoring Polynomials Over Large Finite Fields," *Mathematics of Computation*, vol. 24, no. 111, pp. 713–735, Jul. 1970.

[6] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, *High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves*, Cryptology ePrint Archive, Report 2010/354, `http://eprint.iacr.org/`, 2010.

[7] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system. I. The user language," *Journal of Symbolic Computation*, vol. 24, no. 3-4, pp. 235–265, Sep. 1997.

[8] D. G. Cantor and H. Zassenhaus, "A New Algorithm for Factoring Polynomials Over Finite Fields," *Mathematics of Computation*, vol. 36, no. 154, pp. 587–592, Apr. 1981.

[9] J. Chung and M. A. Hasan, "Asymmetric Squaring Formulae," in *18th IEEE Symposium on Computer Arithmetic, 2007 – ARITH '07*, Jun. 2007, pp. 113–122.

[10] D. Coppersmith, "Fast Evaluation of Logarithms in Fields of Characteristic Two," *IEEE Transactions on Information Theory*, vol. 30, no. 4, pp. 587–594, Jul. 1984.

[11] ——, "Solving Homogeneous Linear Equations over $GF(2)$ via Block Wiedemann Algorithm," *Mathematics of Computation*, vol. 62, no. 205, pp. 333–350, Jan. 1994.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd. The MIT Press, 2009.

[13] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[14] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[15] P. Flajolet, X. Gourdon, and D. Panario, "The Complete Analysis of a Polynomial Factorization Algorithm over Finite Fields," *Journal of Algorithms*, vol. 40, no. 1, pp. 37–812, Jul. 2001.

[16] K. F. Gauss, *Disquisitiones Arithmeticae, English Edition*, W. C. Waterhouse, Ed., trans. by A. A. Clarke. Springer New York, 1986.

[17] F. Göloğlu, R. Granger, G. McGuire, and J. Zumbrägel, "On the Function Field Sieve and the Impact of Higher Splitting Probabilities," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, R. Canetti and J. A. Garay, Eds., vol. 8043, Springer Berlin Heidelberg, 2013, pp. 109–128.

[18] R. Granger, T. Kleinjung, and J. Zumbrägel, *Breaking '128-bit Secure' Supersingular Binary Curves (Or How to Solve Discrete Logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$)*, Cryptology ePrint Archive, Report 2014/119, `http://eprint.iacr.org/`, 2014.

[19] ——, "Breaking '128-bit Secure' Supersingular Binary Curves (Or How to Solve Discrete Logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$)," in *Advances in Cryptology – CRYPTO 2014*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8617, Springer Berlin Heidelberg, 2014, pp. 126–145.

[20] ——, *On the Powers of 2*, Cryptology ePrint Archive, Report 2014/300, `http://eprint.iacr.org/`, 2014.

[21] R. Granger and M. Scott, "Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions," in *Public Key Cryptography – PKC 2010*, ser. Lecture Notes in Computer Science, P. Q. Nguyen and D. Pointcheval, Eds., vol. 6056, Springer Berlin Heidelberg, 2010, pp. 209–223.

[22] M. Hellman and J. Reyneri, "Fast Computation of Discrete Logarithms in $GF(q)$," in *Advances in Cryptology: Proceedings of CRYPTO '82*, D. Chaum, R. Rivest, and A. Sherman, Eds., Springer US, 1983, pp. 3–13.

[23] C. G. Jacobi, "Über die Kreistheilung und ihre Anwendung auf die Zahlentheorie," *Journal für die reine und angewandte Mathematik*, vol. 30, pp. 166–182, 1846.

[24] A. Joux, "A New Index Calculus Algorithm with Complexity $L(\frac{1}{4}+o(1))$ in Small Characteristic," in *Selected Areas in Cryptography – SAC 2013*, ser. Lecture Notes in Computer Science, T. Lange, K. Lauter, and P. Lisoněk, Eds., Springer Berlin Heidelberg, May 2014, pp. 355–379.

[25] ——, "Faster Index Calculus for the Medium Prime Case Application to 1175-bit and 1425-bit Finite Fields," in *Advances in Cryptology – EUROCRYPT 2013*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881, Springer Berlin Heidelberg, 2013, pp. 177–193.

[26] A. Joux and R. Lercier, "The Function Field Sieve in the Medium Prime Case," in *Advances in Cryptology – EUROCRYPT 2006*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004, Springer Berlin Heidelberg, 2006, pp. 254–270.

[27] A. Joux and C. Pierrot, "Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms," in *Advances in Cryptology – ASIACRYPT 2014*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8873, Springer Berlin Heidelberg, 2014, pp. 378–397.

[28] E. Kaltofen and V. Shoup, "Subquadratic-Time Factoring of Polynomials over Finite Fields," in *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, May 1995, pp. 398–406.

[29] A. A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet Physics Doklady*, vol. 7, Jan. 1963, pp. 595–7596.

[30] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd. Prentice Hall Professional Technical Reference, 1988.

[31] N. Koblitz, *A Course in Number Theory and Cryptography*, 2nd, ser. Graduate Texts in Mathematics. Springer New York, 1994, vol. 114.

[32] S. Lang, *Algebra*, 3rd, ser. Graduate Texts in Mathematics. Springer New York, 2002, vol. 211.

[33] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997, vol. 20.

[34] G. L. Mullen and D. Panario, *Handbook of Finite Fields*, ser. Discrete Mathematics and Its Applications. CRC Press, 2013.

[35] G. Paoloni, "How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures," Intel Corporation, Tech. Rep. 324264-001, Sep. 2010.

[36] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communucations of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[37] M. Scott, "Implementing Cryptographic Pairings," in *Pairing-Based Cryptography – Pairing 2007*, ser. Lecture Notes in Computer Science, T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, Eds., vol. 4575, Springer Berlin Heidelberg, 2007, pp. 197–207.

[38] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd. Cambridge University Press, 2009.

[39] ——, "A New Polynomial Factorization Algorithm and its Implementation," *Journal of Symbolic Computation*, vol. 20, no. 4, pp. 363–397, Oct. 1995.

[40] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd. Cambridge University Press, 2013.

[41] J. von zur Gathen and D. Panario, "Factoring Polynomials Over Finite Fields: A Survey," *Journal of Symbolic Computation*, vol. 31, no. 1–2, pp. 3–17, Jan. 2001.

[42] J. von zur Gathen and V. Shoup, "Computing Frobenius Maps and Factoring Polynomials," *Computational complexity*, vol. 2, no. 3, pp. 187–224, Sep. 1992.

[43] A. Weimerskirch and C. Paar, *Generalizations of the Karatsuba Algorithm for Efficient Implementations*, Cryptology ePrint Archive, Report 2006/224, `http://eprint.iacr.org/`, 2006.

[44] D. Wiedemann, "Solving Sparse Linear Equations Over Finite Fields," *IEEE Transactions on Information Theory*, vol. 32, no. 1, pp. 54–62, Jan. 1986.