Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Plataforma de Experimentación Criptográfica Basada en Geometría Algebraica

Tesis que presenta

Eliver Pérez Villegas

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis:

Dr. Guillermo Morales Luna

Ciudad de México Agosto, 2017

Dedicatoria

A Dios

Por haberme permitido llegar hasta este punto y haberme dado salud para lograr mis objetivos, además de su infinita bondad y amor.

A mi padre Eliver

Gracias por habernos educado así. Estoy orgulloso de ser como soy y eso te lo debo a ti. Te amo papá.

A mi madre María

Por siempre creer en mí. Por haberme formado como un hombre de bien, y por ser la mujer que me dio la vida y me enseñó a vivirla. A quien agradezco enormemente su amor incondicional Te amo mamá.

A mi abuela Ma. Luz

Una de mis más grandes inspiraciones para seguir superándome día con día. Gracias por tu amor incondicional, donde sea que estés, te amaré por siempre.

A mi novia Alondra

A quien amo con todo mi corazón y me ha acompañado hasta el final, siempre apoyándome en cada paso que damos juntos.

A Andrea Lizetth

Quien siempre me apoyo en cada paso que di en este nuevo logro, aunque ya no este presente, hoy logro llegar a cumplir una meta más que ambos teníamos.

A mi abuela Oralia

Por tu paciencia, por el amor que me has dado y por tu apoyo incondicional en mi vida. Gracias por llevarme en tus oraciones porque estoy seguro que siempre lo haces.

A mi familia

A mis hermanos Martha Alicia, Alexis y Jayro que con su amor me han enseñado a salir adelante, y siempre han estado ahí para mí cuando más los necesito. Los amo con todo mi corazón.

Agradecimientos

Primero que nada a mi familia, que es mi más grande inspiración para seguir logrando cada una mis metas que me propongo.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) por haberme brindado el apoyo económico, permitiéndome el haber culminado mis estudios de posgrado.

Agradezco al Dr. Guillermo Morales por la oportunidad de asesorar mi trabajo de tesis, por sus consejos y todos los conocimientos transmitidos; a mis sinodales Dr. Luis Gerardo de la Fraga y Dr. Feliú Sagols por sus correcciones oportunas, gracias por haberse dado el tiempo de revisar mi trabajo de tesis.

Al M.C.A. Hector Flores Gutierrez, que ha sido como un padre para mi y sin su apoyo tal vez no me hubiera sido posible llegar hasta aquí.

Al Dr. Hector Diez Rodriguez, quien nunca dejo de insistir en que continuara con mis estudios de maestría.

A Sofía Reza por su amabilidad y por apoyarme cuando más necesitaba de alguien.

Finalmente, gracias al Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional, en especial al Departamento de Computación por haberme permitido formarme en lo personal y académico.

Gracias a todos.

Resumen

La criptografía de clave pública es utilizada día a día a la hora de compartir cualquier tipo de información utilizando canales inseguros. A la fecha, en la práctica algunos de los esquemas más utilizados son RSA, ElGamal y los basados en curvas elípticas, los cuales siguen siendo resistentes a ataques utilizando nuestro cómputo actual. Sin embargo, en una computadora cuántica esta criptografía ya estaría rota gracias al algoritmo de Shor para resolver el problema de factorización y el del logaritmo discreto en tiempo polinomial. Por esta razón es que se han estado estudiando otros esquemas criptográficos que sean capaces de reemplazar los esquemas utilizados hoy en día siendo resistentes a ataques empleados desde una computadora cuántica. Una de esas alternativas es la criptografía sobre varias variables que ha sido de gran interés en los últimos años, debido a que resolver sistemas de polinomios de grado dos con más de dos variables ha sido demostrado ser un problema NP-difícil, por lo que dicho paradigma criptográfico cae entre los esquemas de criptografía post-cuántica. Uno de los problemas de esta criptografía es que las claves tiendan a ser muy grandes, las claves públicas siendo sistemas de ecuaciones de grado dos en varias variables y las claves privadas parejas de transformaciones afines en espacios vectoriales sobre un campo finito. Presentamos una plataforma de experimentación llamada `MQCrypto` que incorpora las implementaciones de un total de diez esquemas de criptografía sobre varias variables, en `MQCrypto` es posible generar instancias de claves de cada uno de los esquemas y utilizarlas para realizar procesos de firma y verificación. De igual manera se propone una manera de representar y codificar tanto las claves públicas y las privadas con la que se logra una significante reducción en su tamaño comparándola con la representación usual de las mismas instancias.

Abstract

Public key cryptography is used in day to day communication when transferring data through insecure channels. Nowadays, some of the most used cryptographic schemes in practice are RSA, ElGamal and based on elliptic curve, being resistant to attacks with our actual computing resources. However, when quantum computing becomes a reality, these schemes would turn to be insecure, as Shor published in 1994 a quantum algorithm capable of solving the factorization and discrete logarithm problem in polynomial time. Hence, some cryptographic schemes capable of supporting attacks from a quantum computer have been widely studied. One of those cryptographic schemes is Multivariate Cryptography, whose security relies in solving a system of multivariate polynomials over a finite field. Multivariate cryptography has been of great interest in recent years, due to the fact that solving systems of quadratic n-variate polynomials has been proven to be NP-hard, falling in the category of post-quantum cryptography schemes. So far, every multivariate cryptography scheme proposed has been broken. One disadvantage of multivariate cryptography is related to the keys size, producing such big keys compared to the ones from RSA or ECC, the public key being a system of quadratic n-variate polynomials and the private key containing two affine transformations on vector spaces over a Galois field. We present a system called `MQCrypto` which contains the implementation of ten multivariate cryptography schemes and is capable of generating keypairs from each of them, that can be used for signing and verifying signatures. Furthermore, an encoding using Abstract Syntax Notation One structure is presented to represent instances of this last problem with reduced sizes. The encoding is shown to be significantly reduced compared with the conventional polynomial representation of the same instances.

Índice general

Índice de figuras

Índice de tablas

Índice de Algoritmos

Capítulo

1

Introducción

"To me, mathematics, computer science, and the arts are all insanely related. They're all creative expressions."

Sebastian Thrun

Hoy en día el ser humano tiene una gran dependencia de los sistemas electrónicos para almacenar o enviar información de un lugar a otro, sin embargo, se suelen utilizar canales de comunicación y medios de almacenamiento inseguros, existiendo así la posibilidad de que un adversario (una entidad distinta al remitente y destinatario) sea capaz de interceptar la comunicación y así obtener información que puede ser considerada valiosa.

Es por esto, que la criptografía se ha vuelto una parte esencial en este tipo de comunicaciones, proporcionando así otro nivel de seguridad, en el que el adversario tendrá que resolver un problema cualquiera dependiendo del esquema criptográfico que se esté utilizando para de esta manera obtener la información en claro. Como ejemplo, RSA[1] es un esquema de criptografía asimétrica que basa su seguridad en el problema de factorización en el cual teniendo un número n es necesario encontrar sus factores primos p y q, donde n suele ser un número gigantesco.

1.1. Motivación

El problema de factorizar un número entero en sus factores primos es la base de seguridad del esquema de criptografía asimétrica RSA [44], mientras que el problema del logaritmo discreto es

[1] Propuesto por Ron Rivest, Adi Shamir y Len Adleman

la base de seguridad del protocolo de intercambio de claves Diffie-Hellman [12] y de los esquemas de criptografía asimétrica ElGamal [17] y la criptografía de curvas elípticas [33]. En el problema del logaritmo discreto se tiene un grupo multiplicativo G de orden finito n y un generador $g \in G$, el problema consiste en, dado un $h \in G$ encontrar un $x \in [0, \ldots, n-1]$, tal que $h = g^x$.

Debido a la seguridad que pueden proporcionar considerando el cómputo con el que se cuenta hoy en día, los criptosistemas mencionados anteriormente son de los más utilizados para mantener las comunicaciones seguras, ya que aún no existe una manera de resolver en tiempo polinomial tanto el problema de factorización como el problema del logaritmo discreto con nuestros recursos de cómputo actual. Sin embargo, un gran inconveniente es que estos dos problemas pueden resolverse en tiempo polinomial por medio del algoritmo de Shor [47] en un hipotético computador cuántico.

Matsumoto e Imai propusieron el primer esquema criptográfico que basa su seguridad en resolver sistemas de polinomios cuadráticos sobre varias variables (problema $\mathcal{MQ}$) en 1988 [35]. A partir de ahí han surgido varios esquemas criptográficos que basan su seguridad en el problema $\mathcal{MQ}$ y el problema de isomorfismo en polinomios.

Estos tipos de esquemas criptográficos han sido de gran interés en los últimos años debido a que encontrar soluciones a estos sistemas ha sido demostrado que es un problema NP-difícil [29], por lo que dicho paradigma criptográfico cae entre los esquemas de criptografía post-cuántica. Aunque hasta la fecha no existe un esquema propuesto que no haya sido quebrantado y se encuentre puesto en práctica.

Existen algunas herramientas que nos permiten probar la robustez de estos criptosistemas, una de esas herramientas es `PolyBoRi` [4] `(Polynomials over Boolean Rings)`, el cual es una biblioteca desarrollada en C++ que provee una interfaz en `Python` y está incorporada en el sistema de álgebra computacional `SageMath`. `PolyBoRi` utiliza bases de Gröbner para resolver sistemas de ecuaciones polinomiales con soluciones booleanas. Debido a falta de mantenimiento, `PolyBoRi` puede llegar a ser removido en un futuro del sistema de álgebra computacional `SageMath`.

Para probar si un esquema criptográfico es seguro, es necesario contar con una plataforma en la que se puedan realizar algunas pruebas de ataques. Al igual que con algunas implementaciones de dichos esquemas en las que sea posible construir instancias reales del problema para posteriormente ser atacadas.

1.2. Planteamiento del problema

En un campo finito, de orden $q = p^k$, donde p es un primo y $k \in \mathbb{Z}_{\geq 1}$, los polinomios con soluciones booleanas se realizan como elementos del anillo cociente

$$\mathbb{F}_q[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$$

donde los coeficientes son [0,1] y por las condiciones impuestas, se tiene que $x_i^2 = x_i$.
En un protocolo criptográfico de autenticación existen dos entidades, el *Probador* P y el *Verificador* V.
P posee dos claves,

P_b - La clave pública

P_p - La clave privada

P_b debe de construirse fácilmente a partir de P_p, pero P_p no debe ser recuperable a partir de P_b.
Este protocolo consiste de desafíos y respuestas, donde las respuestas correctas deben de ser constructibles a partir de P_p, solamente por aquel que tiene acceso a la clave privada, y los desafíos son constructibles a partir de la clave pública P_b, cualquier verificador V conoce la clave pública y puede construir sus desafíos y solo aquel que conozca la clave privada correspondiente puede proporcionar las respuestas correctas. V debe aceptar o rechazar las credenciales o pruebas aportadas por P.
Las claves públicas son polinomios cuadráticos en un anillo de polinomios $\mathbb{K}[\mathbf{X}]$ del tipo

$$f(\mathbf{X}) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}X_iX_j + \sum_{i=1}^{n} b_iX_i + \sum_{j=1}^{n} c_jX_j + d$$

Las claves privadas son puntos en las variedades algebraicas correspondientes.
P_b se construye a partir de las transformaciones afines $S : \mathbb{K}^n \to \mathbb{K}^n$ y $T : \mathbb{K}^n \to \mathbb{K}^n$,

$$P_b = T \circ P_p \circ S$$

El *probador* cuenta con un conjunto de polinomios $\mathcal{P}$ y para este conjunto de polinomios los conjuntos que lo satisfacen nos dan una variedad algebraica.
Los desafíos pueden ser puntos en la clave privada que satisfagan alguna condición. Por ejemplo, un desafío puede ser, pedir que en un determinado sector nos den ciertos valores fijos, entonces

habría que ver en la variedad cual es esa traza y dar cualquier punto, y esa sería la respuesta al desafío, la persona que quiera suplantar al *probador*, tendría que calcular esa variedad algebraica, y eso puede ser difícil. La introducción de las funciones afines es con el fin de disfrazar el problema. Conociendo las transformaciones afines S y T se puede resolver facilmente el sistema de polinomios con las condiciones que se generarían los desafíos. Pero el que no conozca esas transformaciones se encuentra con un problema computacionalmente difícil.

Descripción del problema

Es necesario contar con una plataforma de experimentación para criptografía sobre varias variables, que cuente con implementaciones de algunos de estos esquemas en los que sea posible crear instancias reales de claves públicas y privadas y puedan ser utilizadas para realizar operaciones de firma, verificación, cifrado y descifrado.

Dos problemas principales para quebrantar este tipo de esquemas criptográficos son el problema $\mathcal{MQ}$ y el problema de isomorfismo en polinomios.

Problema $\mathcal{MQ}$

Problema $\mathcal{MQ}$ (polinomios cuadráticos sobre varias variables)
Entrada: Un conjunto de polinomios cuadráticos $F = (f_1, \ldots, f_n) \subseteq \mathbb{K}[x_1, \ldots, x_n]$:

$$
\begin{aligned}
f_1(X_1, \ldots, X_n) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^{(1)} X_i X_j + \sum_{i=1}^{n} b_i^{(1)} X_i + \sum_{j=1}^{n} c_j^{(1)} X_j + d \\
f_2(X_1, \ldots, X_n) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^{(2)} X_i X_j + \sum_{i=1}^{n} b_i^{(2)} X_i + \sum_{j=1}^{n} c_j^{(2)} X_j + d \\
&\vdots \qquad\qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
f_m(X_1, \ldots, X_n) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^{(m)} X_i X_j + \sum_{i=1}^{n} b_i^{(m)} X_i + \sum_{j=1}^{n} c_j^{(m)} X_j + d
\end{aligned}
$$

y un vector $\mathbf{y}= (y_1, \ldots, y_m) \in \mathbb{K}^m$
Salida: Un vector $\mathbf{x}= (x_1, \ldots, x_n) \in \mathbb{K}^n$ (si existe) tal que

$$
\begin{aligned}
f_1(x_1, \ldots, x_n) &= y_1 \\
f_2(x_1, \ldots, x_n) &= y_2 \\
\vdots \quad & \quad \vdots \quad \quad \vdots \\
f_m(x_1, \ldots, x_n) &= y_m
\end{aligned}
$$

Problema de isomorfismo en polinomios

> **Problema de isomorfismo en polinomios**
>
> **Entrada:** Dos conjuntos de polinomios cuadráticos $\mathcal{P} = (p_1, \ldots, p_m), \mathcal{P}' = (p'_1, \ldots, p'_m) \subseteq \mathbb{K}[x_1, \ldots, x_n]$
>
> **Salida:** Transformaciones lineales S, T (si existen) tales que $\mathcal{P} = T \circ \mathcal{P}' \circ S$

Propuesta de solución

Se desarrolló una plataforma basada en OpenSSL que cuenta con las implementaciones de algunos esquemas criptográficos que basan su seguridad en el problema $\mathcal{MQ}$ y de esta manera es posible realizar generaciones de claves públicas y privadas de esquemas de criptografía sobre varias variables, y con ellas realizar operaciones de firma, verificación, cifrado y descifrado.

Estado del arte

Desde el año 1985 se han sugerido [13] trabajar con esquemas de clave pública en el que la clave pública sea dada como un conjunto de ecuaciones cuadráticas sobre varias variables (o de un grado superior) sobre un campo finito $\mathbb{K}$ de característica pequeña.

En la conferencia Eurocrypt de 1988, T. Matsumoto y H. Imai [35] propusieron el esquema de cifrado y firmas digitales C^* o MI el cual esta basado en sistemas de polinomios cuadráticos sobre varias variables. El hecho de que estuviera basado en este tipo de sistemas polinomiales llamó tanto la atención que fue propuesto para ser utilizado como un estándar por el gobierno Japonés [13]. Sin embargo, poco antes de la selección final en 1995, MI fue roto por el francés Jacques Patarin haciendo uso de un ataque algebraico que utilizaba ecuaciones de linearización [38].

Este podría haber parecido el fin de MI, sin embargo el criptosistema propuesto por Matsumoto e Imai trajo consigo una idea matemática totalmente nueva la cual sería explorada y ampliamente extendida. Han aparecido algunas variantes de MI como lo es *Hidden Field Equations* (HFE)

propuesto por Jacques Patarin en la Eurocrypt de 1996 [13] siguiendo la idea del criptosistema de Matsumoto e Imai, sin embargo HFE ya ha sido roto empleando más de un ataque conocido.
Otra propuesta basada en la idea de MI fue SFLASH [9] el cual en 2001 fue seleccionado como candidato por la NESSIE (New European Schemes for Signatures, Integrity and Encryption) y finalmente en 2004 fue aceptado para ser utilizado en tarjetas inteligentes, aunque unos años después este esquema fue roto. SFLASH es una variante de C^{*--} que a su vez es una variante de C^* y es el único esquema criptográfico basado en sistemas polinomiales sobre varias variables que ha sido puesto en práctica hasta la fecha.
Quartz es otra variante de C^* propuesta por Patarin y Courtois en 2001 [8]. Una de las partes interesantes de Quartz es que produce firmas muy cortas y no existe un método conocido para vulnerar este esquema aún. Sin embargo, a pesar de que aún no ha sido quebrantado, éste no es utilizado en la práctica, y esto se debe a que su tiempo de respuesta es muy largo, tomando cerca de 11 segundos para generar una firma [42]. Con un tiempo de respuesta tan largo deja de ser una alternativa para poder ser utilizado para fines prácticos, ya que 11 segundos no es un tiempo aceptable.
Existen algunas herramientas de álgebra simbólica que nos permiten probar la resistencia de este tipo de esquemas criptográficos. `PolyBoRi` [4] es una plataforma que fue desarrollada en C++ y utiliza cómputos de bases de Gröbner para resolver sistemas de ecuaciones polinomiales sobre anillos booleanos. `PolyBoRi` provee una interfaz en `Python` para poder trabajar con el, de igual manera puede ser utilizado en el sistema de álgebra computacional `SageMath`. `PolyBoRi` esta desarrollado utilizando `Python 2` y no se le ha dado mantenimiento a esta plataforma últimamente por lo que se cree pudiera ser removido de `SageMath` más adelante. En el 2015, `PolyBoRi` cambió su nombre a `BRiAl` en un movimiento para actualizar la plataforma, sin embargo, la plataforma parece no tener avances y hasta la fecha se encuentra aún en mantenimiento.

1.3. Objetivos y contribuciones

Objetivo general

El objetivo principal de este trabajo de tesis es desarrollar una plataforma de experimentación criptográfica que cuente con las implementaciones de esquemas de criptografía sobre varias variables y sea capaz de generar instancias de claves de cada una de ellas para realizar operaciones de firma

y cifrado.

Objetivos particulares

Para lograr el objetivo general se espera cumplir con los siguientes objetivos particulares:

1. Desarrollar una estrategia de búsqueda de polinomios de manera que se pueda encontrar una instancia difícil mientras que la clave pública no sea tan larga.

2. Generar instancias del problema que resulten intratables por las herramientas actuales de álgebra simbólica como `PolyBoRi` o el sistema desarrollado en esta tesis.

3. Probar implementaciones de algunos esquemas de criptografía sobre varias variables, para analizar su incorporación en la plataforma de experimentación.

4. Realizar las implementaciones de algunos esquemas de criptografía sobre varias variables para posteriormente ser incorporados a la plataforma de experimentación.

5. Ofrecer un sistema con capacidad para ser ejecutado de manera eficiente en equipos de cómputo que cuenten con diferentes recursos.

Contribuciones

La principal contribución de este trabajo de tesis es la construcción de una plataforma de experimentación criptográfica que incorpora las implementaciones de un total de diez esquemas de criptografía sobre varias variables, en la que siete de ellos son de diversos autores mientras que las tres restantes son implementaciones propias. Con estos esquemas es posible generar parejas de claves para posteriormente ser utilizadas en procesos de firma y verificación.

La plataforma cuenta con una sintaxis similar a la de `OpenSSL` tanto para la generación de las claves, como para el proceso de firma y verificación.

Además se propone una estrategia para compresión de polinomios de tal manera que se pueda a su vez generar instancias imposibles de resolver mediante las herramientas computacionales actuales en un tiempo aceptable mientras que las claves públicas no sean tan largas.

1.4. Organización de la tesis

Tomando en cuenta este primer capítulo en el que se aborda la motivación, el planteamiento del problema, la propuesta de solución y los objetivos, esta tesis consta de un total de 6 capítulos.
En el capítulo 2 se introducen algunos conceptos básicos necesarios para comprender este trabajo de tesis. Se tratan algunas definiciones, tales como las bases de Gröbner y como es que éstas pueden ser calculadas así como las pruebas de conocimiento nulo.
Posteriormente en el capítulo 3 se introduce la criptografía sobre varias variables *(Multivariate Cryptography)*, como se componen las claves públicas y privadas, el primer esquema de este tipo que fue introducido por Matsumoto e Imai en 1988. De igual manera se presentan algunos esquemas que ya han sido quebrantados.
En el capítulo 4 se presenta una propuesta para la codificación en ASN.1 de las claves de la criptografía sobre varias variables. El propósito de la codificación es el de reducir el tamaño de las claves, siendo que este es un problema en este tipo de criptografía.
El sistema propuesto se presenta en el capítulo 5. Este capítulo contiene los detalles del diseño del sistema que ha sido implementado. De igual manera se presentan algunos resultados de pruebas realizadas en las que tomando en cuenta los esquemas implementados y sus construcciones se modificaron sus números de variables y polinomios para así analizar el crecimiento de las claves producidas por cada uno de ellos. Finalmente, el capítulo 6 presenta las conclusiones y el trabajo a futuro.

Capítulo

2

Conceptos preliminares

"I am always doing that which I cannot do, in order that I may learn how to do it."

Pablo Picasso

En este capítulo se introducen algunos conceptos básicos que serán utilizados con frecuencia en esta tesis, esto con el fin de facilitar al lector la lectura de este documento. Se definen las bases de Gröbner y su utilidad para el cálculo de puntos en variedades algebraicas en sistemas de polinomios cuadráticos sobre varias variables. El material de este capítulo ha sido tomado de los libros [11, 13, 36, 48, 26, 32, 34].

2.1. Conceptos algebraicos

Definición 2.1.1. ***(Función de Euler)*** *Para $n \geq 1$, $\phi(n)$ denota el número de enteros en el intervalo $[1, n]$ que son primos relativos de n, es decir, cualesquiera $m \in [1, n]$, tal que $\mathrm{mcd}(m, n) = 1$. La función ϕ se llama la* ***función de Euler****.*

Algunas propiedades de la función ϕ son las siguientes:

1. Si p es primo, entonces $\phi(p) = p - 1$

2. La función de Euler es multiplicativa. Es decir, si $\mathrm{mcd}(m, n) = 1$, entonces $\phi(mn) = \phi(m) \cdot \phi(n)$.

3. Si $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ es la factorización en números primos de n, entonces

$$\phi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_k}\right)$$

Definición 2.1.2. ***(Operación binaria)*** *Una* ***operación binaria*** $\times$ *sobre un conjunto* G *es una función*

$$G \times G \to G$$

Una operación binaria es ***asociativa*** *si,* $\forall a, b, c \in G$

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

Una operación binaria es ***conmutativa*** *si,* $\forall a, b \in G$

$$a \times b = b \times a$$

Definición 2.1.3. ***(Monoide)*** *Una estructura matemática formada por un conjunto no vacío* G *y una operación binaria* $(G, \times)$ *se dice ser un* ***monoide*** *si satisface las propiedades de:*

- ***Asociatividad.*** $\forall a, b, c \in G$, $a \times (b \times c) = (a \times b) \times c$.
- ***Existencia del elemento identidad.*** $\exists! e \in G$, *tal que* $\forall a \in G$ *se cumple que* $a \times e = e \times a = a$.

Definición 2.1.4. ***(Grupo)*** *Una estructura algebraica formada por un conjunto no vacío* G *y una operación binaria* $\times$ *se denomina* ***grupo****, si además de ser un monoide satisface las siguientes propiedades:*

- ***Cerradura.*** $\forall a, b \in G$, $a \times b \in G$.
- ***Existencia de inversos.*** $\forall a \in G, \exists a' \in G$, *tal que* $a \times a' = a' \times a = e$, *siendo* e *el elemento identidad.*

Definición 2.1.5. ***(Grupo Abeliano)*** *Un grupo* $(G, \times)$*, es un* ***grupo abeliano****, si además de cumplir las propiedades de un grupo, también es conmutativo, tal que* $\forall a, b \in G, a \times b = b \times a$.

Ejemplo 2.1.1. El conjunto de los números enteros con la operación suma $(\mathbb{Z}, +)$, es un grupo:

- **Cerradura.** La suma de dos números enteros da como resultado otro número entero.

- **Asociatividad.** Para cualesquiera $a, b, c \in \mathbb{Z}$, $a + (b + c) = (a + b) + c$.
- **Existencia del elemento identidad.** Para cualquier entero a, se tiene que $a+0 = 0+a = a$, por lo tanto 0 es el elemento identidad en el grupo $(\mathbb{Z}, +)$.
- **Existencia de inversos.** Para cualquier número entero a existe un elemento $-a$, tal que $a + (-a) = (-a) + a = 0$.
- **Conmutatividad.** Para todo $a, b \in \mathbb{Z}$, se cumple que $a + b = b + a$, por lo tanto $(\mathbb{Z}, +)$ es un grupo abeliano.

Definición 2.1.6. ***(Orden del grupo)*** *Sea G un grupo, el **orden del grupo** es la cardinalidad de G y se denota como $|G|$. Si el orden de G es un número finito, G se dice ser un grupo finito. Para cada elemento $g \in G$, el orden de g, esta definido como el número m más pequeño para el cual*

$$g^m = \underbrace{g \times \cdots \times g}_{m \text{ veces}} = e$$

Si $|G| = \infty$, entonces no existe un entero m para el cual $g \in G$ satisfaga la condición anterior y g se dice ser de orden infinito.

Ejemplo 2.1.2. El conjunto $\mathbb{Z}_n$, con la operación suma módulo n, forma un grupo de orden n. El conjunto $\mathbb{Z}_n$, con la operación producto módulo n no es un grupo, debido a que no todos los elementos tienen inversos multiplicativos. Sin embargo, el grupo $\mathbb{Z}_n^*$, que incluye a todos los enteros $m \in [1, n]$ tal que, $\mathrm{mcd}(m, n) = 1$, si es un grupo de orden $\phi(n)$ con la operación producto módulo n y 1 es el elemento identidad.

Definición 2.1.7. *Una manera de denotar a un grupo es $(G, +, 0)$ que se suele llamar **grupo aditivo**, 0 es el elemento identidad en G, el inverso para todo $a \in G$ es $-a \in G$, el orden de a es m y se denota como*

$$ma = \underbrace{a + \cdots + a}_{m \text{ veces}}$$

*a $(G, \cdot, 1)$ se le llama el **grupo multiplicativo**, 1 es el elemento identidad, el inverso para todo $a \in G$ es $a^{-1} \in G$ y el orden de a es m y se denota como*

$$a^m = \underbrace{a \cdot \cdots \cdot a}_{m \text{ veces}} = e$$

Definición 2.1.8. ***(Anillo)*** *Un* ***anillo*** *es una estructura matemática formada por un conjunto R con dos operaciones binarias $(R, +, \cdot, 0, 1)$ que satisface las siguientes propiedades:*

- *$(R, +)$ es un grupo abeliano, $\forall a, b \in R, a + b = b + a$, con 0 como el elemento identidad.*
- *El producto $\cdot$ es asociativo, es decir, $\forall a, b, c \in R$ se cumple que $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.*
- *El producto $\cdot$ es distributivo sobre la suma $+$. Es decir, $\forall a, b, c \in R, a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ y $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$.*
- *$(R\backslash\{0\}, \cdot)$ es cerrado y asociativo bajo el producto, además de la existencia del elemento identidad, siendo 1 el elemento identidad.*

El anillo es un anillo conmutativo *si $a \times b = b \times a$, para todo $a, b \in R$.*

Ejemplo 2.1.3. El conjunto de los números enteros $\mathbb{Z}$ bajo la suma y el producto es un anillo conmutativo.

Ejemplo 2.1.4. El conjunto $\mathbb{Z}_n$ con la suma y producto módulo n es un anillo conmutativo.

Definición 2.1.9. ***(Campo)*** *Un* ***campo*** *es una estructura matemática formada por un conjunto $\mathbb{F}$ y dos operaciones binarias $(\mathbb{F}, +, \cdot)$ que satisface las siguientes propiedades:*

- *$(\mathbb{F}, +)$ es un grupo abeliano donde 0 es el elemento identidad, tal que para cualquier elemento $a \in \mathbb{F}, a + 0 = a$.*
- *$(\mathbb{F}\backslash\{0\}, \cdot)$ es un grupo abeliano donde 1 es el elemento identidad, tal que para cualquier elemento $a \in \mathbb{F}, a \cdot 1 = a$.*
- *$(\mathbb{F}, +, \cdot)$ es un dominio de integridad, es decir, para ningunos $a, b \in \mathbb{F}$, nunca se cumple que $a \cdot b = 0$, para todo a y b distintos de 0. Para esto, la característica del campo tiene que ser un número primo.*

Definición 2.1.10. ***(Característica del campo)*** *La* ***característica*** *de un campo $\mathbb{F}$ es 0 si $\overbrace{1 + 1 + \cdots + 1}^{m \text{ veces}}$ nunca es igual a 0, para cualquier $m \geq 1$. De otra manera, la característica del campo es el entero positivo más pequeño para el cual $\sum_{i=1}^{m} 1$ es igual a 0.*

Ejemplo 2.1.5. El conjunto de los números enteros bajo la suma y el producto no es un campo, ya que los únicos elementos que cuentan con inversos multiplicativos son 1 y -1. Sin embargo, los números racionales $\mathbb{Q}$, los números reales $\mathbb{R}$ y los números complejos $\mathbb{C}$ forman campos de característica 0.

Ejemplo 2.1.6. $\mathbb{Z}_n$ es un campo (bajo la suma y el producto módulo n), si y solo si n es un número primo. Si n es primo, entonces $\mathbb{Z}_n$ tiene característica n.

Si la característica de un campo $car(\mathbb{F}) = 0$, entonces $\mathbb{F}$ es **infinito**. Un campo es **finito** si la característica $car(\mathbb{F}) = p$, con $p \neq 0$, por lo cual tiene un número finito de elementos. En general, un campo finito contiene $q = p^m$ elementos, para $m \geq 1$, y se denota como $\mathbb{F}_q$.

Definición 2.1.11. ***(Extensión de campo)*** *Sea E un campo, un subconjunto F del campo E es un subcampo de E, si el conjunto F por si solo también forma un campo con respecto a las operaciones binarias de E. En este caso, E se dice ser una **extensión de campo** de F.*

Ahora que ya se definieron los conceptos básicos de un campo, se define lo que son los polinomios sobre n variables $x_1, \ldots, x_n$ con coeficientes en un campo $\mathbb{F}$. Se empieza por definir los monomios.

Definición 2.1.12. ***(Monomio)*** *Un **monomio** sobre $x_1, \ldots, x_n$, es un producto de la forma*

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}$$

*donde todos los exponentes $\alpha_1, \ldots, \alpha_n$ son enteros no negativos. El **grado absoluto** de un monomio es la suma $\alpha_1 + \ldots + \alpha_n$.*

La notación de los monomios se puede simplificar de la siguiente manera: sea $\alpha = (\alpha_1, \ldots, \alpha_n)$ una n-tupla de enteros no negativos. Entonces tenemos

$$x^\alpha = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}$$

Cuando $\alpha = (0, \ldots, 0), x^\alpha = 1$. De igual manera $|\alpha| = \alpha_1 + \cdots + \alpha_n$ denota el grado absoluto del monomio x^α.

Definición 2.1.13. ***(Polinomio)*** *Un **polinomio** f consiste de la combinación lineal de n monomios. Un polinomio f puede escribirse como*

$$f = \sum_n a_\alpha x^\alpha$$

donde la suma es sobre una cantidad finita de n-tuplas $\alpha = (\alpha_1, \ldots, \alpha_n)$.

Si R es un anillo conmutativo, un polinomio en la variable x sobre el anillo R, es una expresión de

la forma

$$f(x) = a_n x^n + \cdots a_2 x^2 + a_1 x + a_0$$

donde $a_i \in R$ y $n \geq 1$. El elemento a_i es el coeficiente de x^i en $f(x)$. El entero m más grande para el cual $a_m \neq 0$ es el *grado del polinomio* $f(x)$, se denota como $\operatorname{gr} f(x)$; a a_m se le conoce como el *coeficiente principal* de $f(x)$. Si todos los coeficiente de $f(x)$ son ceros, a excepción de la parte constante, es decir, $f(x) = a_0, a_0 \neq 0$, $f(x)$ es un polinomio de grado 0 y se le llama *polinomio constante*. Un *polinomio cero*, es aquel en el que todos los coeficientes tienen un valor de 0, el grado de este polinomio está definido como $-\infty$. Un polinomio $f(x)$ se dice ser *mónico*, si el coeficiente principal es 1.

Definición 2.1.14. ***(Anillo de polinomios)*** *Si R es un anillo conmutativo, el **anillo de polinomios** $R[x]$ es el anillo que se forma a partir del conjunto de polinomios sobre la variable x con coeficientes en R. Estos polinomios son de la forma*

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

donde los coeficientes a_i son elementos en R.

Ejemplo 2.1.7. Sean $f(x) = x^3 + x + 1$ y $g(x) = x^2 + x$ elementos del anillo de polinomios $\mathbb{Z}_2[x]$. En $\mathbb{Z}_2$ se llevan a cabo las operaciones,

$$f(x) + g(x) = x^3 + x^2 + 1$$

y

$$f(x)g(x) = x^5 + x^4 + x^3 + x$$

Definición 2.1.15. ***(División de polinomios)*** *Sean $g(x), h(x) \in \mathbb{F}[x]$, con $h(x) \neq 0$. En el proceso de dividir $g(x)$ por $h(x)$ se generan dos nuevos polinomios $q(x), r(x) \in \mathbb{F}[x]$, tales que*

$$g(x) = q(x)h(x) + r(x), \;\; \operatorname{gr} r(x) < \operatorname{gr} h(x)$$

*$q(x)$ se denomina el **cociente**, mientras que $r(x)$ es el **residuo**. El residuo de la división en ocasiones se denota como $g(x) \mod h(x)$, y el cociente como $g(x) \operatorname{div} h(x)$.*

Ejemplo 2.1.8. Sean $g(x) = x^6 + x^5 + x^3 + x^2 + x + 1$ y $h(x) = x^4 + x^3 + 1$, ambos en $\mathbb{Z}_2[x]$.

La división de $g(x)$ por $h(x)$, tiene el siguiente resultado

$$g(x) = x^2 h(x) + (x^3 + x + 1)$$

por lo tanto, $g(x) \mod h(x) = x^3 + x + 1$ y $g(x)$ div $h(x) = x^2$.

Sea $f(x), g(x) \in \mathbb{F}[x]$, se dice que $g(x)$ divide a $f(x)$ si existe un polinomio $h(x) \in \mathbb{F}[x]$, tal que $f(x) = g(x)h(x)$ y se denota como $g(x) \mid f(x)$. En caso de que no exista algún $h(x)$ que cumpla la condición anterior, $g(x)$ no es un divisor de $f(x)$ y se denota como $g(x) \nmid f(x)$.

Definición 2.1.16. ***(Polinomio irreducible)*** *Sea $f(x) \in \mathbb{F}[x]$ un polinomio de grado a lo menos 1. $f(x)$ se dice ser un **polinomio irreducible** sobre $\mathbb{F}$, si no puede ser escrito como el producto de dos polinomios de grados positivos en $\mathbb{F}[x]$, tal que $f(x) = g(x)h(x)$ con $g(x), h(x) \in \mathbb{F}[x]$.*

Definición 2.1.17. ***(Polinomios cuadráticos sobre varias variables)*** *Consideremos el anillo R y las variables $\boldsymbol{x} = \{x_1, \dots x_n\}$, un polinomio de grado dos sobre estas variables con coeficientes en R se puede escribir como*

$$\sum_{i=0}^{n} \sum_{j=0}^{n} a_{i,j} x_i x_j + \sum_{i=0}^{n} b_i x_i + c$$

Definición 2.1.18. ***(Ideal)*** *Sea R un anillo. Un **ideal** de R es un subgrupo aditivo I de R, tal que $ar \in I$ para todo $a \in I$ y $r \in R$. I es cerrado bajo el producto por los elementos en R.*

Teorema 2.1.1. *Sea R un anillo, y $a \in R$. Entonces $aR = \{ar : r \in R\}$ es un ideal de R.*

Demostración. Para todo $ar, ar' \in aR$ y $r'' \in R$, se tiene que $ar + ar' = a(r + r') \in aR$ y $(ar)r'' = a(rr'') \in aR$. □

El ideal del teorema anterior se hace llamar *ideal de R generado por a*. Un ideal de esta forma es llamado *ideal principal*. Ya que R es conmutativo, este ideal también puede ser escrito como $Ra = \{ra : r \in R\}$. Este ideal, es el ideal más pequeño que contiene al elemento a.

Definición 2.1.19. ***(Anillo cociente)*** *Sea $\mathbb{F}[X]$ un anillo de polinomios y $f(x) \in \mathbb{F}[X]$ el anillo cociente $\mathbb{F}[X]/f(x)$ se define como el conjunto que contiene a la clase de residuos módulo $f(x)$.*

Definición 2.1.20. ***(Espacio afín)*** *Dado un campo $\mathbb{F}$ y un entero positivo n, el **espacio afín** de tamaño n sobre $\mathbb{F}$ se dice ser el conjunto*

$$\mathbb{F}^n = \{(a_1, \dots, a_n) : a_1, \dots a_n \in \mathbb{F}\}$$

Ahora veremos como es que un espacio afín se relaciona con un polinomio. La idea principal de un polinomio $f = \sum_\alpha a_\alpha x^\alpha \in \mathbb{F}[x_1, \ldots, x_n]$ es que regrese una función

$$f : \mathbb{F}^n \to \mathbb{F}$$

de la siguiente manera: dado el espacio afín $(a_1, \ldots, a_n) \in \mathbb{F}^n$, hay que reemplazar cada valor de x_i por a_i en un polinomio. Como todos los coeficientes también están en $\mathbb{F}$, esta operación da como resultado un elemento $f(x_1, \ldots, x_n) \in \mathbb{F}$.

Definición 2.1.21. ***(Variedad afín)*** *Sea $\mathbb{F}$ un campo, y $f_1, \ldots, f_n$ polinomios en $\mathbb{F}[x_1, \ldots, x_n]$. Entonces se puede obtener*

$$\boldsymbol{V}(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \in \mathbb{F}^n : f_i(a_1, \ldots, a_n) = 0, \text{ para todo } 1 \leq i \leq s\}$$

*Llamamos a $\boldsymbol{V}(f_1, \ldots, f_s)$ la **variedad afín** definida por $f_1, \ldots, f_s$.*

Así que, una variedad afín $\mathbf{V}(f_1, \ldots, f_s) \subset \mathbb{F}^n$ es el conjunto de soluciones del sistema de ecuaciones $f_1(x_1, \ldots, x_n) = \cdots = f_s(x_1, \ldots, x_n)$.
El ideal generado por los polinomios $f_1, \ldots, f_s$ se denota como $I = \langle f_1, \ldots, f_s \rangle$:

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=0}^{s} h_i f_i : h_i \in \mathbb{F}[x_1, \ldots, x_n], i = 1, \ldots, s \right\}$$

I es un ideal, tal que, si $f, g \in I$, también $f+g \in I$. De igual manera, sea $f \in I$ y $h \in \mathbb{F}[x_1, \ldots, x_n]$, se cumple que $hf \in I$.
Las soluciones en $\mathbf{V}(I)$, son también soluciones en $\mathbf{V}(f_1, \ldots, f_s)$. Entonces, si se tiene dos conjuntos de polinomios $f_1, \ldots, f_s$ y $g_1, \ldots, g_s$, ambos en $\mathbb{F}[x_1, \ldots, x_n]$ y comparten el mismo ideal $I = \langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_s \rangle$, se tiene que $\mathbf{V}(I) = \mathbf{V}(f_1, \ldots, f_s) = \mathbf{V}(g_1, \ldots, g_s)$. De esta manera se tiene que

$$f_1(a_1, \ldots, a_n) = 0, \ldots, f_s(a_1, \ldots, a_n) = 0$$

$$y$$

$$g_1(a_1, \ldots, a_n) = 0, \ldots, g_s(a_1, \ldots, a_n) = 0$$

comparten las mismas soluciones, y se puede concluir que las variedades de un sistema de ecua-

ciones están determinadas por un ideal y no por el conjunto de ecuaciones en sí.

2.2. Bases de Gröbner

Las bases de Gröbner, introducidas por Bruno Buchberger [5] en 1965, son una herramienta muy útil que son de ayuda para proporcionar soluciones a una gran variedad de problemas en álgebra conmutativa y geometría algebraica. Uno de estos problemas que las bases de Gröbner puede resolver, es el de encontrar puntos en variedades algebraicas en sistemas de polinomios cuadráticos. Buchberger nombro así a las bases de Gröbner en honor a su asesor de su tesis doctoral Wolfgang Gröbner quien sugirió a Buchberger el uso del polinomio S.

Cuando las bases de Gröbner son utilizadas para encontrar soluciones a sistemas de ecuaciones lineales, es similar a cuando se utiliza eliminación de Gauss-Jordan. En la eliminación gaussiana el orden en que las variables son eliminadas no es tan importante, por lo regular depende del orden que se tiene en la matriz de coeficientes. Por otro lado, en el cálculo de las bases de Gröbner el orden de las variables y términos es muy importante. Existen algunas instancias en los que el algoritmo puede tener éxito en encontrar las bases utilizando algún orden, pero puede fracasar utilizando un orden diferente debido a limites de tiempo o memoria.

Las bases de Gröbner son un mejor conjunto generador del ideal $I = \langle f_1, \ldots, f_s \rangle$, denotado como G, que tiene algunas propiedades que los hacen especial, una de ellas, que es posible encontrar soluciones en sistemas de ecuaciones sobre varias variables.

El orden de los términos en las bases de Gröbner es un ingrediente esencial. Por ejemplo, en la división sobre $\mathbb{F}[x]$ con una sola variable y los polinomios $f(x) = x^5 - 3x + 1$ y $g(x) = x^2 - 4x + 7$, se tendría que:

- Escribir los términos de los polinomios en orden decreciente, según el grado de x.
- Como primer paso, se tiene que el *término principal* (el término con el grado mas grande) es $x^5 = x^3 \cdot x^2 = x^3 \cdot$*(el término principal en g)*. Entonces, se resta $x^3 \cdot g(x)$ de $f(x)$ para cancelar el término principal, por lo tanto ahora $f(x) = 4x^4 - 7x^3 - 3x^2 + 1$.
- Se sigue repitiendo el mismo proceso hasta obtener un polinomio de grado menor a dos.

Trabajando sobre una variable x, el orden que se sigue es según el grado de x sobre los monomios:

$$\cdots x^{m+1} > x^m > \cdots > x^2 > x > 1.$$

Un monomio $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ puede ser construido a partir de la n-tupla de exponentes $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_{\geq 0}^n$, estableciendo una relación uno-a-uno entre los monomios en $\mathbb{F}[x_1, \ldots, x_n]$ y $\mathbb{Z}_{\geq 0}^n$. Cualquier orden $>$ establecido sobre $\mathbb{Z}_{\geq 0}^n$ genera un orden sobre monomios: si $\alpha > \beta$, de acuerdo a este orden $x^\alpha > x^\beta$.

Un polinomio esta construido a partir de una suma de monomios, para poder ordenar cada uno de los términos en un polinomio es necesario poder realizar comparaciones entre cada uno de los monomios para establecer su posición. Para esto, es fundamental que el orden sea un *orden total*. Es decir, para todo par de monomios x^α y x^β, exactamente una de las siguientes tres relaciones deberá de cumplirse:

$$x^\alpha > x^\beta, \qquad x^\alpha = x^\beta, \qquad x^\alpha < x^\beta.$$

Es necesario que el orden que se utilice tenga la siguiente propiedad: sea x^γ cualquier monomio y $x^\alpha > x^\beta$, entonces deberá cumplirse que $x^\alpha x^\gamma > x^\beta x^\gamma$. Es decir, en términos de las n-tuplas de exponentes, $\alpha > \beta$ en $\mathbb{Z}_{\geq 0}^n$ implica que, para todo $\gamma \in \mathbb{Z}_{\geq 0}^n, \alpha + \gamma > \alpha + \beta$.

Definición 2.2.1. ***(Orden de términos)*** *Un orden de términos en $\mathbb{F}[x_1, \ldots, x_n]$ es una relación $>$ sobre el conjunto de términos $x^\alpha, \alpha \in \mathbb{Z}_{\geq 0}^n$, que satisface las siguientes propiedades:*

- *$>$ es un orden total sobre $\mathbb{Z}_{\geq 0}^n$.*
- *Si $\alpha > \beta$ y $\gamma \in \mathbb{Z}_{\geq 0}^n$, entonces $\alpha + \gamma > \beta + \gamma$.*
- *$>$ es un conjunto bien ordenado (*well-ordering*) sobre $\mathbb{Z}_{\geq 0}^n$. Es decir, todo subconjunto no vacío $\mathbb{Z}_{\geq 0}^n$ tiene un elemento menor bajo $>$.*

Vamos a considerar las siguientes tres ordenes sobre los términos

Definición 2.2.2. ***(Orden lexicográfico)*** *Sea $\alpha = (\alpha_1, \ldots, \alpha_n), \beta = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}_{\geq 0}^n$. Decimos que $\alpha >_{lex} \beta$ si, en la diferencia de vectores $\alpha - \beta \in \mathbb{Z}_{\geq 0}^n$, la primera entrada a la extrema del lado izquierdo diferente de cero, es positiva. Entonces podemos escribir $x^\alpha >_{lex} x^\beta$ si $\alpha >_{lex} \beta$.*

Ejemplo 2.2.1. Algunos ejemplos del orden lexicográfico (lex):

- $(1, 2, 2) >_{lex} (0, 4, 3)$, ya que $\alpha - \beta = (1, -2, -1)$.
- $(1, 2, 1) >_{lex} (1, 1, 3)$, ya que $\alpha - \beta = (0, 1, -2)$.
- Las variables $x_1, \ldots, x_n$ ordenadas de manera usual, utilizando el orden lexicográfico:

$$(1, 0, \ldots, 0) >_{lex} (0, 1, 0, \ldots, 0) >_{lex} \cdots >_{lex} (0, \ldots, 0, 1)$$

así que $x_1, >_{lex} x_2 >_{lex} \cdots >_{lex} x_n$.

Definición 2.2.3. ***(Orden lexicográfico graduado)*** *En el orden lexicográfico graduado (**graded lex order**) se dice que $\alpha >_{grlex} \beta$, para $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$, si*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i, \quad o \quad |\alpha| = |\beta| \text{ y } \alpha >_{lex} \beta$$

Ejemplo 2.2.2. El orden lexicográfico graduado ($grlex$) ordena en base al grado total del término, cuando el grado es igual, rompe esa igualdad utilizando lex, algunos ejemplos:

- $(1,4,2) >_{grlex} (1,2,2)$, ya que $|(1,4,2)| = 7 > |(1,2,2)| = 5$.
- $(3,0,3) >_{grlex} (2,1,3)$, ya que $|(3,0,3)| = |(2,1,3)|$ y $(3,0,3) >_{lex} (2,1,3)$.
- Las variables $x_1, \ldots, x_n$ están ordenadas en base al orden lexicográfico, tal que $x_1 >_{grlex} \cdots >_{grlex} x_n$

Definición 2.2.4. ***(Orden lexicográfico graduado inverso)*** *En el orden lexicográfico graduado inverso (**graded reverse lex order**) se dice que $\alpha >_{grevlex} \beta$ si,*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i, \quad o \quad |\alpha| = |\beta| \text{ y la entrada extrema derecha}$$

diferente de cero $\alpha - \beta \in \mathbb{Z}^n_{\geq 0}$ es negativa.

Ejemplo 2.2.3. El orden lexicográfico graduado inverso ($grevlex$), al igual que $grlex$ ordena en base al grado del término solo que cuando el grado es igual, rompe la igualdad de una manera diferente a $grlex$, algunos ejemplos:

- $(4,7,1) >_{grevlex} (4,2,3)$, ya que $|(4,7,1)| = 12 > |(4,2,3)| = 7$.
- $(1,5,2) >_{grevlex} (4,1,3)$, ya que $|(1,5,2)| = |(4,1,3)|$ y $(1,5,2) - (4,1,3) = (-3, 4-1)$.
- $grevlex$ da el mismo orden sobre las variables $x_1, \ldots, x_n$ que lex, tal que

$$(1,0,\ldots,0) >_{grevlex} (0,1,0,\ldots,0) >_{grevlex} \cdots >_{grevlex} (0,\ldots,0,1)$$

o

$$x_1 >_{grevlex} x_2 >_{grevlex} \cdots >_{grevlex} x_n$$

Definición 2.2.5. *Sea f un polinomio en $\mathbb{F}[x_1, \ldots, x_n]$ diferente al polinomio cero, definimos un orden de términos $>$*

1. *El **grado múltiple** de f es*

$$grmul(f) = \max(\alpha \in \mathbb{Z}^n_{\geq 0} : a_\alpha \neq 0)$$

   (el máximo es tomado con respecto a $>$)

2. *El **coeficiente principal** de f es*

$$LC(f) = a_{grmul(f)} \in \mathbb{F}$$

3. *El **monomio principal** de f es*

$$LM(f) = x^{grmul(f)}$$

4. *El **término principal** de f es*

$$LT(f) = LC(f) \cdot LM(f)$$

Ejemplo 2.2.4. Sea $f = 4xy^2z + 4z^2 - 5x^3 + 7x^2z^2$ y orden lexicográfico $>$. Entonces

$$\begin{aligned} grmul(f) &= (3, 0, 0), \\ LC(f) &= -5, \\ LM(f) &= x^3, \\ LT(f) &= -5x^3 \end{aligned}$$

Algoritmo de división

En un algoritmo de división en $\mathbb{F}[x_1, \ldots, x_n]$ se realiza la división de $f \in \mathbb{F}[x_1, \ldots, x_n]$ por $f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n]$. Con esto se busca eliminar los términos principales, con el objetivo de que los nuevos términos sean menores a los términos cancelados. Este proceso se repite hasta que ya no sea posible ninguna reducción adicional.

Definición 2.2.6. *En el caso especial de división de f entre g, donde $f, g, h \in \mathbb{F}[x_1, \ldots, x_n]$ y $g \neq 0$.*

Se dice que f se reduce a h módulo g en un paso y se denota como:

$$f \xrightarrow{g} h,$$

si y sólo si $LM(g)$ divide a un término diferente X de cero en f y

$$h = f - \frac{X}{LT(g)}g$$

Con esto, se busca eliminar por completo el término X en f, y que este sea sustituido estrictamente por términos menores que X. Este proceso puede continuar hasta eliminar de f todos los términos que sean divisibles por $LM(g)$.

Definición 2.2.7. *Sea $f, h \in \mathbb{F}[x_1, \ldots, x_n]$ y $F = \{f_1, \ldots, f_s : f_i \in \mathbb{F}[x_1, \ldots, x_n]$ y $f_i \neq 0, 1 \leq i \leq s\}$. Entonces, se dice que f se reduce a h módulo F y se denota como:*

$$f \xrightarrow{F}_{+} h,$$

si y sólo si existe una sucesión de índices $i_1, \ldots, i_t \in \{1, \ldots, s\}$ y una sucesión de polinomios $h_1, \ldots, h_{t-1} \in \mathbb{F}[x_1, \ldots, x_n]$, tal que

$$f \xrightarrow{f_{i_1}} h_1 \xrightarrow{f_{i_2}} h_2 \xrightarrow{f_{i_3}} \cdots \xrightarrow{f_{i_{t-1}}} h_{t-1} \xrightarrow{f_{i_t}} h$$

Definición 2.2.8. *Un polinomio r se dice ser reducido con respecto a un conjunto $F = \{f_1, \ldots, f_s :$ $f_i \in \mathbb{F}[x_1, \ldots, x_n]$ y $f_i \neq 0\}$ si $r = 0$ o no existe algún término de r que pueda ser dividido por los términos principales $LT(f_i), i = 1, \ldots, s$. Es decir, no es posible que r pueda ser reducido módulo F.*

Definición 2.2.9. *Llamamos a r un residuo con respecto a F si r esta reducido con respecto a F, $f \xrightarrow{F}_{+} r$.*

La reducción permite definir un algoritmo de división sobre $\mathbb{F}[x_1, \ldots, x_n]$. Dados los polinomios $f, f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n]$, con $f_i \neq 0$ para $i = 1, \ldots, s$, este algoritmo regresa los cocientes $a_1, \ldots, a_s$ y el residuo r, ambos en $\mathbb{F}[x_1, \ldots, x_n]$, tales que

$$f = a_1 f_1 + \cdots + a_s f_s + r$$

En el Algoritmo 1 se presenta el proceso de división para polinomios sobre varias variables.

Algoritmo 1 Algoritmo de división para polinomios sobre varias variables

Entrada: $f, f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n]$ con $f_i \neq 0$ para $i = 1, \ldots, s$

Salida: $r, a_1, \ldots, a_s \in \mathbb{F}[x_1, \ldots, x_n]$, tales que $f = a_1 f_1 + \cdots + a_s f_s + r$ y que r este reducido con respecto a $f_1, \ldots, f_s$ y que $\max(LM(a_1)LM(f_1), \ldots, LM(a_s)LM(f_s), LM(r)) = LM(f)$

$u_1 \leftarrow 0, \ldots, u_s \leftarrow 0$
$r \leftarrow 0$
$h \leftarrow f$
while $h \neq 0$ **do**
    if Existe un i, tal que $LM(f_i)$ divide $LM(h)$ **then**
        Escoger el i menor, tal que, $LM(f_i)$ divida $LM(h)$
        $u_i \leftarrow u_i + \frac{LT(h)}{LT(f_i)}$
        $h \leftarrow h - \frac{LT(h)}{LT(f_i)} f_i$
    else
        $r \leftarrow r + LT(h)$
        $h \leftarrow h - LT(h)$
    end if
end while
return $r, u_1, \ldots, u_s$

Algoritmo de Buchberger

Antes de mostrar como es que se calculan las bases de Gröbner para un ideal $I = \langle f_1, \ldots, f_s \rangle$, primero es necesario definir qué es una base de Gröbner así como también los polinomios S, los cuales son utilizados en el algoritmo de Buchberger.

Definición 2.2.10. *A un conjunto de polinomios $G = \{g_1, \ldots, g_t\}$ diferentes de cero contenidos en un ideal I, se les conoce como la base de Gröbner de I, si y sólo si, para toda $f \in I$, tal que, $f \neq 0$ existe un índice $i \in \{1, \ldots, t\}$, tal que $LM(g_i)$ divide $LM(f)$.*

Esto quiere decir que, si G es una base de Gröbner de I, entonces no existen polinomios diferentes de cero en I reducidos respecto a G.

Teorema 2.2.1. *Sea I un ideal en $\mathbb{F}[x_1, \ldots, x_n]$ diferente de cero. Las siguientes afirmaciones son equivalentes para un conjunto de polinomios $G = \{g_1, \ldots, g_s\} \subseteq I$ diferentes de cero:*

1. *G es una base de Gröbner para I*
2. *$f \in I$ si y sólo si $f \xrightarrow{G}_+ 0$*
3. *$f \in I$ si y sólo si $f = \sum_{i=1}^{t} h_i g_i$ con $LM(f) = \max_{1 \leq i \leq t}(LM(h_i)LM(g_i))$*

4. $LT(G) = LT(I)$

Corolario 2.2.1. *Si $G = \{g_1, \ldots, g_t\}$ es una base de Gröbner para un ideal I, entonces $I = \langle g_1, \ldots, g_t \rangle$.*

Lemma 2.2.1. *Sea I un ideal generado por un conjunto de polinomios diferentes de cero S y $f \in \mathbb{F}[x_1, \ldots, x_n]$. Entonces, $f \in I$ si y sólo si para cada término X que aparece en f existe un $Y \in S$, tal que, ese Y divide a X. Si para cada término X que aparece en f existe un $Y \in S$, tal que, Y divide X, entonces tales X's estan en $I = \langle S \rangle$ y por lo tanto f esta en I.*

Corolario 2.2.2. *Todo ideal I diferente de cero en $\mathbb{F}[x_1, \ldots, x_n]$ tiene una base de Gröbner.*

Definición 2.2.11. *Un subconjunto $G = \{g_1, \ldots, g_t\}$ en $\mathbb{F}[x_1, \ldots, x_n]$ es una base de Gröbner, si y sólo si, es una base de Gröbner para el ideal $I = \langle G \rangle$ que genera.*

Teorema 2.2.2. *Sea un conjunto de polinomios $G = \{g_1, \ldots, g_t\}$ diferentes de cero en $\mathbb{F}[x_1, \ldots, x_n]$. G es una base de Gröbner si y sólo si para toda $f \in \mathbb{F}[x_1, \ldots, x_n]$ el residuo de la división de f por G es único.*

Definición 2.2.12. *Sea $f, g \in \mathbb{F}[x_1, \ldots, x_n]$ con $f, g \neq 0$. Sea $L = mcm(LM(f), LM(g))$. Al polinomio*

$$S(f, g) = \frac{L}{LT(f)} f - \frac{L}{LT(g)} g$$

se le conoce como polinomio-S de f y g.

Teorema 2.2.3. *Sea un conjunto de polinomios $G = \{g_1, \ldots, g_t\}$ diferentes de cero en $\mathbb{F}[x_1, \ldots, x_n]$. G es una base de Gröbner para el ideal $I = \langle g_1, \ldots, g_t \rangle$ si y sólo si para toda $S(g_i, g_j) \xrightarrow{G}_+ 0$ para toda $i \neq j$.*

Corolario 2.2.3. *Sea $G = \{g_1, \ldots, g_t\}$ con $g_i \neq 0$, para $1 \leq i \leq t$. Entonces G es una base de Gröbner si y sólo si para toda $i \neq j$, tal que, $i \neq 1$ y $j \neq 1$, se tiene*

$$S(g_i, g_j) = \sum_{\nu=1}^{t} h_{ij\nu} g_\nu, \quad \text{donde } LM(S(g_i, g_j)) = \max_{1 \leq \nu \leq t} (LM(h_{ij\nu}), LM(g_\nu))$$

Bases de Gröbner reducidas

El orden de términos utilizado para un conjunto de polinomios $F = \{f_1, \ldots, f_s\} \in \mathbb{F}[x_1, \ldots, x_n]$ utilizando el Algoritmo 2 puede ocasionar diferentes bases de Gröbner para el mismo conjunto de polinomios. Entonces se tiene la siguiente definición:

Algoritmo 2 Algoritmo de Buchberger para calcular bases de Gröbner

Entrada: $F = \{f_1, \ldots, f_s\} \subseteq \mathbb{F}[x_1, \ldots, x_n]$ con $f_i \neq 0$, para $1 \leq i \leq s$

Salida: $G = \{g_1, \ldots, g_t\}$, una base de Gröbner para el ideal $I = \langle f_1, \ldots, f_s \rangle$

1: $G \leftarrow F$
2: $\mathcal{G} \leftarrow \{\{f_i, f_j\} | f_i \neq f_j \in G\}$
3: **while** $\mathcal{G} \neq 0$ **do**
4: Escoger cualquier par de funciones $\{f, g\} \in \mathcal{G}$
5: $\mathcal{G} \leftarrow \mathcal{G} - \{\{f, g\}\}$
6: $S(f, g) \xrightarrow{G}_{+} h$, donde h está reducida con respecto a G
7: **if** $h \neq 0$ **then**
8: $\mathcal{G} \leftarrow \mathcal{G} \cup \{\{u, h\} \forall u \in G\}$
9: $G \leftarrow G \cup \{h\}$
10: **end if**
11: **end while**
12: **return** G

Definición 2.2.13. *Una base de Gröbner $G = \{g_1, \ldots, g_t\}$ se conoce como mínima si para toda $i, LC(g_i) = 1$ y para toda $i \neq j, LM(g_i)$ no divide $LM(g_j)$.*

Lemma 2.2.2. *Sea $G = \{g_1, \ldots, g_t\}$ una base de Gröbner para un ideal $I = \langle f_1, \ldots, f_s \rangle$. Si $LM(g_2)$ divide $LM(g_1)$, entonces $\{g_2, \ldots, g_t\}$ también es una base de Gröbner de I.*

Con base a esto último, a partir de una base de Gröbner ya encontrada se puede generar una base de Gröbner mínima.

Corolario 2.2.4. *Sea $G = \{g_1, \ldots, g_t\}$ una base de Gröbner de el ideal $I = \langle f_1, \ldots, f_s \rangle$. Una base de Gröbner mínima se puede obtener con base a G, eliminando todos los g_i para los cuales exista un $j \neq i$, tal que $LM(g_j)$ divida $LM(g_i)$ y finalmente, dividir cada g_i restante por el $LC(g_i)$.*

Aun así, las bases de Gröbner mínmas siguen sin ser únicas. Para encontrar una base de Gröbner única se requiere encontrar:

Definición 2.2.14. *Una base de Gröbner $G = \{g_1, \ldots, g_t\}$ se conoce como base de Gröbner reducida si para todas $i, LC(g_i) = 1$ y g_i esta reducida respecto a $G - \{g_i\}$. Es decir, para toda i, ningún término diferente de cero en g_i es divisible por algún $LM(g_j)$ para cualquier $j \neq i$.*

2.3. Criptografía

Las transacciones electrónicas cada día tienen un rol mas importante en nuestras vidas [49]. Es necesario proveer la seguridad necesaria para que esta información que es transmitida por medios electrónicos pueda viajar segura, para eso es necesario proporcionar algunos servicios de seguridad necesarios [25, 45]:

- **Confidencialidad:** Garantiza que la información solo puede ser revelada por los usuarios/entidades que tengan autorización para acceder a ella, manteniendo así todos los datos en secreto.
- **Integridad de datos:** Es necesario asegurarse que los datos no han sido alterados por alguna entidad maliciosa.
- **Autenticación de origen de datos:** Corroborar que el mensaje recibido proviene del emisor correcto y no se trata de un impostor.
- **Autenticación de entidades:** Corroborar que las entidades son quienes dicen ser y no impostores.
- **No repudio:** Prevenir que una entidad pueda negar acciones realizadas en la comunicación.

En criptografía, al mensaje se le llama *texto en claro*. *Cifrar* es el codificar un mensaje de una manera que no pueda ser leído por una entidad que no cuente con la autorización. El mensaje cifrado se le conoce como *texto cifrado*. Al proceso de obtener el texto en claro a partir del texto cifrado se le conoce como *descifrado*. Por lo general, el proceso de cifrado y descifrado hacen uso de una *clave*, esta clave es utilizada para cifrar y descifrar respectivamente.
Existen dos categorias importantes en criptografía: la criptografía de clave privada (*criptografía simétrica*) y la criptografía de clave pública (*criptografía asimétrica*). Ambas categorías juegan un rol importante en aplicaciones criptográficas.

Criptografía de clave privada

Definición 2.3.1. ***(Criptografía simétrica)*** *Un criptosistema de clave privada se puede definir como la quintupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, donde:*

- $\mathcal{P}$ *representa al conjunto finito de todos los posibles textos en claro.*

- *$\mathcal{C}$ representa al conjunto finito de todos los posibles textos cifrados.*
- *$\mathcal{K}$ representa el espacio de claves. Es decir, el conjunto finito de todas las posibles claves.*
- *$\forall K \in \mathcal{K}, \exists E_K \in \mathcal{E}$ (regla de cifrado), $\exists D_K \in \mathcal{D}$ (regla de descifrado).*
- *Cada $E_K : \mathcal{P} \to \mathcal{C}$ y $D_K : \mathcal{C} \to \mathcal{P}$ son funciones bien definidas, tales que, $\forall x \in \mathcal{P}, D_K(E_K(x)) = x$.*

En un esquema de criptografía simétrica, tanto la clave para cifrar y para descifrar deben de mantenerse en secreto. Sólo las entidades que cuenten con estas claves tienen acceso a la información que será intercambiada en la comunicación entre dos o más entidades. Las claves deberán conocerse tanto por el emisor como el receptor para poder intercambiar información, la Figura 2-1 muestra un intercambio de mensajes por dos entidades.

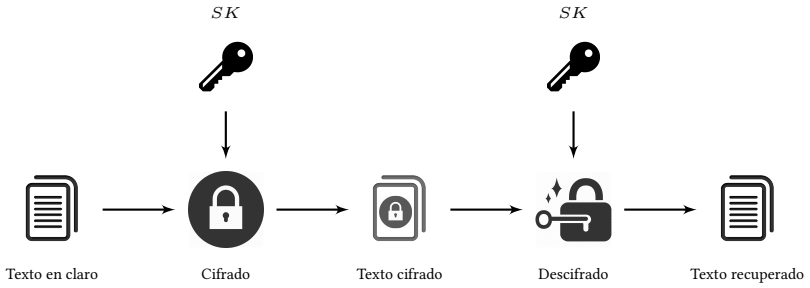

Figura 2-1: Criptografía simétrica

Criptografía de clave pública

En 1976, se logra un gran avance en la criptografía con la propuesta de Diffie y Hellman para la criptografía de clave pública [12]. Esto no solo solucionó algunos problemas, como el de intercambio de claves, sino que también se proporcionan las herramientas necesarias para poder proveer los servicios de seguridad de autenticación y no repudio de manera eficiente.
Los algoritmos en criptografía asimétrica utilizan una clave diferente para cifrar y descifrar la información. A estas claves se les conoce como clave pública y clave privada. La clave pública como su nombre lo dice, es de conocimiento público y es utilizada para cifrar la información, mientras que la clave privada se utiliza para descifrar y deberá mantenerse en secreto, de lo contrario un adversario puede interceptar la comunicación y obtener los textos en claro.

Definición 2.3.2. ***(Criptografía asimétrica)*** *Un criptosistema de clave pública se puede definir como la quintupla ($\mathcal{P}, \mathcal{C}, \mathcal{K}_{pub} \times \mathcal{K}_{priv}, \mathcal{E}, \mathcal{D}$), donde:*

- *$\mathcal{P}$ representa al conjunto finito de todos los posibles textos en claro.*
- *$\mathcal{C}$ representa al conjunto finito de todos los posibles textos cifrados.*
- *$\mathcal{K}_{pub}$ representa el espacio de claves públicas y $\mathcal{K}_{priv}$ representa el espacio de claves privadas.*
- *$\forall K_{pub} \times K_{priv} \in \mathcal{K}_{pub} \times \mathcal{K}_{priv}, \exists E_{K_{pub}} \in \mathcal{E}$ (regla de cifrado), $\exists D_{K_{priv}} \in \mathcal{D}$ (regla de descifrado).*
- *Cada $E_{K_{pub}} : \mathcal{P} \to \mathcal{C}$ y $D_{K_{priv}} : \mathcal{C} \to \mathcal{P}$ son funciones bien definidas, tales que, $\forall x \in \mathcal{P}, D_{K_{priv}}(E_{K_{pub}}(x)) = x$.*

Definimos dos entidades que quieren comunicarse, *Alicia* y *Beto*. Alicia le quiere enviar un mensaje a Beto utilizando un canal de comunicación inseguro. *Eva*, es un adversario que intenta conocer que es de lo que Alicia y Beto hablan, por lo tanto Alicia hace uso de la criptografía para asegurarse que nadie más pueda leer lo que en la comunicación se está enviando. En un esquema de criptografía asimétrica Alicia, A genera dos pares de llaves A_{pub} y A_{priv} al igual que Beto, B. Alicia quiere enviar un mensaje a Beto, por lo tanto hace uso de la clave pública de Beto B_{priv} para cifrar el mensaje m que desea enviar. Entonces, Alicia obtiene el mensaje cifrado $c = E_{B_{pub}}(m)$ y se lo envía a Beto. Beto recibe el mensaje cifrado c de Alicia y procede a realizar el descifrado utilizando su clave privada B_{priv} para así obtener el mensaje en claro $m = D_{B_{priv}}(c)$. En la Figura 2-2 se muestra un diagrama de comunicación utilizando criptografía asimétrica.

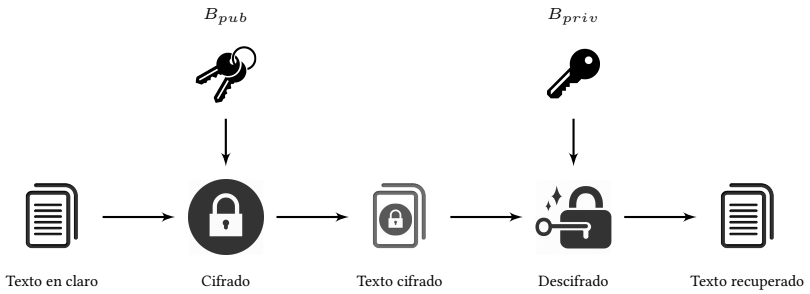

Figura 2-2: Criptografía asimétrica

En criptografía asimétrica todas las entidades conocen las claves públicas sin embargo, en los esquemas criptográficos de clave pública, el obtener la clave privada a partir de la clave pública debe ser un problema difícil, es por esto que este tipo de criptosistemas se basan en problemas ma-

temáticos difícil o funciones trampa[1]. Los esquemas criptográficos de clave pública, también son utilizados para generar firmas digitales que no pueden ser repudiadas.

2.4. Pruebas de conocimiento nulo

Los protocolos (pruebas) de conocimiento nulo (*PCN*), introducidos por Goldwasser, Micali y Rackoff [23] en el año 1985, son métodos que permiten demostrar la posesión de cierta información sin dar la necesidad de mostrar esa información. Los PCN consisten de un proceso de verificación entre dos entidades, un *verificador* lanza un reto, tal que otra entidad llamada el *probador* sea capaz de responder de tal manera que pueda convencer al verificador que el poseé la información que dice poseer.
El proceso de verificación tiene que cumplir con ciertas propiedades, tal como que el encontrar la prueba para el proceso de verificación sea una tarea computacionalmente difícil para una entidad distinta al probador.

Definición 2.4.1. *Una **prueba de conocimiento nulo interactiva** sobre un conjunto S consiste de una interacción entre el probador y el verificador, por lo que ambos deben estar presentes. El verificador V ejecuta una serie de retos que ya sea rechaza o acepta y el probador genera las respuestas a los retos.*

Una *estrategia* describe la siguiente acción a realizar por cada entidad en cualquier punto del proceso de verificación. Una prueba interactiva de conocimiento nulo consta de las propiedades de *completitud* y *consistencia*:

Definición 2.4.2. ***(Completitud)** Para todo $x \in S$, el verificador V siempre acepta la prueba después de interactuar con el autentico probador P con una entrada en común x.*

Definición 2.4.3. ***(Consistencia)** Para todo $x \notin S$ y para toda estrategia del probador P^*, el verificador V rechaza la prueba con una probabilidad de $\frac{1}{2}$ después de interactuar con P^* en una entrada en común x.*

Definición 2.4.4. ***(Conocimiento nulo)** La estrategia de un probador P^*, se dice ser de conocimiento nulo sobre un conjunto S si, por cada estrategia de un verificador en tiempo polinomial probabilístico V^*, existe un simulador en tiempo polinomial probabilístico $A*$, tal que para cada diferenciador en*

[1] Una *función trampa* consiste de una función matemática, en la que el cálculo directo es sencillo, pero calcular su función inversa es mucho más complejo, de manera que el número de pasos se eleva de manera considerable haciendo que computacionalmente sea un problema difícil de resolver.

tiempo polinomial probabilístico D, se cumple que:

$$d(n) \stackrel{def}{=} \max_{x \in S \cap \{0,1\}^n} \{|P_r[D(x,(P,V^*)(x)) = 1] - P_r[D(x,A^*(x)) = 1]|\}$$

es una función despreciable.

Definición 2.4.5. ***(Conocimiento nulo perfecto)*** *La estrategia de un probador P se dice ser de conocimiento nulo perfecto sobre un conjunto S, si por cada estrategía de un verificador en tiempo polinomial probabilístico V^*, existe un algoritmo en tiempo polinomial probabilístico A^*, tal que*

$$(P,V^*)(x) \equiv A^*(x), \quad \textit{para cada } x \in S,$$

donde $(P,V^*)(x)$ es una variable aleatoria que representa la salida del verificador V^* después de interactuar con el probador P en una entrada en común x y $A^*(x)$ es una variable aleatoria que representa la salida del algoritmo A^* en la entrada x.

Conocimiento nulo en criptografía sobre varias variables

En [24] se presenta un protocolo de autenticación de conocimiento nulo basado en el problema de isomorfismo y el problema $\mathcal{MQ}$. Sea P_1 y P_2 dos conjuntos de polinomios isomorfos, el secreto esta formado por las transformaciones afines S, T que generan un isomorfismo entre P_1 y P_2. El probador tiene en sus manos el secreto (S,T) y para convencer al verificador de que tiene esta información, se puede proceder de la siguiente manera:

1. El probador genera un conjunto de polinomios P_3, seleccionando dos transformaciones afines S', T' de manera aleatoria, tal como se muestra en el siguiente diagrama

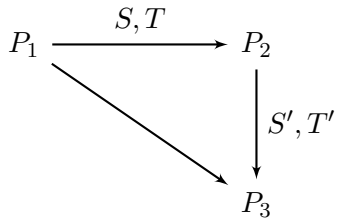

Figura 2-3: Protocolo de conocimiento nulo en criptografía sobre varias variables.

entonces $P_3 = S' \circ P_2 \circ T' = S' \circ S \circ P_1 \circ T \circ T'$.

2. El probador envía P_3 al verificador.

3. El verificador ahora puede lanzar un reto al probador, ya sea que

   - probar que P_1 y P_3 son isomorfos, o
   - probar que P_2 y P_3 son isomorfos.

   La selección entre estas dos opciones se realiza con una probabilidad de $\frac{1}{2}$.

4. Si la declaración del probador es aceptada por el verificador, entonces el probador deberá de revelar:

   - $S' \circ S$ y $T' \circ T$ si se pregunta por la primer opción, o
   - S' y T' si se pregunte por la segunda opción.

Después del proceso de verificación el secreto (S, T) sigue a sin revelarse y el probador es capaz de convencer al verificador que tiene la información sobre el isomorfismo entre P_1 y P_2. Para convencer al verificador con una alta probabilidad de que el probador es autentico, es necesario realizar varios retos, debido a que una entidad distinta al probador puede acertar a la respuesta con una probabilidad de $\frac{1}{2}$.

2.5. PolyBoRi

`PolyBoRi` (*Polynomials over Boolean Rings*) es una biblioteca para el cálculo de bases de Gröbner sobre anillos de polinomios booleanos. `PolyBoRi` está desarrollado en C++ y proporciona una interfaz en `Python` para trabajar. De igual manera la biblioteca se encuentra incorporada en el sistema de álgebra computacional `SageMath`.

Conjunto de Polinomios Booleanos

`PolyBoRi` trabaja con polinomios booleanos. Es decir, polinomios en el anillo cociente $Q = \mathbb{Z}_2[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x^n \rangle$. Por lo tanto, trata con elementos en el anillo de polinomios $P = \mathbb{Z}_2[x_1, \cdots, x_n]$ donde los coeficientes pueden tomas los valores $\{0, 1\}$ y por las restricciones impuestas

$$x_1^2 = x^1, x_2^2 = x_2^2, \ldots, x_n^2 = x^n$$

Capítulo

3

Criptografía sobre varias variables

"Sometimes it is the people no one can imagine anything of, who do the things no one can imagine."

Alan Türing

En este capítulo se describe la criptografía sobre varias variables, se presentan algunos de los esquemas que han sido propuesto y algunos que ya han sido rotos. Para el contenido de este capítulo se utilizó el libro [13].

3.1. Criptografía post-cuántica

La criptografía de clave pública con los esquemas más utilizados hoy en día se encuentra en riesgo debido a la inminente salida de un computador cuántico, en el que utilizando los algoritmos propuestos por Shor en 1997 [47] se logra resolver el problema de factorización y el problema del logaritmo discreto en tiempo polinomial. De tal manera que esta criptografía deja de ser útil al ya no ser capaz de proveer la seguridad necesaria.

Existe un gran interés en algunos esquemas de criptografía, los cuales a pesar de que se pudiera contar hoy en día con un computador cuántico siguen siendo seguros a ataques, a este tipo de criptografía se le conoce como *criptografía post-cuántica* en donde se encuentran criptosistemas [3] basados en:

- **Hash-based cryptography** - funciones picadillo,

- **Code-based cryptography** - códigos de corrección de errores,
- **Lattice-based cryptography** - retículos,
- **Multivariate-quadratic-equations cryptography** - sistemas de polinomios de grado dos.

En este proyecto se trabaja con criptografía sobre varias variables (*Multivariate Cryptography*), la seguridad de esta criptografía está basada en el problema $\mathcal{MQ}$ y el problema de isomorfismo en polinomios, ambos mencionados en el capítulo 1.

Forma bipolar estándar

Sea $\mathbb{K}$ un campo finito. En la forma bipolar sobre un criptosistema sobre varias variables, el cifrado está dado como una función $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^m$:

$$\mathcal{P}(x_1, \ldots, x_n) = (p_1, \ldots, p_m),$$

donde cada p_i es un polinomio en $\mathbb{K}[x_1, \ldots, x_n]$. Una construcción de este tipo de sistema se empieza por definir la función $\mathcal{Q} : \mathbb{K}^n \to \mathbb{K}^m$, de tal manera que

$$\mathcal{Q}(x_1, \ldots, x_n) = (q_1, \ldots, q_m), \text{ donde } q_i \in \mathbb{K}[x_1, \ldots, x_n].$$

Sean dos transformaciones afines biyectivas $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ y $\mathcal{T} : \mathbb{K}^m \to \mathbb{K}^m$. La función $\mathcal{P}$ se construye a partir de la composición

$$\mathcal{P} = \mathcal{T} \circ \mathcal{Q} \circ \mathcal{S}$$

$$\mathbb{K}^n \xrightarrow{\mathcal{S}} \mathbb{K}^n \xrightarrow{\mathcal{Q}} \mathbb{K}^m \xrightarrow{\mathcal{T}} \mathbb{K}^m$$

Figura 3-1: Composición de funciones

Para cifrar un mensaje $X' = (x'_1, \ldots, x'_n)$, se ha de calcular $Y' = \mathcal{P}(X') = (y'_1, \ldots, y'_m)$. Para descifrar un mensaje $Y' = (y'_1, \ldots, y'_m)$, se ha de resolver un sistema de ecuaciones definidos por

$$\mathcal{P}(x_1, \ldots, x_n) = Y'$$

esto se logra resolviendo primero $Y_1 = \mathcal{T}^{-1}(Y')$, después $Y_2 = \mathcal{Q}^{-1}(Y_1)$ y por último $X' = \mathcal{S}^{-1}(Y_2)$.

Para generar una firma $X' = (x'_1, \ldots, x'_n)$ a partir de un mensaje $Y' = (y'_1, \ldots, y'_m)$ se utiliza el procedimiento antes mencionado para descifrar un mensaje

$$Y_1 = \mathcal{T}^{-1}(Y')$$
$$Y_2 = \mathcal{Q}^{-1}(Y_1)$$
$$X' = \mathcal{S}^{-1}(Y_2)$$

Para validar la firma es suficiente con verificar que $\mathcal{P}(X') = Y'$.

Forma ímplicita

En este caso, el criptosistema utiliza una función cuadrática $\mathcal{P} : \mathbb{K}^{n+m} \to \mathbb{K}^l$ como su clave pública

$$\mathcal{P}(x_1, \ldots, x_n, y_1, \ldots, y_m) = (p_1, \ldots, p_l) \tag{3.1}$$

donde cada p_i es un polinomio en $\mathbb{K}[x_1, \ldots, x_n, y_1, \ldots, y_m]$. Para desarrollar un esquema de este tipo, es necesario construir una función $\mathcal{Q} : \mathbb{K}^{n+m} \to \mathbb{K}^l$

$$\mathcal{Q}(x_1, \ldots, x_n, y_1, \ldots, y_m) = (q_1, \ldots, q_l) \tag{3.2}$$

donde cada q_i es un polinomio en $\mathbb{K}[x_1, \ldots, x_n, y_1, \ldots, y_m]$, tal que

- Dado un $\boldsymbol{x} = (x_1, \ldots, x_n)$ es fácil resolver el sistema de ecuaciones

$$\mathcal{Q}(\boldsymbol{x}, \boldsymbol{y}) = (0, \ldots, 0), \text{ tal que } \boldsymbol{y} = (y_1, \ldots, y_m)$$

- Dado un $\boldsymbol{y} = (y_1, \ldots, y_m)$ es fácil resolver el sistema de ecuaciones

$$\mathcal{Q}(\boldsymbol{x}, \boldsymbol{y}) = (0, \ldots, 0), \text{ tal que } \boldsymbol{x} = (x_1, \ldots, x_n)$$

Sean $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ y $\mathcal{T} : \mathbb{K}^m \to \mathbb{K}^m$ dos transformaciones afines biyectivas y una transformación

lineal biyectiva $\mathcal{L} : \mathbb{K}^l \to \mathbb{K}^l$. $\mathcal{P}$ se construye a partir de la composición

$$\mathcal{P} = \mathcal{L} \circ \mathcal{Q} \circ (\mathcal{T} \times \mathcal{S})$$

Para cifrar un mensaje $X' = (x'_1, \ldots, x'_n)$ es necesario sustituir en 3.1 y encontrar las soluciones al sistema de ecuaciones

$$\mathcal{P}(x'_1, \ldots, x'_n, y_1, \ldots, y_m) = (0, \ldots, 0)$$

Para descifrar un mensaje cifrado $Y' = (y'_1, \ldots, y'_m)$, primero se calcula $\widetilde{Y} = \mathcal{L}^{-1}(Y')$. Ahora, dado que $\widetilde{Y} = (\widetilde{y}_1, \ldots, \widetilde{y}_m)$ se sustituye en 3.2 y se resuelve el sistema de ecuaciones

$$\mathcal{Q}(x_1, \ldots, x_n, \widetilde{y}_1, \ldots, \widetilde{y}_m) = (0, \ldots, 0)$$

siendo $\widetilde{X}$ la solución a esta ecuación, el texto en claro esta dado por $X' = \mathcal{T}^{-1}(\widetilde{X})$.

3.2. Matsumoto e Imai

El primer criptosistema sobre varias variables (*MPKC*, por sus siglas en inglés, *Multivariate Public Key Cryptosystem*) fue C^* o MI propuesto por Matsumoto e Imai [35] en 1988.

Sea $\mathbb{K}$ un campo finito de característica dos y cardinalidad q y $\mathbb{K}[X]$ un anillo de polinomios. Definamos una extensión de grado n, $\mathbb{K}_1 \approx \mathbb{K}[X]/(P(X))$ con $P(X) \in \mathbb{K}[X]$ de grado n irreducible. Sea $\phi : \mathbb{K}_1 \to \mathbb{K}^n$ una transformación lineal que identifica el campo $\mathbb{K}_1$ sobre $\mathbb{K}$ dada por

$$\phi(a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}) = (a_0, a_1, \ldots, a_{n-1})$$

Seleccionamos un θ, tal que $0 < \theta < n$ y $\mathrm{mcd}(q^\theta + 1, q^n - 1) = 1$ y definimos la función sobre $\mathbb{K}$

$$\widetilde{\mathcal{Q}}(X) = X^{1+q^\theta}$$

por las condiciones impuestas sobre θ, se asegura que $\widetilde{\mathcal{Q}}$ es una función que se puede invertir. Dado $t \in \mathbb{Z}$, tal que

$$t(1 + q^\theta) \equiv 1 \mod (q^n - 1)$$

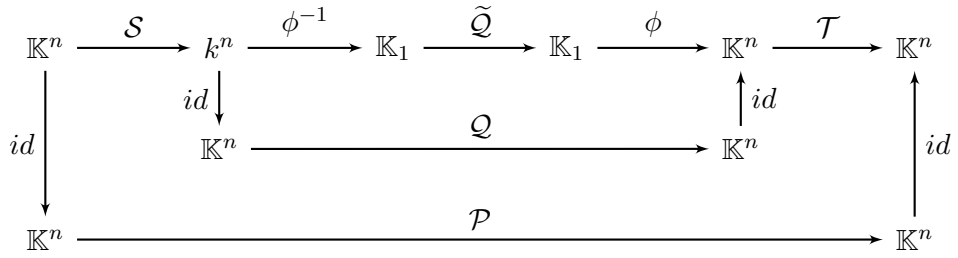

Figura 3-2: Diagrama de la construcción del criptosistema de C^*.

el inverso de $\widetilde{\mathcal{Q}}$ está definido por

$$\widetilde{\mathcal{Q}}^{-1}(X) = X^t$$

Ahora, sea $\mathcal{Q}$ la función sobre $\mathbb{K}^n$ definida por

$$\mathcal{Q}(x_1, \ldots, x_n) = \phi \circ \widetilde{\mathcal{Q}} \circ \phi^{-1}(x_1, \ldots, x_n) = (f_1, \ldots, f_n)$$

donde $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$. Para terminar la construcción del esquema de Matsumoto e Imai, habremos de seleccionar dos transformaciones afines biyectivas $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ y $\mathcal{T} : \mathbb{K}^n \to \mathbb{K}^n$. Definamos $\mathcal{P}$ como

$$\mathcal{P}(x_1, \ldots, x_n) = \mathcal{T} \circ \mathcal{Q} \circ \mathcal{S}(x_1, \ldots, x_n) = (p_1, \ldots, p_n),$$

con $p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_n]$. La figura 3-2 muestra un diagrama de la construcción de este criptosistema.

La clave pública está conformada por el campo $\mathbb{K}$ y el conjunto de polinomios $p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_n]$. Mientras que la clave privada contiene a las transformaciones afines $\mathcal{S}$ y $\mathcal{T}$.

Cifrar un mensaje es una tarea que puede ser realizada por cualquier entidad ya que la clave pública es de dominio público como su nombre lo indica. Por lo tanto, al cifrar el mensaje $X' = (x'_1, \ldots, x'_n)$ se obtiene el texto cifrado $Y' = (y'_1, \ldots, y'_n)$, donde

$$y'_i = p_i(x'_1, \ldots, x'_n).$$

Descifrar un texto cifrado es una tarea que solo puede ser realizada por aquella entidad que tenga conocimiento de la clave privada. El texto cifrado $Y' = (y'_1, \ldots, y'_n)$ puede ser descifrado al

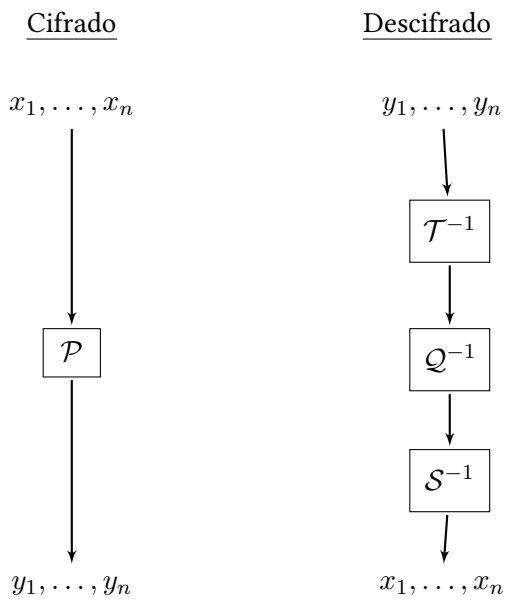

Figura 3-3: Representación gráfica del cifrado y descifrado con MI.

calcular

$$\begin{aligned}\mathcal{P}^{-1}(y'_1, \ldots, y'_n) &= \mathcal{S}^{-1} \circ \mathcal{Q}^{-1} \circ \mathcal{T}^{-1}(y'_1, \ldots, y'_n) \\ &= \mathcal{S}^{-1} \circ \phi \circ \mathcal{Q}^{-1} \circ \phi^{-1} \circ \mathcal{T}^{-1}(y'_1, \ldots, y'_n)\end{aligned}$$

La figura 3-3 muestra una representación gráfica de como funciona el cifrado y el descifrado utilizando el criptosistema C^*.

3.3. Variantes de Matsumoto-Imai

Después de que C^* fuera quebrantado mediante un ataque de linealización por Patarin [38] se propusieron algunas variantes para incrementar la seguridad de los esquemas. Dos de esas variantes son las llamadas $plus$ $(\oplus)$ y $minus$ $(-)$.

Variante minus $(-)$

La variante **Minus** fue propuesta por Shamir en 1993 [46], y la aplicación de esta variante elimina la posibilidad de ser atacado con un ataque de linealización, si el valor minus r no es muy pequeño. La variante minus consiste en eliminar r polinomios de una clave pública dada. Por ejemplo, digamos que tenemos un anillo de polinomios $\mathbb{K}[x_1, \ldots, x_n]$ y una clave pública $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^l$ que está formada por un conjunto de l polinomios $p_1, \ldots, p_l \in \mathbb{K}[x_1, \ldots, x_n]$. Una vez aplicada la variante

minus sobre el criptosistema se tendrá una función $\mathcal{P}^- : \mathbb{K}^n \to \mathbb{K}^{l-r}$ definida por

$$\mathcal{P}^-(x_1, \ldots, x_n) = (p_1, \ldots, p_{l-r})$$

La clave pública consiste de la estructura del campo $\mathbb{K}$ y el conjunto de polinomios: $(p_1, \ldots, p_{l-r}) \in \mathbb{K}[x_1, \ldots, x_n]$. La clave privada al igual que el criptosistema original consiste de las transformaciones afines $\mathcal{S}$ y $\mathcal{T}$. Cabe destacar, que la variante minus convierte a un esquema de cifrado en un esquema de firmas debido a que se pierde la inyectividad.

Sflash

Sflash es un esquema de firmas digitales y una variante de C^* propuesto por Courtois [9]. La NESSIE (por sus siglas en ingés, *New European Schemes for Signatures, Integrity and Encryption*) después de un proceso de evaluación de dos años seleccionó Sflash como un estándar para ser utilizado en tarjetas inteligentes de bajo costo. La propuesta inicial Sflash^{v1} contaba con una pequeña vulnerabilidad por la que fue quebrantado [22]. El esquema de firmas utilizaba $GF(2)$ para los elementos del campo, que fue seleccionado de manera intencional para así reducir el tamaño de las claves. Este problema fue corregido rápidamente en Sflash^{v2} seleccionando $GF(2^7)$ para los elementos del campo [1, 41]. Sflash^{v2} es llamado Flash por la NESSIE.

Sflash^{v2} tiene un tamaño de firma de 259-bits y claves públicas de 15KB. Debido a algunas preocupaciones acerca de la seguridad, los autores propusieron una nueva versión Sflash^{v3} [37], que en esencia es Sflash^{v2} con un tamaño de firma más grande. Esta nueva versión tiene un tamaño de firma de 469-bits y claves públicas de 112KB.

Sflash^{v2} utiliza los parámetros $n = 37$ y $r = 11$, por lo tanto se tiene $\mathcal{P}^- : \mathbb{K}^{37} \to \mathbb{K}^{26}$, definido como

$$\mathcal{P}^-(x_1, \ldots, x_n) = (p_1, \ldots, p_{n-r}),$$

donde $p_1, \ldots, p_{26} \in \mathbb{K}[x_1, \ldots, x_{37}]$.

Las claves públicas en Sflash consisten del campo $\mathbb{K} = GF(2^7)$ y su estructura aditiva y multiplicativa. En particular, $\mathbb{K} = GF(2)[x]/(x^7 + x + 1)$. Además los 26 polinomios $p_1, \ldots, p_{26} \in \mathbb{K}[x_1, \ldots, x_{37}]$ son parte de la clave pública. Mientras las claves privadas incluyen a las dos transformaciones afines biyectivas $\mathcal{S}, \mathcal{T}$ y la función $\mathcal{Q}$ además de un parámetro Δ de 80-bits de longitud seleccionado de manera arbitraria.

Variante plus ($\oplus$)

En el caso de la variante **Plus** se añaden s polinomios aleatorios a un esquema criptográfico sobre varias variables y se mezclan en la clave pública a través de una transformación afín biyectiva. Por ejemplo, supongamos que se tiene $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^l$, se agregan s polinomios seleccionados de manera aleatoria $g_1, \ldots, g_s \in \mathbb{K}[x_1, \ldots, x_n]$ para crear una función $\mathcal{P}^+ : \mathbb{K}^n \to \mathbb{K}^{n+s}$ definida por

$$\mathcal{P}^+ = \mathcal{L} \circ (p_1, \ldots, p_l, g_1, \ldots, g_s),$$

donde $\mathcal{L} : \mathbb{K}^{l+s} \to \mathbb{K}^{l+s}$ es una transformación afín biyectiva que mezcla los polinomios plus en el criptosistema.

La variante plus no mejora la seguridad de un criptosistema aplicada directamente, los ataques de linealización siguen siendo efectivos. El objetivo de esta variante es el de convertir una función no inyectiva $\mathcal{P}$ en una función inyectiva para así poder ser utilizada para cifrar mensajes. De esta manera al utilizar las dos variantes se puede convertir un esquema de firmas en un esquema de cifrado y así es como se logra proponer el criptosistema de clave pública Matsumoto-Imai-Plus-Minus.

Sea una función $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^n$, tal que

$$p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_n]$$

forma la clave pública de un criptosistema de Matsumoto e Imai. Se eliminan r polinomios, se añaden s polinomios de grado dos seleccionados de manera aleatoria $g_1, \ldots, g_s \in \mathbb{K}[x_1, \ldots, x_n]$ y se define la función $\mathcal{P}^{\pm} : \mathbb{K}^n \to \mathbb{K}^m$

$$\mathcal{P}^{\pm} = \mathcal{L} \circ (p_1, \ldots, p_{l-r}, g_1, \ldots, g_s) = (p_1^{\pm}, \ldots, p_m^{\pm})$$

donde $r \leq s, m = n - r + s$ y $\mathcal{L} : \mathbb{K}^m \to \mathbb{K}^m$ es una transformación afín biyectiva.

Las claves públicas en un esquema Matsumoto-Imai-Plus-Minus incluyen el campo $\mathbb{K}$ y los $m = n - r + s$ polinomios de grado dos $p_1^{\pm}, \ldots, p_m^{\pm} \in \mathbb{K}[x_1, \ldots, x_n]$. Mientras que las claves privadas consisten del conjunto de polinomios $g_1, \ldots, g_s \in \mathbb{K}[x_1, \ldots, x_n]$ y las tres transformaciones afines $\mathcal{S}, \mathcal{T}$ y $\mathcal{L}$.

3.4. Aceite y vinagre

Después de haber quebrantado C^*, Jacques Patarin empezó con un enfoque muy diferente sobre los criptosistemas de clave pública, convirtiendo el ataque de linearización sobre C^* en los esquemas de firmas digitales *Oil-Vinegar* (aceite y vinagre). Los esquemas de aceite y vinagre se dividen en tres grupos: aceite y vinagre equilibrado *(Balanced Oil-Vinegar)* [40], aceite y vinagre no equilibrado *(Unbalanced Oil-Vinegar)* [30] y arcoíris (Rainbow) [14].
Sea $\mathbb{K}$ un campo un campo de cardinalidad q. Llamaremos a las variables $x_1, \ldots, x_o$ las variables *aceite* y $x'_1, \ldots, x'_v$ las variables *vinagre*, tal que $n = o + v$.

Definición 3.4.1. *Un **polinomio aceite y vinagre** es cualquier polinomio de grado dos de la forma:*

$$p = \sum_{i=1}^{o}\sum_{j=1}^{v} \gamma_{ij} x_i x'_j + \sum_{i=1}^{v}\sum_{j=1}^{v} \lambda_{ij} x'_i x'_j + \sum_{i=1}^{o} \xi_i x_i + \sum_{j=1}^{v} \xi'_j x'_j + \delta$$

donde $p \in \mathbb{K}[x_1, \ldots, x_o, x'_1, \ldots, x'_v]$

El nombre de polinomios aceite-vinagre es por el hecho de que las variables nunca se mezclan por completo en sus términos cuadráticos.

Definición 3.4.2. *Sea $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^o$ una función cuadrática de la forma*

$$\mathcal{P}(x_1, \ldots, x_o, x'_1, \ldots, x'_v) = (p_1, \ldots, p_o)$$

*donde $p_1, \ldots, p_o \in \mathbb{K}[x_1, \ldots, x_o, x'_1, \ldots, x'_v]$ son polinomios aceite y vinagre. Entonces, $\mathcal{P}$ se hace llamar una **función aceite y vinagre**.*

Para la construcción de este criptosistema una vez teniendo una función aceite y vinagre $\mathcal{Q}$. Tal y como se hace en las construcciones anteriores es necesario ocultar la función $\mathcal{Q}$ con la ayuda de una transformación afín biyectiva $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ de la forma

$$(x_1, \ldots, x_o, x'_1, \ldots, x'_v) = \mathcal{S}(y_1, \ldots, y_n)$$

La composición genera la función cuadrática $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^o$, definida por

$$\mathcal{P} = \mathcal{Q} \circ \mathcal{S} = (p_1, \ldots, p_o), p_i \in \mathbb{K}[x_1, \ldots, x_o, x'_1, \ldots, x'_v]$$

Las claves públicas en los esquemas de aceite y vinagre incluyen el campo $\mathbb{K}$ y la función $\mathcal{P} = \mathcal{Q} \circ \mathcal{S} = (p_1, \ldots, p_o)$, tal que

$$p_1, \ldots, p_o \in \mathbb{K}[x_1, \ldots, x_n]$$

Las claves privadas están compuestas por la transformación afín $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ y la función aceite y vinagre $\mathcal{Q} = (q_1, \ldots, q_o)$, tal que

$$q_1, \ldots, q_o \in \mathbb{K}[x_1, \ldots, x_n]$$

Arcoíris

Arcoíris es una extensión de aceite y vinagre que está basado en una construcción por capas.

Sea S el conjunto $\{1, 2, 3, \ldots, n\}$, sean u enteros $v_1, \ldots, v_u$, $u \leq n$, tales que $0 < v_1 < v_2 < \cdots < v_u = n$ y definamos los conjuntos de enteros $S_l = \{1, 2, \ldots, v_l\}$ para $l = 1, 2, \ldots, u$, tales que se tiene

$$S_1 \subset S_2 \subset \cdots \subset S_u = S$$

v_i es la cardinalidad de S_i.

Para $i = 1, \ldots, u-1$, sea $o_i = v_{i+1} - v_i$ y $O_i = S_{i+1} - S_i$, tal que o_i es la cardinalidad de O_i. Sea P_l el espacio lineal de polinomios cuadráticos que contiene a polinomios de la forma

$$\sum_{i \in O_l, j \in S_l} \alpha_{i,j} x_i x_j + \sum_{i,j \in S_l} \beta_{i,j} x_i x_j + \sum_{i \in S_{l+1}} \gamma_i x_j + \eta$$

Se puede ver que estos son polinomios del tipo aceite y vinagre, tales que $x_i, i \in O_l$ son las variables aceite y $x_j, j \in S_l$ son las variables vinagre. Llamamos a $x_i, i \in O_l$ una *variable aceite de la l-ésima capa* y a $x_j, j \in S_l$ una *variable vinagre de la l-ésima capa.* A un polinomio P_l se le llama un *polinomio aceite y vinagre de la l-ésima capa.* Es claro que $P_i \subset P_j$, para todo $i < j$.

De esta manera, cada $P_l, l = 1, \ldots, u-1$ es un polinomio aceite y vinagre. Cada polinomio en P_l tiene a $x_i, i \in O_l$ como sus variables aceite y a $x_j, j \in S_l$ como sus variables vinagre. Podemos ver que las variables vinagre en la $(l+1)$-ésima capa son todas las variables en la l-ésima capa ya que

$$S_{i+1} = O_i \cup S_i$$

Ahora, definamos la función $F : \mathbb{K}^n \to \mathbb{K}^{n-v_1}$ del esquema de firmas Rainbow, es una función tal

que

$$\begin{aligned} F(x_1,\ldots,x_n) &= (\widetilde{F}_1(x_1,\ldots,x_n),\ldots,\widetilde{F}_{u-1}(x_1,\ldots,x_n)) \\ &= (F_1(x_1,\ldots,x_n),\ldots,F_{n-v_1}(x_1,\ldots,x_n)), \end{aligned}$$

cada $\widetilde{F}_i$ consiste de o_i polinomios cuadráticos en P_i seleccionados de manera arbitraria. Polinomios seleccionados de manera arbitraria se refiere a que se seleccionaron los coeficientes de manera arbitraria.

De esta manera, se puede observar que F tiene $u-1$ capas de construcción de aceite y vinagre. La primer capa consiste de o_1 polinomios $F_1,\ldots,F_{o_1}$, tales que $x_i, i \in O_1$ son las variables aceite y $x_j, j \in S_1$ son las variables vinagre. La i-ésima capa consiste de los o_i polinomios $F_{v+1},\ldots,F_{v_{i+1}}$, tales que $x_i, i \in O_i$ son las variables aceite y $x_j, j \in S_i$ son las variables vinagre. A partir de esto se puede construir un *Arcoiris* de las n variables

$[x_1,\ldots,x_{v_1}];\{x_{v_1+1},\ldots,x_{v_2}\}$

$[x_1,\ldots,x_{v_1},x_{v_1+1},\ldots,x_{v_2}];\{x_{v_2+1},\ldots,x_{v_3}\}$

$[x_1,\ldots,x_{v_1},x_{v_1+1},\ldots,x_{v_2},x_{v_2+1},\ldots,x_{v_3}];\{x_{v_3+1},\ldots,x_{v_4}\}$

$\vdots$

$[x_1,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,\ldots,x_{v_{u-1}}];\{x_{v_{u-1}},\ldots,x_n\}$

Cada fila representa una capa del arcoíris. En la l-ésima capa las variables vinagre son las que se encuentra dentro de los paréntesis cuadrado $[\ldots]$ mientras que las variables aceite se encuentran encerradas entre corchetes $\{\ldots\}$. Llamamos a F una función arcoíris con $u-1$ capas.

Sea dos transformaciones afines biyectivas $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ y $\mathcal{T} : \mathbb{K}^{n-v_1} \to \mathbb{K}^{n-v_1}$ y definimos $\mathcal{P} : \mathbb{K}^n \to \mathbb{K}^{n-v_1}$ como

$$\mathcal{P} = \mathcal{T} \circ F \circ \mathcal{S} = (p_i,\ldots,p_{n-v_1})$$

donde cada $p_i \in \mathcal{P}$ es un polinomio en $\mathbb{K}[x_1,\ldots,x_n]$.

La clave pública en un esquema de firmas arcoíris consiste del campo $\mathbb{K}$ y los $n-v_1$ polinomios en $\mathbb{K}[x_1,\ldots,x_n]$. Mientras que en la clave privada se incluyen las transformaciones afines $\mathcal{S}$, $\mathcal{T}$ y la función $F : \mathbb{K}^n \to \mathbb{K}^{n-v_1}$.

3.5. Hidden field equations

Jacques Patarin propuso en 1996 el esquema criptográfico sobre varias variables *Hidden Field Equations* (HFE) [39], el que era el esquema más resistente en ese tiempo. Su diseño está basado en seleccionar un parámetro d que determina la eficiencia del criptosistema. Sin embargo, en 1999 Kipnis y Shamir encontraron una manera de revelar la clave privada utilizando un ataque de rango (*MinRank*) [31] cuando d es suficientemente pequeño. Posteriormente, Courtois mejoró el ataque de Kipnis y Shamir proponiendo dos nuevos ataques eficientes sobre HFE [10].

Sea $\mathbb{K}$ un campo finito de cardinalidad q y $\mathbb{K}_1$ una extensión de campo de $\mathbb{K}$ de grado n. A diferencia de C^* no se requiere que la característica de $\mathbb{K}$ sea dos. Si $P(X) \in \mathbb{K}[X]$ es un polinomio irreducible de grado n entonces $\mathbb{K}_1 \cong \mathbb{K}[X]/P(X)$. Sea $\phi : \mathbb{K}_1 \to \mathbb{K}^n$ una transformación lineal entre $\mathbb{K}_1$ y $\mathbb{K}^n$, donde

$$\phi(a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}) = (a_0, a_1, a_2, \ldots, a_{n-1})$$

La construcción de HFE es muy parecido al de C^*, la diferencia más que nada se encuentra en que la función $\widetilde{\mathcal{Q}} = X^{1+q^\theta}$ es reemplazada por

$$\widetilde{\mathcal{Q}}(X) = \sum_{i=0}^{r_2-1} \sum_{j=0}^{i} \alpha_{ij} X^{q^i+q^j} + \sum_{i=0}^{r_1-1} \beta_i X^{q^i} + \gamma,$$

donde los coeficientes $\alpha_{ij}, \beta_i, \gamma \in \mathbb{K}$ son seleccionados de manera arbitraria y r_1, r_2 se seleccionan de manera que el grado de $\widetilde{\mathcal{Q}}$ sea menor a un parámetro d dado. Los polinomios de la clave pública se forman de la composición

$$\mathcal{P} = \mathcal{T} \circ \mathcal{Q} \circ \mathcal{S} = (p_1, \ldots, p_n),$$

donde $p_i \in \mathbb{K}[x_1, \ldots, x_n]$, $\mathcal{Q} = \phi \circ \widetilde{\mathcal{Q}} \circ \phi^{-1}$ y $\mathcal{T} : \mathbb{K}^n \to \mathbb{K}^n$ y $\mathcal{S} : \mathbb{K}^n \to \mathbb{K}^n$ son dos transformaciones afines biyectivas.

Las claves públicas en HFE incluyen el campo $\mathbb{K}$ y los n polinomios en $\mathcal{P} = (p_1, \ldots, p_n) \in \mathbb{K}[x_1, \ldots, x_n]$. Mientras que las claves privadas consisten de la función $\widetilde{\mathcal{Q}}$ y las dos transformaciones afines biyectivas $\mathcal{S}$ y $\mathcal{T}$.

Se han presentado algunas variantes de HFE utilizando las variantes plus y minus, tal como se hizo con C^*. Por ejemplo HFE$^-$ puede ser utilizado como un esquema de firmas, mientras que HFE$^\pm$

puede utilizarse como un esquema de cifrado. Una variante un poco más sofisticada fue la que se logró en la combinación de HFE con aceite y vinagre, llamado HFEv. Otras extensiones de HFE se lograron entre aceite y vinagre mezclado con las variantes plus y minus tal como lo es el esquema de firmas Quartz, un ejemplo de HFEv-Minus [39].

En HFEv la función $\widetilde{\mathcal{Q}}$ es reemplazada por la función $\widetilde{\mathcal{Q}} : \mathbb{K}_1 \times \mathbb{K}^v \to \mathbb{K}_1$ definida por

$$\begin{aligned}\widetilde{\mathcal{Q}}(X, x'_1, \ldots, x'_v) = &\sum_{i=0}^{r_2-1}\sum_{j=0}^{i}\alpha_{ij}X^{q^i+q^j} + \sum_{i=0}^{r_1}\beta_i X^{q^i} \\ &+ \sum_{i=0}^{r_1}\gamma_i\Omega_i(x'_1, \ldots, x', v)X^{q^i} + \Gamma(x'_1, \ldots, x'_v),\end{aligned}$$

donde $\alpha_{ij}, \beta_i, \gamma_i \in \mathbb{K}_1, \Omega_i : \mathbb{K}^v \to \mathbb{K}_1$ y $\Gamma : \mathbb{K}^v \to \mathbb{K}_1$. Las variables asociadas con X actúan como variables aceite, mientas que las variables $(x'_1, \ldots, x'_v)$ actúan como variables vinagre. Si $v = 0$ entonces HFEv se reduce a HFE.

Sea $\pi : \mathbb{K}^v \to \mathbb{K}^n, \pi(a_1, \ldots, a_v) = (a_1, \ldots, a_v, 0, \ldots, 0)$, entonces $\widetilde{\mathcal{Q}}$ se puede rescribir como una función de $\mathbb{K}_1 \times \mathbb{K}_1 \to \mathbb{K}_1$:

$$\begin{aligned}\widetilde{\mathcal{Q}}(X, V) = &\sum_{i=0}^{r_2-1}\sum_{j=0}^{i}\alpha_{ij}X^{q^i+q^j} + \sum_{i=0}^{r_1-1}\beta_i X^{q^i} \\ &+ \sum_{i=0}^{r_1}\sum_{j=0}^{n-1}\gamma_{ij}X^{q^i}V^{q^j} \\ &+ \sum_{i=0}^{n-1}\sum_{j=0}^{i}\lambda_{ij}V^{q^i+q^j} + \sum_{i=0}^{n-1}\xi_i V^{q^i} + \eta\end{aligned}$$

donde $V = \phi^{-1} \circ \pi(x'_1, \ldots, x'_v)$ y $\alpha_{ij}, \beta_i, \gamma_{ij}, \lambda_{ij}, \xi_i, \eta \in \mathbb{K}_1$.

Ahora definamos la función $\mathcal{Q} : \mathbb{K}^{n+v} \to \mathbb{K}^n$ como

$$\begin{aligned}\mathcal{Q}(x_1, \ldots, x_n, x'_1, \ldots, x'_v) &= \phi \circ \widetilde{\mathcal{Q}} \circ (\phi^{-1} \times \phi^{-1} \circ \pi)(x_1, \ldots, x_n, x'_1, \ldots, x'_v) \\ &= (q_1, \ldots, q_n),\end{aligned}$$

donde $q_1, \ldots, q_n \in \mathbb{K}[x_1, \ldots, x_n, x'_1, \ldots, x'_v]$ son todos de grado dos. Para terminar la construcción, sea dos transformaciones afines biyectivas $\mathcal{T} : \mathbb{K}^n \to \mathbb{K}^n$ y $\mathcal{S} : \mathbb{K}^{n+v} \to \mathbb{K}^{n+v}$, se construye $\mathcal{P} : \mathbb{K}^{n+v} \to \mathbb{K}^n$ a partir de la composición

$$\mathcal{P} = \mathcal{T} \circ \mathcal{Q} \circ \mathcal{S} = (p_1, \ldots, p_n),$$

tal que $p_i \in \mathbb{K}[x_1, \ldots, x_n, x'_1, \ldots, x'_v]$.

Las claves públicas de HFEv al igual que los esquemas anteriores incluye al campo $\mathbb{K}$ y los n polinomios $p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_n, x'_1, \ldots, x'_v]$. Mientras que las claves privadas consisten de la función $\widetilde{\mathcal{Q}}$ y las dos transformaciones afines $\mathcal{S}$ y $\mathcal{T}$.

3.6. Ataques

Son varios los ataques que se han empleado para quebrantar los esquemas criptográficos propuestos, ya sea utilizando técnicas para encontrar las soluciones a un sistema $\mathcal{P}$, para de esta manera poder generar firmas sin la necesidad de conocer la clave privada que generalmente se compone por $\mathcal{S}, \mathcal{Q}$ y $\mathcal{T}$. O intentando quebrantar la estructura algebraica de los sistemas propuestos, tales como los ataques de linearización, diferenciales y de rango.

Solo existen una cantidad limitada de técnicas para encontrar las soluciones a un sistema de ecuaciones sobre un campo finito, y estas pueden ser agrupadas de las siguiente manera:

- Bases de Göbner
- XL
- Zhuang-Zi

El primero siendo el mas importante de los tres, dado un sistema de polinomios

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0, \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0, \end{aligned} \tag{3.3}$$

el problema se centra en encontrar una base para el ideal generado por el conjunto $F = \{f_1, \ldots, f_m\}$. Una vez calculada esa base es posible saber que tipo de soluciones habitan ahí y conocer esas soluciones. Las bases de Gröbner se definen con más detalle en el capítulo 2.

El método XL (*eXtended Linearization*) busca convertir un sistema de ecuaciones no lineales en un sistema de ecuaciones lineales introduciendo variables adicionales que representan los términos no lineales. Los términos no lineales son derivados a partir de la generación del ideal correspondiente, a través de la multiplicación de monomios.

El método de Zhuang-Zi busca sacar ventaja del hecho de que las soluciones se encuentran dentro de un campo finito. El conjunto de ecuaciones y sus variables se llevan a un anillo de polinomios con coeficientes en una extensión de campo $\mathbb{K}[X]$, de tal manera que encontrar las soluciones en 3.3 es lo mismo que encontrar las raíces a un único polinomio sobre la variable X.

Linealización

El criptosistema C^* propuesto por Matsumoto e Imai fue quebrantado utilizando un ataque de linealización.

Definición 3.6.1. *Sea $\mathcal{G} = \{g_1, \ldots, g_m\}$ un conjunto de m polinomios en $\mathbb{K}[x_1, \ldots, x_n]$. Una **ecuación de linealización** para $\mathcal{G}$ es un polinomio en $\mathbb{K}[x_1, \ldots, x_n, y_1, \ldots, y_m]$ de la forma*

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}x_i y_j + \sum_{i=1}^{n} b_i x_i + \sum_{j=1}^{m} c_j y_j + d,$$

tal que se pueden obtener los ceros en $\mathbb{K}[x_1, \ldots, x_n]$ substituyendo en g_j por y_j, para $j = 1, \ldots, m$. De igual forma, una ecuación de linealización es cualquier ecuación en $\mathbb{K}[x_1, \ldots, x_n]$ de la forma

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}x_i g_j(x_1, \ldots, x_n) + \sum_{i=1}^{n} b_i x_i + \sum_{j=1}^{m} c_j g_j(x_1, \ldots, x_n) + d = 0$$

que se cumple para todo $(x'_1, \ldots, x'_n) \in \mathbb{K}^n$.

Bases de Gröbner

Para el cálculo de bases de Gröbner existen algunas variantes del algoritmo de Buchberger que tienden a ser más eficientes, dos de esas variantes son los algoritmos F_4 [18] y F_5 [19] de Jean-Charles Faugère. El algoritmo F_5 fue utilizado por Faugère para quebrantar el llamado reto HFE (*HFE Challenge 1*) de Patarin en 96 horas en un equipo de computo con un procesador de 833 MHz y 4GB de memoria RAM [20, 21].

Faugère F_4

La principal diferencia en comparación al algoritmo de Buchberger es que en lugar de seleccionar un único par crítico $(f_i, f_j) \in B$, Faugère propone el seleccionar un subconjunto de elementos críticos $C \subset B$. La estrategia propuesta de Faugère se basa en seleccionar de manera simultánea todos las parejas en B cuyo grado de $T(i, j)$ sea mínimo.

Definición 3.6.2. *Sea $T(x_1, \ldots, x_n)$ el conjunto de todos los términos sobre esas variables denotado como T y $\mathbb{K}[x]$ un anillo de polinomios. Un* ***par crítico*** *de dos polinomios (f_i, f_j) es un elemento en $T^2 \times \mathbb{K}[x] \times T \times \mathbb{K}[x], Par(f_i, f_j) := (\mathrm{mcm}_{ij}, t_i, f_i, t_j, f_j)$, tal que*

$$\mathrm{mcm}(Par(f_i, f_j)) = \mathrm{mcm}_{ij} = LM(t_i, f_i) = LM(t_j, f_j) = \mathrm{mcm}(LT(f_i), LM(f_j))$$

Definición 3.6.3. *Decimos que el grado de un par crítico $p_{i,j} = Par(f_i, f_j), \mathrm{gr}(p_{i,j})$, es $\mathrm{gr}(\mathrm{mcm}_{i,j})$. Se definen las dos proyecciones $Izq(p_{i,j}) := (t_i, f_i)$ y $Der(p_{i,j}) := (t_j, f_j)$. Si $(t, p) \in T \times \mathbb{K}[x]$ entonces denotamos como $mult((t, p))$ el producto evaluado en $t * p$.*

Sea

$$d = \min\{\mathrm{gr}(\mathrm{mcm}(LM(f_i), LM(f_j))) : \forall (f_i, f_j) \in B\},$$

entonces el subconjunto seleccionado es

$$\mathcal{S}el(B) = \{(f_i, f_j) \in B : \mathrm{gr}(\mathrm{mcm}(LM(f_i), LM(f_j))) = d\} \tag{3.4}$$

Definición 3.6.4. *Sea $F = \{f_1, \ldots, f_s\}$ el conjunto de polinomios en $\mathbb{K}[x_1, \ldots, x_n]$. Denotamos como $X = \{t_1, \ldots, t_n\}$ al conjunto de monomios en orden decreciente de acuerdo al orden σ. La representación matricial de un polinomio es:*

$$\boldsymbol{F} = \boldsymbol{A}\boldsymbol{X},$$

donde $\boldsymbol{F}$ y $\boldsymbol{X}$ son vectores columnas y la matriz $\boldsymbol{A}$ de $s \times m$ tiene valores en $\mathbb{K}$. Sea $\widetilde{\boldsymbol{A}}$ la matriz escalonada de $\boldsymbol{A}$, donde todas las filas de ceros han sido removidas. Entonces $\widetilde{\boldsymbol{F}}$ es el conjunto de polinomios correspondiente a

$$\widetilde{\boldsymbol{F}} = \widetilde{\boldsymbol{A}}\boldsymbol{X}$$

Definimos a

$$\widetilde{F}^+ = \{f \in \widetilde{F} \mid LM(f) \notin LM(F)\},$$

tal que $\widetilde{F}^+$ consiste de todos los polinomios en $\widetilde{F}$ cuyos monomios principales no son monomios principales en F.

El Algoritmo 3 presenta el procedimiento F_4 básico de Faugère. Además de la entrada que es un conjunto de polinomios en $\mathbb{K}[x]$, se tiene la función $\mathcal{S}el$ que se mostró en 3.4, la cual retorna un

subconjunto de parejas en B y al mismo tiempo remueve esas parejas del conjunto B.

Algoritmo 3 Algoritmo F_4 de Faugère

Entrada: Un conjunto de polinomios $F = \{f_1, \ldots, f_s\} \in \mathbb{K}[x]$

Salida: Una base de Gröbner reducida G

1: $G := F, \widetilde{F}_0^+ := F$ y $d := 0$
2: $P := \{Par(f, g) \mid f, g \in G \text{ y } f \neq g\}$
3: **while** $P \neq \emptyset$ **do**
4: $\quad d := d + 1$
5: $\quad P_d := \mathcal{S}el(P)$
6: $\quad P := P \backslash P_d$
7: $\quad L_d := Izq(P_d) \cup Der(P_d)$
8: $\quad \widetilde{F}_d^+ := Reduccion(L_d, G)$
9: $\quad$ **for** $h \in \widetilde{F}_d^+$ **do**
10: $\quad\quad P := P \cup \{Par(h, g) \mid g \in G\}$
11: $\quad\quad G := G \cup \{h\}$
12: $\quad$ **end for**
13: **end while**
14: **return** G

El Algoritmo 4 de reducción toma en cuenta que se esta reduciendo un subconjunto $\mathbb{K}[x]$ por G. Espera dos parámetros de entrada $L \subset T^n \times G$ y G y regresa un conjunto de polinomios reducidos cuyos coeficientes principales no han aparecido aún como coeficientes principales en G.

Algoritmo 4 Algoritmo de reducción en F_4

Entrada: Un subconjunto $L \subset T \times G$ y G

Salida: Un subconjunto finito en $k[x]$, posiblemente un conjunto vacío

1: $F :=$ *Preprocesamiento-Simbólico*(L, G)
2: $\widetilde{F} :=$ utilizar eliminación de Gauss-Jordan en F
3: $\widetilde{F}^+ = \left\{ f \in \widetilde{F} \mid LM(f) \notin LM(F_d) \right\}$
4: **return** $\widetilde{F}^+$

Faugère llama *Preprocesamiento-Simbólico* a la parte anterior a la eliminación de Gauss-Jordan, debido a que puede ser calculada de una manera muy eficiente en un tiempo lineal sobre el tamaño de la entrada. El Algoritmo 5 muestra el preprocesamiento-simbólico.

Algoritmo F_4 Mejorado

En una versión mejorada del algoritmo F_4 se realizan algunos cambios. En la primer versión se utilizan solo las filas de la matriz reducida, eliminando la filas que se encontraban en la matriz

Algoritmo 5 Preprocesamiento simbólico en F_4

Entrada: Un subconjunto $L \subset T \times G$ y G

Salida: Un subconjunto finito en $k[x]$.

1: $F_d := \{t * f \mid (t, f) \in L\}$
2: $D := LM(F_d)$
3: **while** $T(F_d) \neq D$ **do**
4: seleccionar un $m \in T(F_d) \backslash D$
5: $D := D \cup \{m\}$
6: **if** Si existe un $f \in G$, tal que $m = m' * LM(f)$ **then**
7: $m = m' * LM(f)$ para un $f \in G$ y algún $m' \in T$.
8: $F := F \cup \{m' * f\}$
9: **end if**
10: **end while**
11: **return** F

original. En la versión mejorada se mantienen esas filas, y se reemplazan algunos productos $m * f$ que ocurren en la matriz F por unos productos nuevos equivalentes $m' * f'$, donde $m \geq m'$. Esta es la tarea de la función $Simplificar : T \times \mathbb{K}[x] \times Lista(Subconjunto(\mathbb{K}[x])) \rightarrow T \times \mathbb{K}[x]$. El tercer parámetro en la función Simplificar es la lista de todas las matrices que ya han sido calculadas. El Algoritmo 6 muestra el funcionamiento de la mejora en F_4. De nuevo se cuenta con la función $\mathcal{S}el$ y se actualiza la parte del algoritmo de Buchberger, el cual selecciona las parejas en la función $Actualizar$.

El algoritmo de reducción es idéntico al de la primera versión, con una única diferencia de que existe un nuevo parámetro que implica un cambio en la función *Preprocesamiento-Simbólico* al ser un parámetro de entrada. El Algoritmo 7 muestra la reducción en el F_4 mejorado.

Algoritmo 6 Algoritmo F_4 mejorado de Faugère

Entrada: Un conjunto de polinomios $F = \{f_1, \ldots, f_s\} \in k[x]$

Salida: Una base de Gröbner reducida G

1: $G := \emptyset, P := \emptyset$ y $d := 0$
2: **while** $F \neq \emptyset$ **do**
3: $\quad f := primer(F)$
4: $\quad F := F \backslash \{f\}$
5: $\quad (G, P) := Actualizar(G, P, f)$
6: **end while**
7: **while** $P \neq \emptyset$ **do**
8: $\quad d := d + 1$
9: $\quad P_d := \mathcal{S}el(P)$
10: $\quad P := P \backslash P_d$
11: $\quad L_d := Izq(P_d) \cup Der(P_d)$
12: $\quad (\widetilde{F}_d^+, F_d) := Reduccion(L_d, G, (F_i)_{d=1,\ldots,(d-1)}$
13: $\quad$ **for** $h \in \widetilde{F}_d^+$ **do**
14: $\quad\quad (G, P) := Actualizar(G, P, h)$
15: $\quad$ **end for**
16: **end while**
17: **return** G

Algoritmo 7 Algoritmo de reducción en F_4 mejorado

Entrada: Un subconjunto $L \subset T \times G, G$ y $\mathcal{F}_{k=1,\ldots,(d-1)}$, donde $F_k \in k[x]$

Salida: Dos subconjuntos finitos en $k[x]$

1: $F :=$ *Preprocesamiento-Simbólico*$(L, G, \mathcal{F})$
2: $\widetilde{F} :=$ utilizar eliminación de Gauss-Jordan en F
3: $\widetilde{F}^+ = \left\{ f \in \widetilde{F} \mid LM(f) \notin LM(F_d) \right\}$
4: **return** $(\widetilde{F}^+, F)$

Algoritmo 8 Preprocesamiento simbólico en F_4 mejorado

Entrada: Un subconjunto $L \subset T \times G, G$ y $\mathcal{F}_{k=1,\ldots,(d-1)}$, donde $F_k \in k[x]$

Salida: Un subconjunto finito en $k[x]$.

1: $F_d := \{muld(Simplificar(m, f, \mathcal{F})) \mid (m, f) \in L\}$
2: $D := LM(F_d)$
3: **while** $T(F_d) \neq D$ **do**
4: seleccionar un $m \in T(F_d) \backslash D$
5: $D := D \cup \{m\}$
6: **if** Si existe un $f \in G$, tal que $m = m' * LM(f)$ **then**
7: $m = m' * LM(f)$ para un $f \in G$ y algún $m' \in T$.
8: $F := F \cup \{mult(Simplificar(m', f, \mathcal{F}\}$
9: **end if**
10: **end while**
11: **return** F

Algoritmo 9 Algoritmo Simplificar en F_4 mejorado

Entrada: Un término $t \in T$, un polinomio $f \in k[x]$ y $\mathcal{F}_{k=1,\ldots,(d-1)}$, donde $F_k \in k[x]$

Salida: Un producto no evaluado en $T \times k[x]$.

1: **for** $u \in$ lista de divisores de t **do**
2: **if** $\exists j (1 \leq j < d)$, tal que $(u * f) \in F_j$ **then**
3: $\widetilde{F}_j$ es la matriz escalonada de F_j bajo un orden $<$
4: $!\exists p \in \widetilde{F}_j^+$, tal que $LM(p) = LM(u * f)$
5: **if** $u \neq t$ **then**
6: **return** $Simplificar(\frac{t}{u}, p, \mathcal{F})$
7: **else**
8: **return** $(1, p)$
9: **end if**
10: **end if**
11: **end for**
12: **return** (t, f)

Capítulo

4

Codificación en ASN.1

"To know, is to know you know nothing. That is the meaning of true knowledge."

Socrates

La codificación de las claves públicas y privadas se lleva a cabo utilizando el estándar ASN.1 para así reducir el tamaño de las claves a la hora de almacenarlas, este capítulo introduce unas propuestas para las estructuras de los objetos como se codifican las claves.

4.1. Abstract syntax notation one

Abstract Syntax Notation One (*ASN.1*) es una notación formal utilizada para describir como es que la información debería de ser transmitida por protocolos de comunicación, ofreciendo un rico conjunto de tipos de datos que permiten estructurar la información de manera que ésta pueda ser interpretada independientemente de la máquina utilizada. ASN.1 es un estándar que define un formalismo para la especificación de tipos de datos abstractos. Fue desarrollado como parte de la capa 6 (presentación) del modelo OSI (en inglés, *Open System Interconnection*). Permite definir varios tipos de datos, desde tipos simples como enteros hasta tipos estructurados como conjuntos y secuencias, además de tipos complejos definidos en función de otros.

Un tipo (type), es una clase de datos. Los tipos de datos se clasifican según si son: simples (primitivos), compuestos (construidos a partir de otros, simples o compuestos) o definidos, con una etiqueta de nombre para identificarlos.

Tipos primitivos

Algunos de los tipos de datos simples se muestran en la Tabla 4.1.

Tipos Primitivos	Etiqueta	Uso
BOOLEAN	1	Representa un operador lógico, toma valores de verdadero o falso.
INTEGER	2	Representa un tipo de dato que toma valores enteros.
BIT STRING	3	Representa cadenas binarias de tamaño arbitrario.
OCTET STRING	4	Representa cadenas binarias cuya longitud es múltiplo de ocho.
NULL	5	Representa la ausencia de valor.
OBJECT IDENTIFIER	6	Representa los identificadores de los objetos.
REAL	9	Representa un tipo de dato que toma valores reales.
ENUMERATED	10	Representa valores de variables con por lo menos tres estados.
CHARACTER STRING	*	Representa cadenas de caracteres.

Tabla 4.1: Tipos de datos primitivos en ASN.1.

Tipos construidos

Los tipos construidos son tipos compuestos, utilizados para almacenar registros de valores. Algunos de ellos se muestran en la Tabla 4.2.

Tipos Compuestos	Etiqueta	Uso
SEQUENCE	16	Representa una estructura de datos, una lista ordenada de tipos de datos diferentes.
SEQUENCE OF	16	Representa una estructura de datos similar a la de SEQUENCE, excepto que todos los tipos han de ser iguales.
SET	17	Representa una estructura de datos, equivalente al de SEQUENCE pero la lista no esta ordenada.
SET OF	17	Representa una estructura de datos, equivalente al de SEQUENCE OF pero la lista no esta ordenada.
CHOICE	*	Representa un tipo de dato en el que hay que elegir uno de entre los tipos disponibles en una lista.

Tabla 4.2: Tipos de datos construidos en ASN.1.

Tipos definidos

Los tipos definidos se derivan de los tipos construidos pero con un nombre más descriptivo. Algunos de los más importantes se pueden observar en la Tabla 4.3.

Tipos Definidos	Uso
`IpAddress`	Almacena una dirección IP. Son 4 bytes y se define como OCTET `STRING (SIZE(4))`.
`Counter`	Representa un contador que incrementa su valor y vuelve a 0 al alcanzar su valor máximo. Se define como un tipo entero que solo puede tomar valores positivos.
`Gauge`	Representa un valor que puede incrementar o decrementar. Puede llevar asociadas acciones a tomar en caso de que se superen unos umbrales. Es un entero de 32 bits.
`TimeTicks`	Representa un tipo de dato usado para medir tiempos. Indica el número de centésimas de segundo que han transcurrido desde un determinado evento temporal. Es un entero de 32 bits.
`Opaque`	Define datos arbitrarios codificados como OCTET `STRING`.
`NetworkAddress`	Representa un CHOICE que permite seleccionar varios formatos de direcciones. Atualmente sólo `IpAddress`.

Tabla 4.3: Tipos de datos definidos en ASN.1.

Reglas de codificación

El estándar depende de un conjunto de reglas, las cuales especifican como es que los datos habrán de codificarse. Algunas de esas reglas son las siguientes:

- **BER** (Basic Encoding Rules)
- **CER** (Canonical Encoding Rules)
- **DER** (Distinguished Encoding Rules)
- **PER** (Packed Encoding Rules)
- **XER** (XML Encoding Rules)

BER

Las reglas de codificación básicas (BER) son las reglas originales de ASN.1, siendo parte del estándar X.409 [43] antes de que fuera dividido en dos partes en 1985.
La sintaxis de transferencia de BER siempre utiliza el formato de una tripleta llamada TLV [16] (en inglés, ⟨ *Tag, Length, Value* ⟩) como se muestra en la Figura 4-1. Cada campo es una serie de bytes que definen:

- **Tag** (Etiqueta/Tipo) - Los primeros dos bits indican la clase de dato
  - Universal: 00

- Propio de la aplicación: 01
- Específico del contexto: 10
- Privado: 11

Un bit indica si el tipo de dato es primitivo o construido, mientras que los últimos 5 bits indican un número de etiqueta, el cual identifica el tipo de dato. Si el número es mayor a 31 significa que el campo tipo ocupa más de un byte.

- **Length** (Longitud) - Indica cuantos bytes son necesarios para representar el valor, si el primer bit es 0 significa que un byte es necesario para representar al campo longitud. En caso de que el primer bit sea 0, los 7 bits restantes indican la longitud del campo que representa la longitud del valor.

- **Value** (Valor) - Este campo almacena el valor concreto al que hacen referencia los dos campos anteriores, ocupando una determinada longitud en bytes.

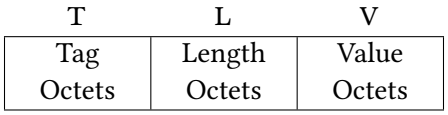

Figura 4-1: Estructura Tag-Length-Value utilizada por BER.

El campo V puede incluso almacenar una estructura TLV si el tipo de dato es construido, tal como se puede observar en la Figura 4-2.

Figura 4-2: Estructura Tag-Length-Value dentro de una TLV utilizada por BER.

4.2. PKCS

En criptografía, Public Key Cryptography Standards (PKCS) son un conjunto de especificaciones técnicas desarrolladas por RSA Laboratories con el objetivo de uniformizar las técnicas y protocolos en la criptografía. La primer publicación se hizo en el año 1991 como resultado de una reunion con un pequeño grupo que empezaba a trabajar con criptografía de clave pública.

A la fecha, existen 13 documentos con títulos genéricos que van desde PKCS #1 hasta PKCS #15 [28]. PKCS #1 es el primero de la familia de estándares PKCS, provee las definiciones y recomendaciones para implementar el algoritmo de clave pública RSA. Define las propiedades matemáticas de las claves públicas y privadas, operaciones primitivas para el cifrado y firmas digitales, esquemas criptográficos seguros y una propuesta para su representación en ASN.1.

PKCS #1: RSA Criptography Standard

En [27] se describe el criptosistema de clave pública RSA. La clave pública en RSA consiste de dos componentes:

- $n \in \mathbb{Z}^+$ - el módulo en RSA,
- $e \in \mathbb{Z}^+$ - el exponente público en RSA,

En una clave pública válida de RSA, el módulo n es el producto de los u primos $r_i, i = 1, \ldots, u$, y $u \geq 2$, y el exponente público e es un entero entre 3 y $n - 1$, tal que $mcd(e, \lambda(n)) = 1$, donde $\lambda(n) = mcm(r_1 - 1, \ldots, r_u - 1)$. A los dos primeros primos r_1 y r_2 también se les denota como p y q.
La clave privada en RSA puede tener dos representaciones, la primer representación consiste de la pareja (n, d) donde las componentes tienen el siguiente significado:

- $n \in \mathbb{Z}^+$ - el módulo en RSA,
- $d \in \mathbb{Z}^+$ - el exponente privado en RSA,

El exponente privado d en RSA, es un entero positivo menor a n, tal que:

$$e \cdot d \equiv 1 \pmod{\lambda(n)}$$

donde e es el exponente público en RSA y $\lambda(n)$ se definió anteriormente.

La segunda representación contiene información para mejorar la eficiencia utilizando el Teorema Chino del Residuo (CRT) y consiste de la quintupla $(p, q, dP, dQ, dInv)$, donde las componentes tienen los siguientes significados:

- $p \in \mathbb{Z}^+$ - el primer factor,

- $q \in \mathbb{Z}^+$ - el segundo factor,
- $dP \in \mathbb{Z}^+$ - el exponente del primer factor utilizando el CRT,
- $dQ \in \mathbb{Z}^+$ - el exponente del segundo factor utilizando el CRT,
- $qInv \in \mathbb{Z}^+$ - el coeficiente del CRT,
- $r_i \in \mathbb{Z}^+$ - el factor i-ésimo,
- $d_i \in \mathbb{Z}^+$ - el exponente del factor i-ésimo utilizando el CRT,
- $t_i \in \mathbb{Z}^+$ - el coeficiente del factor i-ésimo,

En la segunda representación, se incluye una tripleta (r_i, d_i, t_i) por cada factor extra además de p y q. Los exponentes dP y dQ son enteros positivos menores a p y q respectivamente, tal que:

$$e \cdot dP \equiv 1 \pmod{(p-1)}$$
$$e \cdot dQ \equiv 1 \pmod{(q-1)},$$

y $qInv$ es un entero positivo menor a p, que satisface la siguiente propiedad:

$$q \cdot qInv \equiv 1 \pmod{p}$$

Codificación de RSA en ASN.1

El documento PKCS #1 propone una manera en la que las claves públicas y privadas deberán de codificarse. Se especifican dos objetos `RSAPublicKey` y `RSAPrivateKey` los cuales representan la clave pública y privada respectivamente.
Una clave pública en RSA deberá de ser representado con el objeto `RSAPublicKey`:

```
RSAPublicKey ::= SEQUENCE {
    modulus INTEGER, -- n
    publicExponent INTEGER, -- e
}
```

El objeto `RSAPublicKey` contiene los siguientes campos:

- `modulus` - el módulo n en RSA,

- `publicExponent` - el exponente público e en RSA.

En el documento se especifica el objeto `RSAPrivateKey` para codificar la clave privada de la siguiente manera:

```
RSAPrivateKey ::= SEQUENCE {
    version Version,
    modulus INTEGER, -- n
    publicExponent INTEGER, -- e
    privateExponent INTEGER, -- d
    prime1 INTEGER, -- p
    prime2 INTEGER, -- q
    exponent1 INTEGER, -- d mod (p-1)
    exponent2 INTEGER, -- d mod (q-1)
    coefficient INTEGER, -- (q^-1) mod p
    otherPrimeInfos OtherPrimeInfos OPTIONAL
}
```

El objeto `RSAPrivateKey` contiene los siguientes campos:

- `modulus` - el módulo n en RSA,
- `publicExponent` - el exponente público e en RSA,
- `privateExponent` - el exponente privado d en RSA,
- `prime1` - el primer factor primo p,
- `prime2` - el segundo factor primo q,
- `exponent1` - el exponente dP del CRT,
- `exponent2` - el exponente dQ del CRT,
- `coefficient` - el coeficiente $qInv$ del CRT.

Los campos `version` y `otherPrimeInfos` son necesarios en caso de utilizar una implementación de RSA que haga uso de más de dos factores primos.

4.3. Representación de polinomios

Cada polinomio cuadrático en n variables puede ser representado utilizando la parte triangular superior de una matriz de tamaño $(n+1)\times(n+1)$. Cada uno de los valores de la matriz representa a los coeficientes en el polinomio, como se puede observar en la Tabla 4.4, donde $a_{i,j} \in \mathbb{F}_{p^k}$.

	1	X_1	$\dots$	X_n
1	$a_{0,0}$	$a_{0,1}$	$\dots$	$a_{0,n}$
X_1	0	$a_{1,1}$	$\dots$	$a_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
X_n	0	0	$\dots$	$a_{n,n}$

Tabla 4.4: Representación de un polinomio por medio de la parte triangular superior de una matriz de $(n+1)\times(n+1)$.

De esta manera, cada polinomio cuadrático sobre $\mathbb{F}_{p^k}$ puede ser representado como una cadena binaria de tamaño

$$N = \frac{(n+1)(n+2)}{2} \cdot k \tag{4.1}$$

que representa la concatenación de los valores en la parte triangular superior de la matriz de $(n+1)\times(n+1)$.

La matriz en la Tabla 4.5 representa a un polinomio cuadrático sobre 3 variables con coeficientes en $\mathbb{F}_{2^3}$.

	1	X_1	X_2	X_3
1	110	010	000	111
X_1	0	011	101	001
X_2	0	0	101	111
X_3	0	0	0	110

Tabla 4.5: Matriz polinomial sobre tres variables con coeficientes en $\mathbb{F}_{2^3}$.

Este polinomio se puede representar como la siguiente cadena binaria

$$110010000111011101001101111110$$

y puede ser dividido en partes de 3, donde cada uno representa un elemento en

$$\mathbb{F}_{2^3} = \{a_2x^2 + a_1x + a_0 \mid a_i \in \mathbb{F}_2\}$$

De esta manera la cadena binaria "11001000011101110100110111111 0" tiene la siguiente representación:

$$\underbrace{110}_{x^2+x}\ \underbrace{010}_{x}\ \underbrace{000}_{0}\ \underbrace{111}_{x^2+x+1}\ \underbrace{011}_{x+1}\ \underbrace{101}_{x^2+1}\ \underbrace{001}_{1}\ \underbrace{101}_{x^2+1}\ \underbrace{111}_{x^2+x+1}\ \underbrace{110}_{x^2+x}$$

que representa al polinomio

$$(x^2+1)X_1X_2 + X_1X_3 + (x^2+x+1)X_2X_3 + X_1 + (x^2+1)X_2 + X_3 + (x^2+x)$$

en $\mathbb{F}_{2^3}$.

4.4. Codificación propuesta

Clave pública

```
MPKCPublicKey ::= SEQUENCE {
    variables INTEGER, -- n
    polynomials INTEGER, -- m
    fieldchar INTEGER, -- p
    fielddegree INTEGER, -- k
    PolynomialSet INTEGER, -- P
}
```

- `variables` - un entero positivo que define el número de variables en el anillo de polinomios.
- `polynomials` - un entero positivo que define el número de polinomios en la clave pública.
- `fieldchar` - un entero positivo que define la característica del campo base.
- `fielddegree` - un entero positivo que define el grado del campo base.
- `PolynomialSet` - un entero positivo de tamaño $\left(\frac{(n^2+3n+2}{2} \cdot k \cdot m\right)$-bits que representa al conjunto de polinomios $\mathcal{P} = \{p_1(\boldsymbol{X}), \ldots, p_n(\boldsymbol{X})\}$, $p_i(\boldsymbol{X}) \in \mathbb{F}_q[\boldsymbol{X}]$.

Clave privada

```
MPKCPrivateKey ::= SEQUENCE {
    fieldchar INTEGER, -- p
    fielddegree INTEGER, -- k
    extfield INTEGER, -- e
    theta INTEGER, -- t
    affine1size INTEGER, -- n
    affine1 INTEGER, -- S
    affine2size INTEGER, -- m
    affine2 INTEGER, -- T
}
```

- `fieldchar` - un entero positivo que define la característica del campo base.
- `fielddegree` - un entero positivo que define el grado del campo base.
- `extfield` - un entero positivo que define la extensión del campo.
- `theta` - un entero positivo $0 < \theta < n$, tal que, $\gcd(q^\theta + 1, q^n - 1) = 0$.
- `affine1size` - un entero positivo que define el tamaño de la transformación afín invertible $\mathcal{S}$.
- `affine1` - transformación afín invertible $\mathcal{S}$ de tamaño n.
- `affine2size` - un entero positivo que define el tamaño de la transformación afín invertible $\mathcal{T}$.
- `affine2` - transformación afín invertible $\mathcal{T}$ de tamaño m.

Capítulo

5

Sistema de experimentación

"By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest."

Confucius

En el capítulo 3 se introdujeron algunos esquemas de la criptografía sobre varias variables y sus construcciones para obtener las claves públicas y privadas y así poder realizar operaciones de cifrado, descifrado, firma y verificación tomando en cuenta las propiedades del esquema que se utilize. En el capítulo 4 se introdujo una manera de representar los polinomios y así codificarlos utilizando ASN.1 con el objetivo de lograr una reducción en el tamaño en disco que ocupan tanto las claves públicas como las privadas. Este capítulo muestra la implementación de una plataforma que incorpora las implementaciones de distintos esquemas criptográficos, con la que se pueden realizar operaciones como generaciones de claves, firma y verificación.

5.1. Diseño del sistema

Para la implementación de esta plataforma llamada `MQCrypto`, por ser un sistema que incorpora implementaciones de esquemas criptográficos que basan su seguridad en el problema $\mathcal{MQ}$, se tomó como base la sintaxis que utiliza `OpenSSL`, para de esta manera hacerlo lo mas parecido posible en cuanto a las instrucciones que pueden ser realizadas.

`MQCrypto` incorpora la implementación de un total de diez esquemas, entre los que se encuen-

tran:

- rainbow5640
- rainbow6440
- rainbow16242020
- rainbow256181212
- pflash1
- sflash^{v_1}
- sflash^{v_2}
- uov
- 3icp
- tts6440

Las implementaciones de Rainbow, PFlash, 3ICP y TTS6440 fueron tomadas de [50] y están programas en C++ y C. Las implementaciones de SFlash y UOV (Aceite y Vinagre no Equilibrado) están programadas en SageMath.

Rainbow5640 y Rainbow6440 fueron diseñados por Jintai Ding y Bo-Yin Yang e implementados por Anna Inn-Tung Chen, Tien-Ren Chen, Ming-Shing Chen [7, 6, 2] ambos sobre $\mathbb{F}_{31}$. Los parámetros de Rainbow5640 son $(\mathbb{F}_{31}, 16, 20, 20)$, tiene un total de 56 variables y 40 polinomios. Por otro lado, los parámetros de Rainbow6440 son $(\mathbb{F}_{31}, 24, 20, 20)$, con un total de 64 variables y 40 polinomios.

Las implementaciones de rainbow16242020 y rainbow256181212 son de Ming-Shing Chen [7]. Ambos llevan sus parámetros en sus nombres, $(\mathbb{F}_{2^4}, 24, 20, 20)$ son los parámetros de rainbow16242020 que nos indica que tiene un total de 64 variables y 40 polinomios sobre $\mathbb{F}_{2^4}$. Por otro lado, $(\mathbb{F}_{2^8}, 16, 12, 12)$ nos indica que rainbow256181212 tiene un total de 42 variables y 24 polinomios sobre $\mathbb{F}_{2^8}$. PFlash fue implementado por Chia-Hsin Owen Chen y Ming-Shing Chen [7] y tiene un total de 37 variables y 26 polinomios sobre un campo $\mathbb{F}_{2^4}$. 3icp fue propuesto por Ding [15] y se encuentra en $\mathbb{F}_{31}$.

SFlash^{v_1} y SFlash^{v_2}, ambos siendo implementaciones nuestras, cuentan con un total de 37 variables y 26 polinomios, la diferencia entre ellos es que en SFlash^{v_1} los coeficientes se encuentran sobre $\mathbb{F}_2$ mientras que en SFlash^{v_2} se encuentran sobre $\mathbb{F}_{2^7}$.
Otra implementación nuestra es UOV (aceite y vinagre no equilibrado) sobre $\mathbb{F}_{2^4}$ y cuenta con un total de 120 variables, entre las que se encuentran 40 variables de aceite y 80 de vinagre. De esta manera, las claves públicas de UOV tienen 40 polinomios.
A continuación se presenta una lista de instrucciones que puede ser utilizada en MQCrypto.

`mqcrypto list_schemes`	
`-list_schemes`	Despliega una lista con los esquemas que pueden ser utilizados así como también una descripción de cada uno de ellos.

Tabla 5.1: Lista de esquemas implementados en MQCrypto.

La tabla 5.1 introduce la instrucción `-list_schemes` que permite desplegar un listado de los esquemas que se encuentran implementados y pueden ser utilizados por MQCrypto. La lista de esquemas que se despliega de igual manera da detalles de cada una de las implementaciones, datos tales como: sobre que campo trabajan, el número de variables y número de polinomios. El nombre con el que despliegan es el que se requiere ingresar en la generación de las claves.

`mqcrypto genKeys`	
`genKeys`	Genera un par de claves a partir de un esquema de criptografía sobre varias variables seleccionado que se encuentre implementado.
`-scheme <scheme>`	Selecciona un esquema utilizando su nombre de la lista de esquemas que se encuetrán implementados en MQCrypto.
`-out <private.pem>`	Captura la ruta y nombre del archivo de salida para guardar el par de claves generado por MQCrypto.
`-zip`	La salida codificada en ASN.1 y Base64 pasará por un método de compresión utilizando bzip2 produciendo un archivo binario comprimido.

`-symencryption`	Selecciona un algoritmo de criptografía simétrica para cifrar el par de claves, los algoritmos que pueden ser utilizados son los siguientes: ▪ `aes-128-cbc` ▪ `aes-192-cbc` ▪ `aes-256-cbc`
`-passout pass:<phrase>`	Contraseña utilizada para cifrar el par de claves generada utilizando el algoritmo simétrico seleccionado.

Tabla 5.2: Sintaxis para la generación de un nuevo par de claves.

La sintaxis de la instrucción `genKeys` se muestra en la tabla 5.2 que permite la generación de un nuevo par de claves seleccionando algún esquema implementado por `MQCrypto`. La generación de las claves puede producir dos diferentes tipos de archivos según los parámetros que sean ingresados. El par de claves siempre se codifica en ASN.1, de ahí las salidas que se pueden obtener son las siguientes:

- Un archivo ASCII con el par de claves en ASN.1 codificado en Base64.
- Un archivo en binario con el par de claves codificado en ASN.1 y Base64, aplicando un método de compresión utilizando bzip2.

`mqcrypto keys <keypair.pem>`	
`keys <keypair.pem>`	Extrae la clave pública o privada en archivos separados y descifrados.
`-pubout <public-key.pem>`	Captura la ruta y nombre del archivo de salida para guardar la clave pública que será extraída del par de claves.
`-privout <private-key.pem>`	Captura la ruta y nombre del archivo de salida para guardar la clave privada que será extraída del par de claves. (La clave privada será almacenada descifrada)
`-passin pass:<phrase>`	Contraseña necesaria para descifrar el par de claves.

Tabla 5.3: Sintaxis para la extracción de claves.

La generación de claves produce un único archivo con el par de claves codificada en un objeto en ASN.1, a partir de ese archivo es posible extraer la clave pública para así poder enviarla a las entidades con quien se desee establecer una comunicación segura utilizando algún esquema implementado en `MQCrypto`. Para realizar la extracción de la clave pública o privada del par de claves es necesario utilizar la función `keys`, la sintaxis para realizar esa extracción se muestra en la tabla 5.3. De igual manera se puede extraer la clave privada del archivo que contiene el par de claves, sin embargo se recomienda no hacerlo ya que la clave privada que será extraida se almacena en un archivo descifrado y por lo tanto es inseguro.

`mqcrypto sign <private-key.pem>`	
`sign <private-key.pem>`	Recibe un archivo de entrada y produce una firma dada una clave privada generada previamente.
`-in <file>`	Captura la ruta y nombre del archivo de entrada que será firmado.
`-out <signature>`	Captura la ruta y nombre del archivo de salida para guardar la firma producida.
`-dgst`	Función hash a utilizar para la producción de la firma, las funciones que se pueden utilizar son las siguientes: ▪ `sha256` ▪ `sha512`
`-passin pass:<phrase>`	Contraseña necesaria para descifrar el par de claves.

Tabla 5.4: Sintaxis para firmar un documento.

La tabla 5.4 muestra la sintaxis para firmar un documento. Para producir una firma digital es necesario seleccionar el documento que se desea firmar, y la clave privada a utilizar. Seleccionando una función picadillo la firma será almacenada en un archivo que haya sido ingresado como parámetro de entrada. Para la producción de una firma digital es necesario utilizar un par de claves de un

esquema que se encuentre implementado en MQCrypto.

mqcrypto verify <public-key.pem>	
verify <public-key.pem>	Dado un archivo de entrada y una firma realiza la verificación de la firma sobre el archivo.
-in <file>	Captura la ruta y nombre donde se encuentra el archivo de entrada.
-signature <signature>	Captura la ruta y nombre donde se encuentra almacenada la firma.
-passin pass:<phrase>	Contraseña necesaria para descifrar el par de claves.

Tabla 5.5: Sintaxis para la verificación de un firma digital.

Dada una firma digital, un documento, y una clave pública la verificación se lleva a cabo. Se verifica que el esquema utilizado para la firma digital sea igual al esquema de la clave pública y se procede con la verificación. La sintaxis necesaria para realizar la verificación de una firma digital se encuentra en la tabla 5.5.

5.2. Implementación

Para la implementación de MQCrypto se utilizaron las siguientes bibliotecas:

- asn1c: para el proceso de codificación en ASN.1 en C/C++.
- openssl: algoritmos de criptografía simétrica aes-128-cbc, aes-192-cbc y aes-256-cbc.
- libsodium: un generador pseudoaleatorio y las funciones hash SHA-256 y SHA-512.
- libb64: para el proceso de codificación en Base64.
- libbzip2: comprimir utilizando bzip2.

La plataforma se encuentra en un repositorio en **Github** donde puede ser descargada para realizar pruebas:

https://github.com/eliverperez/MQCrypto

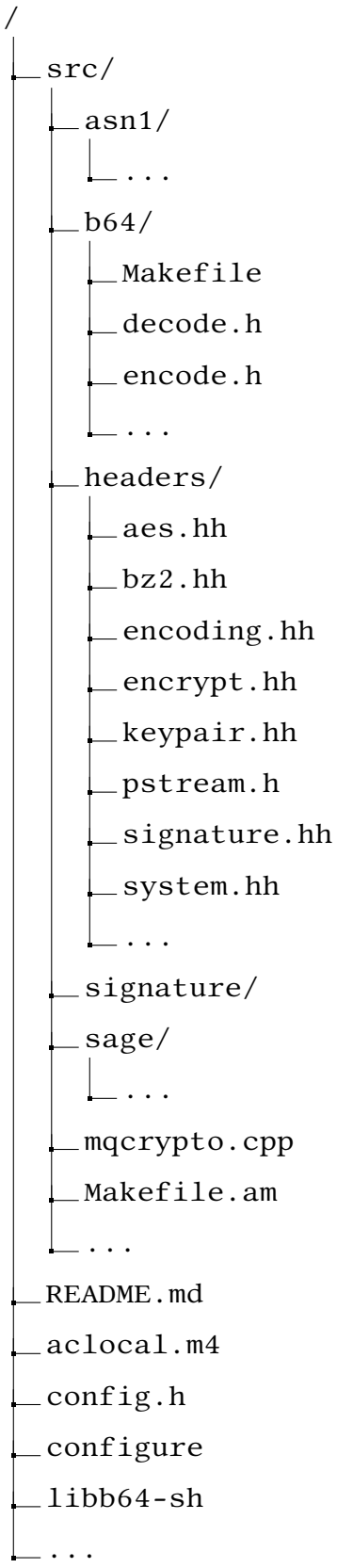

Figura 5-1: Estructura de archivos de la implementación de MQCrypto.

El contenido del repositorio tiene una estructura como la que se muestra en la figura 5-1, con los archivos necesarios para la ejecución de la plataforma.
En el directorio `asn1` se encuentra almacenado el código con las estructuras de las claves públicas y privadas así como también las de un mensaje cifrado y una firma digital, de igual manera los archivos de la biblioteca `asn1c` para el manejo del tipo de datos en ASN.1 se encuentran ahí. El directorio `b64` contiene la biblioteca para codificar/decodificar en Base64.
En `headers` se encuentran almacenados los encabezados para las distintas operaciones que se han de ejecutar en `MQCrypto`. El archivo `aes.hh` contiene la implementación del cifrado utilizando AES haciendo uso de las herramientas de `OpenSSL`, de esta manera se logra el cifrado con los algoritmos: aes-128-cbc, aes-192-cbc y aes-256-cbc. El archivo `bz2.hh` contiene las funciones para comprimir/descomprimir tanto un archivo ASCII como un archivo binario utilizando el compresor bzip2. En `encoding.hh` se encuentra el código necesario para realizar las codificaciones tanto en ASN.1 como en Base64. Las funciones para la generación de un par de claves se encuentran en el archivo encabezado `keypair.hh`. Los archivos `rainbow5640.h`, `rainbow6440.h`, `rainbow16242020.h`, `rainbow256181212.h`, `3icp.h` y `tts6440` contienen las funciones para la generación de un nuevo par de claves, firmar un documento y verificar una firma utilizando los esquemas `rainbow5640`, `rainbow6440`, `rainbow16242020`, `rainbow256181212`, `3icp` y `tts6440` respectivamente. En el encabezado `signature.hh` se encuentra lo necesario para ejecutar la instrucción cuando se quiere firmar un documento o realizar la verificación de una firma. `system.hh` contiene algunas funciones básicas como la lectura de un archivo, la ejecución de la terminal y el parsing de las instrucciones ingresadas entre otras cosas más.
En el directorio `signature` se encuentran las implementaciones de los esquemas de firmas digitales: `pflash`, `rainbow5640`, `rainbow6440`, `rainbow16242020`, `rainbow256181212`, `tts6440` y `3icp` en C y C++. Mientras que en `sage` se encuentran las implementaciones de $\texttt{sflash}^{v_1}$, $\texttt{sflash}^{v_2}$ y UOV (Aceite y Vinagre no Equilibrado) en `SageMath`.

5.3. Instalación

Para la instalación de `MQCrypto` en sistemas Linux/Unix es necesario contar con `OpenSSL`, `libsodium` y `libbzip2` ya instalados previamente. La instalación de la plataforma se puede realizar de manera muy sencilla sólo ejecutando las siguientes instrucciones:

```
./libb64-sh
```

```
./configure
make
sudo make install
```

en donde se empieza primero que nada por realizar la compilación de la biblioteca `libb64`, ejecutando el archivo `libb64-sh`. Después de eso el archivo **Makefile** hará el resto, realizando la compilación del sistema y las biblioteca `asn1c`, posteriormente moviendo los archivos a las carpetas correspondientes para una correcta instalación de `MQCrypto`.

5.4. Resultados

Generación de un par de claves

Como se mostró en la sección 5.1, es posible generar un par de claves ejecutando la instrucción:

```
genKeys -scheme <scheme> ...
```

Los esquemas que pueden ser utilizados para la generación de un nuevo par de claves se mostraron en la sección 5.2. A la hora de generar un nuevo par de claves, independientemente del esquema que se quiera utilizar, se genera un único archivo con el par de claves contenido dentro de el, la clave pública y la clave privada pueden ser extraídas posteriormente. El archivo generado es cifrado utilizando un algoritmo de criptografía simétrica, el que es seleccionado a la hora de la generación. Si no se indica que algoritmo utilizar, de manera predeterminada se utiliza `aes_256_cbc`. `-passout pass:<phrase>` indica la contraseña a utilizar para cifrar la pareja de claves generada, si no se indica una contraseña en el llamado a la generación de las claves, esta se requerirá a la hora de realizar la ejecución de la instrucción. La captura de la contraseña no será mostrada en pantalla, tal como se muestra en la figura 5-2, para una mayor seguridad se recomienda utilizar este método, pues el historial de la terminal puede ser obtenido por una entidad maliciosa que de este modo puede tomar control del par de claves generado.

La figura 5-3 muestra la generación de un nuevo par de claves utilizando el esquema de firma digital `rainbow16242020`, el cual esta sobre $\mathbb{F}_{2^4}$ y tiene un total de 64 variables y 40 polinomios. Al ejecutar la instrucción para generar el nuevo par de claves se imprime un mensaje con el nombre del esquema que ha sido seleccionado, en este caso `"Generating Rainbow16242020 Keypair..."`, seguido de un mensaje que indica que el par de claves ha sido generado de

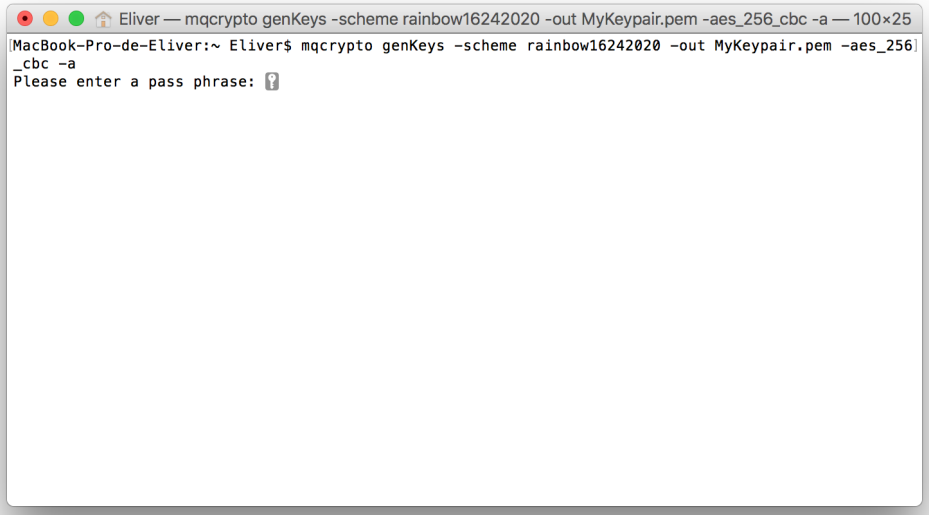

Figura 5-2: Solicitud de contraseña en la generación de un nuevo par de claves.

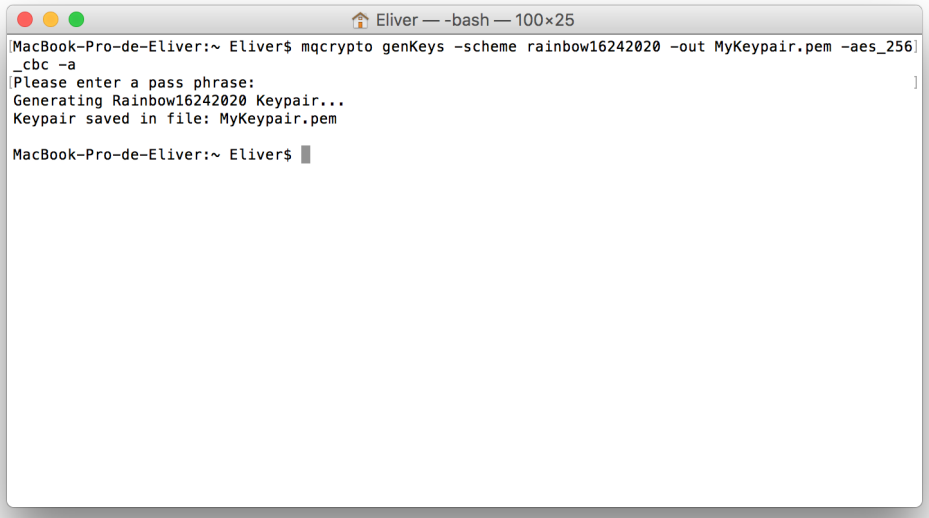

Figura 5-3: Generación de un nuevo par de claves utilizando `rainbow16242020`.

manera satisfactoria en un archivo que fue indicado previamente utilizando `-out <file>`. La pareja de claves tiene dos opciones para su salida: puede ser almacenada como un archivo ASCII codificada en Base64, así como también se le puede aplicar un método de compresión utilizando bzip2 sólo con agregar la etiqueta `-zip` a la ejecución de `genKeys`. En la ejecución de la figura 5-3 se obtiene una pareja de claves codificada en Base64.

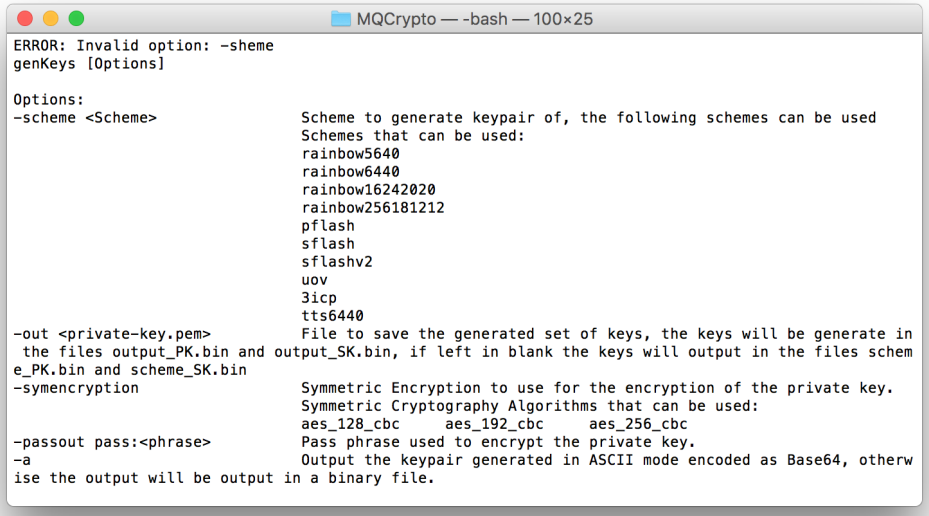

Figura 5-4: Menú de ayuda de la función `genKeys`.

En caso de que exista algún error en la sintaxis a la hora de realizar la ejecución de `genKeys`, se imprime un mensaje donde se indica el de error cometido seguido de las instrucciones necesarias para generar un nuevo par de claves utilizando la función `genKeys`, desplegando cada uno de los parámetros que se pueden recibir como entrada. La figura 5-4 muestra el menú que se despliega cuando existe un error en la ejecución.

Una vez generado el par de claves, se almacena un archivo codificado en Base64, este archivo contiene la información necesaria para saber si es un par de claves (Keypair), una clave pública (Public Key) o una clave privada (Private Key) y el algoritmo de cifrado que fue utilizado en caso de ser un archivo que contiene el par de claves y esta cifrado utilizando un algoritmo de criptografía simétrica.

Para la pareja de claves generadas utilizando `MQCrypto` se obtiene un archivo con información que indica que el contenido codificado en Base64 almacena un par de claves y que éstas se encuentran cifradas utilizando `aes_256_cbc`. El contenido del archivo es similar al siguiente:

```
-----BEGIN MPKC KEYPAIR-----
ENC: AES-256-CBC

MIMDLQYWC3JhaW5ib3c2NDQwBILhABoLuYqiqDGywtuZOqnZtGrm2YpwW7HbOS
JfC8YhoDfWLm+jdqHolSE33SYALt4NdaPPrOsMPBPMmV0mCIcjBkGj7qZcoGGF
IKifQARrBxl4Qpc4yiPSCYiyTUreA+fZfOiNunkJT/iCkhP7jrguO6QJwgMkMJ
                    .                .                .
                    .                .                .
                    .                .                .
/wYACwD5/wsAAAAGAPX/DgAFAAoA9P/3/w0ADQDz///BwAAAA4ADAD+/woA/v
/wcACwDz//b///8CAPr/DgALAPn//P8IAAIA9P8PAA8A8f/7//z//P/6//X/+f
/wEABAD2/wkAAgD0//H/9P/+/wUA/P8=
-----END MPKC KEYPAIR-----
```

A partir de este archivo es posible realizar la extracción tanto de la clave pública como de la clave privada. Esto es posible mediante la instrucción:

```
keys <keypair.pem> ...
```

Recibiendo como parámetro el archivo donde se encuentran almacenados el par de claves generado, es posible realizar la extracción de la clave pública y privada. La clave pública en `keypair.pem` puede ser extraída utilizando `-pubout public-key.pem` para almacenarla de manera independiente y descifrada en caso de que quiera ser compartida con alguna otra entidad con la que se desee establecer una comunicación. Por otro lado, `-privout <private-key.pem>` realiza la extracción de la clave privada, almacenandola de manera independiente y descifrada. Se recomienda no realizar la extracción de la clave privada, por fines de seguridad, siendo que al ser almacenada de manera descifrada esta vulnerable a que pueda ser tomada por una entidad maliciosa. Con `-passin pass:<phrase>`, es posible ingresar la contraseña para descifrar el par de claves, de manera similar a la generación de claves, no es necesario ingresar la contraseña en la linea de comandos (donde quedará almacenada en el registro de la terminal) y puede ser ingresada durante la ejecución para que así pueda ocultarse.
Si existe un error en la sintaxis de la instrucción, se despliega el menú de ayuda para la extracción de las claves, tal como se puede observar en la figura 5-5.

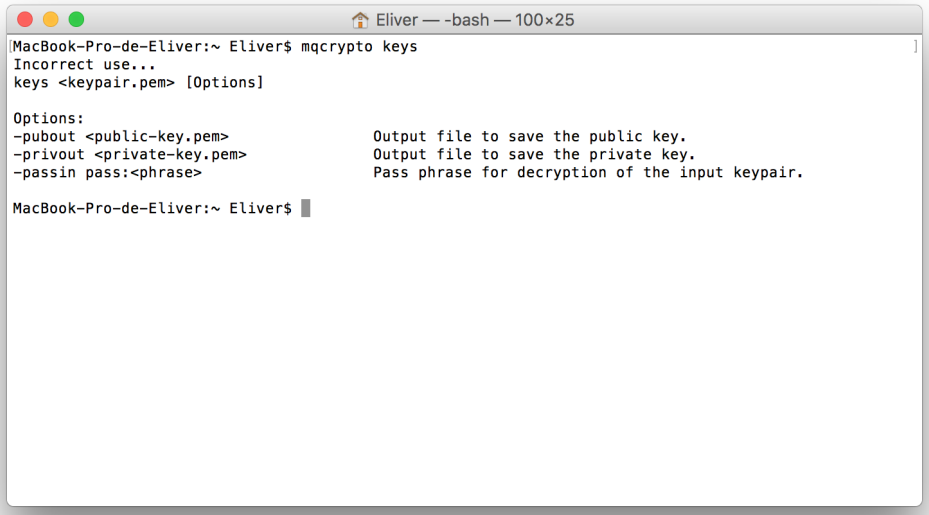

Figura 5-5: Menú de ayuda de la función `keys`.

Una vez que sea ejecutada la extracción del par de claves, `mqcrypto` lee la clave pública, identifica el esquema al que pertenece el par de claves y separa la clave pública de la privada, almacenando cada una en distintos archivos, la figura 5-6 muestra una correcta ejecución en la que se separa el par de claves almacenada en un archivo llamado `keypair.pem`, la clave pública se almacena en un archivo llamado `public-key.pem` mientras que la clave privada es almacenada en el archivo `private-key.pem`.

Una vez extraídas la clave pública y privada son almacenadas en archivos no cifrados, éstas pueden ser codificadas en Base64 o comprimidas utilizando bzip2. Almacenada en Base64, la clave pública y la clave privada pueden ser identificadas por medio del encabezado. Por ejemplo, la clave pública tiene una estructura como la siguiente:

```
-----BEGIN MPKC PUBLIC KEY-----
MILRqhYQcmFpbmJvdzI1NjE4MTIxMgSCdiB5F//tDMFF/d50jT/rfwig74UnDG
AAAAACyBIJ5PzuhI9DlUi4VYVpFbdGIxoQWEpAAAAAAAAAAAQKKFwMiA9ivr1r
eV1SpIMHAAAAAAAAAADviW+W1SsF6uNw/vdexkWd6ioL7QKpO6wAAAAAAAAAAI
                    .                   .                   .
                    .                   .                   .
                    .                   .                   .
```

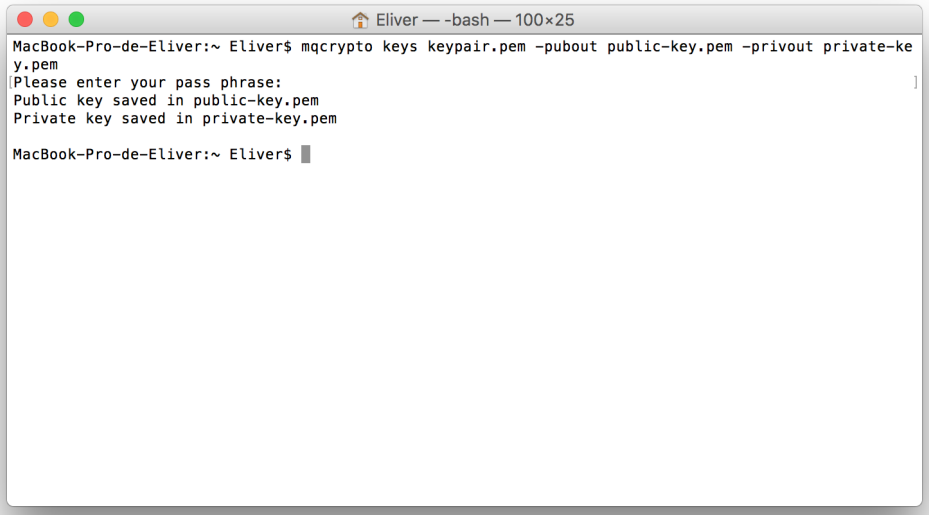

Figura 5-6: Extracción de clave pública y privada utilizando la función `keys`.

```
GA4Wef0ra8HlQAAAAADjYg1z2JwMcSbkZkIAAAAA4qohCGM800Sqfb8vAAAAAB
VrUnTAAAAABLUroKScLBFoHwzDgAAAAAn8pvQixpsQXTgSpJAAAAAMhz4gud9I
AAA=
-----END MPKC PUBLIC KEY-----
```

Por medio del encabezado ——BEGIN MPKC PUBLIC KEY—— es posible identificar, que la codificación en Base64 pertenece a una clave pública que contiene un objeto en ASN.1 con la información necesaria para su uso.

De manera similar a una clave pública, la clave privada una vez extraída de la pareja de claves, tiene una estructura como la siguiente:

```
-----BEGIN MPKC PRIVATE KEY-----
MIMBL3sWBnBmbGFzaASDARm8TbYM0xB3gYgk4YDXsHZY4uFeZbX/8frh5TAyYy
rEPi5aeOYx4AJShHvbJE+9SvU/OPlIPgFZiglKQNIWoHQboFHQbV0Hldx2+tPK
iqN703HGYsVUy0FMdVARHxF0QkZBzAuvZdmEEUSM5w16Tq7Nhk0hJYBd4vFI8/
                    .                .                .
                    .                .                .
                    .                .                .
5oboB/bpysHrnYDxTp1P6HaTpT5bUEGgf576ME7GY2FJ8L/KrXfn2dMMrktT+m
```

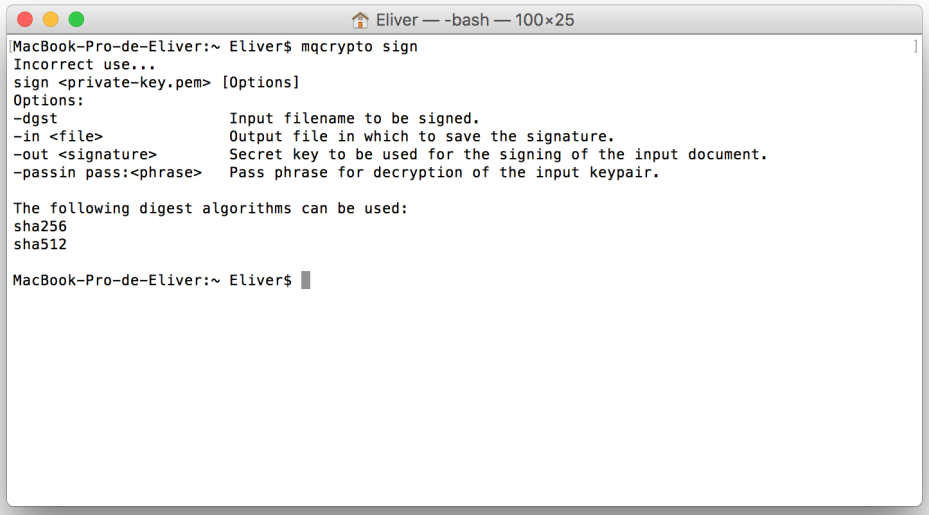

Figura 5-7: Menú de ayuda de la función `sign`.

```
R7p83oYpAmrSnPZnRz8kxbsyJ+Zt9bMV3LUbzEEeQLVK3JbAYbCDW+Yts8cJUr
OBOBYy3Kj/Gpg7kll4KU3W0F9XYcRArvB1Rn695uKQsF5n5W7///jqSujds=
-----END MPKC PRIVATE KEY-----
```

Por medio del contenido a simple vista es difícil saber la diferencia de si es una clave pública o una clave privada, la principal diferencia se encuentra en los encabezados como mencionamos anteriormente, en el caso de las clave privadas el encabezado ——BEGIN MPKC PRIVATE KEY—— indica que la codificación pertenece a una clave privada.

Firma

Una vez generado un par de claves es posible firmar un documento. La función `sign` es la encargada de llevar a cabo la ejecución de esta operación. La sintaxis de esta función se mostró previamente en la sección 5.1. Tal como sucede con las operaciones anteriores, en caso de cometer un error en la sintaxis se despliega el menú de ayuda como se muestra en la figura 5-7, que indica las instrucciones necesarias para ejecutar esta operación y poder firmar de manera correcta un documento utilizando una clave privada que haya sido generada previamente en `MQCrypto`.
La figura 5-8, muestra una correcta ejecución del firmado de un documento. Se utiliza como par de claves el archivo `keypair.pem`, y se desea realizar la firma digital del archivo `file.txt`

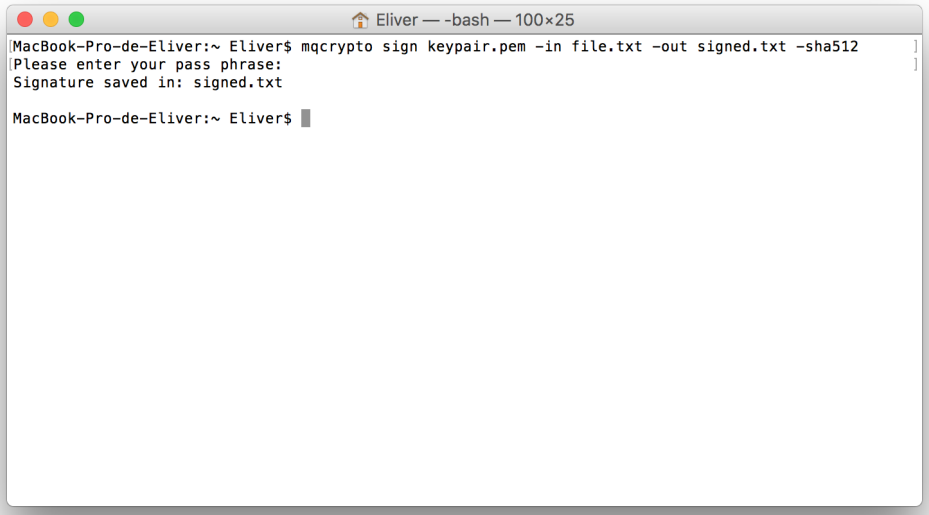

Figura 5-8: Firma de un documento.

utilizando la clave privada que se encuentra almacenada en esta pareja de claves. La firma generada es almacenada en un archivo nombrado `signed.txt` y se utiliza la función picadillo SHA-512. Como es una pareja de claves la que se utiliza en este caso, hay una gran probabilidad de que se encuentren cifradas utilizando algún algoritmo de criptografía simétrica por lo cual si es así será necesario ingresar la contraseña que fue utilizada para cifrar el par de claves, para de esta manera lograr descifrar y utilizar la clave privada con el objetivo de realizar el proceso de la firma digital. En caso de que se utilice la clave privada que haya sido extraída previamente y ésta se encuentra descifrada se omite este paso. La firma en esta ejecución es almacenada en ASCII codificada en Base64, de igual manera sin utilizar la etiqueta `-a` la salida que se produce puede ser almacenada en binario y con `-zip` aplicar un método de compresión.

Al igual que las claves públicas y privadas codificadas en Base64, se puede identificar que un archivo contiene una firma digital si se encuentra almacenado en ASCII. Una firma digital, tiene la siguiente estructura:

```
-----BEGIN MPKC SIGNATURE-----
MBcWC3JhaW5ib3c2NDQwFgZzaGE1MTIEAA==
-----END MPKC SIGNATURE-----
```

El encabezado ——BEGIN MPKC SIGNATURE—— indica que el contenido codificado perte-

nece a una firma digital, la estructura en ASN.1 codificada contiene los elementos necesarios para identificar que esquema fue utilizado para generar la firma, al igual que la función picadillo.

Verificación

La sintaxis necesaria para realizar la verificación de una firma digital se mostró en la sección 5.1. En caso de cometer algún error en la sintaxis, se indica cual es el error seguido de el menú de ayuda de la función `verify` como se puede observar en la figura 5-9.

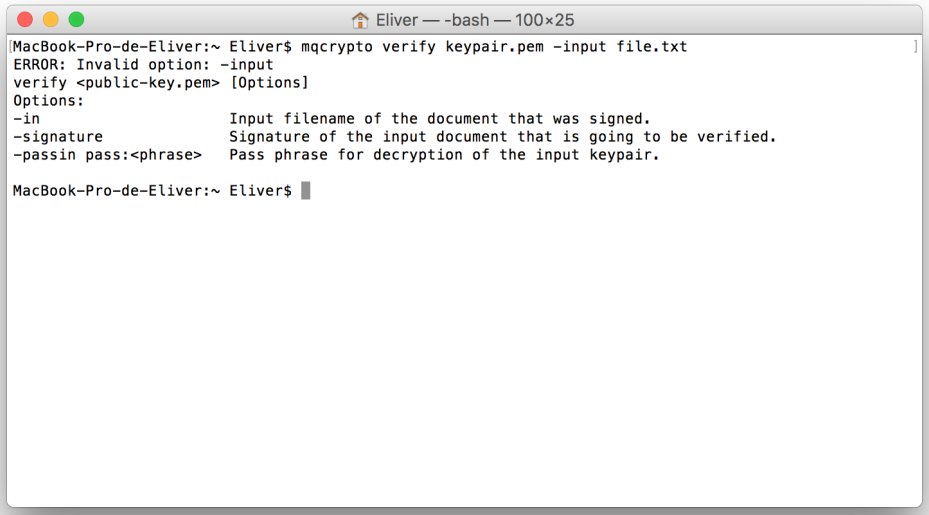

Figura 5-9: Menú de ayuda de la función `verify`.

La figura 5-10 muestra la verificación de la firma digital del documento `file.txt` generado anteriormente. La firma se encuentra en el archivo `signed.txt`, y se utiliza la clave pública en `keypair.pem`. Si la clave ingresada se encuentra cifrada, es necesario ingresar la contraseña, puede ser introducida utilizando `-passin pass:<phrase>`, o posteriormente en la ejecución de la verificación. El proceso termina con un mensaje en el que indica el estado de la verificación, en caso de que la firma haya sido verificada con éxito se imprime el mensaje `Verify: OK`, de lo contrario `Verify: Fail` en caso de que la firma haya sido rechazada utilizando la clave pública ingresada.

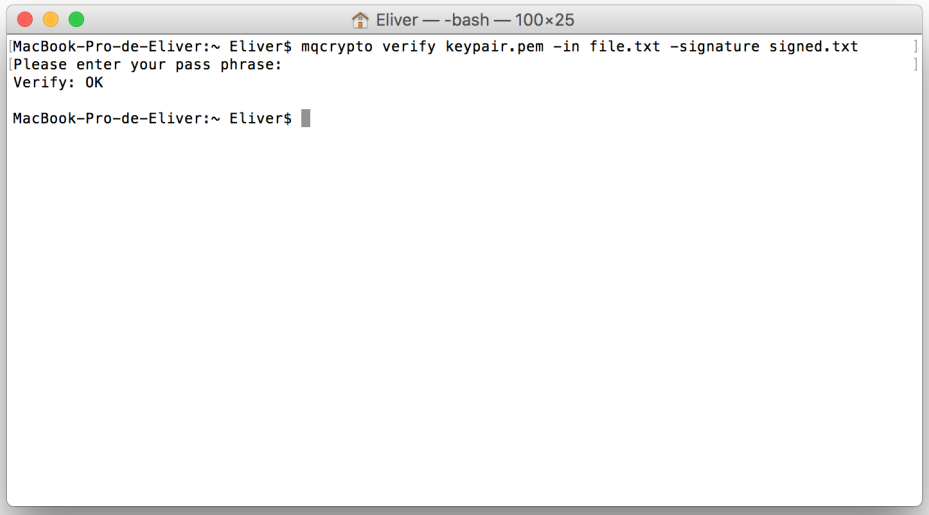

Figura 5-10: Verificación de una firma digital.

5.5. Tamaño del par de claves

Tomando en cuenta las diferentes construcciones de los esquemas que fueron implementados, se realizaron algunas pruebas para calcular el crecimiento de las claves. En estas pruebas se utilizaron las construcciones de los esquemas de `Rainbow` en los campos $\mathbb{F}_{31}$, $\mathbb{F}_{2^4}$ y $\mathbb{F}_{2^8}$ el cual es una variante de `Aceite y Vinagre` con una construcción por capas. `TTS6440`, una variante de `Rainbow` en $\mathbb{F}_{31}$. $\texttt{SFlash}^{v_1}$ y $\texttt{SFlash}^{v_2}$ con coeficientes en $\mathbb{F}_2$ y $\mathbb{F}_{2^7}$ respectivamente. `PFlash`, siendo una variante de `SFlash` con coeficientes en $\mathbb{F}_{2^4}$.

En el crecimiento del tamaño de las claves se tomarón en cuenta cuatro diferentes salidas: cuando se encuentran almacenadas en un archivo binario con las claves tal y como fueron generadas, las claves codificadas en un objeto en ASN.1 obteniendo una salida en un archivo binario, la pareja de claves una vez en ASN.1 se le aplica una codificación en Base64 para así proveer un archivo ASCII y por último a la codificación en ASN.1 se le aplica un método de compresión devolviendo un archivo binario.

Par de claves en binario

En la construcción de cada una de las implementaciones se tomó en cuenta el tamaño que producía la clave pública y la clave privada juntas, en algunos esquemas la clave pública tiende a ser mas

grande que la privada mientras que en algunos otros sucede lo opuesto. Las pruebas para observar como es que crecían las claves producidas se llevaron a cabo desde un número de 8 variables y 4 polinomios hasta 120 variables y 80 polinomios.

Tomando en cuenta los resultados producidos de las claves almacenadas en archivos binarios, PFlash es quien produce las claves más grandes acercándose a los 2.4MB en instancias de 120 variables y 80 polinomios. Mientras que por otro lado SFlash^{v_1} es quien produce las claves mas cortas con un tamaño de alrededor de 76KB en instancias de 120 variables y 80 polinomios seguido de TTS. La gráfica 5-11 muestra el crecimiento de las claves con los esquemas utilizados en estas pruebas. Los resultados se pueden observar con más detalle en el Apéndice A en la tabla A.1.

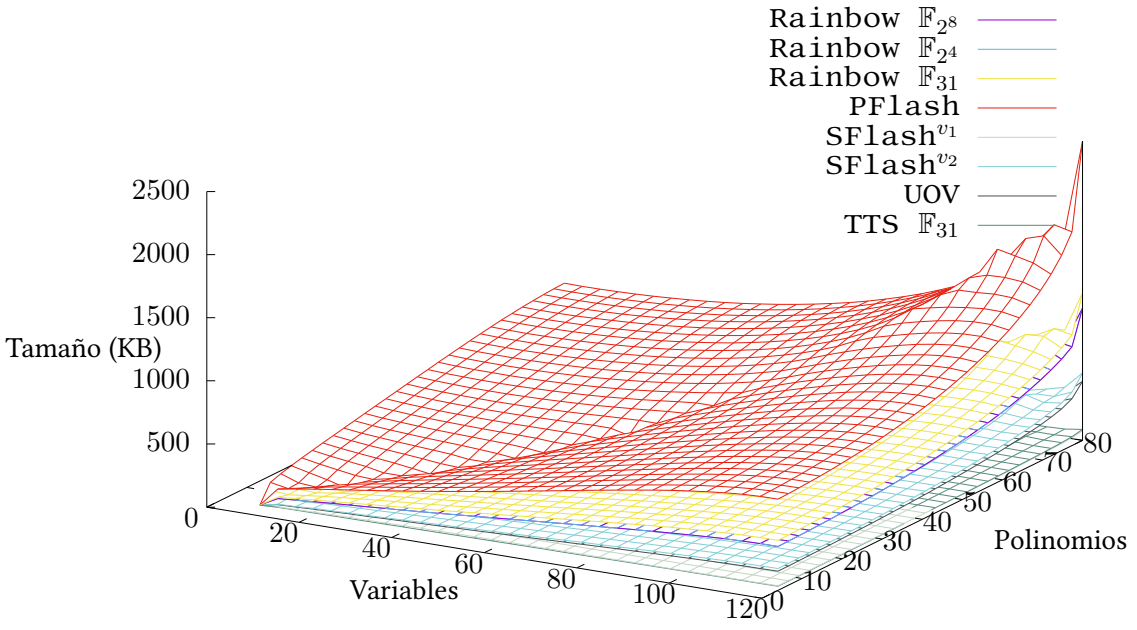

Figura 5-11: Tamaño de par de claves almacenadas en un archivo binario.

Par de claves en ASN.1

Una vez generada la pareja de claves, esta es almacenada en un objeto en ASN.1 que contiene información acerca del esquema al que pertenecen las claves y el contenido tanto de la clave pública como la clave privada. De esta manera se logran encapsular ambas claves en un único objeto. Posteriormente, éstas pueden ser extraídas para ser utilizadas de manera individual. Se realizaron algunas pruebas para analizar el crecimiento de las claves codificadas en ASN.1.

La gráfica 5-12 muestra el crecimiento del par de claves codificadas en ASN.1, como se puede observar, el orden sigue igual a los resultados anteriores, siendo que la codificación añade un crecimiento constante al tamaño original. De esta manera, PFlash sigue contando con las claves de mayor tamaño mientras que SFlash^{v_1} produce las claves más cortas. En la tabla A.2 se despliegan los resultados del crecimiento de las claves cuando se encuentran codificadas en ASN.1.

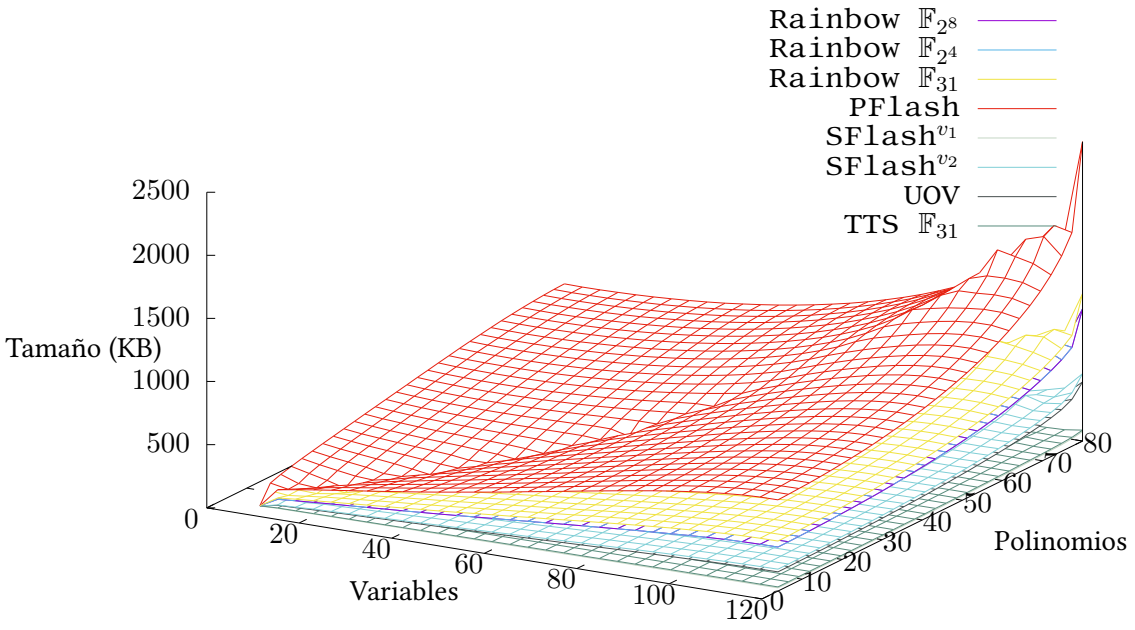

Figura 5-12: Tamaño de par de claves almacenadas codificadas en ASN.1.

Par de claves en Base64

Una vez codificadas en ASN.1 contamos con un arreglo de bytes que puede ser almacenado en un archivo binario. A este contenido se le añade una codificación en Base64 para así poder representarlo en ASCII, sin embargo, la codificación en Base64 produce un crecimiento de aproximadamente un 35 % en el tamaño de la clave codificada en ASN.1.

El crecimiento de nuevo es constante sobre el tamaño de los archivos codificados en ASN.1. Por lo tanto, no existe cambios en el orden y PFlash sigue contando con las claves más grandes tomando en cuenta instancias donde con 8 variables y 2 polinomios tiene un tamaño de ≈1.1KB y ≈3.2MB con 120 variables y 80 polinomios. Mientras que SFlash^{v_1} sigue produciendo las claves más cortas, tal como se puede observar en la gráfica 5-13. Los resultados se encuentran más

detallados en la tabla A.3.

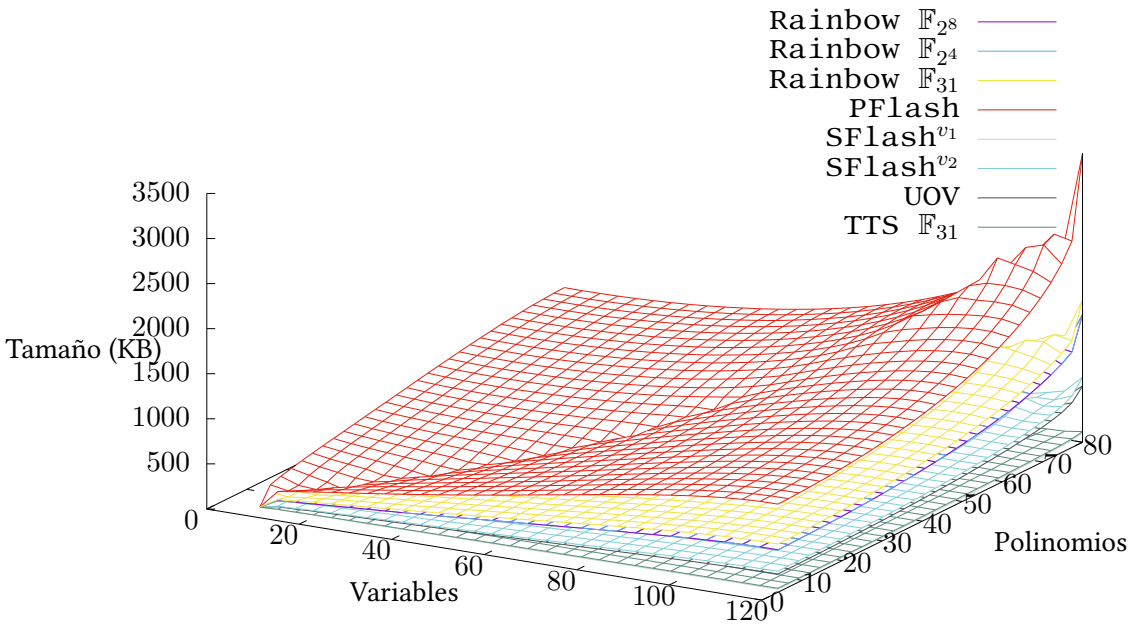

Figura 5-13: Tamaño de par de claves codificadas en ASN.1 almacenadas en ASCII en Base64.

Par de claves comprimidas

A la pareja de claves codificadas en ASN.1 se les aplica un método de compresión, con el objetivo de lograr una reducción en el tamaño de las claves. De esta manera se obtiene de nuevo un archivo binario que por lo regular (se espera) ocupa menos espacio que el original y es posible obtener de nuevo el mismo contenido que fue comprimido sin tener alguna perdida, ya que la perdida de un solo bit sobre la clave pública o privada echaría todo a perder. Se utilizó el método de compresión bzip2, con el cual sin ningún problema es posible recuperar el contenido que fue comprimido.

La compresión provocó algunas modificaciones en el tamaño de las claves, sin embargo hubo algunas que en lugar de reducir su tamaño tuvieron un crecimiento, tal como `PFlash` que de esta manera sigue ocupando el primer puesto en las claves más grandes. `SFlash`^{v_1} sigue con las claves más cortas. `Rainbow` sobre $\mathbb{F}_{2^4}$ tuvo una de las reducciones más grandes, esto se debe a que se utilizaron enteros de 8-bits para representar a los coeficientes en el campo, sin embargo es necesario solo de 4-bits para representar un elemento en $\mathbb{F}_{2^4}$, por lo tanto en cada coeficiente se desperdician 4 bits que quedan en ceros y de esta manera al encontrar tantas repeticiones, la

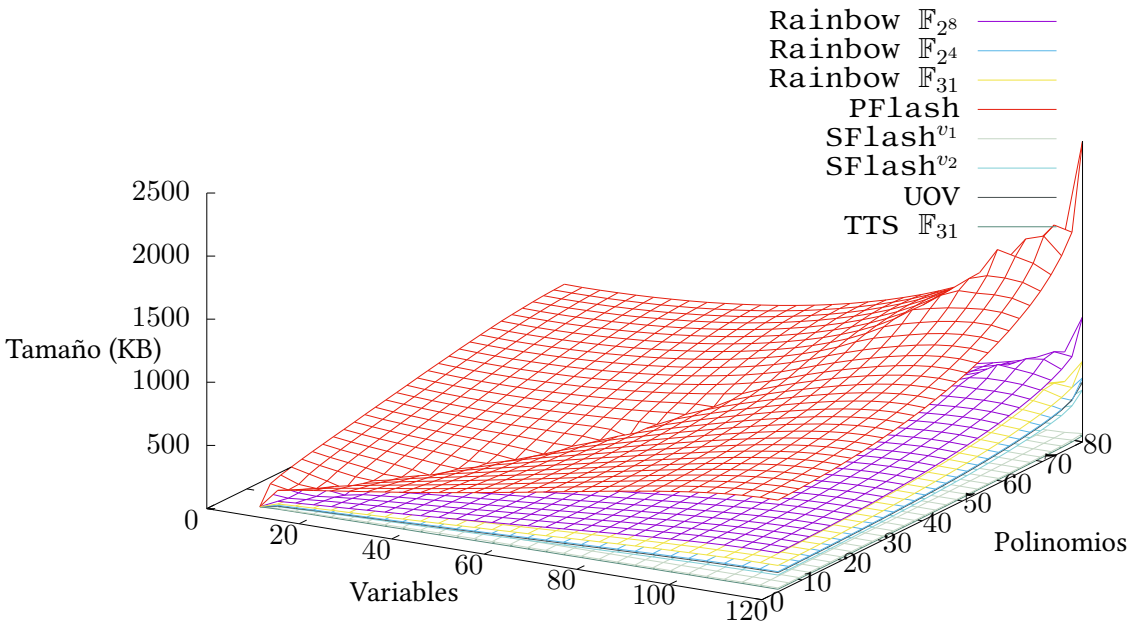

Figura 5-14: Tamaño de par de claves codificadas en ASN.1 y comprimidas utilizando bzip2.

compresión logra ser más eficiente.

La gráfica 5-14 muestra el crecimiento de las claves tomando en cuenta instancias desde 8 variables y 4 polinomios hasta 120 variables y 80 polinomios. La tabla A.4 muestra detalles del crecimiento de las claves utilizando bzip2.

Aunque algunos esquemas con estos parámetros no lograron reducirse utilizando bzip2, con los parámetros pertenecientes a los esquemas si se logra una reducción de su tamaño original. Tal como `PFlash` que con las pruebas realizadas en lugar de lograr una reducción tuvo un crecimiento, sin embargo, con sus parámetros se logra una reducción de ≈76 % de su tamaño original.

Clave pública en binario

Cuando se genera un nuevo par de claves, la clave privada, como su nombre lo indica se mantiene en secreto y no se envía a ningún lugar. Por otro lado la clave pública es de dominio público, y es necesario enviar esta clave a cualquier entidad con la que se desee establecer una comunicación segura, por lo tanto, es una clave que se esta transfiriendo constantemente y es un poco más importante su tamaño y lograr reducirla que la clave privada.

Se realizaron algunas pruebas en las que de las claves anteriormente generadas se toma en cuenta

solamente la clave pública, para de esta manera observar su crecimiento conforme el tamaño de las instancias de claves variaba tomando en cuenta valores desde 8 variables y 4 polinomios hasta 120 variables y 80 polinomios.

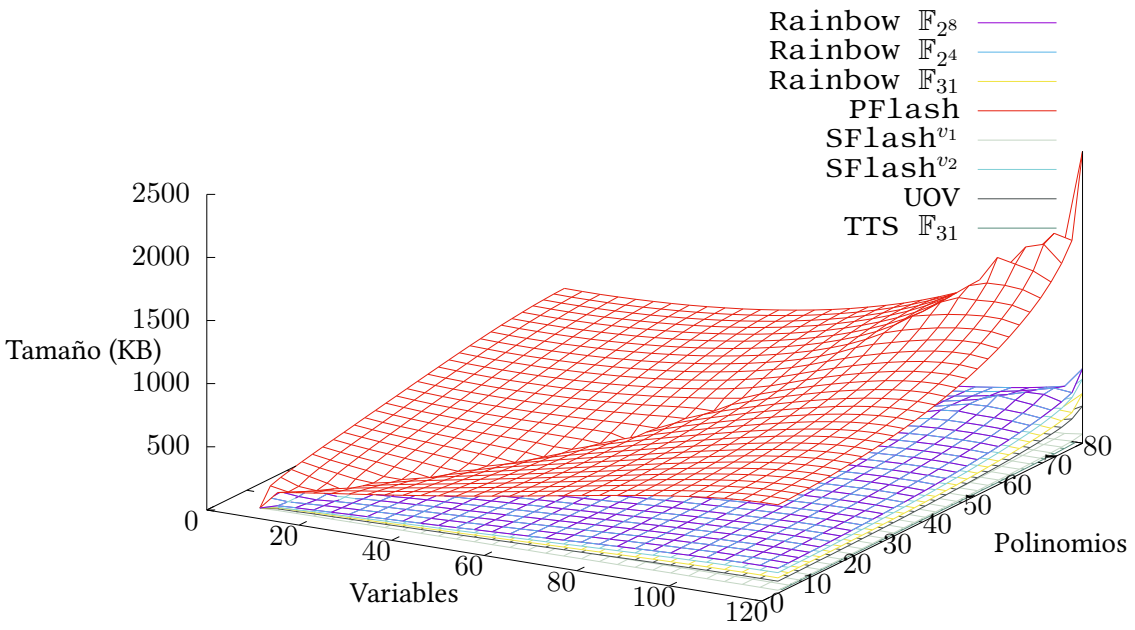

Figura 5-15: Tamaño de clave pública almacenadas en un archivo binario.

De nuevo PFlash cuenta con las claves con mayor tamaño con la clave pública tomando casi todo el espacio del par de claves, dejando lugar a la generación de una clave privada muy pequeña. TTS sobre $\mathbb{F}_{31}$ cuenta con las claves públicas más cortas, mientras que SFlash^{v_1} le sigue con coeficientes en $\mathbb{F}_2$.

La gráfica 5-15 muestra el crecimiento de las claves representadas por un arreglo de bytes a la hora de su generación. Los detalles del crecimiento de la clave pública se muestran en la tabla A.5.

Clave pública en ASN.1

Cuando se genera el par de claves codificada en ASN.1, es posible lograr la extracción tanto de la clave privada como de la clave pública. Una vez extraída la clave pública se almacena en un nuevo objeto en ASN.1 con la información necesaria para describir la clave almacenada. El crecimiento de las claves públicas al ser codificadas en ASN.1 tiene un crecimiento constante, por lo que no provoca cambio alguno en el orden del tamaño de las claves que se mostró anteriormente.

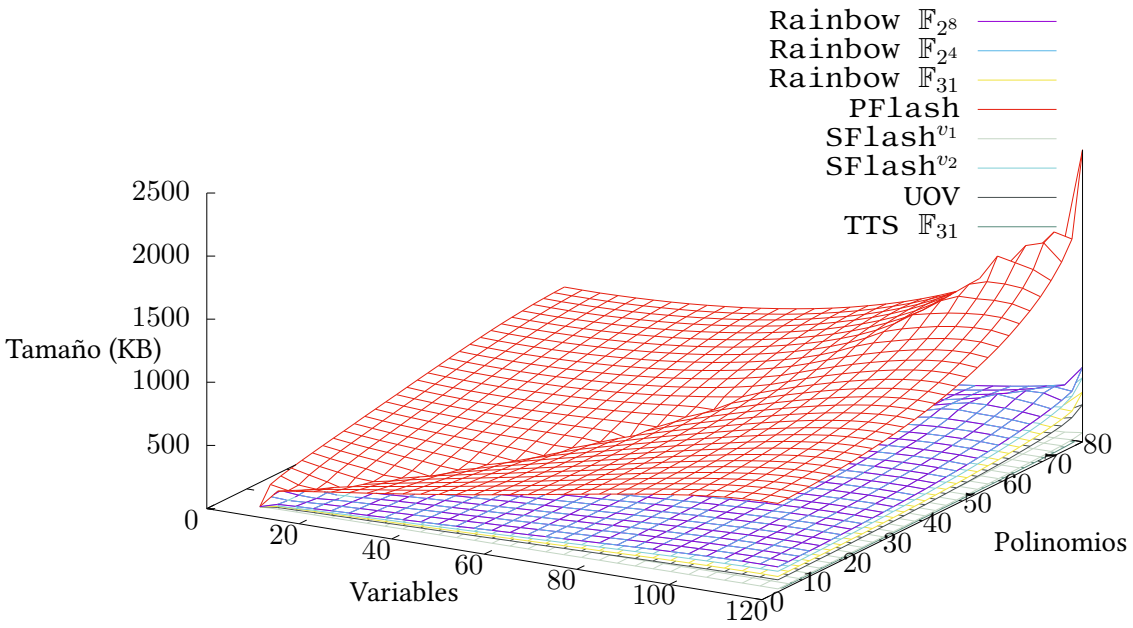

Figura 5-16: Tamaño de clave pública almacenada codificada en ASN.1.

PFlash continúa con las claves públicas más grandes, le sigue Rainbow sobre $\mathbb{F}_{2^4}$ y $\mathbb{F}_{2^8}$ que producen claves de igual tamaño, esto debido a que ambos utilizan un byte para representar un coeficiente, sin embargo 4-bits son suficientes para representar un elemento en $\mathbb{F}_{2^4}$, dejando 4-bits en 0. SFlash^{v_1} produce las claves más cortas en instancias pequeñas, generando claves con un tamaño de ≈47B en instancias de 8 variables y 4 polinomios. Mientras que TTS sobre $\mathbb{F}_{31}$ produce las claves más cortas en instancias más grandes, con claves de ≈33KB en instancias de 120 variables y 80 polinomios.

La tabla A.6 contiene los resultados del crecimiento de las claves variando el número de variables y polinomios. El crecimiento se puede observar en la gráfica 5-16.

Clave pública en Base64

Una vez codificada en ASN.1 se obtiene la clave pública como un archivo binario que es posible codificarlo en Base64 y así producir una salida en ASCII. La codificación en ASCII logra un crecimiento en el tamaño de la clave de ≈35 % de su tamaño original.

Al igual que la codificación en ASN.1 no produce algún cambio en el orden de cual esquema genera las claves de mayor tamaño. PFlash tiene un crecimiento de claves, donde con 120 variables

y 80 polinomios genera claves con un tamaño de ≈3.2MB. Rainbow sobre $\mathbb{F}_{2^4}$ y $\mathbb{F}_{2^8}$ generan claves con un tamaño de ≈798KB, Rainbow sobre $\mathbb{F}_{31}$ genera claves de ≈533KB, mientras que SFlash^{v_2} en las mismas instancias logra producir claves con un tamaño de ≈687KB, UOV de ≈393KB SFlash^{v_1} de ≈98KB y por último TTS produce las claves más cortas con un tamaño de ≈45KB.

En la gráfica 5-17 se puede observar el crecimiento del tamaño de las claves públicas de los distintos esquemas codificadas en Base64. La tabla A.7 contiene detalles del crecimiento de las claves públicas en Base64.

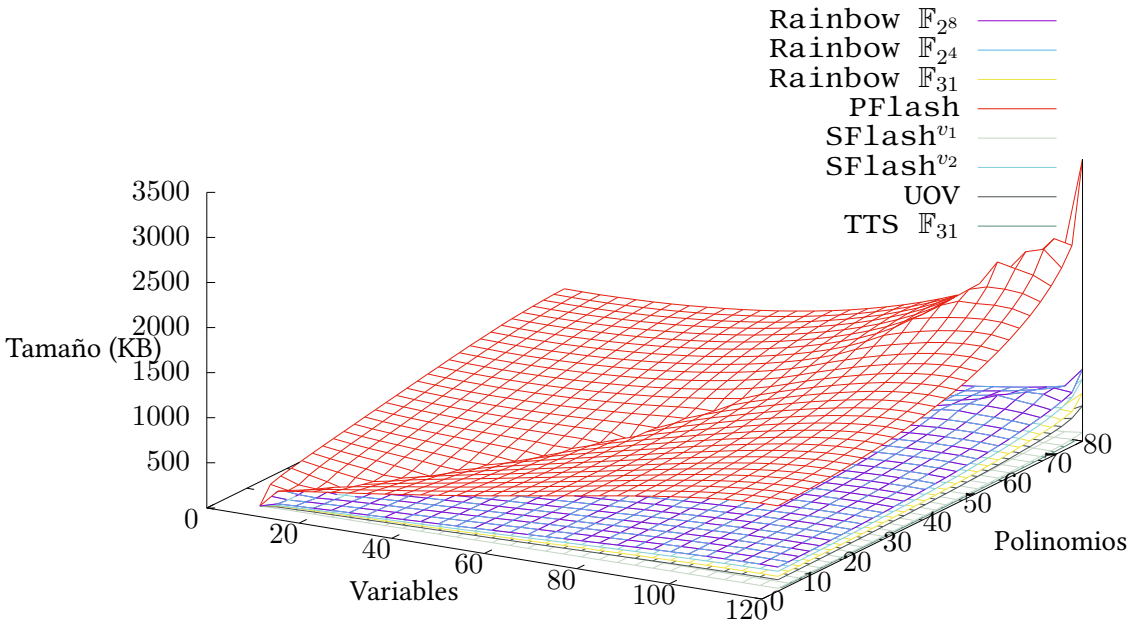

Figura 5-17: Tamaño de clave pública codificada en ASN.1 almacenada en ASCII en Base64.

Clave pública comprimida

Es posible aplicar un método de compresión sobre las claves públicas, ya sea una vez codificadas en ASN.1 o en Base64. Se realizaron algunas pruebas en las que se llevó a cabo la compresión con bzip2 sobre las claves públicas codificadas en ASN.1. En algunos esquemas no es posible lograr una reducción en su tamaño provocando un crecimiento menor, mientras que, en otros se logra una significante reducción.

PFlash sigue con las claves de mayor tamaño no siendo posible lograr una reducción con los

parámetros ingresados para realizar estas pruebas. Rainbow sobre $\mathbb{F}_{2^4}$ es quien logra la mayor reducción de todos los esquemas que participaron en las pruebas utilizando bzip2. De nuevo TTS logra generar las claves más cortas con un tamaño de ≈33KB en instancias de 120 variables y 80 polinomios. Siguiendo SFlash^{v_1} quien produce las claves más cortas sólo después de TTS con un tamaño de ≈55KB.

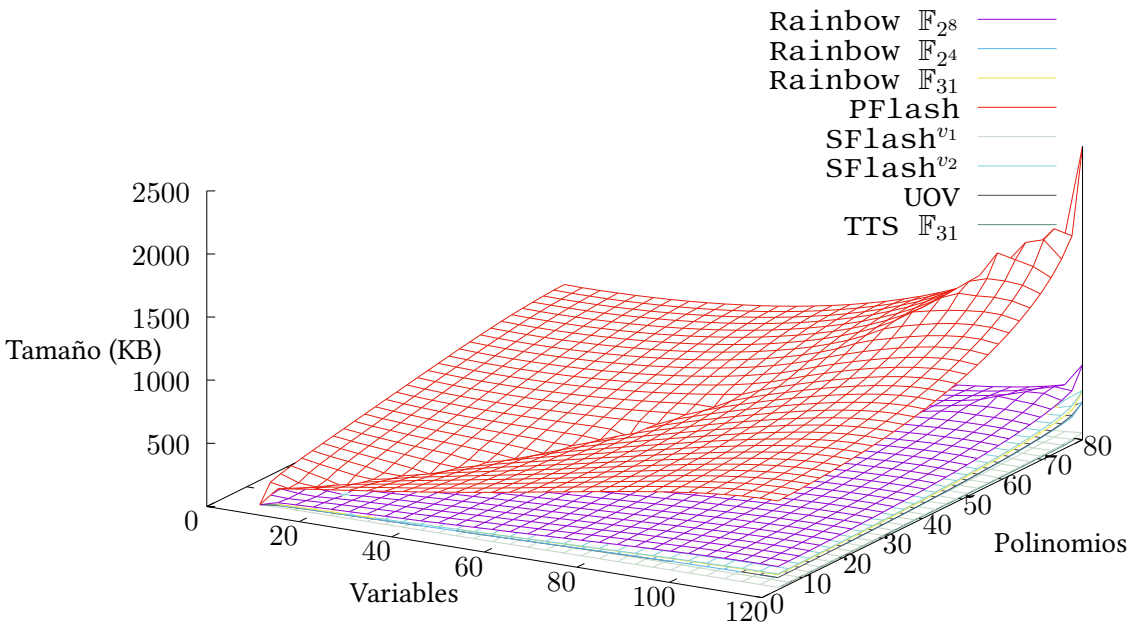

Figura 5-18: Tamaño de clave pública codificada en ASN.1 y comprimida utilizando bzip2.

Capítulo

6

Conclusiones y trabajo a futuro

"I mean the word proof not in the sense of the lawyers, who set two half proofs equal to a whole one, but in the sense of a mathematician, where half proof = 0, and it is demanded for proof that every doubt becomes impossible."

Carl Friedrich Gauss

6.1. Conclusiones

En este trabajo de tesis se propuso una plataforma de experimentación criptográfica para esquemas de criptografía sobre varias variables llamada `MQCrypto` , basando su sintaxis en `OpenSSL`. `MQCrypto` cuenta con la implementación de un total de diez esquemas criptográficos que basan su seguridad en el problema $\mathcal{MQ}$. Con esta plataforma es posible realizar pruebas de generación de claves con cada uno de los diez esquemas implementados, posteriormente es posible realizar procesos de firma y verificación.

Como se mostró en el capítulo 5, `MQCrypto` es capaz de realizar las operaciones de generación de claves, firma y verificación de una manera muy eficiente. Los tiempos de ejecución no se presentaron en la tesis, sin embargo, cuenta con implementaciones muy eficientes, la mayoría programadas en C y C++ por lo que los tiempos para las diferentes operaciones suelen tomar del orden de milisegundos.

De igual manera, en el capítulo 4 se presentó una manera de codificar las claves públicas y privadas

con el objetivo de lograr una reducción en su tamaño. Los tamaños de claves se reducen de manera eficiente tomando en cuenta su representación usual, sin embargo, si estas claves se comparan con algunas generadas por esquemas tales como RSA, ElGamal o ECC se puede observar que siguen siendo muy grandes.

El apéndice B, en la página 101 contiene un manual de referencia en donde se indica como es que puede ser descargado e instalado `MQCrypto`, así como también algunas instrucciones de cada una de las operaciones que pueden ser ejecutadas.

Todos las implementaciones que incorpora `MQCrypto` son de esquemas que pueden ser utilizados para realizar operaciones de firma y verificación. El código queda libre en un repositorio en Github, donde en un futuro se le pueden ir añadiendo más implementaciones de algunos otros esquemas que tal vez puedan incluir propiedades para cifrar y descifrar.

6.2. Trabajo a futuro

Algunas implementaciones generan claves más grandes de lo que deberían de producir, tal es el caso de `rainbow16242020` por ejemplo. Es posible realizar algunas modificaciones a estas implementaciones con el objetivo de lograr una reducción en el tamaño de claves que estas generan. En `rainbow16242020` se utiliza un byte para cada coeficiente, cuando para representar un elemento en $\mathbb{F}_{2^4} = \{a_3x^3 + a_2x^2 + a_1x + a_0 \mid a_i \in \mathbb{F}_2\}$ es necesario sólo de 4-bits, de esta manera los 4-bits restantes no se utilizan y se mantienen en ceros, produciendo una clave lo doble de grande de lo que debería de ser.

En un futuro, se puede añadir una autoridad certificadora, de manera que sea posible crear certificados digitales siguiendo un estándar como el X.509, que define el formato que deben de llevar los certificados de las claves públicas. De esta forma, `MQCrypto` podría ser utilizado para generar certificados digitales con claves de los esquemas incorporados en el sistema.

Apéndices

Apéndice

A

Crecimiento de claves

Como se mencionó en el capítulo 5, se realizaron algunas pruebas para analizar el crecimiento de las claves generadas por cada uno de los esquemas implementados en `MQCrypto`. Se calculó el tamaño de las parejas de claves que producía cada uno de los esquemas propuestos, y se obtuvieron sus tamaños en 4 etapas: cuando se generaban y se obtenían como un archivo binario, una vez codificadas en un objeto en ASN.1, codificadas en Base64 de modo que se obtuviera un archivo ASCII de salida, y utilizando un método de compresión sobre la codificación en ASN.1. Las tablas A.1, A.2, A.3 y A.4 contienen los resultados del crecimiento de la pareja de claves generadas por los esquemas en `MQCrypto`.

Las pruebas igual se realizaron con las claves públicas, debido a la importancia del tamaño de esta clave que se requiere transferir constantemente a las distintas entidades con quienes se desee establecer una comunicación segura, a diferencia de la clave privada que se busca mantenerse en secreto. Las pruebas tomaron valores desde 8 variables y 4 polinomios hasta 120 variables y 80 polinomios. Los resultados del tamaño claves que producen estos esquemas para la clave pública se muestran en las tablas A.5, A.6, A.7 y A.8.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{2^4}$)	Rainbow($\mathbb{F}_{2^8}$)	PFlash	SFlash^{v_1}	SFlash^{v_2}	UOV($\mathbb{F}_{2^4}$)	tts6440	3icp
8	4	2064 B	1968 B	1968 B	812 B	36 B	252 B	200 B	≈531 B	-
16	10	6608 B	6064 B	6064 B	6326 B	238 B	1666 B	1396 B	≈976 B	-
24	16	12112 B	12720 B	12720 B	21152 B	750 B	5250 B	4412 B	≈2947 B	-
32	20	37456 B	31248 B	31248 B	45740 B	1584 B	11088 B	9652 B	≈5084 B	-
40	26	60464 B	47680 B	47680 B	90694 B	3075 B	21525 B	18648 B	≈6592 B	-
48	32	82976 B	66768 B	66768 B	158272 B	5092 B	37044 B	31928 B	≈10739 B	-
56	36	157648 B	151568 B	151568 B	240428 B	7980 B	55860 B	49120 B	≈15507 B	-
64	42	214480 B	194608 B	194608 B	363222 B	11960 B	83720 B	73212 B	≈18448 B	-
72	48	264048 B	244160 B	244160 B	521952 B	17082 B	119574 B	104052 B	≈25859 B	-
80	52	415760 B	355376 B	355376 B	695468 B	22680 B	158760 B	140108 B	≈33795 B	-
88	58	519728 B	427392 B	427392 B	934502 B	30349 B	212443 B	186592 B	≈39040 B	-
96	64	608832 B	504480 B	504480 B	1222784 B	39576 B	277032 B	242288 B	≈50547 B	-
104	68	862608 B	794992 B	794992 B	1521452 B	49140 B	343980 B	304120 B	≈62419 B	-
112	74	1030096 B	915888 B	915888 B	1915126 B	61698 B	431886 B	380292 B	≈73184 B	-
120	80	1167760 B	1047248 B	1047248 B	2371360 B	76230 B	533610 B	468140 B	≈87171 B	-

Tabla A.1: Tamaño de par de claves en binario.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{2^4}$)	Rainbow($\mathbb{F}_{2^8}$)	PFlash	SFlash^{v_1}	SFlash^{v_2}	UOV($\mathbb{F}_{2^4}$)	tts6440	3icp
8	4	2102 B	2007 B	2009 B	843 B	74 B	290 B	236 B	≈563 B	-
16	10	6647 B	6103 B	6105 B	6359 B	276 B	1704 B	1432 B	≈1009 B	-
24	16	12151 B	12759 B	12761 B	21185 B	788 B	5288 B	4448 B	≈2980 B	-
32	20	37498 B	31290 B	31292 B	45773 B	1622 B	11126 B	9688 B	≈5117 B	-
40	26	60506 B	47722 B	47724 B	90729 B	3113 B	21563 B	18684 B	≈6627 B	-
48	32	83019 B	66811 B	66813 B	158307 B	5330 B	37082 B	31964 B	≈10774 B	-
56	36	157692 B	151613 B	151615 B	240463 B	8018 B	55898 B	49156 B	≈15542 B	-
64	42	214524 B	194653 B	194655 B	363257 B	11998 B	83758 B	73248 B	≈18483 B	-
72	48	264093 B	244205 B	244207 B	521987 B	17120 B	119612 B	104088 B	≈25894 B	-
80	52	415805 B	355421 B	355423 B	695503 B	22718 B	158798 B	140144 B	≈33830 B	-
88	58	519773 B	427437 B	427439 B	934537 B	30387 B	212481 B	186628 B	≈39075 B	-
96	64	608877 B	608877 B	608879 B	1222819 B	39614 B	277070 B	242324 B	≈50582 B	-
104	68	862653 B	795037 B	795039 B	1521488 B	49178 B	344018 B	304156 B	≈62455 B	-
112	74	1030141 B	915933 B	915935 B	1915162 B	61736 B	431924 B	380328 B	≈73221 B	-
120	80	1167805 B	1047293 B	1047295 B	2371396 B	76268 B	533648 B	468176 B	≈87208 B	-

Tabla A.2: Tamaño de par de claves codificado en ASN.1.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{24}$)	Rainbow($\mathbb{F}_{28}$)	PFlash	SFlash^{v_1}	SFlash^{v_2}	UOV($\mathbb{F}_{24}$)	tts6440	3icp
8	4	2843 B	2714 B	2718 B	1140 B	100 B	0392 B	316 B	≈763 B	-
16	10	8988 B	8254 B	8258 B	8598 B	373 B	2303 B	1912 B	≈1367 B	-
24	16	16430 B	17249 B	17249 B	28641 B	1065 B	7148 B	5932 B	≈4032 B	-
32	20	50695 B	42300 B	42304 B	76131 B	2192 B	15041 B	12920 B	≈6919 B	-
40	26	81797 B	64516 B	64520 B	122653 B	4208 B	29151 B	24912 B	≈8959 B	-
48	32	112230 B	90322 B	90326 B	214008 B	7205 B	50131 B	42620 B	≈14568 B	-
56	36	213177 B	204960 B	204964 B	325074 B	10839 B	75568 B	65544 B	≈21012 B	-
64	42	290005 B	263145 B	263149 B	491071 B	16220 B	113232 B	97664 B	≈24987 B	-
72	48	357015 B	330131 B	330135 B	705651 B	23144 B	161703 B	138784 B	≈35008 B	-
80	52	562109 B	480478 B	480482 B	940220 B	30712 B	214679 B	186860 B	≈45735 B	-
88	58	702658 B	577832 B	577836 B	1263359 B	41080 B	287253 B	248840 B	≈52824 B	-
96	64	823112 B	682044 B	682048 B	1653073 B	53554 B	374570 B	323100 B	≈68381 B	-
104	68	1166180 B	1074775 B	1074777 B	2056828 B	66483 B	465077 B	405544 B	≈84433 B	-
112	74	1392601 B	1238206 B	1238210 B	2589018 B	83460 B	583918 B	507104 B	≈98984 B	-
120	80	1578703 B	1415787 B	1415791 B	3205779 B	103106 B	721438 B	624236 B	≈117895 B	-

Tabla A.3: Tamaño de par de claves codificadas en ASN.1 almacenada en ASCII en Base64.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{2^4}$)	Rainbow($\mathbb{F}_{2^8}$)	PFlash	SFlash^{v1}	SFlash^{v2}	UOV($\mathbb{F}_{2^4}$)	tts6440	3icp
8	4	1094 B	628 B	855 B	1110 B	31 B	194 B	362 B	≈240 B	-
16	10	3468 B	2058 B	3829 B	6849 B	206 B	1282 B	1829 B	≈588 B	-
24	16	7119 B	5321 B	10352 B	21646 B	650 B	4042 B	4928 B	≈1488 B	-
32	20	18177 B	11787 B	22082 B	46320 B	1373 B	8537 B	10157 B	≈2706 B	-
40	26	30509 B	22304 B	41954 B	91511 B	2666 B	16574 B	19143 B	≈4375 B	-
48	32	45831 B	34131 B	67466 B	159418 B	4588 B	28523 B	32454 B	≈6857 B	-
56	36	79167 B	56568 B	107683 B	241939 B	6918 B	43012 B	49704 B	≈9907 B	-
64	42	110929 B	83437 B	159134 B	365316 B	10369 B	64464 B	73916 B	≈13407 B	-
72	48	144471 B	115064 B	222881 B	524888 B	14810 B	92071 B	104918 B	≈18684 B	-
80	52	213112 B	157040 B	301456 B	699120 B	19663 B	122245 B	141131 B	≈23336 B	-
88	58	271975 B	208962 B	401614 B	939680 B	26312 B	163581 B	187862 B	≈25709 B	-
96	64	333105 B	256966 B	507239 B	1228798 B	34312 B	213314 B	243845 B	≈31831 B	-
104	68	448121 B	337885 B	650228 B	1529688 B	42604 B	264864 B	305993 B	≈37597 B	-
112	74	543911 B	420783 B	811808 B	1925001 B	53492 B	332552 B	382554 B	≈42423 B	-
120	80	638445 B	506871 B	990818 B	2383406 B	66091 B	410879 B	470814 B	≈48022 B	-

Tabla A.4: Tamaño de par de claves codificadas en ASN.1 y comprimidas utilizando bzip2.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{24}$)	Rainbow($\mathbb{F}_{28}$)	PFlash	SFlash^{v_1}	SFlash^{v_2}	UOV($\mathbb{F}_{24}$)	tts6440	3icp
8	4	128 B	704 B	704 B	540 B	18 B	126 B	92 B	≈403 B	-
16	10	1344 B	2432 B	2432 B	5270 B	170 B	1190 B	768 B	≈720 B	-
24	16	3584 B	5184 B	5184 B	18800 B	600 B	4200 B	2600 B	≈1923 B	-
32	20	7680 B	17920 B	17920 B	41580 B	1320 B	9240 B	5612 B	≈3484 B	-
40	26	16128 B	27520 B	27520 B	84214 B	2665 B	18655 B	11196 B	≈3904 B	-
48	32	26624 B	39168 B	39168 B	148960 B	4704 B	32928 B	19600 B	≈6131 B	-
56	36	39744 B	79296 B	79296 B	227772 B	7182 B	50274 B	29756 B	≈8595 B	-
64	42	63360 B	102912 B	102912 B	346710 B	10920 B	76440 B	45048 B	≈9488 B	-
72	48	86784 B	129600 B	129600 B	501072 B	15768 B	110376 B	64824 B	≈12803 B	-
80	52	115648 B	212480 B	212480 B	669708 B	21060 B	147420 B	86348 B	≈16323 B	-
88	58	160320 B	256256 B	256256 B	903350 B	28391 B	198737 B	116148 B	≈17536 B	-
96	64	203776 B	304128 B	304128 B	1185728 B	37248 B	260736 B	152096 B	≈21875 B	-
104	68	252416 B	445120 B	445120 B	1477980 B	46410 B	324870 B	189212 B	≈26515 B	-
112	74	327104 B	515200 B	515200 B	1864726 B	58534 B	409738 B	238320 B	≈29408 B	-
120	80	394240 B	590400 B	590400 B	2313520 B	72600 B	508200 B	295240 B	≈33411 B	-

Tabla A.5: Tamaño de clave pública en binario.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{2^4}$)	Rainbow($\mathbb{F}_{2^8}$)	PFlash	SFlash^{v1}	SFlash^{v2}	UOV($\mathbb{F}_{2^4}$)	tts6440	3icp
8	4	128 B	739 B	741 B	567 B	47 B	155 B	118 B	$\approx$432 B	-
16	10	1344 B	2467 B	2469 B	5299 B	199 B	1219 B	794 B	$\approx$749 B	-
24	16	3584 B	5219 B	5221 B	18829 B	629 B	4229 B	2626 B	$\approx$1952 B	-
32	20	7680 B	17958 B	17960 B	41609 B	1349 B	9269 B	5638 B	$\approx$3513 B	-
40	26	16128 B	27558 B	27560 B	84245 B	2694 B	18684 B	11222 B	$\approx$3935 B	-
48	32	26624 B	39206 B	39208 B	148991 B	4733 B	32957 B	19626 B	$\approx$6162 B	-
56	36	39744 B	84952 B	84954 B	227803 B	7211 B	50303 B	29782 B	$\approx$8626 B	-
64	42	63360 B	102952 B	102954 B	346741 B	10949 B	76469 B	45074 B	$\approx$9519 B	-
72	48	86784 B	129640 B	129642 B	501103 B	15797 B	110405 B	64850 B	$\approx$12834 B	-
80	52	115648 B	212520 B	212522 B	669739 B	21089 B	147449 B	86374 B	$\approx$16354 B	-
88	58	160320 B	256296 B	256298 B	903381 B	28420 B	198766 B	116174 B	$\approx$17567 B	-
96	64	203776 B	304168 B	304170 B	1185759 B	37277 B	260765 B	152122 B	$\approx$21906 B	-
104	68	252416 B	445160 B	445162 B	1478012 B	46439 B	324899 B	189238 B	$\approx$26547 B	-
112	74	327104 B	515240 B	515242 B	1864758 B	58563 B	409767 B	238346 B	$\approx$29440 B	-
120	80	394240 B	590440 B	590444 B	2313552 B	72629 B	508229 B	295266 B	$\approx$33443 B	-

Tabla A.6: Tamaño de clave pública codificada en ASN.1.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{24}$)	Rainbow($\mathbb{F}_{28}$)	PFlash	SFlash^{v_1}	SFlash^{v_2}	UOV($\mathbb{F}_{24}$)	tts6440	3icp
8	4	219 B	1002 B	1006 B	767 B	63 B	209 B	160 B	≈585 B	-
16	10	1866 B	3338 B	3342 B	7167 B	269 B	1647 B	1060 B	≈1014 B	-
24	16	4896 B	7057 B	7061 B	25457 B	850 B	5717 B	3504 B	≈2641 B	-
32	20	10435 B	24277 B	24281 B	56251 B	1823 B	12530 B	7520 B	≈4750 B	-
40	26	21856 B	37255 B	37259 B	113889 B	3642 B	25258 B	14964 B	≈5321 B	-
48	32	36046 B	53003 B	53007 B	201416 B	6398 B	44554 B	26168 B	≈8331 B	-
56	36	53781 B	107254 B	107258 B	307959 B	9748 B	68004 B	39712 B	≈11664 B	-
64	42	85707 B	139179 B	139183 B	468746 B	14801 B	103378 B	60100 B	≈12869 B	-
72	48	117376 B	175257 B	175261 B	677420 B	21355 B	149256 B	86468 B	≈17350 B	-
80	52	156395 B	287296 B	287300 B	905391 B	28510 B	199336 B	115168 B	≈22111 B	-
88	58	216786 B	346475 B	346479 B	1221238 B	38420 B	268711 B	154900 B	≈23750 B	-
96	64	275531 B	411193 B	411197 B	1602971 B	50394 B	352528 B	202832 B	≈29614 B	-
104	68	341284 B	601792 B	601796 B	1998055 B	62780 B	439230 B	252320 B	≈35888 B	-
112	74	442251 B	696530 B	696534 B	2520877 B	79171 B	553964 B	317796 B	≈39802 B	-
120	80	533010 B	798191 B	798194 B	3127580 B	98187 B	687074 B	393688 B	≈45212 B	-

Tabla A.7: Tamaño de clave pública codificada en ASN.1 almacenada en ASCII en Base64.

n	m	Rainbow($\mathbb{F}_{31}$)	Rainbow($\mathbb{F}_{2^4}$)	Rainbow($\mathbb{F}_{2^8}$)	PFlash	SFlash^{v1}	SFlash^{v2}	UOV($\mathbb{F}_{2^4}$)	tts6440	3icp
8	4	232 B	234 B	376 B	746 B	13 B	97 B	192 B	≈145 B	-
16	10	1474 B	1035 B	2130 B	5816 B	130 B	916 B	1032 B	≈439 B	-
24	16	4049 B	2768 B	5717 B	19301 B	462 B	3234 B	3083 B	≈1024 B	-
32	20	8164 B	6401 B	12367 B	42133 B	1016 B	7114 B	6139 B	≈2002 B	-
40	26	15596 B	12425 B	23833 B	84997 B	2052 B	14364 B	11699 B	≈3009 B	-
48	32	26865 B	20081 B	39744 B	150048 B	3622 B	25354 B	20093 B	≈4754 B	-
56	36	39754 B	32306 B	61993 B	229275 B	5530 B	38710 B	30268 B	≈6872 B	-
64	42	60434 B	48407 B	93275 B	348756 B	8408 B	58858 B	45612 B	≈9203 B	-
72	48	85513 B	66077 B	130602 B	503820 B	12141 B	84989 B	65492 B	≈12891 B	-
80	52	113509 B	92033 B	177714 B	673177 B	16216 B	113513 B	87156 B	≈16479 B	-
88	58	152466 B	123237 B	238444 B	907960 B	21861 B	153027 B	117093 B	≈17570 B	-
96	64	198843 B	154677 B	305939 B	1192066 B	28680 B	200766 B	153195 B	≈22020 B	-
104	68	245848 B	199557 B	387068 B	1485953 B	35735 B	250149 B	190474 B	≈26412 B	-
112	74	311326 B	250701 B	486796 B	1874535 B	45071 B	315498 B	239788 B	≈29498 B	-
120	80	383019 B	300143 B	593534 B	2325223 B	55902 B	391314 B	297032 B	≈33192 B	-

Tabla A.8: Tamaño de clave pública en binario ASN.1 y comprimida utilizando bzip2.

Apéndice

B

Manual de referencia

El código de MQCrypto se encuentra en un repositorio en Github desde donde puede ser descargado para realizar algunas pruebas:

`https://github.com/eliverperez/MQCrypto`

Para instalar MQCrypto es necesario contar con las bibliotecas OpenSSL, `libsodium` y `libbzip2` previamente instaladas. De ahí en adelante la instalación se puede realizar de una manera muy sencilla utilizando el archivo `Makefile`.

```
./libb64-sh
./configure
make
sudo make install
```

donde se empieza por realizar la compilación de la biblioteca `libb64`, y se continúa con la instalación tradicional en sistemas Unix/Linux, empezando por realizar la configuración, en la que se verifica que se cuente con el resto de las bibliotecas necesarias, posteriormente realizando la compilación de MQCrypto para finalmente moverlo a sus carpetas correspondientes para su correcta instalación.

Una vez instalado, MQCrypto puede ser utilizado de dos maneras: solamente ejecutando `mqcrypto` en la terminal, lo cual abrirá una terminal del software donde puede ser utilizado tal como con OpenSSL, la segunda manera es al igual mandarlo llamar mediante su nombre e ingresando los parámetros para la operación que se requiera ejecutar como se muestra en la sección 5.4.

MQCrypto incorpora los siguientes esquemas que pueden ser utilizados para generar un par de claves:

- rainbow5640
- rainbow6440
- rainbow16242020
- rainbow256181212
- pflash
- sflashv1
- sflashv2
- uov
- 3icp
- tts6440

Para obtener una lista de los esquemas que pueden ser utilizados, así como también algunos detalles, tales como sobre que en campo trabajan, el número de variables y el número de polinomios se puede utilizar la etiqueta -list_schemes:

```
mqcrypto -list_schemes
```

Para generar un par de claves basta ejecutar la función genKeys de MQCrypto de la siguiente manera:

```
mqcrypto genKeys -scheme pflash -out Keypair.pem -aes_192_cbc -zip
```

lo cual producirá un par de claves de pflash cifradas utilizando aes_192_cbc y comprimidas utilizando bzip2. Una vez ejecutada esta línea se pide la contraseña utilizada para cifrar el par de claves utilizando AES.
Para consultar el menú de ayuda donde se describe la sintaxis para la generación de las claves, así como también cada uno de los esquemas que pueden utilizarse, se puede ejecutar:

```
mqcrypto genKeys help
```

Cuando se produce la generación de las claves, se obtiene un archivo cifrado con el contenido de la clave pública y la clave privada. Si se desea compartir la clave pública, ésta puede ser extraída de la pareja de claves utilizando la función `keys` de la siguiente manera:

```
mqcrypto keys Keypair.pem -pubout public-key.pem
```

en la que se obtiene un archivo `Keypair.pem` que contiene la pareja de claves y extrae la clave pública en el archivo `public-key.pem`. Como la pareja de claves se encuentra cifrada se requiere ingresar la contraseña. De igual manera la clave privada puede ser extraída, sin embargo produce un archivo con la clave privada descifrada por lo que queda vulnerable y por lo tanto se recomiendo no hacerlo.
Para mayor detalle en la extracción de las claves públicas y privadas, un menú de ayuda de la función `keys` puede ser mandado llamar ejecutando:

```
mqcrypto keys help
```

Ya que se cuenta con un par de claves de uno de los diez esquemas en `MQCrypto`, es posible realizar el proceso de firmar un documento. Para generar una firma digital se utiliza la función `sign` de la siguiente manera:

```
mqcrypto sign private-key.pem -in file.txt -out signed.txt -sha256
```

donde recibe la clave privada del archivo `private-key.pem` y firma el contenido de `file.txt` utilizando la función picadillo SHA-256 y almacenando la firma en `signed.txt`. De igual manera se puede ingresar como parámetro de entrada la pareja de claves. La función devuelve como salida un mensaje que indica el estatus de la generación de la firma.
Un menú de ayuda para utilizar la función `sign` en `MQCrypto`, donde se muestran detalles de cada una de las etiquetas que pueden ser utilizadas y su función, puede ser desplegado utilizando:

```
mqcrypto sign help
```

Al contar con una firma digital, es posible realizar el proceso de verificación si se cuenta con la clave pública. El proceso de verificación puede ser ejecutado de la siguiente manera:

```
mqcrypto verify public-key.pem -in file.txt -signature signed.txt
```

en el que utilizando la clave pública almacenada en el archivo `public-key.pem` procede a realizar el proceso de verificación de la firma del documento `file.txt` almacenada en `signed.txt`.

De igual manera, puede utilizarse la pareja de claves para el proceso de verificación de la cual se utilizará únicamente la clave pública. El proceso de verificación devuelve como salida un único mensaje del estatus de la firma: `Verify: OK` en caso de que la firma haya sido verificada con éxito, de lo contrario `Verify: Fail` si la firma fue rechazada.
Para consultar todos los parámetros que se pueden recibir en la verificación de una firma, es posible consultar un menú de ayuda para la función `verify` mediante:

```
mqcrypto verify help
```

Bibliografía

[1] Mehdi-Laurent Akkar, Nicolas T. Courtois, Romain Duteuil, and Louis Goubin. *A Fast and Secure Implementation of Sflash*, pages 267–278. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[2] Côme Berbain, Olivier Billet, and Henri Gilbert. *Efficient Implementations of Multivariate Quadratic Systems*, pages 174–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post Quantum Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[4] M. Brickenstein and A. Dreyer. *POLYBORI: A Gröbner basis framework for Boolean polynomials*. J. Symb. Comput. 44, No. 9, 1326-1345 (2009).

[5] Bruno Buchberger. Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41(3):475 – 511, 2006.

[6] Anna Inn-Tung Chen, Chia-Hsin Owen Chen, Ming-Shing Chen, Chen-Mou Cheng, and Bo-Yin Yang. *Practical-Sized Instances of Multivariate PKCs: Rainbow, TTS, and ℓIC-Derivatives*, pages 95–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[7] Anna Inn-Tung Chen, Ming-Shing Chen, Tien-Ren Chen, Chen-Mou Cheng, Jintai Ding, Eric Li-Hsiang Kuo, Frost Yu-Shuang Lee, and Bo-Yin Yang. *SSE Implementation of Multivariate PKCs on Modern x86 CPUs*, pages 33–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[8] N. Courtois, L. Goubin, and J. Patarin. *Quartz, an asymmetric signature scheme for short signatures on PC Primitive specification and supporting documentation*, 2001. Available at: http://www.minrank.org/quartz-b.pdf, Accessed on: Sept. 24, 2016.

[9] N. Courtois, L. Goubin, and J. Patarin. *SFLASH, a fast asymmetric signature scheme for low-cost smartcards - Primitive specification and supporting documentation*, 2001. Available at: http://www.minrank.org/sflash-b-v2.pdf, Accessed on: Sept. 24, 2016.

[10] Nicolas Courtois. The security of hidden field equations (hfe). In *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, CT-RSA 2001, pages 266–281, London, UK, UK, 2001. Springer-Verlag.

[11] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[12] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[13] J. Ding, J. E. Gower, D. S. Schmidt, Penpan Youngkhong, and Yupaporn Kemprasit. *Multivariate Public Key Cryptosystems*. Springer, USA, 1st edition, 2006.

[14] Jintai Ding and Dieter Schmidt. *Rainbow, a New Multivariable Polynomial Signature Scheme*, pages 164–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[15] Jintai Ding, Christopher Wolf, and Bo-Yin Yang. ℓ-invertible cycles for multivariate quadratic (mq) public key cryptography. In *Proceedings of the 10th International Conference on Practice and Theory in Public-key Cryptography*, PKC'07, pages 266–281, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] Olivier Dubuisson and Philippe Fouquart. *ASN.1: Communication Between Heterogeneous Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[17] Taher ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, pages 10–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[18] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1):61 – 88, 1999.

[19] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.

[20] Jean-Charles Faugère. Algebraic cryptanalysis of HFE using Gröbner bases. Research Report RR-4738, INRIA, 2003.

[21] Jean-Charles Faugère and Antoine Joux. *Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases*, pages 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[22] Henri Gilbert and Marine Minier. *Cryptanalysis of SFLASH*, pages 288–298. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[23] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[24] Edgar González, Guillermo Morales-Luna, and Feliú D. Sagols. Zero knowledge authentication protocols with algebraic geometry techniques. Cryptology ePrint Archive, Report 2016/737, 2016. `http://eprint.iacr.org/2016/737`.

[25] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[26] R. Hartshorne. *Algebraic Geometry*. Encyclopaedia of mathematical sciences. Springer, 1977.

[27] J. Jonsson and B. Kaliski. Public-key cryptography standards (pkcs) #1: Rsa cryptography specifications version 2.1, 2003.

[28] Burton S. Kaliski Jr. An overview of the PKCS standards. Technical report, RSA Laboratories, November 1993.

[29] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings*, volume 1592 of LNCS, pages 206–222. Springer Berlin Heidelberg, 1999.

[30] Aviad Kipnis, Jacques Patarin, and Louis Goubin. *Unbalanced Oil and Vinegar Signature Schemes*, pages 206–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[31] Aviad Kipnis and Adi Shamir. *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization*, pages 19–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[32] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[33] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[34] S. Lang. *Algebra*. Graduate Texts in Mathematics. Springer New York, 2005.

[35] T. Matsumoto and H. Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In D. Barstow, W. Brauer, P. B. Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and C. G. Günther, editors, *Workshop on the Theory and Application of Cryptographic Techniques - Advances in Criptology - EUROCRYPT'88*, volume 330 of LNCS, pages 419–453, Davos, Switzerland, May 25âĂŞ27 1988. Springer-Verlag.

[36] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[37] Louis Goubin Nicolas T. Courtois and Jacques Patarin. Sflashv3, a fast asymmetric signature scheme. Cryptology ePrint Archive, Report 2003/211, 2003. `http://eprint.iacr.org/2003/211`.

[38] J. Patarin. Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt'88. In D. Coppersmith, editor, *15th Annual International Cryptology Conference - Advances in Cryptology - CRYPTO'95*, volume 963 of LNCS, pages 248–261, Santa Barbara, California, USA, August 27âĂŞ31 1995. Springer Berlin Heidelberg.

[39] Jacques Patarin. *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms*, pages 33–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[40] Jacques Patarin. The oil and vinegar signature scheme. In *Dagstuhl Workshop on Cryptography*, September,1997.

[41] Jacques Patarin, Nicolas Courtois, and Louis Goubin. *FLASH, a Fast Multivariate Signature Algorithm*, pages 298–307. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[42] A. Petzoldt, M. Chen, B. Yang, C. Tao, and J. Ding. Design principles for HFEv- based multivariate signature schemes. In T. Iwata and J. H. Cheon, editors, *In Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security - Advances in Cryptology - ASIACRYPT 2015*, volume 9452 of LNCS, pages 311–334, Auckland, New Zealand, November 29 - December 3 2015. Springer Berlin Heidelberg.

[43] Arthur R. Pope. Encoding ccitt x.409 presentation transfer syntax. *SIGCOMM Comput. Commun. Rev.*, 14(4):4–10, October 1984.

[44] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[45] Francisco Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and Cetin Kaya Koc. *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[46] Adi Shamir. *Efficient Signature Schemes Based on Birational Permutations*, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[47] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

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

[49] J. von zur Gathen. *CryptoSchool*. Springer Berlin Heidelberg, 2015.

[50] Christian Wenzel-Benner and Jens Gräf. *XBX: eXternal Benchmarking eXtension for the SUPERCOP Crypto Benchmarking Framework*, pages 294–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.