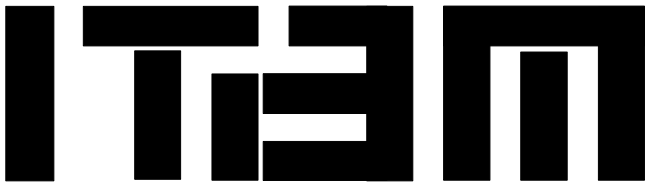
INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



CRIPTOGRAFÍA CON GRUPOS DE TRENZAS

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS

P R E S E N T A

CID ARMANDO ISAAC REYES BUSTOS

MÉXICO, D.F. **2011**

Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimoniales de la obra titulada **“CRIPTOGRAFÍA CON GRUPOS DE TRENZAS”** , otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Bailléres Jr., autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación

Cid Armando Isaac Reyes Bustos

_____

Fecha

_____

Firma

A la memoria de mis abuelos.

Agradecimientos

- A mi familia, en especial mi madre y mi hermano, quienes siempre han depositado su confianza en mí y me han brindado todo el apoyo posible.
- A las directoras de esta tesis, Marcela González Peláez y Begoña Albizuri Romero, por guiar este trabajo desde su concepción hasta su terminación, así como por las amenas sesiones de seminario que tuvimos durante la duración del mismo.
- A los sinodales de esta tesis, María del Carmen López Laiseca, César Luis García García, Javier Alfaro Pastor, Silvia del Carmen Guardati Buemo y Fernando Esponda Darlington, por aceptar una petición egoísta y revisar con prontitud este trabajo, haciendo indicaciones y comentarios que permitieron mejorarlo.
- A mis amigos y compañeros, Alejandro, Luis Alfonzo, Alfredo, Juan José, Bruno, y muchos otros, que me acompañaron a lo largo de la carrera.

Índice general

Capítulo 1

Introducción

1.1. Motivación

Desde que existe información privada ha existido la necesidad de transmitirla a otras personas de manera segura. Para responder a esta necesidad se desarrolla la criptografía, en la cual se produce una versión cifrada de un texto original que debe poderse descifrar sólo por el destinatario y no por terceros. Para lograr esto, el cifrado se hace utilizando una llave, un secreto entre el remitente y el destinatario, que permite cifrar y descifrar el mensaje.

Los sistemas simétricos usan una llave, o una transformación de la misma, para el cifrado y el descifrado. De esta forma, el conocimiento de la llave por parte de un tercero compromete la seguridad del sistema. El criptosistema AES (Advanced Encryption Standard, descrito en [19]), utilizado por el gobierno de los Estados Unidos de América, es un ejemplo de criptosistema simétrico.

Para utilizar de manera apropiada un sistema simétrico es necesario entonces que las partes puedan establecer un valor para la llave de manera segura. En 1976, en [6] Martin Hellman y Whitfield Diffie proponen un protocolo para el intercambio de llaves entre dos partes a través de un canal público de manera segura. Este protocolo es conocido como el protocolo de intercambio de llaves de Diffie-Hellman, y marca el inicio de la criptografía de llave pública.

En la criptografía de llave pública, tanto el remitente como el destinatario cuentan con llaves privadas y públicas. Para cifrar un mensaje se utiliza la llave pública del

destinatario y solamente él puede descifrar el mensaje, usando su llave privada. De esta forma, el proceso de cifrado y descifrado se vuelve independiente, es decir, conocer la llave para encriptar un mensaje no permite al usuario descifrar el mensaje.

Entre los sistemas de llave pública más conocidos está el RSA (por Rivest, Shamir y Adleman en [20]) que utiliza la dificultad de la factorización en números primos de números enteros grandes y el sistema ElGamal, que es un sistema basado en el protocolo de Diffie-Hellman y debe su seguridad al problema del logaritmo discreto en un grupo cíclico. Es un principio establecido por la criptografía y el criptoanálisis que la seguridad de un sistema no debe depender del conocimiento del atacante del sistema usado, es decir, el sistema debe ser seguro aunque el atacante conozca todos los parámetros usados con excepción de las llaves utilizadas para cifrar. Se han desarrollado métodos para comprometer la seguridad de los sistemas usados en la actualidad; sin embargo, no se han encontrado ataques que pongan en duda la seguridad de estos sistemas con los parámetros recomendados.

La criptografía forma parte de lo que hoy se conoce como seguridad informática, la protección de la información de robo, daño o destrucción. En particular, utilizando la criptografía se puede hacer que solamente las personas que tienen permiso para utilizar la información estén autorizadas para hacerlo y esto ha permitido el desarrollo de sistemas de autenticación de usuarios, almacenamiento remoto de información, sistemas de comercio electrónico, entre otros. La seguridad de estos sistemas depende directamente de la seguridad de los sistemas criptográficos que los componen, por lo que es necesario estudiar cuidadosamente la teoría que los respalda y la implementación particular de los mismos para asegurar que no existen riesgos en ninguno de ellos.

El reciente estudio de la computación cuántica y la posible implementación de computadoras cuánticas pone en riesgo muchos de los sistemas actuales, ya que existen algoritmos cuánticos que pueden determinar fácilmente las llaves privadas a partir de la información pública. En caso de que esto ocurriera, grandes cantidades de información privada podrían verse comprometidas así como la integridad de muchos de los grandes sistemas usados en prácticamente todas las áreas (cualquier sistema que utilice la identificación de usuarios por medio de contraseñas podría verse afectado).

En vista de lo anterior, en los últimos años se han investigado distintas áreas de las matemáticas para desarrollar nuevas alternativas a los sistemas actuales. Una de las

áreas que ha recibido gran atención es la de las estructuras algebraicas no conmutativas y, en especial, los grupos de trenzas se han estudiado con gran detalle. Se han propuesto diversos sistemas criptográficos basados en grupos de trenzas y para algunos de ellos se han encontrado ataques eficientes que comprometen la seguridad de los mismos. La investigación en esta área sigue abierta y constantemente se proponen nuevos criptosistemas en diferentes estructuras algebraicas.

En este trabajo se estudia de manera general a los sistemas criptográficos basados en teoría de grupos no abelianos y de manera particular a los basados en grupos de trenzas, así como algunos de los ataques conocidos.

Así mismo se realizó la implementación computacional de los sistemas basados en grupos de trenzas; dicha implementación tiene como propósito mostrar el correcto funcionamiento de los sistemas descritos en el trabajo así como establecer las bases para la experimentación y el desarrollo de nuevos sistemas en el grupo de trenzas. Adicionalmente, la implementación también incluye uno de los principales ataques a estos sistemas para probar la seguridad de los mismos y así intentar establecer si éstos son seguros para su uso en aplicaciones reales.

1.2. Objetivos

El objetivo principal de este trabajo es el estudio y desarrollo de sistemas criptográficos basados en grupos de trenzas, así como de una implementación que pueda ser utilizada como material de estudio y pruebas para estudiantes e investigadores. De manera específica se tienen los siguientes objetivos:

- Investigar y desarrollar los antecedentes de la teoría de grupos necesarios para la comprensión y desarrollo de los algoritmos criptográficos del presente trabajo.
- Desarrollar de manera completa los sistemas criptográficos basados en grupos de trenzas, sus propiedades, particularidades y consideraciones de seguridad.
- Estudiar los principales ataques conocidos a los sistemas criptográficos estudiados.
- Analizar y desarrollar estructuras de datos y algoritmos para la implementación de los sistemas criptográficos estudiados.

- Implementar los sistemas criptográficos, incluyendo los detalles del diseño y las consideraciones técnicas.
- Documentar el desarrollo, así como las pruebas y resultados de la implementación.

1.3. Alcance

Como alcance del proyecto se define el estudio y desarrollo completo de algunos de los sistemas criptográficos basados en los grupos de trenzas, así como el análisis de la seguridad de los mismos.
Se hace también el diseño e implementación de un programa funcional de los sistemas criptográficos mencionados. Para lo anterior se cuenta con extensa bibliografía en el estudio de grupos de trenzas y sus aplicaciones criptográficas.

1.4. Organización del documento

El documento está integrado por seis capítulos, que se listan a continuación:

1 Introducción. Se presenta la motivación, objetivo y alcance del proyecto y una breve descripción de los temas tratados en el mismo.

2 Antecedentes. Se presentan los antecedentes necesarios para la comprensión de los sistemas criptográficos desarrollados en los siguientes capítulos.

3 Grupos de trenzas. Se desarrolla a detalle la teoría básica y las particularidades de los grupos de trenzas.

4 Criptografía. Se describen dos sistemas criptográficos basados en grupos de trenzas y se analiza uno de los ataques conocidos.

5 Diseño e Implementación. Se muestra el diseño del sistema que implementa los sistemas criptográficos estudiados, se decriben las características de la implementación del sistema y se presentan las pruebas y los resultados obtenidos.

6 Conclusiones. Se establecen las observaciones y conclusiones obtenidas del desarrollo del proyecto y posibles líneas futuras.

Capítulo 2

Antecedentes

En este capítulo se estudiarán los antecedentes necesarios de la teoría de grupos y criptografía. Se desarrollará el material necesario para la comprensión del presente trabajo. La teoría de grupos es un área de las matemáticas que tiene varios orígenes, pero se puede decir que inicia con los estudios de Galois para la solución de ecuaciones algebraicas. Desarrollos posteriores de matemáticos como Cayley y Hamilton extendieron el uso de grupos a otras áreas y actualmente su uso se extiende a casi todas las ramas de las matemáticas.
También resulta ser muy antiguo el origen de la criptografía, que se remonta a la época de los grandes imperios. El célebre cifrado de Julio César es uno de los ejemplos más representativos de la criptografía clásica. Los desarrollos criptográficos que utilizan, de manera importante, las herramientas de las matemáticas modernas se dan en el siglo XX, como resultado del uso de las computadoras.

2.1. Antecedentes de la teoría de grupos

Aquí se exponen las definiciones y resultados básicos de la teoría de grupos que permiten el desarrollo de los sistemas criptográficos que se realizan en esta tesis. Para una exposición completa de la teoría de grupos se pueden consultar [12, 15].
Se incluyen las siguientes definiciones a manera de completitud.[1]

[1]En los casos en los que no se presente confusión se omitirá el símbolo de las operaciones y se denotará por medio de la yuxtaposición.

Definición (Semigrupo)**.** Un semigrupo $\mathcal{S}$ es un conjunto no vacío con una operación binaria $\cdot : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ que es asociativa, es decir, $a(bc) = (ab)c \quad \forall a, b, c \in \mathcal{S}$.

Definición (Monoide)**.** Un monoide $\mathcal{M}$ es un semigrupo que contiene un elemento neutro, llamado también idéntico, es decir, $\exists e \in \mathcal{M}$ tal que $ea = a = ae \quad \forall a \in \mathcal{M}$.

Definición (Grupo)**.** Un grupo $\mathcal{G}$ es un monoide en donde todo elemento tiene un inverso, es decir, $\forall a \in \mathcal{G} \quad \exists a^{-1} \in \mathcal{G}$ tal que $aa^{-1} = e = a^{-1}a$. Si $ab = ba$ para todo $a, b \in G$ se dice que G es un grupo es abeliano.

Las estructuras anteriores se suelen denotar como la pareja ordenada formada por el conjunto y la operación binaria, es decir, si $\mathcal{G}$ es un grupo con la operación $\cdot$, entonces se escribe $(\mathcal{G}, \cdot)$ es un grupo.
Para poder definir los grupos que se utilizan en los sistemas criptográficos que se estudiarán en el presente trabajo es necesario el estudio de los grupos libres. La exposición presente está basada en los textos [13, 21].
Sea $\mathcal{X}$ un conjunto no vacío, entonces se define al conjunto $\mathcal{X}^{-1}$ como $\mathcal{X}^{-1} = \{x^{-1} \mid x \in \mathcal{X}\}$ y sea 1 un elemento que no pertenece a $\mathcal{X} \cup \mathcal{X}^{-1}$. La notación x^{-1} es puramente formal.

Definición (Palabra)**.** Una palabra en $\mathcal{X}$ es una sucesión $(a_i)_{i \in \mathbb{N}}$ con
$a_i \in \mathcal{X} \cup \mathcal{X}^{-1} \cup \{1\}$, y $\exists n \in \mathbb{N}$ tal que $a_k = 1 : \forall k \geq n$.

La palabra $(1, 1, \ldots)$ se conoce como la *palabra vacía* y se denota con el símbolo 1.
Si una palabra no incluye términos consecutivos con exponentes con signo contrario, entonces se dice que es una palabra reducida, es decir,

Definición (Palabra reducida)**.** Una palabra $(a_1, a_2, \ldots)$ en $\mathcal{X}$ es una palabra reducida si $\forall i \in \mathbb{N}$ y $x \in \mathcal{X}$ se tiene que $a_i = x \Rightarrow a_{i+1} \neq x^{-1}$ y $a_i = x^{-1} \Rightarrow a_{i+1} \neq x$.

Una palabra reducida no vacía $(x_1^{\lambda_1}, x_2^{\lambda_2}, \ldots, x_n^{\lambda_n} \ldots)$ en $\mathcal{X}$ con $\lambda_i \in \{-1, 1\}$ para $1 \leq i \leq n$ y $x_k = 1$, $\forall k > n$ se denotará como $x_1^{\lambda_1} x_2^{\lambda_2} \ldots x_n^{\lambda_n}$, donde $x_i^1 = x_i$.
Se denota con $\mathcal{F}(\mathcal{X})$ al conjunto de palabras reducidas en $\mathcal{X}$. La función $i : \mathcal{X} \to \mathcal{F}(\mathcal{X})$ definida como $x \mapsto x^1$ es inyectiva y actúa como función de inclusión, por lo que se puede pensar a $\mathcal{X}$ como un subconjunto de $\mathcal{F}(\mathcal{X})$ si se considera su imagen bajo dicha función.

Se define la operación binaria $\cdot : \mathcal{F}(\mathcal{X}) \times \mathcal{F}(\mathcal{X}) \to \mathcal{F}(\mathcal{X})$ como la yuxtaposición(o concatenación) y reducción de la palabra resultante. La palabra vacía 1 actuará como elemento idéntico para dicha operación.
Si $x_1^{\lambda_1}x_2^{\lambda_2}\dots x_m^{\lambda_m}$ y $y_1^{\delta_1}y_2^{\delta_2}\dots y_n^{\delta_n} \in \mathcal{F}(\mathcal{X})$ con $m \leq n$ y $0 \leq k \leq m$ es el mayor entero tal que $x_{m-j}^{\lambda_{(m-j)}} = y_{j+1}^{-\delta_{(j+1)}}$ para $0 \leq j \leq k-1$, se puede entonces definir la operación de manera precisa como sigue:

$$(x_1^{\lambda_1}\dots x_m^{\lambda_m})\cdot(y_1^{\delta_1}\dots y_n^{\delta_n}) = \begin{cases} x_1^{\lambda_1}\dots x_{m-k}^{\lambda_{(m-k)}} y_{k+1}^{\delta_{(k+1)}}\dots y_n^{\delta_n} & \text{si } k < m, \\ y_{m+1}^{\delta_{(m+1)}}\dots y_n^{\delta_n} & \text{si } k = m < n \\ 1 & \text{si } k = m = n \end{cases}$$

En el caso $k = 0$ se tiene que

$$(x_1^{\lambda_1}\dots x_m^{\lambda_m})\cdot(y_1^{\delta_1}\dots y_n^{\delta_n}) = x_1^{\lambda_1}\dots x_m^{\lambda_m}y_1^{\delta_1}\dots y_n^{\delta_n}$$

La operación se define de manera análoga cuando $m > n$.
Como ejemplo se considera $\mathcal{X} = \{a, b, c\}$ el conjunto de las primeras tres letras del alfabeto y los elementos $a^1b^1c^{-1}$ y c^1b^1 del conjunto de palabras reducidas $\mathcal{F}(\mathcal{X})$. Utilizando la notación de la definición se tiene que $m = 3$ y $n = 2$. Entonces como $x_{3-0}^{\lambda_{(3-0)}} = c^{-1} = c^{-(1)} = y_{0+1}^{-\delta_{(0+1)}}$ pero $x_{3-1}^{\lambda_{(3-1)}} = b^1 \neq b^{-(1)} = y_{1+1}^{-\delta_{(1+1)}}$ se tiene que $k = 1$ y el resultado de la operación es el siguiente:

$$(a^1b^1c^{-1})(c^1b^1) = a^1b^1b^1$$

Con la operación anterior resulta que $\mathcal{F}(\mathcal{X})$ es un grupo. Pero para probar eso necesitamos el siguiente lema.

Lema 1. *Si $(X, \cdot)$ es un conjunto con una operación binaria*[2]*, $(G, \star)$ es un grupo y existe una función inyectiva $\phi : X \hookrightarrow G$ tal que $\phi(a \cdot b) = \phi(a) \star \phi(b) \quad \forall a, b \in X$, entonces la operación $\cdot$ es asociativa y $(X, \cdot)$ es un semigrupo.*

Demostración. Sean $a, b, c \in X$ elementos arbitrarios, entonces como la operación en G es asociativa:

$$\begin{aligned} \phi(a(bc)) &= \phi(a) \star \phi(bc) = \phi(a) \star (\phi(b) \star \phi(c)) = (\phi(a) \star \phi(b) \star \phi(c)) \\ &= \phi(ab) \star \phi(c) = \phi((ab)c)); \end{aligned}$$

[2]Dicha estructura se conoce como grupoide.

como la función es inyectiva tenemos que $a(bc) = (ab)c$ y entonces la operación $\cdot$ es asociativa. □

Teorema 2. *$\mathcal{F} = \mathcal{F}(\mathcal{X})$ con la yuxtaposición y reducción es un grupo. $\mathcal{F}(\mathcal{X})$ se conoce como el grupo libre sobre $\mathcal{X}$ y el conjunto $\mathcal{X}$ es llamado la base de $\mathcal{F}(\mathcal{X})$.*

Demostración. Por definición, $1 \in \mathcal{F}$ actúa como idéntico para la operación. Sea $x = x_1^{\lambda_1} \dots x_m^{\lambda_m} \in \mathcal{F}$, entonces su elemento inverso es $x^{-1} = x_m^{-\lambda_m} \dots x_1^{-\lambda_1}$.
Ahora se probará que la operación definida anteriormente es asociativa. Para $x \in \mathcal{X}$ y $\delta = \pm 1$ se define $|x^\delta| : \mathcal{F} \to \mathcal{F}$ como sigue:

$$x_1^{\delta_1} \dots x_m^{\delta_m} \mapsto \begin{cases} x^\delta x_1^{\delta_1} \dots x_m^{\delta_m} & \text{si } x^\delta \neq x_1^{-\delta_1} \\ x_2^{\delta_2} \dots x_m^{\delta_m} & \text{si } x^\delta = x_1^{-\delta_1} \end{cases}$$

Se cumple:

$$|x||x^{-1}| = 1_{\mathcal{F}} = |x^{-1}||x|$$

Entonces $|x^\delta|$ es una permutación de $\mathcal{F}$ (y su inversa es $|x^{-\delta}|$).
Sea $\mathcal{A}(\mathcal{F})$ el grupo simétrico en $\mathcal{F}$ y $\mathcal{F}_0$ el subgrupo generado por $\{|x||x \in \mathcal{X}\}$. Se define entonces la función $\varphi : \mathcal{F} \to \mathcal{F}_0$ con las siguientes reglas $1 \mapsto 1_F$ y $x_1^{\delta_1} \dots x_m^{\delta_m} \mapsto |x_1^{\delta_1}| \dots |x_m^{\delta_m}|$. Se mostrará que la función es biyectiva y cumple además la condición del lema 1.
Si $\sigma \in \mathcal{F}_o$, entonces σ puede escribirse como $\sigma = |x_1^{\delta_1}| \dots |x_m^{\delta_m}|$ y el elemento $x_1^{\delta_1} \dots x_m^{\delta_m} \in \mathcal{F}$ es tal que su imagen $\varphi(x_1^{\delta_1} \dots x_m^{\delta_m})$ es igual a $|x_1^{\delta_1}| \dots |x_m^{\delta_m}|$, entonces φ es suprayectiva.
Ahora, por definición $\varphi(xy) = \varphi(x)\varphi(y)$ en donde podría ser necesario insertar elementos de la forma $x_i x_i^{-1}$ ó $x_i^{-1} x_i$ para formar las palabras x y y a partir de la palabra reducida xy.
Sean $x = x_1^{\delta_1} \dots x_m^{\delta_m}$ y $y = y_1^{\lambda_1} \dots y_n^{\lambda_n}$ elementos de $\mathcal{F}$. Si $\varphi(x) = \varphi(y)$ entonces $|x_1^{\delta_1}| \dots |x_m^{\delta_m}|$ y $|y_1^{\lambda_1}| \dots |y_n^{\lambda_n}|$ son iguales; como $\varphi(x) \in \mathcal{F}_0$ se puede aplicar al elemento $1 \in \mathcal{F}$, y se tiene que $(|x_1^{\delta_1}| \dots |x_m^{\delta_m}|)(1) = x_1^{\delta_1} \dots x_m^{\delta_m}$, de manera análoga se tiene que $(|y_1^{\lambda_1}| \dots |y_n^{\lambda_n}|)(1) = y_1^{\lambda_1} \dots y_n^{\lambda_n}$. Como $\varphi(x) = \varphi(y)$ es una igualdad entre funciones, se tiene que $(\varphi(x))(1) = (\varphi(y))(1)$ y entonces $x_1^{\delta_1} \dots x_m^{\delta_m} = y_1^{\lambda_1} \dots y_n^{\lambda_n}$, de donde se concluye que φ es inyectiva.
Entonces se cumplen las condiciones del lema 1 y se tiene que la operación de yuxtaposición es asociativa. Por lo tanto $\mathcal{F}$ es un grupo y φ es un isomorfismo de grupos.

$$\begin{array}{ccc} & & \mathcal{A}(\mathcal{F}) \\ & & \uparrow{\scriptstyle i_{\mathcal{F}_0}} \\ \mathcal{F} & \xrightarrow[\text{es isomorfismo}]{\varphi} & \mathcal{F}_0 \end{array}$$

□

Teorema 3. *(Propiedad universal) Sea $\mathcal{F}$ el grupo libre sobre $\mathcal{X}$ y $i : \mathcal{X} \to \mathcal{F}$ la inclusión en $\mathcal{F}$. Si G es un grupo y $f : \mathcal{X} \to G$ un mapeo de conjuntos, existe un único homomorfismo de grupos $\bar{f} : \mathcal{F} \to G$ tal que $\bar{f}i = f$, como se ilustra en el siguiente diagrama.*

$$\begin{array}{ccc} \mathcal{F} & & \\ {\scriptstyle i}\uparrow & \searrow^{\bar{f}} & \\ \mathcal{X} & \xrightarrow[f]{} & G \end{array}$$

Demostración. Primero se define la función $\bar{f}$. Sea $\bar{f}(1) = e$, donde e es el neutro de G, y para un elemento arbitrario $x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n}$ del grupo libre, se define $\bar{f}(x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n})$ como $f(x_1)^{\delta_1}f(x_2)^{\delta_2}\dots f(x_n)^{\delta_n}$ (elemento de G bien definido).
Se probará ahora que $\bar{f}$ es un homomorfismo. Se consideran dos elementos arbitrarios de $\mathcal{F}$, que se expresan como $x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n}$ y $y_1^{\lambda_1}y_2^{\lambda_2}\dots\ y_m^{\lambda_m}$. Entonces

$$\begin{aligned} & \bar{f}(x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n}y_1^{\lambda_1}y_2^{\lambda_2}\dots y_m^{\lambda_m}) \\ = \ & f(x_1)^{\delta_1}f(x_2)^{\delta_2}\dots f(x_n)^{\delta_n}f(y_1)^{\lambda_1}f(y_2)^{\lambda_2}\dots f(y_m)^{\lambda_m} \\ = \ & (f(x_1)^{\delta_1}f(x_2)^{\delta_2}\dots f(x_n)^{\delta_n})(f(y_1)^{\lambda_1}f(y_2)^{\lambda_2}\dots f(y_m)^{\lambda_m}) \\ = \ & \bar{f}(x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n})\bar{f}(y_1^{\lambda_1}y_2^{\lambda_2}\dots y_m^{\lambda_m}) \end{aligned}$$

Por lo tanto $\bar{f}$ es un homomorfismo. Ahora se probará la unicidad. Sea $g : \mathcal{F} \to G$ un homomorfismo tal que $gi = f$ y sea $x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n} \in \mathcal{F}$, entonces

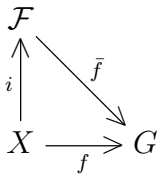
$$\begin{aligned} & g(x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n}) \\ = \ & g(x_1^{\delta_1})g(x_2^{\delta_2})\dots g(x_n^{\delta_n}) \\ = \ & g(x_1)^{\delta_1}g(x_2)^{\delta_2}\dots g(x_n)^{\delta_n} \\ = \ & gi(x_1)^{\delta_1}gi(x_2)^{\delta_2}\dots gi(x_n)^{\delta_n} \\ = \ & f(x_1)^{\delta_1}f(x_2)^{\delta_2}\dots f(x_n)^{\delta_n} \\ = \ & \bar{f}(x_1^{\delta_1}x_2^{\delta_2}\dots x_n^{\delta_n}) \end{aligned}$$

Por lo tanto $\bar{f} = g$. □

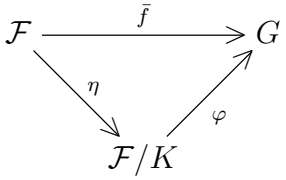
El siguiente resultado muestra una forma de caracterizar a un grupo arbitrario por medio de grupos libres y justifica el término *propiedad universal*.

Corolario 4. *Todo grupo es un cociente de un grupo libre.*

Demostración. Sea G un grupo, entonces se construye el conjunto $X = \{x_g | g \in G\}$, al que le corresponde un elemento formal por cada elemento del grupo G y la biyección natural $f : X \to G$ definida como $f(x_g) = g$.
Sea $\mathcal{F}$ el grupo libre sobre X. Por el teorema anterior $\exists \bar{f} : \mathcal{F} \to G$ un único homomorfismo tal que $\bar{f}i = f$, como se observa en el siguiente diagrama.



Como f es una función biyectiva, entonces $\bar{f}$ es suprayectiva. Si $K = Nu(\bar{f})$, por el primer teorema de isomorfismos, se tiene que existe $\varphi : \mathcal{F}/K \to G$ isomorfismo tal que el siguiente diagrama conmuta:



Por lo tanto $F/K \simeq G$.

□

En el corolario anterior se observa que K queda determinado por cualquier subconjunto de X que lo genere y que si $k = x_{g_1}^{\delta_1} \dots x_{g_r}^{\delta_r}$ es un generador de K, entonces $\bar{f}(k) = g_1^{\delta_1} \dots g_r^{\delta_r} = e$.
De la misma forma se puede caracterizar un grupo a partir de un grupo libre y un subgrupo normal del mismo.

Definición (Presentación de grupo)**.** Sea X un conjunto y Δ un conjunto de palabras reducidas en X. Se dice que un grupo G está definido por los generadores $x \in X$ y las relaciones $w = 1$ (con $w \in \Delta$) si:

$$G \simeq F/R,$$

donde F es el grupo libre con base X y R es el subgrupo normal de F generado por Δ, esto es, es la intersección de los subgrupos normales de F que contienen a Δ. Se dice que la pareja ordenada:

$$< X|\Delta >$$

es una presentación del grupo G.

Partiendo de la definición se ve claramente que los elementos que pertenecen a R son equivalentes al elemento neutro en $F/R \simeq G$.

Ejemplo. $\mathbb{Z}_6 \simeq < x|x^6 >$.

Considérese el conjunto $X = \{x\}$ y su grupo libre $F = F(X) =< x >$, sea $\Delta = \{x^6\}$, el subgrupo normal generado por las relaciones es $R =< x^6 >$. Entonces se tiene que encontrar un homomorfimo suprayectivo $\varphi : F \to \mathbb{Z}_6$ con $Nu(\varphi) = R$. Se define $f : X \to \mathbb{Z}_6$ como

$$f(x) = 1.$$

Por la propiedad universal se puede extender a un homomorfismo único $\varphi : F \to \mathbb{Z}_6$ con $\varphi i = f$. Se tiene que:

$$Nu(\varphi) = \{x^n \in F \mid \varphi(x^n) = 0\}$$

ya que:

$$\begin{aligned} \varphi(x^n) = 0 &\Leftrightarrow n\varphi(x) = 0 \\ &\Leftrightarrow n = 6k \quad k \in \mathbb{Z} \\ &\Leftrightarrow x^n = x^{6k} \\ &\Leftrightarrow x^n \in < x^6 >= R \end{aligned}$$

Definición. Si G es un grupo y $a, b \in G$, se denota $[a, b] = aba^{-1}b^{-1}$, $[a, b]$ es llamado el *conmutador* de a y b.

Ejemplo. *El grupo de trenzas con n hilos B_n es el grupo con la presentación:*

$$< \sigma_1, \sigma_2, ..., \sigma_{n-1} \mid [\sigma_i, \sigma_j] = 1 \ si \ |j - i| > 1, \ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} > .$$

Ahora se definen a los grupos abelianos libres, que se usaran en las siguientes partes del capítulo.

Definición. La *base* de un grupo abeliano F es un subconjunto $X \subset F$ tal que $F =< X >$ y para elementos distintos $x_1, x_2, \ldots, x_k \in X$ y $n_i \in \mathbb{Z}$

$$n_1x_1 + n_2x_2 + \ldots + n_kx_k = 0$$

implica que $n_i = 0$ para toda i.

Definición. Un *grupo abeliano libre* F con base X es un grupo abeliano F con base X no vacía.

Ejemplo. *El grupo libre en X tiene la presentación*

$$F =< X \mid \emptyset > ,$$

y el grupo libre abeliano en X tiene la presentación

$$F =< X \mid xyx^{-1}y^{-1} = 1 \ \forall x, y \in X > .$$

Antes de comenzar el estudio de los grupos de trenzas se mostrarán algunos resultados adicionales que permiten caracterizar a un grupo libre por el número de elementos en su base. Se utilizarán para esto algunos resultados de grupos abelianos libres, que se pueden consultar en [13].
En particular se hará uso del siguiente teorema, análogo a la propiedad universal para grupos abelianos libres, que se enuncia sin demostración.

Teorema 5. *F es el grupo abeliano libre con base X si, y solo si dado un grupo abeliano G y una función $f : X \to G$ existe un único homomorfismo de grupos $\bar{f} : F \to G$ tal que $\bar{f}i = f$, donde $i : X \to F$ es la inclusión de X en F.*

Es necesario también introducir el concepto de *subgrupo conmutador* de un grupo G.

Definición (Subgrupo Conmutador)**.** Si G es un grupo, el subgrupo conmutador G' es el subgrupo dado por:

$$G' =< [a,b] \mid a,b \in G > .$$

El subgrupo conmutador también se conoce como *subgrupo derivado* de G. Ahora se probarán algunas de sus propiedades básicas.

Teorema 6. *El subgrupo conmutador G' es un subgrupo normal de G. Además se tiene que el grupo cociente G/G' es un grupo abeliano.*

Demostración. Primero se observa que si $a, b, g \in G$

$$\begin{aligned} g^{-1}[a,b]g &= g^{-1}aba^{-1}b^{-1}g \\ &= g^{-1}aba^{-1}eb^{-1}g \\ &= g^{-1}aba^{-1}gb^{-1}bg^{-1}b^{-1}g \\ &= ((g^{-1}a)b(a^{-1}g)b^{-1})(bg^{-1}b^{-1}g) \\ &= [g^{-1}a,b][b,g^{-1}] \in G' \end{aligned}$$

de donde se tiene que $g^{-1}[a,b]g \in G'$ $\forall g \in G$ y para todo $[a,b]$. Ahora, si $x \in G'$ y $g \in G$, entonces x se puede escribir como un producto finito de conmutadores $x = [a_1,b_1]\dots[a_n,b_n]$; si se multiplica $g^{-1}xg$ por $e = gg^{-1}$ entre cada uno de los conmutadores se tiene:

$$\begin{aligned} g^{-1}xg &= g^{-1}[a_1,b_1]e\dots e[a_n,b_n]g \\ &= g^{-1}[a_1,b_1]gg^{-1}\dots gg^{-1}[a_n,b_n]g \\ &= (g^{-1}[a_1,b_1]g)\dots(g^{-1}[a_n,b_n]g) \in G' \end{aligned}$$

ya que cada uno de los elementos $g^{-1}[a_i,b_i]g \in G'$ por la observación anterior, de aquí que $g^{-1}G'g \subset G'$ para todo $g \in G'$. Por lo tanto G' es un subgrupo normal de G. Para la segunda parte del teorema, sea H un grupo tal que $H \simeq G/G'$ y por el primer teorema de isomorfismos se puede encontrar $\varphi : G \to H$ un epimorfismo con

$Nu(\varphi) = G'$. Sean $u, v \in H$ arbitrarios y $x, y \in G$ tales que $\varphi(x) = u$ y $\varphi(y) = v$, entonces se considera $uvu^{-1}v^{-1}$, la imagen del elemento $xyx^{-1}y^{-1}$ bajo φ, y como $xyx^{-1}y^{-1} \in G' = Nu(\varphi)$ se tiene que $uvu^{-1}v^{-1} = e$ y $uv = vu$ en H. Por lo tanto G/G' es un grupo abeliano. □

Lema 7. *Sea $\mathcal{F}$ un grupo libre con base $\mathcal{X}$ y $\mathcal{F}'$ el subgrupo conmutador, entonces $\mathcal{F}/\mathcal{F}'$ es un grupo abeliano libre con base $X_{\#} = \{x\mathcal{F}' \mid x \in X\}$.*

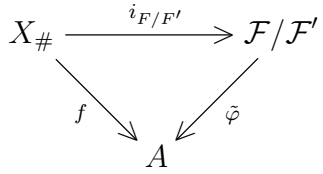
Demostración. Sea A un grupo abeliano y $f : X_{\#} \to A$ una función. Se define $f_{\#} : X \to A$ para cada elemento de X como $x \mapsto f(x\mathcal{F}')$. Entonces por la propiedad universal $\exists\varphi : \mathcal{F} \to A$ que extiende a $f_{\#}$.
Como A es un grupo abeliano, si $x, y \in \mathcal{F}$, entonces

$$\varphi(xyx^{-1}y^{-1}) = \varphi(x)\varphi(y)\varphi(x)^{-1}\varphi(y)^{-1} = e,$$

de aquí que $[x, y] \in Nu(\varphi)$ y $\mathcal{F}' \leq Nu(\varphi)$. Por lo tanto existe un homomorfismo $\tilde{\varphi} : \mathcal{F}/\mathcal{F}' \to A$ dado por

$$\tilde{\varphi}(w\mathcal{F}') = \varphi(w) = f(w\mathcal{F}')$$

Ahora se probará que $\tilde{\varphi}$ es único. Sea $\theta : \mathcal{F}/\mathcal{F}' \to A$ con $\theta(x\mathcal{F}') = f(x\mathcal{F}')$ y $\eta : \mathcal{F} \to \mathcal{F}/\mathcal{F}'$ el homomorfismo natural, entonces $\theta\eta : F \to A$ es un homomorfismo con $\theta\eta(x) = \theta(xF') = f(xF') = \varphi(x)$ para toda $x \in X$. Como $\theta\eta(x) = \varphi(x) = \tilde{\varphi}(x\mathcal{F}') = \tilde{\varphi}(\eta(x))$ y η es un epimorfismo, tenemos que $\theta = \tilde{\varphi}$.
Por lo tanto, dada una función $f : X_{\#} \to A$, donde A es un grupo abeliano, existe un único homomorfismo $\tilde{\varphi} : \mathcal{F}/\mathcal{F}' \to A$ que extiende a f. Entonces $\mathcal{F}/\mathcal{F}'$ es un grupo libre abeliano con base $X_{\#}$.



□

Definición (Rango de un grupo libre)**.** El rango de un grupo libre $\mathcal{F}$ es el número de elementos en la base del mismo, denotado como $rango(\mathcal{F})$.

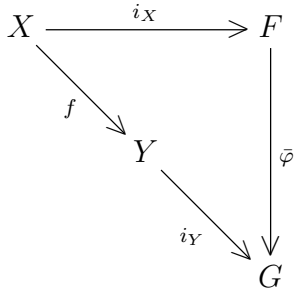
Se enunciará el siguiente teorema sin demostración. Para una demostración se puede consultar [15].

Teorema 8. *Dos grupos libres abelianos $\mathcal{F}$ y $\mathcal{G}$ con bases X y Y, respectivamente, son isomorfos* $\Leftrightarrow |X| = |Y|$.

Entonces ahora se demostrará la versión más general del teorema.

Teorema 9. *Sean $\mathcal{F}$ y $\mathcal{G}$ grupos libres con bases X y Y, respectivamente, entonces* $\mathcal{F} \simeq \mathcal{G} \Leftrightarrow |X| = |Y|$.

Demostración. $\Rightarrow$] Sea $\varphi : \mathcal{F} \to \mathcal{G}$ un isomorfismo. Entonces $\mathcal{F}/\mathcal{F}' \simeq \mathcal{G}/\mathcal{G}'$ y $\mathcal{F}/\mathcal{F}'$ es un grupo libre abeliano con base $X_\# = \{x\mathcal{F}' | x \in X\}$. Se mostrará que $|X_\#| = |X|$. Se suponen $x, y \in X$, $x \neq y$ y $x\mathcal{F}' = y\mathcal{F}'$, entonces $x^{-1}y \in \mathcal{F}'$, lo cual es una contradicción ya que $X_\#$ es base de $\mathcal{F}/\mathcal{F}'$, de donde se tiene que si $x \neq y \in X$, $x\mathcal{F}' \neq y\mathcal{F}'$ y $|X| = |X_\#|$. Entonces $|X| = rango(\mathcal{F}/\mathcal{F}')$ y $|Y| = rango(\mathcal{G}/\mathcal{G}')$ y como $\mathcal{F}/\mathcal{F}' \simeq \mathcal{G}/\mathcal{G}'$, por el teorema anterior se tiene que $|X| = |Y|$.
$\Leftarrow$] Como $|X| = |Y|$ entonces $\exists f : X \to Y$ biyectiva. Sean $\bar{\varphi} : \mathcal{F} \to \mathcal{G}$ el homomorfismo que extiende a $i_Y \circ f$,



y $\bar{\psi} : G \to F$ el homomorfismo que extiende a $i_X \circ f^{-1}$. Entonces $\bar{\psi}\bar{\varphi} : \mathcal{F} \to \mathcal{F}$ es tal que $\bar{\psi}\bar{\varphi}|_X = i_X$, entonces $\bar{\psi}\bar{\varphi}$ es un homomorfismo que extiende 1_X, pero $1_F : \mathcal{F} \to \mathcal{F}$ también extiende a 1_X, por la propiedad universal se tiene entonces que $\bar{\psi}\bar{\varphi} = 1_F$. Análogamente, se prueba que $\bar{\varphi}\bar{\psi} = 1_G$ y se concluye que $\bar{\varphi}$ es un isomorfismo. $\square$

La teoría desarrollada en la presente sección es suficiente para comenzar a desarrollar la teoría de los grupos de trenzas en el siguiente capítulo.

2.2. Antecedentes de criptografía

En esta sección se definen los conceptos básicos de criptografía que se usarán en los siguientes capítulos. La exposición presentada aquí está basada en [17, 18]. En el contexto de este trabajo un *bit* es una elemento del conjunto $\{0,1\}$ y un vector de n *bits* es un elemento del conjunto $\{0,1\}^n$.

Definición (Cifrado simétrico de n bits)**.** La notación V_m se usa para el conjunto de vectores de m bits. $\mathcal{K} = V_{\bar{n}}$ para alguna $\bar{n} \in \mathbb{N}$ es el conjunto de llaves.
Un cifrado simétrico de n bits es una función $E : V_n \times \mathcal{K} \to V_n$ tal que para cada $k \in \mathcal{K}$, $E(P,k)$ es una función invertible que se denota $E_k(P)$ y la inversa se denotará $D_k(C)$, para toda $P, C \in V_n$. E_k es llamada la función de cifrado y D_k la función de descifrado.

Con una función de cifrado simétrico se puede establecer una comunicación entre dos partes de manera segura. Como usualmente se hace, se denotarán como Alicia y Beto (A y B) a cada una de las partes, un mensaje plano se refiere al mensaje antes de ser cifrado. Ambas partes se ponen de acuerdo en una llave secreta $k \in \mathcal{K}$. Alicia tiene un mensaje plano $P \in V_n$ que desea mandar de manera segura a Beto, para eso obtiene el mensaje cifrado $C = E_k(P)$ y lo manda a Beto. Al recibir el mensaje cifrado, Beto utiliza la función inversa para obtener el mensaje plano $P = D_k(C)$. Esto se ilustra en el siguiente diagrama.

$$\text{Alicia} \xrightarrow{C=E_k(P)} \text{Beto}$$
$$C = E_k(P) \qquad\qquad D_k(C) = P$$

La seguridad de la comunicación depende de la elección de la función de cifrado E, así como de la confidencialidad de la llave k.
El tipo de cifrado antes definido pertenece a la categoría de cifrados de bloque debido a que el texto plano es una cadena de n bits ($n > 1$) y a la categoría de cifrados de tamaño fijo, ya que la cadena cifrada tiene la misma longitud que la cadena original.

Definición (Función Hash)**.** Una función Hash es una función

$$H : V_\infty \to V_n$$

donde V_∞ es el conjunto de vectores con número arbitrario de bits.

Una de las propiedades deseables de una función Hash es que sea una función de una sola dirección, es decir, que a partir de un valor de la imagen sea un problema difícil encontrar la preimagen del mismo. También se busca que dos cadenas arbitrarias tengan la misma imagen con muy baja probabilidad, si H tiene esta propiedad se dice que H es una función libre de colisiones. Si el dominio de la función Hash es V_s con $s \in \mathbb{N}$ se dice que es un Hash de entrada fijo y normalmente se elige n tal que $n \geq log_2(s)$.
Si Alicia y Beto establecen una llave secreta k, entonces podrían utilizar una función Hash para cifrar mensajes entre ellos.

Definición (Cifrado con Hash)**.** Sea $H : V_s \to V_n$ una función Hash, $P \in V_n$ un mensaje plano y $k \in \mathcal{K} = V_s$ una llave, entonces se define el cifrado $E : V_n \to V_n$ como sigue:

$$E(P) = P \oplus H(k).$$

Donde $\oplus$ es la suma módulo 2 elemento a elemento. Para descifrar solamente se necesita aplicar una vez la suma módulo 2 de la imagen de la llave bajo la función Hash.

$$(P \oplus H(k)) \oplus H(k) = P \oplus (H(k) \oplus H(k)) = P.$$

El protocolo de establecimiento de llaves de Diffie-Hellman fue inventado en 1976 y es considerado como el inicio de la criptografía de llave pública, la descripción original fue publicada en [6]. La idea del protocolo es lograr que dos partes puedan establecer un secreto (en particular una llave secreta) que luego les permita usar un sistema criptográfico simétrico.

Definición (Protocolo de intercambio de llaves de Diffie-Hellman)**.** El protocolo básicamente consiste en los siguientes pasos.

1 Alicia y Beto se ponen de acuerdo en un grupo cíclico finito G y un elemento generador g.

2 Alicia escoge al azar $a \in \mathbb{N}$ y envía el elemento g^a a Beto.

3 Beto escoge al azar $b \in \mathbb{N}$ y envía el elemento g^b a Alicia.

4 Alicia calcula $K_A = (g^b)^a = g^{ba}$.

5 Beto calcula $K_B = (g^a)^b = g^{ab}$.

El elemento $K = K_A = K_B$ es el secreto compartido por Alicia y Beto.

La seguridad de este algoritmo depende de la dificultad de resolver el problema de Diffie-Hellman. Para lo siguiente G representa un grupo cíclico finito.
El problema de Diffie-Hellman consiste en obtener g^{ab} a partir de g^a y g^b, $g \in G$, para algunos $a, b \in \mathbb{N}$.
Relacionado con el problema de Diffie-Hellman, existe también el problema del logaritmo discreto.
El problema del logaritmo discreto consiste en obtener c a partir de g^c, $g \in G$, donde $c \in \mathbb{N}$.
Claramente, si se encuentra una solución para el problema del logaritmo discreto entonces se tiene una solución para el problema de Diffie-Hellman. Es desconocido si la solución del problema de Diffie-Hellman implica la solución al problema del logaritmo discreto.

Capítulo 3

El grupo de trenzas

Los grupos de trenzas fueron estudiados por primera vez por Artin en 1925 y sus resultados más importantes fueron presentados en el artículo [3]. Estos grupos aparecen en distintas áreas de las matemáticas y han sido estudiados desde diversos puntos de vista. El nombre de los grupos de trenzas viene de las estructuras geométricas que caracterizan. En este capítulo solamente se utilizarán las propiedades algebraicas de estos grupos y su representación geométrica será utilizada sólo de manera ilustrativa.

3.1. Presentación del grupo

Definición (Grupo de trenzas)**.** El grupo de trenzas B_n con n hilos es el grupo con la presentación:

$$< \sigma_1, \sigma_2, .., \sigma_{n-1} |\ [\sigma_i, \sigma_j] = 1,\ \text{si}\ |i-j| > 1, \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} > . \tag{3.1}$$

Esta presentación se conoce como la de Artin y los elementos σ_i son llamados generadores de Artin. El elemento idéntico del grupo se denotará con el símbolo ϵ. A los elementos del grupo se les conoce como trenzas. La notación B_n proviene del nombre en inglés(Braid group).

El grupo de trenzas con n hilos tiene la siguiente representación geométrica. Se consideran n hilos alineados paralelamente de manera vertical. Entonces el generador σ_i representa el cruce de los hilos i e $i+1$, de manera que el hilo i pasa por debajo del $i+1$. El generador σ_i^{-1} es también el cruce de los hilos i e $i+1$, pero de manera que el

hilo i pasa por encima del $i+1$ y los otros hilos quedan fijos. Los ejemplos se mostrarán en el grupo de trenzas con 5 hilos B_5.

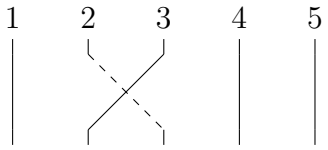


Figura 3.1: Trenza σ_2.

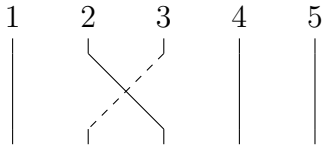


Figura 3.2: Trenza σ_2^{-1}.

La concatenación de generadores representa una serie de cruces en los hilos en el orden que indican los generadores.

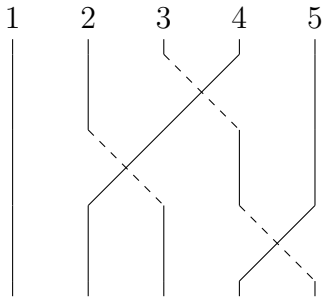


Figura 3.3: Concatenación de generadores: $\sigma_3\sigma_2\sigma_4$.

Dos trenzas son equivalentes en la representación geométrica si se puede transformar una en otra moviendo los hilos de manera que no se crucen.
Como se observa en las figuras 3.4, 3.5 y 3.6, la representación geométrica de dos trenzas equivalentes tiene los mismos hilos en las mismas posiciones iniciales y finales y se puede pasar de una a otra sin cruzar los hilos.

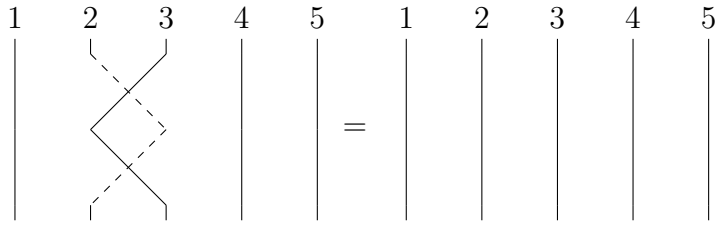


Figura 3.4: Trenzas equivalentes: $\sigma_2\sigma_2^{-1}$ y ϵ.

Se pueden ver también las representaciones geométricas de las relaciones de la presentación de grupo.

$$\sigma_i\sigma_j = \sigma_j\sigma_i \text{ si } |i-j| > 1 \tag{3.2}$$

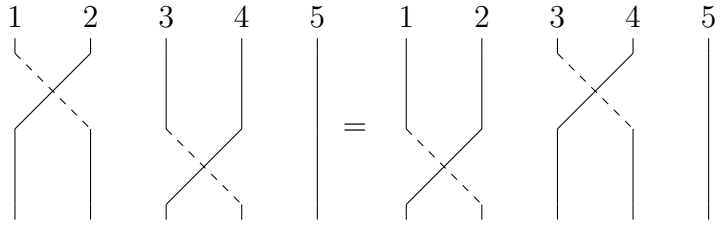


Figura 3.5: Trenzas equivalentes: $\sigma_1\sigma_3$ y $\sigma_3\sigma_1$.

En la representación geométrica la primera relación (figura 3.5) resulta evidente, ya que los cruces no involucran hilos comunes.

$$\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} \tag{3.3}$$

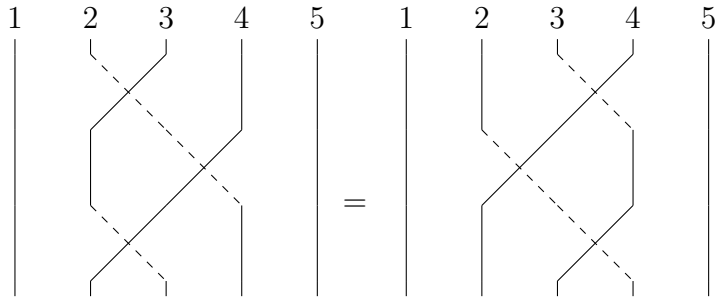


Figura 3.6: las trenzas $\sigma_2\sigma_3\sigma_2$ y $\sigma_3\sigma_2\sigma_3$.

La segunda relación no resulta ser tan evidente, pero se observa que para transformar la primera trenza en la segunda no se necesita pasar ninguno de los hilos por encima de otro.

Debido a la importancia de las relaciones anteriores y a su uso frecuente en este capítulo se hará referencia a las mismas como **propiedad 3.2** y **propiedad 3.3**, respectivamente.

3.2. La trenza fundamental Δ_n

Una forma normal para un elemento de un grupo con presentación finita es una manera de representar al elemento de manera única. Como se ha visto, utilizando la presentación por generadores no se tiene una representación única, debido a las relaciones de igualdad que se ilustran en las figuras 3.5 y 3.6. Tener una forma normal es necesario para poder representar y almacenar un elemento del grupo de trenzas en una computadora, así como para realizar operaciones entre los elementos.
Para poder encontrar una forma normal para el grupo de trenzas es necesario desarrollar algunas herramientas teóricas. En lo sucesivo se considerará siempre el grupo de trenzas de n hilos B_n, con n fija.

Definición (Trenza positiva)**.** Una trenza positiva es una trenza cuya representación con generadores de Artin no incluye inversos de generadores. Se denotará B_n^+ al conjunto de trenzas positivas de B_n.

Así, la trenza $\sigma_1\sigma_3\sigma_2$ es una trenza positiva, mientras que la trenza $\sigma_2\sigma_1^{-1}\sigma_3$ no lo es. Se puede observar que en una trenza positiva los hilos siempre se cruzan en el mismo sentido. Así que si se considera la representación en un espacio de tres dimensiones, para cada pareja de hilos i, j con $i < j$ se tiene que el hilo i está en un nivel de profundidad mayor al del hilo j.

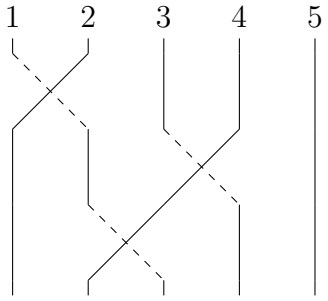


Figura 3.7: Trenza $\sigma_1\sigma_3\sigma_2$

Como B_n tiene una estructura de grupo, el conjunto de trenzas positivas B_n^+ tiene también una estructura, aunque es más débil, como se muestra en el siguiente resultado.

Teorema 10. *El conjunto de trenzas positivas B_n^+ es un monoide con la operación de concatenación.*

Demostración. Primero se considera el hecho de que la operación de concatenación en B_n es asociativa y que el elemento ϵ está contenido en B_n^+. Como todos las trenzas positivas no contienen inversos de los generadores, entonces en su concatenación no habrá inversos de generadores, por lo que la operación es cerrada. □

Podemos definir un homomorfismo entre el grupo de trenzas y el grupo aditivo de los números enteros.

Definición (Peso de una trenza)**.** Sea $wt : B_n \to \mathbb{Z}$, definida para cada generador de la siguiente forma:

$$wt(\sigma_i) = 1$$

Al aplicar este homomorfismo a una trenza positiva, obtenemos su longitud lexicográfica, es decir, el número de generadores que la representan, por ejemplo:

$$wt(\sigma_1\sigma_3^2\sigma_2) = wt(\sigma_1) + 2wt(\sigma_3) + wt(\sigma_2) = 1 + 2 + 1 = 4$$

Ahora se presentará un elemento distinguido del conjunto de trenzas positivas, la trenza fundamental Δ_n.

Definición (Trenza fundamental)**.** La trenza fundamental es el elemento de B_n^+ con la descripción:

$$\Delta_n = (\sigma_1\sigma_2 \dots \sigma_{n-1})(\sigma_1\sigma_2 \dots \sigma_{n-2}) \dots (\sigma_1\sigma_2)\sigma_1$$

También se puede definir la trenza fundamental como $\Delta_2 = \sigma_1$ y $\Delta_n = (\sigma_1 \dots \sigma_{n-1})\Delta_{n-1}$[1]. En el grupo de trenzas B_5, la trenza fundamental es:

$$\Delta_5 = (\sigma_1\sigma_2\sigma_3\sigma_4)(\sigma_1\sigma_2\sigma_3)(\sigma_1\sigma_2)\sigma_1,$$

que tiene la representación gráfica que se muesta en la figura 3.8.

[1]Este tipo de definiciones suelen llamarse definiciones inductivas.

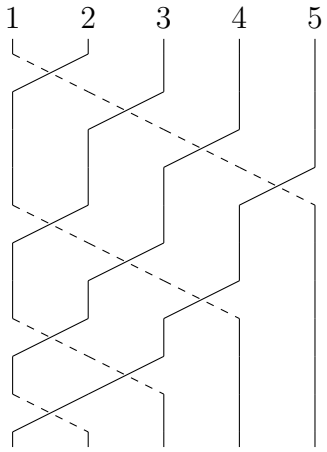


Figura 3.8: Trenza fundamental Δ_5

Como se observa, la trenza fundamental es la trenza donde cada pareja de hilos se cruza de manera positiva exactamente una vez. Como se verá más adelante, la trenza fundamental es esencial para la construcción de la forma normal de Garside. Las propiedades de la trenza fundamental se resumen en el siguiente teorema.

Teorema 11. *La trenza fundamental Δ_n tiene las siguientes propiedades:*

- *Para cada generador σ_i, $\Delta_n = \sigma_i A = B\sigma_i$ con $A, B \in B_n^+$.*
- *Para todo generador σ_i, se tiene que $\sigma_i \Delta_n = \Delta_n \sigma_{n-i}$.*
- *El centro del grupo B_n es el subgrupo cíclico generado por $\Delta_n^2 = \Delta_n \Delta_n$. Es decir, $C(B_n) =< \Delta_n^2 >$.*

Debido a la extensión de la demostración del teorema, ésta se presentará al final del presente capítulo.
El siguiente ejemplo en B_4 ilustra la segunda propiedad de las trenzas fundamentales.

$$\begin{aligned}
\sigma_3\Delta_4 &= \sigma_3(\sigma_1\sigma_2\sigma_3)(\sigma_1\sigma_2)\sigma_1 \\
&= \sigma_1\sigma_3\sigma_2\sigma_3(\sigma_1\sigma_2)(\sigma_1) \\
&= (\sigma_1\sigma_2\sigma_3)\sigma_2\sigma_1\sigma_2\sigma_1 \\
&= (\sigma_1\sigma_2\sigma_3)(\sigma_1\sigma_2\sigma_1)(\sigma_1) \\
&= (\sigma_1\sigma_2\sigma_3)(\sigma_1\sigma_2)(\sigma_1)\sigma_1 \\
&= \Delta_4\sigma_1
\end{aligned}$$

Se definirá el siguiente homomorfismo que permitirá generalizar las segunda propiedad de la trenza fundamental y comprender mejor su representación geométrica.

Definición. Sea $\tau : B_n \to B_n$ definido para cada uno de los generadores de las siguiente manera:

$$\tau(\sigma_i) = \Delta_n\sigma_i\Delta_n^{-1}.$$

Por su definición, τ es un automorfismo interior de B_n. Por la segunda propiedad de las trenzas fundamentales se comprueba que $\tau(\sigma_i) = \sigma_{n-i}$. Por otro lado y como consecuencia de la tercera propiedad de la trenza fundamental se tiene que

$$\sigma_i\Delta_n = \sigma_i\Delta_n\Delta_n\Delta_n^{-1} = \Delta_n\Delta_n\sigma_i\Delta_n^{-1} = \Delta_n\tau(\sigma_i).$$

Se puede entonces generalizar dicha propiedad de la siguiente manera:

$$A\Delta_n = \Delta_n\tau(A),\ A \in B_n^+.$$

Debido a la tercera propiedad de la trenza fundamental, el automorfismo τ es su propio inverso.

$$\tau^2(A) = \tau(\Delta_n A\Delta_n^{-1}) = \Delta_n^2 A\Delta_n^{-2} = A\Delta_n^2\Delta_n^{-2} = A$$

El automorfismo τ tiene también una representación geométrica interesante, como se observa en la siguiente figura.

Como se observa en la figura 3.9, si se consideran las trenzas anteriores en un espacio de tres dimensiones, la transformación τ correspondería a rotar la trenza sobre sí misma,

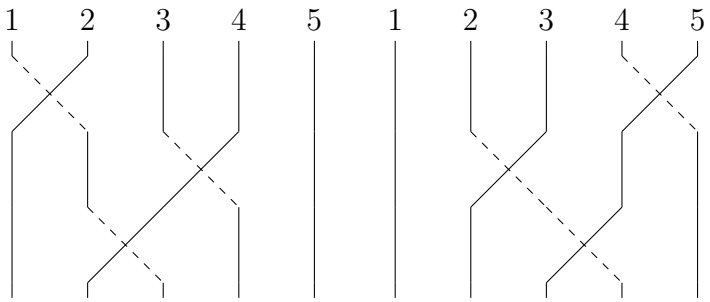


Figura 3.9: Trenzas $\sigma_1\sigma_3\sigma_2$ y $\tau(\sigma_1\sigma_3\sigma_2)$

de manera que el primer hilo quede en la última posición y el último hilo en la primera posición.

El automorfismo τ también puede definirse como $\tau(\sigma_i) = \Delta_n^{-1}\sigma_i\Delta_n$ y se verifica que ambas definiciones son iguales debido a la segunda propiedad de las trenzas fundamentales ya que:

$$\begin{aligned}\Delta_n^{-1}\sigma_i\Delta_n &= \Delta_n^{-1}\Delta_n\sigma_{n-i} \\ &= \sigma_{n-i} \\ &= \sigma_{n-i}\Delta_n\Delta_n^{-1} \\ &= \Delta_n\sigma_i\Delta_n^{-1}\end{aligned}$$

Por lo tanto se puede utilizar cualquiera de las dos definiciones anteriormente mencionadas del automorfismo según sea necesario.

Definición (Orden parcial para B_n)**.** Sean A y B dos trenzas de B_n, decimos que $A \leq B$ si y solo si $\exists C, D$ trenzas positivas tales que:

$$\begin{aligned}B = &\quad CAD \quad \text{ó} \\ B = &\quad CA \quad \text{ó} \\ B = &\quad AD.\end{aligned}$$

Se puede verificar que se cumplen las siguientes propiedades básicas.

- $B \in B_n^+ \Leftrightarrow \epsilon \leq B$
- $A \leq B \Leftrightarrow B^{-1} \leq A^{-1}$

La primera propiedad nos indica que las trenzas positivas B_n^+, son, de hecho, los elementos de la clase positiva en este orden parcial.
Como se verá con los siguientes resultados, la trenza fundamental Δ_n tiene un papel importante dentro de este orden parcial y determina una serie de trenzas que serán usadas para encontrar la forma normal de cualquier trenza arbitraria.

Lema 12. *Si $A \leq \Delta_n^s$, con $s \in \mathbb{Z}$, entonces $\Delta_n^s = D_1 A = A D_2$ para algunos $D_1, D_2 \in B_n^+$.*

Demostración. Como $A \leq \Delta_n^s$ existen $C_1, C_2 \in B_n^+$ tal que $C_1 A C_2 = \Delta_n^s$ y

$$\begin{aligned} AC_2 = C_1^{-1}\Delta_n^s \quad &\Rightarrow AC_2 = \Delta_n^s \tau^s(C_1^{-1}) = \Delta_n^s \tau^{-s}(C_1) \\ &\Rightarrow AC_2\tau^s(C_1) = \Delta_n^s \end{aligned}$$

La demostración de la parte faltante del lema es análoga. □

De la misma forma se puede demostrar el siguiente lema.

Lema 13. *Si $\Delta_n^s \leq A$, entonces $A = E_1 \Delta_n^s = \Delta_n^s E_2$ para algunos $E_1, E_2 \in B_n^+$.*

Los dos lemas anteriores muestran que cuando se considera a la trenza fundamental o una de sus potencias, el orden parcial bilateral antes definido puede considerarse como unilateral.

Corolario 14. *Si $\Delta_n^{r_1} \leq B \leq \Delta_n^{s_1}$ y $\Delta_n^{r_2} \leq C \leq \Delta_n^{s_2}$, entonces:*

$$\Delta_n^{r_1+r_2} \leq BC \leq \Delta_n^{s_1+s_2}.$$

Demostración. Por los lemas anteriores se tiene que $B = E_1\Delta_n^{r1} = \Delta_n^{r1}E_2$ y $C = D_1\Delta_n^{r2} = \Delta_n^{r2}D_2$, con $E_i, D_j \in B_n^+$. Sustituyendo $BC = E_1\Delta_n^{r1}\Delta_n^{r2}D_2 = E_1\Delta_n^{r_1+r_2}D_2$, y entonces $\Delta_n^{r_1+r_2} \leq BC$.
De la misma forma $\Delta_n^{s_1} = E_3 B$ y $\Delta_n^{s_2} = CD_3$. Sustituyendo $\Delta_n^{s_1+s_2} = E_3 BCD_3$ y entonces $BC \leq \Delta_n^{s_1+s_2}$. □

El siguiente teorema muestra que la trenza fundamental sirve para acotar una trenza arbitraria del grupo de trenzas.

Teorema 15. *Toda trenza $B \in B_n$ satisface*

$$\Delta_n^r \leq B \leq \Delta_n^s$$

para algunos $s, r \in \mathbb{Z}$.

Demostración. Para cada generador de Artin se tiene que $\epsilon \leq \sigma_i \leq \Delta_n$ y $\Delta_n^{-1} \leq \sigma_i^{-1} \leq \epsilon$. Entonces, se escribe B como una sucesión de generadores de Artin y se aplica el corolario 14. □

El resultado de este teorema permite definir el ínfimo y supremo de una trenza.

Definición. Sea $B \in B_n$, entonces

$$\text{ínf}(B) = \text{máx}\{r : \Delta_n^r \leq B\}.$$
$$\sup(B) = \text{mín}\{s : B \leq \Delta_n^s\}.$$

Se define ahora la longitud canónica de una trenza, la cual no depende de la representación de la trenza.

Definición (Longitud canónica de una trenza)**.** La longitud canónica $\mathcal{L}$ de una trenza $B \in B_n$ está definida como:

$$\mathcal{L}(B) = \sup(B) - \text{ínf}(B).$$

La notación de intervalos para conjuntos de trenzas, se presenta en la siguiente definición.

Definición. Sean $r, s \in \mathbb{Z}$, el subconjunto $[r, s] \subset B_n$, donde:

$$[r, s] = \{B \in B_n : \Delta_n^r \leq B \leq \Delta_n^s\}$$

es un intervalo de trenzas, se llamará el intervalo rs.

Se tiene entonces la siguiente propiedad:

$$[r, s] = \Delta_n^r[0, s - r].$$

Para verificar la igualdad anterior se considera primero una trenza $B \in [r, s]$, entonces se tiene que $\Delta_n^r \leq B$ y por el lema 13 existe $E \in B_n^+$ tal que

$B = \Delta_n^r E$. Se verifica que $\Delta_n^r \leq \Delta_n^r E \leq \Delta_n^s$ y multiplicando por Δ_n^{-r} se sigue que $E \leq \Delta_n^{-r}\Delta_n^s$ y $\epsilon \leq E \leq \Delta_n^{s-r}$, es decir, $E \in [0, r-s]$ y se concluye que $B \in \Delta_n^r[0, r-s]$. Por otro lado, si $B \in \Delta_n^r[0, r-s]$ se puede escribir a B como $\Delta_n^r C$ con $\epsilon \leq C \leq \Delta_n^{s-r}$, multiplicando por Δ_n^r se obtiene $\Delta_n^r \leq \Delta_n^r C = B \leq \Delta_n^s$, por lo tanto $B \in [r, s]$ y se tiene la igualdad de los dos conjuntos.

Es de especial interés el subconjunto $[0, 1]$, las trenzas positivas menores a la trenza fundamental, ya que serán éstas las que harán posible encontrar una forma normal para el grupo. El subconjunto $[0, 1]$, de la manera en que lo hemos definido, no es fácil de caracterizar. Más adelante se introducirán las trenzas de permutación, concepto que permitirá relacionar a los elementos de $[0, 1]$ con elementos más simples, permutaciones, lo que también servirá para implementar las operaciones del grupo de trenzas en la computadora.

Se expondrá cierta notación y un lema técnico, que se utilizarán para el desarrollo de la forma normal de Garside, el trabajo original se puede consultar en [10].

$$\sigma_i * \sigma_j = \begin{cases} \sigma_i & \text{si } i = j \\ \sigma_i\sigma_j & \text{si } |i-j| > 1 \\ \sigma_i\sigma_j\sigma_i & \text{si } |i-j| = 1 \end{cases}$$

Lema 16 (Garside)**.** *Si $P = \sigma_i P_1 = \sigma_j P_2$, con $P_1, P_2 \in B_n^+$, entonces $\exists P_3 \in B_n^+$ tal que $P = (\sigma_i * \sigma_j)P_3$.*

De la definición se observa que $\sigma_i * \sigma_j = \sigma_j * \sigma_i$, está propiedad será de utilidad para demostrar la existencia de la forma canónica de Garside.

También se definirán dos conjuntos de índices que indican los generadores con los que se puede iniciar (o terminar) la representación de una trenza.

Definición. Dada una trenza P positiva, su conjunto inicial $S(P) \subset \{1, \ldots, n-1\}$ es el conjunto

$$S(P) = \{i : P = \sigma_i P_i,\ P_i \geq \epsilon\},$$

y el conjunto terminal $F(P)$

$$F(P) = \{i : P = P_i\sigma_i,\ P_i \geq \epsilon\}.$$

La notación proviene de los nombres en inglés, start set y final set.
Con las definiciones anteriores se puede definir una factorización ponderada por la izquierda. La forma normal de Garside es de hecho una factorización de este estilo.

Definición (Factorización ponderada)**.** Una factorización positiva de una trenza $P = AB,\ A, B \geq \epsilon$ es ponderada por la izquierda si $S(B) \subset F(A)$ y ponderada por la derecha si $F(A) \subset S(B)$.

Una factorización ponderada por la izquierda resulta ser entonces una factorización donde los generadores usados para describir inicialmente el segundo factor pueden ser también usados para describir terminalmente al primero. Para obtener la forma normal de Garside de una trenza, se busca una serie de factorizaciones ponderadas por la izquierda del siguiente estilo:

$$P = A_1P_1,\ P_1 = A_2P_2, \ldots \quad \text{donde } A_i \in [0, 1].$$

Para poder desarrollar este tipo de factorizaciones, se introducirán las trenzas de permutación, un subconjunto del grupo de trenzas con ciertas propiedades geométricas que está estrechamente relacionado con el conjunto $[0, 1]$.

3.3. Trenzas de permutación

Si se consideran únicamente los hilos de una trenza en su representación gráfica, normalmente cambian su posición por medio de los cruces que determinan a la trenza. Si consideramos la posición final de cada uno de los hilos podemos definir la permutación de una trenza.

Definición. Dada una trenza $B \in B_n$, la permutación $\pi \in S_n$ asociada a la trenza es aquella que relaciona a i con la posición final de la trenza en la i-ésima posición. Se dice entonces que B tiene la permutación π o que B induce la permutación π en sus hilos.

Ejemplo. *La trenza vacía ϵ tiene la siguiente representación gráfica.*

Figura 3.10: Trenza vacía ϵ.

Como todos los hilos permanecen en su posición original, la permutación que corresponde a la trenza vacía es la identidad I.

Ejemplo. *La trenza $\sigma_2\sigma_1^{-1}\sigma_4$ tiene la representación geométrica siguiente.*

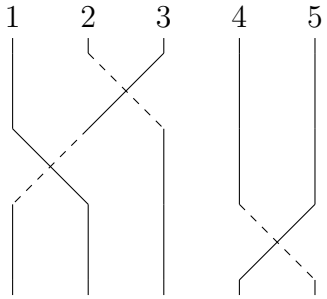


Figura 3.11: Trenza $\sigma_2\sigma_1^{-1}\sigma_4$.

Entonces la permutación que le corresponde es

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix}.$$

Es de importancia notar que diferentes trenzas pueden tener la misma permutación.

Ejemplo. *Las siguientes parejas de trenzas tienen la misma permutación.*

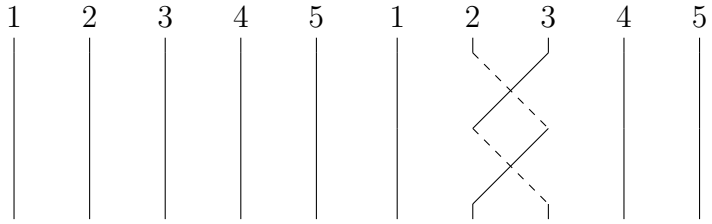


Figura 3.12: Trenzas ϵ y $\sigma_2\sigma_2$

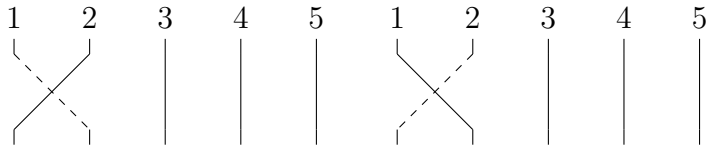


Figura 3.13: Trenzas σ_1 y σ_1^{-1}

En particular, se observa que ninguna de las parejas de trenzas son equivalentes bajo las propiedades 3.2 o 3.3. En el primer caso la misma pareja de hilos se cruza dos veces, haciendo que regresen a su posición original; en el segundo caso tenemos que una trenza formada por un generador tiene la misma permutación que la trenza formada por el inverso de dicho generador. Debido a los anterior, se estudiarán solamente las trenzas positivas y donde cada pareja de hilos se cruza a lo más una vez.

Definición (Trenza de permutación)**.** Una trenza positiva A es una trenza de permutación positiva si en su representación gráfica cada pareja de hilos se cruza a lo más una vez. El conjunto de trenzas positivas de permutación se denota S_n^+.

Ejemplo. *La trenza $\sigma_3\sigma_1\sigma_2$ es una trenza de permutación.*

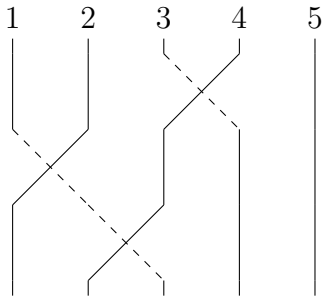


Figura 3.14: Trenza $\sigma_3\sigma_1\sigma_2$.

Por otro lado, las trenzas $\sigma_2\sigma_2$ y σ_1^{-1}, mostradas anteriormente (figuras 3.12 y 3.13) no son trenzas de permutación.

Como el automorfismo τ representa geométricamente rotar una trenza sobre sí misma, si en una trenza no se cruzaba cada pareja de hilos más de una vez, en su imagen bajo τ tampoco lo harán. Se dice que las trenzas de permutación son invariantes bajo τ.

Una vez más la trenza fundamental Δ_n ocupa un lugar distinguido entre las trenzas de permutación, ya que es la trenza donde cada pareja de hilos se cruza exactamente una vez.

El siguiente teorema muestra la relación entre las trenzas de permutación y las permutaciones que inducen.

Teorema 17. *Si A_1, A_2, dos trenzas de permutación, inducen la misma permutación en sus hilos, entonces $A_1 = A_2$. Para cada $\pi \in S_n$ existe $A_\pi \in S_n^+$ tal que A_π induce a la permutación π.*

Para la demostración del teorema se necesitan algunas nociones de topología que exceden el alcance del presente trabajo. Se puede consultar la demostración en [7]. También es importante notar que la igualdad en el teorema se refiere a la igualdad con las relaciones 3.2 y 3.3; así, dos trenzas de permutación podrían parecer diferentes si sólo se observa su representación generador por generador, pero ser equivalentes en el grupo de trenzas.
El teorema anterior establece una relación biyectiva entre el conjunto de trenzas de permutación S_n^+ y el grupo simétrico con n elementos S_n. Una conclusión inmediata de este hecho es que el conjunto de trenzas de permutación tiene $n!$ elementos.

Lema 18. *Si $A_\pi \in S_n^+$ induce una permutación π entonces son equivalentes las siguiente afirmaciones:*

1 .- $i \in S(A_\pi)$.

2 .- Los hilos i *e* $i+1$ *se cruzan en* A_π.

3 .- $\pi(i+1) < \pi(i)$.

Demostración. Si se supone (1) entonces se puede escribir a A_π de manera que inicie con el generador σ_i, entonces los hilos i e $i+1$ se cruzan en A_π. Como todos los cruces son positivos y cada pareja de hilos se cruza a lo más una vez, (2) y (3) son equivalentes. La equivalencia (3) implica (1) se puede consultar en [7]. □

El siguiente resultado caracteriza a los generadores que no pertenecen al conjunto inicial de una trenza de permutación.

Lema 19. *Sea $A \in S_n^+$, entonces:*

$$\sigma_i A \in S_n^+ \Leftrightarrow i \notin S(A)$$

Demostración. $\Rightarrow$) si $i \in S(A)$, entonces los hilos i e $i+1$ se cruzan en A, y en $\sigma_i A$ se cruzarían dos veces, por lo que $i \notin S(A)$.
$\Leftarrow$) Como $i \notin S(A)$, i y $i+1$ no se cruzan en A, entonces en $\sigma_i A$ se cruzan todos su hilos a los más una vez y $\sigma_i A \in S_n^+$. □

Lema 20. *Sea $A \in S_n^+$. Si $i, j \notin S(A)$, entonces $\sigma_i * \sigma_j A \in S_n^+$.*

Demostración. Si $i = j$ entonces por el lema anterior $\sigma_i * \sigma_j A = \sigma_i A \in S_n^+$.
Si $|i-j| > 1$, por el lema $\sigma_i A \in S_n^+$, y como no se cruzan los hilos j y $j+1$ en $\sigma_i A$, $j \notin S(\sigma_i A)$, y aplicando una vez más el lema anterior $\sigma_i * \sigma_j A = \sigma_j \sigma_i A \in S_n^+$.
Si $|i-j| = 1$, entonces los hilos i e $i+1 = j$, j y $j+1 = i+2$ no se cruzan, como todos los cruces son positivos, se tiene que los hilos i e $i+2 = j+1$ no se cruzan tampoco. Entonces $\sigma_i * \sigma_j = \sigma_i \sigma_j \sigma_i$ es una trenza donde cada una de las parejas de hilos antes mencionada se cruza una vez exactamente, y entonces en $\sigma_i * \sigma_j A$ se cruzan una vez exactamente, entonces $\sigma_i * \sigma_j A \in S_n^+$. □

Los lemas anteriores también son aplicables a $F(S)$ si se consideran los elementos de la forma $A\sigma_i$. Las demostraciones son análogas.
Con los resultados anteriores se demuestra que S_n^+ y $[0,1]$ son en efecto el mismo conjunto.

Teorema 21.

$$[0,1] = S_n^+$$

Demostración. Como se mencionó antes $\Delta_n \in S_n^+$, sea $A \in [0,1]$, entonces por el lema 12, existe $B \in B_n^+$ tal que $\Delta_n = AB$. Como Δ_n es una trenza de permutación positiva, cada pareja de sus hilos se cruza a lo más una vez, y como A, B son trenzas positivas, es necesario que en cada una de ellas cada pareja de hilos no se cruce más de una vez. Entonces $A \in S_n^+$, es una trenza de permutación positiva.
Por otro lado, sea A_π una trenza con permutación positiva π. Se define $\delta \in S_n$ como $\delta(i) = n+1-i$ y ρ como la permutación tal que $\pi\rho = \delta$. Sean A_ρ, A_δ las trenzas de permutación que inducen esas permutaciones. En $A_\pi A_\rho$ cada pareja de hilos se cruza a lo más dos veces, pero la trenza resultante tiene también la permutación δ. Como $\delta(i) > \delta(j)$ si $i < j$ y en virtud del lema 18, cada pareja de hilos debe de cruzarse un número impar de veces, entonces se cruza cada pareja de hilos exactamente una

vez. Entonces, $A_\pi A_\rho \in S_n^+$. Y como tiene la misma permutación que Δ_n, tenemos que $A_\pi A_\rho = \Delta_n$ y $A_\pi \leq \Delta_n$

□

Como $\sigma_i \leq \Delta_n$, se tiene que $S(\Delta_n) = F(\Delta_n) = \{1, \ldots, n-1\}$ por el lema 12, ya que se puede escribir $\Delta_n = \sigma_i B = C\sigma_i$ con $B, C \in B_n^+$.

Lema 22. *Sea $A \in S_n^+$, tal que satisface $S(A) = \{1, \ldots, n-1\}$. Entonces $A = \Delta_n$.*

Demostración. Sea π la permutación de A, como $i \in S(A)$ para toda $i \in \{1, \ldots, n-1\}$, se tiene que $\pi(i) > \pi(j)$ si $i < j$, entonces cada pareja de hilos se cruza exactamente una vez, y la única trenza en S_n^+ con esa propiedad es la trenza fundamental, entonces $A = \Delta_n$ □

3.4. La forma canónica de Garside

Ahora se comenzará a estudiar la forma canónica de Garside, para esto son necesarios algunos resultados que demuestran la existencia de una factorización adecuada.

Lema 23. *Si P es una trenza positiva, tiene una factorización positiva ponderada por la izquierda única $P = A_1 P_1$, $A_1 \in [0, 1]$. Cualquier otra factorización positiva ponderada por la izquierda $P = AB$ con $A \in [0, 1]$ satisface $A_1 = AQ$, para alguna $Q \geq \epsilon$.*

Demostración. Primero se probará la existencia de la factorización. Se comienza considerando todas las factorizaciones $P = AB$, con $A \in [0, 1]$ y se selecciona la factorización en la cual $wt(A)$ es máximo. Si $S(B) \not\subset F(A)$ se elije un elemento $i \in S(B)$ tal que $i \notin F(A)$. Entonces $A' = A\sigma_i \in S_n^+$ y se tiene una factorización con $wt(A') > wt(A)$, que contradice la hipótesis. Por lo tanto la factorización presentada debe ser ponderada por la izquierda y será denotada como $P = A_1 P_1$.
Ahora, supóngase que existen factorizaciones $P = AB$, $A \in S_n^+$ en las que A no es un subfactor de A_1. Sea $A = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n}$ la representación por generadores de A. Sea C_j, $j \leq n$, la subtrenza formada por los primeros j generadores que forman a A y sea $C_0 = \epsilon$. Se define el conjunto $\{C_j \mid C_j \leq A_1\}$ de subtrezas de A que son subfactores de A_1; este conjunto es no vacío ya que $C_0 = \epsilon \leq A_1$. Entonces existen factorizaciones $P = C\sigma_i B'$, con $C\sigma_i \in S_n^+$ tales que C sea un subfactor de A_1 pero $C\sigma_i$ no lo es. Se

selecciona la factorización de este tipo tal que $wt(C)$ sea máxima y se escribe $A_1 = CQ$. Por la maximalidad de $wt(A_1)$, se tiene que $wt(A_1) \geq wt(C\sigma_i) > wt(C)$, así que $Q \neq \epsilon$. Ahora se selecciona j de $S(Q)$, y se tiene que $C\sigma_j \leq A_1$, por lo tanto $C\sigma_j \in S_n^+$. Entonces hay una factorización $P = C\sigma_j B''$, se define B como $B = \sigma_i B' = \sigma_j B''$ y se aplica el lema de Garside para obtener B''' tal que $P = C(\sigma_i * \sigma_j)B'''$. Como $i, j \notin F(C)$, se tiene por el lema 20 que $C(\sigma_i * \sigma_j)$ es un elemento de S_n^+. Ahora se consideran dos casos, si $|i - j| > 1$ entonces $\sigma_i * \sigma_j = \sigma_i\sigma_j$ y $P = C(\sigma_i * \sigma_j)B''' = C\sigma_j\sigma_i B'''$ con $C\sigma_j \leq A$, por otro lado $P = C(\sigma_i * \sigma_j)B''' = C\sigma_i\sigma_j B'''$ y $C\sigma_i$ no es una subfactor de A_1, por lo tanto se tiene la factorización $P = (C\sigma_j)\sigma_i B'''$ con $C\sigma_j \leq A_1$ pero $C\sigma_j\sigma_i = C(\sigma_i * \sigma_j)$ no es un subfactor de A_1.

En el otro caso, $|i - j| = 1$ y se tiene que $P = C(\sigma_i * \sigma_j)B''' = C(\sigma_j\sigma_i\sigma_j)B'''$, con $C\sigma_j \leq A_1$ y $C\sigma_j\sigma_i\sigma_j B''' = C(\sigma_i * \sigma_j)$ no es un subfactor de A_1, usando la misma argumentación que el caso anterior. En los dos casos se observa que $wt(C\sigma_j) > wt(C)$, lo que contradice la hipótesis de maximalidad de C. Entonces para cualquier otra factorización $P = AB$, se tiene que A es un subfactor de A_1.

Si existe otra factorización ponderada por la izquierda $P = AB$, se escribe $A_1 = AQ$, con $Q \in S_n^+$. Si $Q = \epsilon$, $A = A_1$, o se selecciona $i \in S(Q)$. Se tiene que $i \notin F(A)$ por el lema 19 ya que $A\sigma_i \leq A_1 \in S_n^+$. Pero $B = QP_1$, entonces $i \in S(B)$, la factorización $P = AB$ no es ponderada por la izquierda. Por lo tanto la factorización ponderada por la izquierda $P = A_1P_1$ es única. □

Corolario 24. *Sea $P \in B_n^+$ con factorización ponderada por la izquierda $P = A_1P_1$, $A_1 \in S_n^+$. Entonces $S(A_1) = S(P)$.*

Demostración. Como $P = A_1P_1$, si $i \in S(A_1)$, A_1 se puede escribir como $A_1 = \sigma_i A'$ y entonces $\sigma_i A'P_1 = A_1P_1 = P$ e $i \in S(P)$. Por lo tanto $S(A_1) \subset S(P)$. Por otro lado, si $i \in S(P)$. Se tiene que $P = \sigma_i B$, con $B \in B_n^+$. Por el teorema anterior $A_1 = \sigma_i Q$ con $Q \geq \epsilon$, entonces $i \in S(A_1)$. □

Ahora se presenta la forma canónica de una trenza positiva.

Teorema 25. *Existe una expresión única para $P \in B_n^+$ como $P = A_1A_2 \ldots A_k$ con $A_i \in [0, 1]$, $A_k \neq \epsilon$ y $S(A_{i+1}) \subset F(A_i)$ para cada i.*

Demostración. Sea $P_0 = P$, entonces se obtiene la factorización única ponderada por la izquierda $P_0 = A_1P_1$, de la misma forma se obtiene $P_1 = A_2P_2$ la factorización única

ponderada por la izquierda de P_1 y así sucesivamente. El proceso termina cuando los dos elementos de la factorización pertenecen a $[0,1]$. Se observa por el lema anterior que $S(Pi) = S(A_{i+1})$ y $S(P_i) \subset F(A_i)$, por lo que la factorización es ponderada por la izquierda. □

Para una trenza $P \in B_n^+$ esta factorización es conocida como la factorización canónica izquierda.

Definición (Forma canónica de Garside)**.** La forma canónica de Garside de una trenza B, es una factorización de la forma:

$$B = (\Delta_n)^s A_1 A_2 \dots A_k,$$

donde $s \in \mathbb{N}$, y $A_1 A_2 \dots A_k$, con $A_1 \neq \Delta_n$, es la factorización canónica de una trenza positiva. Cada uno de los factores A_i es llamado factor canónico de B.

Si B es positiva, la forma canónica de Garside resulta ser igual a la forma canónica por la izquierda. En otro caso, primero se cambian los generadores con signo negativo de la siguiente forma $\sigma_i^{-1} = \Delta_n^{-1} B$, $B \in B_n^+$ utilizando las propiedades del teorema 11, y las propiedades de la trenza fundamental se agrupan todos los elementos Δ_n^{-1} a la izquierda, de manera que se obtiene $\Delta_n^{-1} C$, con C una trenza positiva. Después se utiliza la factorización canónica por la izquierda de C y se obtiene la forma canónica de Garside.
Antes de concluir la sección se demuestra un resultado que será de útilidad para la resolución del problema de conjugación en la proxima sección.

Lema 26. *Sea $P \geq \epsilon$ una trenza con una factorización ponderada por la izquierda $P = A_1 P_1$, con $A_1 \in [0,1]$. Si $B \geq \epsilon$ y $BP \geq \Delta_n$, entonces $BA_1 \geq \Delta_n$.*

Demostración. La prueba se hace por inducción sobre $wt(B)$. Se considera el caso $wt(B) = 0$ como base, de donde se sigue que $B = \epsilon$ y $P \geq \Delta_n$. Entonces se puede escribir $P = \Delta_n Q$, con $Q \geq \epsilon$; de aquí que $S(P) = \{1, \dots, n-1\}$ y por el corolario 24, $S(A_1) = \{1, \dots, n-1\}$ y se concluye por el lema 22 que $S(A_1) = \Delta_n$, luego $A_1 \geq \Delta_n$.
Se supone que la propiedad se cumple para algún $k \in \mathbb{N}$, entonces en el caso $wt(B) = k+1$ se puede escribir $B = B'\sigma_i$ y $P' = \sigma_i P$ con $B', P' \geq \epsilon$ y $1 \leq i \leq n-1$. Se aplica la hipótesis de inducción a B' y P' para obtener $B' A_1' \geq \Delta_n$, donde A_1' es el primer

factor de la factorización ponderada izquierda $P' = A'_1 P'_1$. Así se tiene que $i \in S(P')$ y entonces por 24 $i \in S(A')$. Se escribe $A'_1 = \sigma_i A''_1$. Como $A'_1 \in [0,1]$ se sigue que $A''_1 \in [0,1]$ y se tiene que $P = A''_1 P'_1$. Por el lema 23 se tiene que $A'_1 = A''_1 Q$ para algún $Q \geq \epsilon$. Entonces $BA''_1 = B'A'_1 \geq \Delta_n$ de donde se concluye que $BA_1 = BA''_1 Q \geq \Delta_n$ y queda demostrado el lema. □

Como consecuencia de este lema, si P es una trenza positiva con factorización canónica $P = A_1 A_2 \dots A_k$ se tiene que $P \geq \Delta_n$ si, y sólo si $A_1 = \Delta_n$ Si se aplica sucesivamente esta observación se tiene que $inf(P) = max\{i : A_i = \Delta_n\}$.

Teorema 27. *Sea $P \geq \epsilon$ con forma canónica $P = A_1 A_2 \dots A_k$. Entonces $sup(P) = k$*

Demostración. La demostración se hace por inducción sobre k, el número de factores canónicos en la forma canónica de P. Para el caso $k = 0$ se tiene $P = \epsilon$, entonces $sup(P) = 0$ por definición.
Supóngase que la propiedad se cumple para algún $k \geq 0$ y se considera ahora el caso $P = A_1 A_2 \dots A_{k+1}$. Sea $s = sup(P)$, por el corolario 14 se tiene que $P \in [0, k+1]$, luego $s \leq k+1$. Entonces se puede encontrar $B \geq \epsilon$ tal que $BP = \Delta_n^s$, por el lema 26 se tiene que $BA_1 \geq \Delta_n$. Entonces se obtiene $B_1 \geq \epsilon$ tal que $BA_1 = \Delta_n B_1$. Ahora, $\Delta_n^s = BP = \Delta_n B_1 P_1$ con $P_1 = A_2 \dots A_{k+1}$, entonces $B_1 P_1 = \Delta_n^{s-1}$ y como B_1 y P_1 son trenzas positivas se tiene que $P_1 \leq \Delta_n^{s-1}$. Se aplica la hipotesis de inducción a P_1 y se tiene que $sup(P_1) = k$ y $k \leq s-1$ y se concluye $sup(P) = s = k+1$ lo que termina la demostración. □

Para una trenza $P \geq \Delta_n^r$ no necesariamente positiva, se puede obtener $P' = \Delta_n^{-r} P \geq \epsilon$ y así $sup(P) = r + sup(P')$ y $inf(P) = r + inf(P')$ que se pueden calcular de la forma canónica de P'. Estos resultados se resumen en el siguiente teorema.

Teorema 28. *Sea $P \in B_n$ y $P = \Delta_n^r A_1 A_2 \dots A_k$ la forma canónica de Garside de P, entonces:*

$$\begin{aligned} inf(P) &= r, \\ sup(P) &= r + k. \end{aligned}$$

Demostración. Se tiene que $A_1 \neq \Delta_n$, entonces $inf(P) = r + inf(A_1 \dots A_k) = r$ por la observación que se hizo sobre el lema 26. Por el teorema 27 se sigue que $sup(P) = r + sup(A_1 A_2 \dots A_k) = r + k$. □

3.5. Demostraciones

A manera de referencia se enunciarán otra vez las propiedades del teorema 11.

Teorema. *La trenza fundamental Δ_n tiene las siguientes propiedades:*

- *Para cada generador σ_i, $\Delta_n = \sigma_i A = B\sigma_i$ con $A, B \in B_n^+$.*
- *Para todo generador σ_i, se tiene que $\sigma_i \Delta_n = \Delta_n \sigma_{n-i}$.*
- *El centro del grupo B_n es el subgrupo cíclico generado por Δ_n^2. Es decir, $C(B_n) =< \Delta_n^2 >$.*

Primero se demostrará la primera propiedad. Se darán expresiones explícitas para las trenzas positivas que cumplen con la propiedad y se comenzará con la primera igualdad $\Delta_n = \sigma_i A$. Se comienza con un lema.

Lema 29. *Sea $1 < i \leq j$, entonces*

$$\sigma_i(\sigma_1 \dots \sigma_j) = (\sigma_1 \dots \sigma_j)\sigma_{i-1}.$$

Demostración. Utilizando la propiedad 3.2 en el lado izquierdo se tiene:

$$\begin{aligned} \sigma_i(\sigma_1 \dots \sigma_j) &= \sigma_i \sigma_1 \dots \sigma_j \\ &= \sigma_1 \dots \sigma_{i-2} \sigma_i \sigma_{i-1} \sigma_i \dots \sigma_j. \end{aligned}$$

Utilizando las propiedades 3.2 y 3.3

$$\begin{aligned} \sigma_1 \dots \sigma_{i-1} \sigma_i \sigma_{i-1} \sigma_{i+1} \dots \sigma_j &= (\sigma_1 \dots \sigma_i)(\sigma_{i-1} \sigma_{i+1} \dots \sigma_j) \\ &= (\sigma_1 \dots \sigma_i)(\sigma_{i+1} \dots \sigma_j)\sigma_{i-1} \\ &= (\sigma_1 \dots \sigma_j)\sigma_{i-1}. \end{aligned}$$

□

La interpretación intuitiva del lema es que un generador y un bloque ordenado de trenzas "conmuta" de manera especial, reduciendo en una unidad el índice del generador.
Ahora se demostrarán los casos triviales del teorema.

Demostración. Si $i = 1$, sea $A = (\sigma_2 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1\sigma_2)\sigma_1$. Por definición $\sigma_i A = \Delta_n$. Si $i = n-1$, sea $A = (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1\sigma_2)$.
Entonces:

$$\sigma_{n-1}A = \sigma_{n-1}(\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1\sigma_2)$$

y aplicando el lema 29 sucesivamente a cada uno de los bloques ordenados de generadores se obtiene el Δ_n. □

Como se puede observar, en ambos casos A es similar a la trenza fundamental pero con un generador σ_1 removido del primer o último bloque, respectivamente. Con la misma idea se probarán los casos restantes.

Demostración. Si $1 < i < n-1$, sea

$$A = (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-(i-1)})(\sigma_2 \dots \sigma_{n-i}) \dots \sigma_1.$$

Como se mencionó antes, el elemento A es igual que la trenza fundamental pero quitando el generador σ_1 del i-ésimo bloque. Ahora, utilizando el lema 29 y la propiedad 3.2

$$\begin{aligned}
\sigma_i A &= \sigma_i(\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-i+1})(\sigma_2 \dots \sigma_{n-i}) \dots \sigma_1 \\
&= (\sigma_1 \dots \sigma_{n-1})\sigma_{i-1}(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-i+1})(\sigma_2 \dots \sigma_{n-i}) \dots \sigma_1 \\
&= (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2})\sigma_{i-2} \dots (\sigma_1 \dots \sigma_{n-i+1})(\sigma_2 \dots \sigma_{n-i}) \dots \sigma_1 \\
&\vdots \\
&= (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-i+1})\sigma_{i-i+1}(\sigma_2 \dots \sigma_{n-i}) \dots \sigma_1 \\
&= (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-i+1})(\sigma_1 \dots \sigma_{n-i}) \dots \sigma_1 \\
&= \Delta_n.
\end{aligned}$$

□

Para demostrar la siguiente parte de la primera propiedad $\Delta_n = B\sigma_i$ también se comenzará con un lema técnico.

Lema 30. *Sea σ_i un generador de Artin con $i \geq 3$. Entonces*

$$\begin{aligned}(\sigma_2 \dots \sigma_i)(\sigma_1 \dots \sigma_{i-1})\sigma_i &= \sigma_1(\sigma_2\sigma_1)(\sigma_3\sigma_2)(\sigma_4\sigma_3)\dots(\sigma_i\sigma_{i-1}) \\ &= (\sigma_1 \dots \sigma_i)(\sigma_1 \dots \sigma_{i-1}).\end{aligned}$$

Demostración. La demostración se hace por inducción, se comenzará con el caso base $i = 3$.

$$\begin{aligned}(\sigma_2\sigma_3)(\sigma_1\sigma_2)\sigma_3 &= \sigma_2\sigma_1\sigma_3\sigma_2\sigma_3 \\ &= (\sigma_2\sigma_1\sigma_2)(\sigma_3\sigma_2) \\ &= \sigma_1(\sigma_2\sigma_1)(\sigma_3\sigma_2) \\ &= (\sigma_1\sigma_2\sigma_3)(\sigma_1\sigma_2).\end{aligned}$$

Ahora, se considera que el resultado es válido para $k \in \mathbb{N}$ con $k \geq 3$, y se probará el resultado para $k + 1$.

$$\begin{aligned}&(\sigma_2 \dots \sigma_k\sigma_{k+1})(\sigma_1 \dots \sigma_k)\sigma_{k+1} \\ = \;&(\sigma_2 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1})(\sigma_{k+1}\sigma_k\sigma_{k+1}) \\ = \;&(\sigma_2 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1})\sigma_k(\sigma_{k+1}\sigma_k)\end{aligned}$$

Y se utiliza la hipótesis de inducción y la propiedad 3.2 para obtener que lo anterior es igual a:

$$\begin{aligned}&\sigma_1(\sigma_2\sigma_1)(\sigma_3\sigma_2)\dots(\sigma_k\sigma_{k-1})(\sigma_{k+1}\sigma_k) \\ = \;&(\sigma_1 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1})\sigma_{k+1}\sigma_k \\ = \;&(\sigma_1 \dots \sigma_{k+1})(\sigma_1 \dots \sigma_k)\end{aligned}$$

Entonces se concluye por inducción matemática que el resultado es válido para todo $i \geq 3$. □

Ahora se procederá a probar la segunda parte de la primera propiedad, i.e. $\Delta_n = B\sigma_i$.

Demostración. Sea

$$B = (\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_2 \dots \sigma_i)(\sigma_1 \dots \sigma_{i-1}) \dots \sigma_1.$$

Entonces, utilizando el lema 30 y las propiedades básicas se desarrolla la expresión $B\sigma_i$:

$$\begin{aligned} & (\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_2 \dots \sigma_i)(\sigma_1 \dots \sigma_{i-1}) \dots \sigma_1\sigma_i \\ = \; & (\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_2 \dots \sigma_i)(\sigma_1 \dots \sigma_{i-1})\sigma_i(\sigma_1 \dots \sigma_{i-2}) \dots \sigma_1 \\ = \; & (\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_1 \dots \sigma_i)(\sigma_1 \dots \sigma_{i-1})(\sigma_1 \dots \sigma_{i-2}) \dots \sigma_1 \\ = \; & \Delta_n \end{aligned}$$

□

Para probar la segunda propiedad también se hará uso de un lema técnico.

Lema 31. *Sea* $2 \leq j \leq n-1$*, entonces*

$$\sigma_1(\sigma_1 \dots \sigma_j)(\sigma_1 \dots \sigma_{j-1}) \dots \sigma_1 = (\sigma_1 \dots \sigma_j)(\sigma_1 \dots \sigma_{j-1}) \dots \sigma_1\sigma_j$$

Demostración. Una vez más la demostración se hará por inducción. Se comienza con el caso $j = 2$.

$$\begin{aligned} & \sigma_1(\sigma_1\sigma_2)\sigma_1 \\ = \; & (\sigma_1\sigma_2)\sigma_1\sigma_2 \end{aligned}$$

Solamente se utilizó la propiedad 3.3 del grupo de trenzas. Entonces, se supondrá como válido el resultado para $k \in \mathbb{N}$, $k \geq 2$ y se demostrará para $k+1$.

$$\begin{aligned} & \sigma_1(\sigma_1 \dots \sigma_{k+1})(\sigma_1 \dots \sigma_k) \dots \sigma_1 \\ = \; & \sigma_1(\sigma_1 \dots \sigma_k)(\sigma_1 \dots \sigma_{k+1}\sigma_k)(\sigma_1 \dots \sigma_{k-1}) \dots \sigma_1 \end{aligned}$$

En este caso se puede observar que el generador σ_{k+1}, usando la propiedad 3.2, "conmutó" con los elementos que le siguen hasta quedar en la posición anterior al elemento

σ_k. De la misma forma, el elemento $\sigma_{k+1}\sigma_k$ "conmuta" con los siguientes elementos hasta ocupar la posición anterior al elemento σ_{k-1}. Continuando este proceso y aplicando la hipótesis de inducción se obtiene:

$$\begin{aligned} & (\sigma_1(\sigma_1 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1}) \dots \sigma_1)(\sigma_{k+1}\sigma_k \dots \sigma_1) \\ = \; & (\sigma_1 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1}) \dots \sigma_1\sigma_k(\sigma_{k+1}\sigma_k \dots \sigma_1) \end{aligned}$$

Ahora se puede aplicar la propiedad 3.3 a los elementos $\sigma_k\sigma_{k+1}\sigma_k$ y utilizar la propiedad 3.2 de manera inversa para colocar a cada elemento en la posición deseada.

$$\begin{aligned} & (\sigma_1 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1}) \dots \sigma_1\sigma_{k+1}\sigma_k\sigma_{k+1} \dots \sigma_1 \\ = \; & (\sigma_1 \dots \sigma_k)(\sigma_1 \dots \sigma_{k-1}) \dots \sigma_1\sigma_{k+1}\sigma_k \dots \sigma_1\sigma_{k+1} \\ = \; & (\sigma_1 \dots \sigma_{k+1})(\sigma_1 \dots \sigma_k) \dots \sigma_1\sigma_{k+1} \end{aligned}$$

De esta manera se concluye, por inducción, que el resultado es válido para toda $2 \leq j \leq n-1$. □

La explicación intuitiva de este lema es que el elemento σ_1 "conmuta" con los j bloques ordenados pero al hacerlo, su índice se incrementa $j-1$ veces y se convierte en el elemento σ_j. Con el resultado anterior se puede demostrar la segunda propiedad de la trenza fundamental, i.e. $\sigma_i\Delta_n = \Delta_n\sigma_{n-i}$

Demostración. Se utilizará el lema 29 de manera sucesiva en el elemento σ_i para $i > 1$ y luego el lema 31.

$$\begin{aligned} & \sigma_i\Delta_n \\ = \; & \sigma_i(\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots \sigma_1 \\ = \; & (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-i+1})\sigma_1(\sigma_1 \dots \sigma_{n-i}) \dots \sigma_1 \\ = \; & (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1 \dots \sigma_{n-i+1})(\sigma_1 \dots \sigma_{n-i}) \dots \sigma_1\sigma_{n-i} \\ = \; & \Delta_n\sigma_{n-i} \end{aligned}$$

El caso $i = 1$ es una aplicación directa del lema 31. □

Para la tercera propiedad, sólo se probará que los elementos generados por el elemento Δ_n^2 son parte del centro de B_n. La prueba se hará probando que los generadores de Artin conmutan con Δ_n^2.

Demostración. Sea σ_i una generador de Artin, entonces

$$\sigma_i \Delta_n^2 = \sigma_i \Delta_n \Delta_n = \Delta_n \sigma_{n-i} \Delta_n = \Delta_n^2 \sigma_i$$

Lo anterior es consecuencia de la segunda propiedad de este teorema. □

Capítulo 4

Criptografía

Los sistemas criptográficos que se exponen en esta sección utilizan la teoría de los grupos de trenzas para su desarrollo. En particular utilizan la aparente dificultad del problema de conjugación para hacer que los sistemas sean difíciles de romper. Como se muestra en este capítulo, el problema de la conjugación tiene una solución, pero puede no ser computacionalmente eficiente para ciertas instancias del problema, lo que hace a los sistemas seguros para ciertas selecciones de parámetros.

4.1. Problemas algorítmicos

En un sistema criptográfico es importante tener un proceso en el que si se tienen todos los datos se pueda resolver fácilmente, pero que en ausencia de algunos de ellos no exista una solución sencilla o computacionalmente eficiente. Estos procesos usualmente se conocen como problemas algorítmicos, problemas en los que se busca no sólo la solución, sino una implementación eficiente de la misma. En otas palabras, un problema algorítmico es un problema que es computable (existen algoritmos que solucionan el problema) pero la complejidad de las soluciones conocidas hacen que no sea factible resolver ciertas instancias de los problemas.
Los dos tipos de problemas algorítmicos que se utilizan en la criptografía con teoría de grupos como se describen en [18] son:

- Problemas de decisión, en los que se tiene un objecto $\mathcal{O}$ y una propiedad $\mathcal{P}$ y se decide si el objeto $\mathcal{O}$ tiene o no la propiedad $\mathcal{P}$.

- Problemas de búsqueda, en donde se tiene una propiedad $\mathcal{P}$ y se sabe que existen objetos que tienen la propiedad $\mathcal{P}$ y se debe encontrar al menos uno que cumpla con la propiedad.

El criptosistema RSA utiliza el problema de la factorización de números enteros que son productos de dos números primos. En ese problema se pueden ver los dos tipos de problemas algorítmicos que se acaban de describir.

- Problema de decisión. Dado un número entero n, decir si es primo o no (o equivalentemente, decir si es un número compuesto o no).

- Problema de búsqueda. Dado un número entero n compuesto con dos factores primos, encontrar los p y q enteros primos tales que $n = pq$.

Los dos problemas tienen soluciones algorítmicas conocidas, pero no son eficientes para todas las instancias del problema. En particular no hay una solución computacionalmente eficiente al problema de búsqueda para todas las intancias, esto permite que el sistema criptográfico RSA sea seguro para elecciones correctas de parámetros.
Para los grupos libres se consideran dos tipos de problemas algorítmicos en el presente trabajo. De manera general se puede decir que la dificultad de resolver cada problema algorítmico depende del grupo en el que se esté trabajando y las características del mismo.

4.1.1. El problema de la palabra

El problema de la palabra (WP por su nombre en inglés Word Problem) consiste en: dada una presentación de un grupo G y un elemento $g \in G$, determinar si $g = 1$ en G. El nombre del problema viene del hecho de que se determina si un elemento o palabra del grupo libre G es equivalente a la palabra vacía en G. Esta versión del problema está dada en forma de problema de decisión, ya que la respuesta es *sí* o *no*, algunas veces ambas partes son consideradas por separado y se habla de la parte *sí* o afirmativa del problema de la palabra y la parte *no* o negativa del mismo.
El problema de la palabra también puede formularse como problema de búsqueda y se conoce como el problema de búsqueda de la palabra (o WSP por su nombre en inglés Word Search Problem). Dada una presentación de grupo G y un elemento $g \in G$ tal que

$g = 1$ en G, se busca una presentación del elemento g como producto de las relaciones que son equivalentes a 1 en la presentación del grupo y sus inversos.
En el grupo de trenzas la existencia de la forma canónica de Garside da una solución al problema de la palabra ya que la forma canónica de toda trenza $B = \epsilon$ de algún grupo de trenzas B_n es Δ_n^0 y el cálculo de la forma canónica de Garside se puede realizar de forma eficiente en una computadora. Por esta razón no se pueden definir sistemas criptográficos en el grupo de trenzas que utilicen como base el problema de la palabra.

4.1.2. El problema de conjugación

Dada una presentación de grupo G y una pareja de elementos $a, b \in G$, el problema de la conjugación (CP por su nombre en inglés Conjugacy Problem) consiste en decidir si los elementos forman una pareja conjugada o no, es decir, si existe un elemento $x \in G$ tal que $a = xbx^{-1}$. Este también se trata de un problema de decisión ya que la respuesta del problema es un "sí" o "no".
El problema de conjugación también se puede formular como un problema de búsqueda como sigue: dada una presentación de grupo G y $a, b \in G$ una pareja de elementos que se sabe forman una pareja conjugada, encontrar un elemento particular $x \in G$ tal que $a = xbx^{-1}$. Esta forma del problema se conoce como el problema de búsqueda de conjugación (CSP por su nombre en inglés Conjugacy Search Problem) y siempre tiene una solución ya que se pueden enumerar los conjugados de un elemento, sin embargo, este tipo de solución puede ser ineficiente debido a que los grupos usados suelen ser infinitos.
Para el caso particular del grupo de trenzas existe otra solución al problema de conjugación. Los detalles de esa solución se estudian en la sección correspondiente del presente capítulo. En este capítulo se habla de la *aparente* dificultad del problema de conjugación en el grupo de trenzas, el uso del adjetivo *aparente* responde a que la solución del problema con los método conocidos es computacionalmente costoso; sin embargo, queda la posibilidad de que aparezcan método más eficientes de solución que pudieran hacer que el problema de conjugación no sea apto para usarse como base para sistemas criptográficos.

4.2. Criptografía con grupos de trenzas

Ahora se describen dos sistemas criptográficos basados en los grupos de trenzas, el primero es un esquema de intercambio de llaves que fue propuesto en [1] por Anshel, Anshel y Goldfel de manera general en grupos libres y en [2] por Anshel, Anshel, Fisher y Goldfel. Este sistema utiliza la aparente dificultad del problema de conjugación como base para darle seguridad al sistema.

Por otra parte se estudia un esquema de intercambio de llaves inspirado en el protocolo de Diffie Hellman que utiliza la conmutatividad de algunos subgrupos del grupo de trenzas. El sistema fue propuesto por Ko et al en [14]. La seguridad del sistema está basada en una variante del problema de conjugación. El desarrollo de los dos esquemas sigue la línea de [9], [5] y [16].

Como los dos sistemas son esquemas de intercambio de llaves, no son suficientes por sí mismos para construir un sistema criptográfico completo y para encriptar se puede usar una función hash como se menciona en el capítulo 2 o usar un cifrado simétrico con la llave común que se obtuvo. En la implementación se utiliza el cifrado simétrico AES, aunque podrían utilizarse otros sistemas tales como DES o ElGamal.

4.2.1. El esquema de Anshel, Anshel y Goldfel

El esquema de Anshel, Anshel y Goldfel es un esquema de intercambio de llaves, por lo que para usarlo como un sistema criptográfico completo es necesario tener un sistema criptográfico simétrico que se usará para enviar el mensaje utilizando la llave que se definió previamente. En esta sección los participantes del intercambio se denotarán como A y B.

Para el esquema se considera el grupo de trenzas con n hilos B_n. También se utilizan palabras formadas por los elementos y los inversos de un conjunto de r elementos $X = \{x_1, x_2, \ldots, x_r\}$. Si u es una palabra así y $\{p_1, p_2, \ldots, p_r\}$ un conjunto de trenzas de B_n, entonces se denota como $u(p_1, p_2, \ldots, p_r)$ la sustitución del elemento x_i en la palabra por la trenza p_i del conjunto, para $1 \leq i \leq r$. Entonces se describe el esquema como sigue:

Esquema

- Los elementos públicos son $P = \{p_1, p_2, \ldots, p_k\}$ y $Q = \{q_1, q_2, \ldots, q_m\}$, dos conjunto de trenzas.
- El elemento privado de A es una palabra u en un alfabeto de k elementos, el de B es una palabra v en un alfabeto de m elementos.
- A calcula $s = u(p_1, \ldots, p_k)$, y envía a B los elementos $q_1' = sq_1s^{-1}$, ..., $q_m' = sq_ms^{-1}$.
- B calcula $r = v(q_1, \ldots, q_m)$ y envía a A los elementos $p_1' = rp_1r^{-1}$, ..., $p_k' = rp_kr^{-1}$.
- A calcula $t_A = su(p_1', p_2', \ldots, p_k')^{-1}$.
- B calcula $t_B = v(q_1', q_2', \ldots, q_m')r^{-1}$.
- La llave común $t_B = t_A$.

La igualdad de la llave común se verifica por lo siguiente:

$$\begin{aligned} t_A &= su(p_1', \ldots, p_k')^{-1} = sru(p_1, \ldots, p_k)^{-1}r^{-1} = srs^{-1}r^{-1} \\ &= sv(q_1, q_2, \ldots, q_m)s^{-1}r^{-1} = v(q_1', q_2', \ldots, q_m')r^{-1} = t_B \end{aligned}$$

Lo anterior se debe a la propiedad que se enuncia en el siguiente lema.

Lema 32. *Sean u un alfabeto de m elementos y $p_1, p_2, \ldots, p_m \in B_n$. Si s es un elemento de B_m y se define $p_1' = sp_1s^{-1}, \ldots, p_m' = sp_ms^{-1}$, entonces $u(p_1', p_2', \ldots, p_m') = su(p_1, p_2, \ldots, p_m)s^{-1}$.*

Demostración. La prueba se hace por inducción sobre la longitud de la palabra u. Para la base de inducción se considera una palabra de longitud 1. Entonces se tienen dos casos $u = x_i$ y $u = x_i^{-1}$, con $1 \leq i \leq m$. En el primer caso $u(p_1', \ldots, p_m') = sp_is^{-1} = su(p_1, \ldots, p_m)s^{-1}$, y en el segundo caso $u(p_1', \ldots, p_m') = (sp_is^{-1})^{-1} = sp_i^{-1}s^{-1} = su(p_1, \ldots, p_m)s^{-1}$. Como hipótesis de inducción se supone que existe $k \in \mathbb{N}$ tal que si u es una palabra de longitud k, se tiene que $u(p_1', \ldots, p_m') = su(p_1, \ldots, p_m)s^{-1}$. Si u es una palabra de longitud $k+1$, se define u_k como la palabra formada por los primeros

k elementos de u. Entonces $u = u_k x_i$ o $u = u_k x_i^{-1}$ para alguna $1 \leq i \leq m$. Para el primer caso se tiene:

$$\begin{aligned} u(p'_1, \ldots, p'_m) &= u_k(p'_1, \ldots, p'_m) s p_i s^{-1} \\ &= s u_k(p_1, \ldots, p_m) s^{-1} s p_i s^{-1} \\ &= s(u_k(p_1, \ldots, p_m) p_i) s^{-1} \\ &= s u(p_1, \ldots, p_m) s^{-1} \end{aligned}$$

Y el otro caso:

$$\begin{aligned} u(p'_1, \ldots, p'_m) &= u_k(p'_1, \ldots, p'_m)(s p_i s^{-1})^{-1} \\ &= s u_k(p_1, \ldots, p_m) s^{-1} s p_i^{-1} s^{-1} \\ &= s(u_k(p_1, \ldots, p_m) p_i^{-1}) s^{-1} \\ &= s u(p_1, \ldots, p_m) s^{-1} \end{aligned}$$

Se tiene entonces que la propiedad se cumple para palabras de longitud $k+1$, y queda demostrado el lema por inducción. □

Para que una tercera parte pudiera obtener la llave común de A y B tendría que resolver una variante del problema de conjugación, conocida como el problema de conjugación múltiple. En esta variante se tiene un conjunto finito de parejas de trenzas conjugadas (pi, pi') del que se sabe tienen la misma trenza como conjugador para cada una de las parejas y se debe encontrar dicho elemento conjugador. No se sabe si el problema de la conjugación múltiple es más sencillo que el problema de la conjugación regular, pero resolviendo el problema de conjugación se resuelve el problema de conjugación múltiple. En [2], los autores sugieren trabajar en B_{80} con conjuntos de 20 trenzas y con trenzas de 5 o 10 generadores.

4.2.2. Un esquema inspirado en Diffie-Hellman

En [6] se revisó el esquema de intercambio de llaves de Diffie-Hellman, en el que se utiliza la propiedad $(g^a)^b = (g^b)^a$ donde g es el generador de un grupo cíclico finito y $a, b \in \mathbb{N}$. En [14], Ko et al proponen un esquema inspirado en el de Diffie-Hellman para grupos de trenzas, en lugar de utilizar un generador de grupo cíclico se utiliza una trenza arbitraria de un grupo de trenzas y como operación se utiliza la conjugación.

Los grupos de trenzas no son conmutativos, pero es posible encontrar subconjuntos de trenzas que conmutan entre ellos. Si se considera el grupo de trenzas B_n se define LB_n como el subgrupo de B_n generado por $\sigma_1, \ldots, \sigma_{m-1}$ y UB_n el subgrupo de B_n generado por $\sigma_{m+1}, \sigma_{m+2}, \ldots, \sigma_{n-1}$. Los elementos de LB_n conmutan con los elementos de UB_n ya que para cualquier pareja de trenzas $B \in LB_n$ y $C \in UB_n$, cualesquier generadores σ_i de B y σ_j de C cumplen $|i - j| > 1$. El esquema es como sigue:

Esquema

- La llave pública es una trenza p de B_n.
- La llave privada de A es una trenza $s \in LB_n$, la llave privada de B es una trenza $r \in UB_n$.
- A manda a B $p' = sps^{-1}$.
- B manda a A $p'' = rpr^{-1}$.
- La llave común es $K = srpr^{-1}s^{-1}$.

Como r y s conmutan, A puede calcular $K = sp''s^{-1}$ y B puede calcular, $K = rp'r^{-1}$, y así K sirve como llave común a las dos partes. La seguridad del esquema está basado en el problema de conjugación y a una variante conocida como el problema de conjugación de Diffie-Hellman. El problema consiste en: dada una trenza p en B_n, y las trenzas $p' = sps^{-1}$ y $p'' = rpr^{-1}$, con $s \in LB_n$ y $r \in UB_n$, encontrar la trenza $rp'r^{-1} = sp''s^{-1}$. Se sugiere trabajar en B_{80}, con trenzas formadas por al menos 12 factores canónicos. En el capítulo de pruebas se muestran los resultados de hacer pruebas de eficiencia con los parámetros sugeridos para los dos sistemas.

4.3. Solución al problema de conjugación en B_n

En esta sección se presenta una solución al problema de búsqueda de conjugación. A partir de esta solución se puede desarrollar ataques a los sistemas descritos anteriormente, por lo que es importante estudiar y probar la eficiencia de los mismos en la solución del problema de conjugación. Si se tienen dos trenzas P y Q que se sabe son conjugadas, la solución consiste en buscar todos los conjugados de P en un intervalo $[r, s]$, se busca

que r sea máximo y s sea mínimo, hasta encontrar a Q. El algoritmo consiste en obtener la lista completa de conjugados utilizando un proceso finito de conjugación por trenzas en $[0,1]$. El algoritmo fue presentado en [7] y aquí se sigue la misma línea de desarrollo. Se comienza demostrando un teorema que sirve como base para el algoritmo.

Teorema 33. *Sean $P, Q \geq \Delta_n^r$ conjugadas. Si se supone que $Q = A^{-1}PA$ para alguna $A \geq \epsilon$, entonces $A_1^{-1}PA_1 \geq \Delta_n^r$ donde $A_1 \in [0,1]$ es el primer factor en la forma canónica de A.*

Demostración. Se puede suponer $A \geq \epsilon$ sin perdida de generalidad ya que si $Q = B^{-1}PB$, con $B \in B_n$ se puede encontrar $j \in \mathbb{Z}$ tal que $B \geq \Delta_n^{2j}$ y entonces se tiene que $B = \Delta_n^{2j}A$ para alguna $A \geq \epsilon$ y

$$B^{-1}PB = A^{-1}\Delta_n^{-2j}P\Delta_n^{2j}A = A^{-1}PA\Delta_n^{-2j}\Delta_n^{2j} = A^{-1}PA = Q.$$

Sólo es necesario probar el teorema para los casos $r = 0$ y $r = 1$ ya que se pueden considerar las trenzas $P' = \Delta_n^{-2j}P$ y $Q' = \Delta_n^{-2j}Q$ con $j \in \mathbb{N}$ el mayor entero tal que $2j \leq r$. Se considera la forma canónica de A como $A = A_1A'$, donde A_1 es el primer factor canónico y el factor A' es el producto de los factores canónicos que siguen a A_1, entonces $A_1 \in [0,1]$ y $A' \geq \epsilon$.
Para el caso $r = 0$ se tiene que $A^{-1}PA = Q$ y $P, Q \geq \epsilon$ y se debe demostrar que $A_1^{-1}PA_1 \geq \epsilon$ con $A_1 \in [0,1]$. Se define $\Delta_n A_1^{-1} = A_1^* \in S_n^+$ y se tiene

$$A_1^*PA = A_1^*AQ = A_1^*A_1A'Q = \Delta_n A_1^{-1}A_1A'Q = \Delta_n A'Q \geq \Delta_n.$$

Ahora se aplica el lema 26 a A_1^*PA, con A_1^*P en el lugar de B y A en el lugar de P para concluir $A_1^*PA_1 \geq \Delta_n$, de donde se sigue que $A_1^{-1}PA_1 = \Delta_n^{-1}A_1^*PA_1 \geq \epsilon$.
Para el caso $r = 1$ se tiene que $P, Q \geq \Delta_n$. Se escriben $P = P'\Delta_n$, $Q = Q'\Delta_n$ y $A = A_1A'$ y $A^* = \Delta_n A_1^{-1}$ como en el caso anterior. Ahora como τ es un homomorfismo se tiene que $\tau(A) = \tau(A_1)\tau(A')$ y como $S(\Delta_n) = S(\Delta_n^{-1}) = F(\Delta_n) = S(\Delta_n^{-1})$ se tiene que $\tau(A_1)\tau(A')$ es una factorización ponderada por la izquierda de $\tau(A)$, con $\tau(A_1) \in [0,1]$. Como en el caso anterior se tiene que $A_1^*PA = \Delta_n A'Q$, y

$$\Delta_n A'Q'\Delta_n = \Delta_n A'Q = A_1^*PA = A_1^*P'\Delta_n A = A_1^*P'\tau(A)\Delta_n,$$

entonces $A_1^* P' \tau(A) = \Delta_n A' Q' \geq \Delta_n$. Se aplica el lema 26 para concluir que $A_1^* P' \tau(A_1) \geq \Delta_n$. Se observa primero que:

$$\tau(A_1^{-1}) = \Delta_n^{-1} A_1^{-1} \Delta_n = \Delta_n^{-1} \Delta_n^{-1} A_1^* \Delta_n = \Delta_n^{-1} \tau(A_1^*)$$

y

$$\tau(P) = \Delta_n^{-1} P \Delta_n = \Delta_n^{-1} P' \Delta_n^2 = \Delta_n P'.$$

Luego:

$$\begin{aligned} \tau(A_1^{-1} P A_1) &= \tau(A_1^{-1}) \tau(P) \tau(A_1) \\ &= \Delta_n^{-1} \tau(A_1^*) \Delta_n P' \tau(A_1) \\ &= \tau(\tau(A_1^*)) P' \tau(A_1) \\ &= A_1^* P' \tau(A_1) \\ &\geq \Delta_n \end{aligned}$$

Y aplicando τ a ambos lados de la ecuación se obtiene que $A_1^{-1} P A_1 \geq \Delta_n$ y queda demostrado el teorema. □

El siguiente corolario muestra que si se tienen dos trenzas conjugadas P y Q, se puede transformar una en la otra por medio de un proceso finito de conjugaciones con trenzas de $[0, 1]$.

Corolario 34. *Sean $P, Q \in [r, s]$ dos trenzas conjugadas. Entonces existe una sucesión finita $P = P_0, P_1, \ldots, P_k = Q$ de trenzas, todas en $[r, s]$ tal que cada elemento es conjugado del siguiente por medio de una trenza en $[0, 1]$.*

Demostración. Se escribe $A^{-1} P A = Q$ y se escribe a A en forma canónica como $A = A_1 A_2 \ldots A_m$. Se definen $P_i = A_i^{-1} P_{i-1} A_i$, con $P_0 = P$. La prueba se hace por inducción sobre el índice de P_i. La base $i = 0$ es trivial ya que $P_0 = P \in [r, s]$. La hipótesis de inducción es que existe $k \geq 0$ tal que $P_k \in [r, s]$. Para el caso $P_{k+1} = A_{k+1}^{-1} P_k A_{k+1}$, se considera $A' = A_{k+1} \ldots A_m$. Ahora se tiene que $A'^{-1} P_k A' = Q$ y se sigue por el teorema 33 que $P_{k+1} = A_{k+1}^{-1} P_k A_{k+1} \geq \Delta_n^r$. Ahora, $P_k^{-1}, Q^{-1} \geq \Delta_n^{-s}$ y $A'^{-1} P_k^{-1} A' = Q^{-1}$, y aplicando una vez más el teorema 33 se sigue que $P_{k+1}^{-1} = A_{k+1}^{-1} P_k^{-1} A_{k+1} \geq \Delta_n^{-s}$ y $P_{k+1} \leq \Delta_n^s$, lo que concluye la prueba. □

El algoritmo para resolver el problema de conjugación consiste en buscar todos los conjugados de una trenza en un subconjunto de su clase de conjugación, tal que el conjunto sea finito y los elementos se puedan obtener de manera eficiente.

Definición. El super conjunto cumbre (SSS por su nombre en inglés Super summit set) de una trenza P es el conjunto de conjugados P' de P con $inf(P')$ máximo y $sup(P')$ mínimo. Será denotado como $SSS(P)$.

El hecho de que el conjunto no sea vacío vendrá como resultado del desarrollo del algoritmo. Otra forma de definir el SSS de una trenza es como el conjunto de conjugados que tienen mínima longitud canónica.

4.3.1. Algoritmo

Para resolver el problema de la conjugación para dos trenzas P y Q sólo es necesario determinar el super conjunto cumbre de P y un elemento del super conjunto cumbre de Q y compararlos. Es suficiente esto ya que si P y Q son conjugados, $SSS(P) = SSS(Q)$, y si se tiene un trenza $sQs^{-1} \in SSS(Q)$, con $s \in B_n^+$ que además es un elemento de $SSS(P)$ se puede escribir como $sQs^{-1} = rPr^{-1}$ para alguna trenza $r \in B_n^+$, entonces se tiene que $r^{-1}sQs^{-1}r = P$ y se obtiene el elemento conjugador de las dos trenzas, por lo que también se resuelve también el problema de búsqueda.
Para encontrar a los elementos del $SSS(P)$, por el corolario 34 se puede conjugar P sucesivamente con elementos de $[0, 1]$, desechando los elementos para los cuales sup se incremente o inf disminuya.
Los siguientes resultados sirven para encontrar la potencia de la cumbre de P, es decir, el máximo valor del ínfimo en los elementos de la clase de conjugación de P.
Primero se define una operación sobre las trenzas que se define en términos de la factorización canónica.

Definición. Sea $P \in B_n$ una trenza y $P = \Delta_n^r A_1 A_2 \dots A_k$ su factorización canónica, donde $r = inf(P)$. Se tiene que $A_1 \neq \Delta_n$ y se define $c(P) = \Delta_n^r A_2 \dots A_k \tau^r(A_1)$, y se dice que $c(P)$ se obtiene ciclando a P.

También puede definirse $c(P)$ como $c(P) = \tau^{-r}(A_1) P \tau^r(A_1)$ y se observa que como $\tau^r(A_1)$ es una trenza positiva se tiene que $inf(c(P)) \geq inf(P)$ y $sup(c(P)) \leq sup(P)$.

Para verificar la primera desigualdad se considera que la forma funcional de $c(P)$ satisface:

$$\Delta_n^r A_2 \dots A_k \tau^r(A_1) \geq \Delta_n^r \tag{4.1}$$

ya que todas las trenzas $A_2, A_3, \dots, A_k, \tau^r(A_1)$ son positivas. Entonces, por definición, $inf(c(P) \geq \Delta_n^r = inf(P)$. Para la otra igualdad se verifica primero que la trenza $A_2 A_3 \dots, A_k$ es una factorización ponderada por la izquierda, ya que P estaba en forma canónica, y que la trenza $\Delta_n^r A_2 A_2 \dots A_k$ es una trenza en forma canónica, por la unicidad de la factorización, con supremo igual a $r + k - 1$. Entonces:

$$\begin{aligned} \Delta_n^r A_2 A_2 \dots A_k &\leq \Delta_n^{r+k-1} \\ \Delta_n^r A_2 A_2 \dots A_k \tau^r(A_1) &\leq \Delta_n^{r+k-1} \Delta_n \\ \Delta_n^r A_2 A_2 \dots A_k \tau^r(A_1) &\leq \Delta_n^{r+k} \end{aligned}$$

por lo tanto $c(P) \leq \Delta_n^{r+k} = sup(P)$ y por definición $sup(c(P)) \leq sup(P)$.
De hecho $inf(c(P)) \leq inf(P) + 1$ y $sup(c(P)) \geq sup(P) - 1$, con lo se tienen cotas superiores e inferiores para $inf(c(P))$ y $sup(c(P))$. Estas desiguadades se verifican observando que $P = \tau^r(A_1) c(P) \tau^{-r}(A_1)$, con $c(P) \in [inf(c(P)), sup(c(P))]$, $\tau^r A_1 \in [0, 1]$ y $\tau^{-r}(A_1) \in [-1, 0]$, y por el corolario 14 $P \in [inf(c(P)) - 1, sup(c(P)) + 1]$. Por definición, $inf(P) \geq inf(c(P)) - 1$ y $sup(P) \leq sup(c(P)) + 1$, de donde se sigue que $inf(c(P)) \leq inf(P) + 1$ y $sup(c(P)) \geq sup(P) - 1$.

Lema 35. *Supongase que P y Q son trenzas conjugadas y que $inf(Q) > r = inf(P)$. Entonces al ciclar progresivamente a P se obtendrá $c^j(P)$ con $inf(c^j(P)) > inf(P)$ para alguna j.*

Demostración. Sea $Q = APA^{-1}$ con $A \geq \epsilon$ y sea $inf(Q) > r = inf(P)$. Se puede escribir a $P = \Delta_n^r P'$ y $Q = \Delta_n^r Q'$ con $P' \geq \Delta_n$ y $Q' \geq \Delta_n$. Ahora, como $AP = QA$, sustituyendo se tiene que:

$$\Delta_n^r \tau^r(A) P' = A \Delta_n^r P' = \Delta_n^r Q' A,$$

de lo que se sigue que $\tau^r(A) P' = Q' A \geq \Delta_n$, y utilizando el lema 26 se tiene que $\tau^r(A) P_1 \geq \Delta_n$, donde $P_1 \neq \Delta_n$ es el primer factor de la factorización canónica de P'. Aplicando τ^r se tiene que $A \tau^r(P_1) \geq \Delta_n$, por lo que se puede escribir $A \tau^r(P_1) = A' \Delta_n$

para alguna $A' \geq \epsilon$. Por otro lado, $\tau^r(P_1) \leq \Delta_n$ y se puede escribir $\Delta_n = A''\tau^r(P_1)$ con $A'' \geq \epsilon$.
Por lo tanto:

$$A\tau^r(P_1) = A'\Delta_n = A'A''\tau^r(P_1),$$

de donde, $A = A'A''$ y como $P_1 \neq \Delta_n$ se tiene que $wt(A') = wt(A) - wt(A'') < wt(A)$. Por la definición de $c(P)$ se tiene que $\tau^r(P_1)c(P) = P\tau^r(P_1)$ y entonces:

$$\begin{aligned} QA\tau^r(P_1) &= AP\tau^r(P_1) \\ &= A\tau^r(P_1)c(P). \end{aligned}$$

Sustituyendo $A\tau^r(P_1) = A'\Delta_n$ se obtiene $QA'\Delta_n = A'\Delta_n c(P)$ y finalmente $\tau(Q)\tau(A') = \tau(A')c(P)$, de donde se sigue que $c(P)$ es conjugado de $\tau(Q)$ por la trenza $\tau(A')$ con $wt(\tau(A')) < wt(A)$. Como $inf(\tau(Q)) = inf(Q)$, se tiene que $inf(c(P)) > inf(P)$ o bien en caso contrario, como $wt(\tau(A')) < wt(A)$, si se continua ciclando se obtendrá j tal que $inf(c^j(P)) > inf(P)$, este proceso de ciclar es necesariamente finito ya que cada $c^i(P)$ es conjugado con $\tau^i(Q)$ por medio de trenzas de peso wt estrictamente menor a la anterior. □

Ahora se demuestra que el conjunto SSS de hecho es un conjunto no vacío.

Corolario 36. *Para toda clase de conjugación el valor máximo de inf y el mínimo valor de sup pueden ser alcanzados simultáneamente. Entonces el SSS para una trenza es el subconjunto de la clase de conjugación de esa trenza en el cual la longitud canónica l es mínima.*

Demostración. Se considera la clase de conjugación de una trenza P y sea B una trenza donde se alcanza el mínimo valor de *sup* y Q una trenza donde se alcanza el máximo valor de inf. Entonces B y Q son conjugados, si $inf(B) < inf(Q)$, entonces ciclando repetidamente a B se obtiene $c^j(B)$ $j \in \mathbb{N}$, con $inf(c^j(B)) = inf(Q)$ y $sup(c^j(B)) = sup(B)$. Entonces el elemento $c^j(B)$ pertenece al super conjunto cumbre de P. □

Ahora, para calcular el valor mínimo del *sup* de la clase de conjugación de una trenza P se define la siguiente operación.

Definición. Sea P una trenza con factorización canónica $P = \Delta_n^r A_1 A_2 \dots A_k$, donde $r = inf(P)$, se define $r(P) = \Delta_n^r \tau^r(A_k) A_1 A_2 \dots A_{k-1}$ y se dice que $r(P)$ se obtiene a partir de P ciclando inversamente.

Los siguientes resultados se enuncian sin prueba, para una demostración puede consultarse [7].

Teorema 37. *Sea P una trenza, entonces se cumple:*

- $sup(P) = -inf(P^{-1})$
- $(r(P))^{-1} = \tau(c(P^{-1}))$

Entonces se tiene que $sup(r(P)) = -inf(\tau(c(P^{-1}))) = -inf(c(P^{-1}))$ y entonces $sup(r^i(P)) = -inf(c^i(P^{-1}))$, $\forall i \in \mathbb{N}$. Como se puede encontrar j tal que $c^j(P^{-1})$ tal que $inf(c^j(P^{-1}))$ es el máximo en la clase de conjugación de P^{-1}, entonces se puede encontrar el valor mínimo de $sup(P)$ considerando solamente elementos $r^j(P)$ obtenidos ciclando inversamente P.
De esta manera, se puede encontrar un elemento del $SSS(P)$ utilizando operaciones de ciclado y ciclado inverso hasta encontrar un elemento común. El algoritmo para solucionar el problema de cojugación se enuncia a continuación, basado en la formulación de [9].

Algoritmo

- Entrada: Trenzas $P, Q \in B_n$.
- Salida: Respuesta lógica (verdadero o falso).
- Se calcula un elemento $P' \in SSS(P)$ y $Q' \in SSS(Q)$ por medio de operaciones de ciclado y ciclado inverso. Si $inf(P') \neq inf(Q')$ y $sup(P') \neq sup(Q')$, entonces P y Q no son conjugados.
- En otro caso, se debe calcular $SSS(P)$. Se comienza conjugando P' por cada uno de los elementos de $[0, 1]$ y se forma el conjunto S.
- Se conjuga cada uno de los elementos de S por todos los elementos de $[0, 1]$ y los elementos nuevos se agregan a S. Se continúa hasta que se encuentra que $Q' \in S$ o ya no se producen nuevos elementos con el proceso anterior.

- Si $Q' \in S$ se tiene que las trenzas P y Q son conjugadas.

En el siguiente capítulo se muestran los detalles de la implementación del algoritmo así como resultados de las pruebas que se hicieron del mismo.

Capítulo 5

Diseño e implementación

En este capítulo se describe el diseño y la implementación computacional de los grupos de trenzas y los sistemas criptográficos antes estudiados; la implementación fue realizada en el lenguaje de programación Python. En la primera sección se presenta el diseño del sistema, posteriormente se describen los detalles particulares de la implementación.

5.1. Diseño

A diferencia del enfoque tomado en la sección precedente de este trabajo, un enfoque de abajo hacía arriba, iniciando con las bases de la teoría de grupos de trenzas y culminando con los sistemas criptográficos basados en los mismos, en esta sección se tomará el enfoque complementario y se partirá del problema del cifrado de un mensaje, detallando gradualmente cada una de las partes del sistema hasta llegar a las particularidades del mismo.

El objetivo de la implementación es contar con un sistema que permita cifrar y descifrar cadenas de texto plano, utilizando la teoría desarrollada en los capítulos precedentes; las particularidades de la transmisión de los mensajes se encuentra fuera del alcance de este trabajo. El punto de partida es el sistema mostrado en la figura 5.1, en donde a un mensaje de texto se le aplica un proceso, que se llamará sistema criptográfico, y se obtiene un mensaje cifrado.

En el capítulo 4 se desarrolló la teoría de dos sistemas criptográficos basados en el grupo de trenzas, el esquema de Anshel-Anshel-Golfeld y el esquema inspirado en Diffie-Hellman; el resultado de los dos esquemas es una llave común que comparten dos

Figura 5.1: Primer Diagrama del sistema

personas. Como se menciona en el capítulo 4, para construir un sistema criptográfico completo se utiliza también un sistema criptográfico simétrico que emplea las llaves generadas por alguno de los esquemas mencionados. En la figura 5.2 se muestra el diagrama incluyendo la generación de llaves y el cifrado con un sistema simétrico.

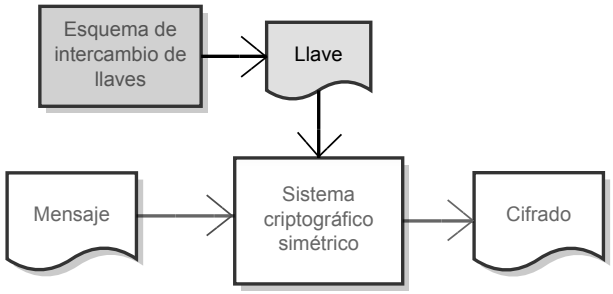


Figura 5.2: Segundo Diagrama del sistema

Como sistema criptográfico simétrico se utilizó el sistema Advanced Encription Standard, o AES, por sus siglas en inglés, para una descripción del estándar se puede consultar [19]. Los detalles concretos de la implementación se detallarán en las siguientes secciones. El esquema de intercambio de llaves es en sí mismo un sistema, como se puede ver en la figura 5.3.

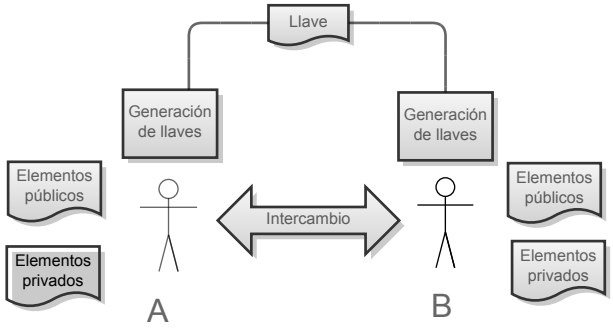


Figura 5.3: Esquema de intercambio de llaves

En la figura se observa a dos partes, que tienen elementos públicos y privados, entre las cuales hay un intercambio de información, típicamente en función de los elementos públicos y privados y a partir de la información intercambiada cada parte obtiene una llave común entre ambas.

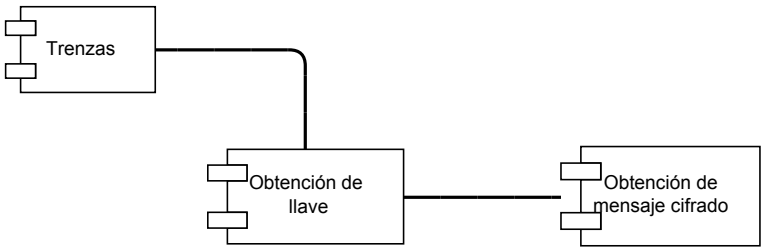


Figura 5.4: Diseño del sistema implementado

La figura 5.4 muestra una representación del diseño del sistema en componentes. Esta separación se hizo para mantener un orden en el diseño del sistema y la implementación.

Obtención de mensaje cifrado

El cifrado del mensaje se realiza empleando el sistema criptográfico AES, se consideró una implementación externa que se utilizó como librería y se adaptó para ser usada en el sistema.

Trenzas

Este componente consiste en una librería de clases que implementan a los elementos del grupo de trenzas con las siguientes características

- Se pueden realizar todas las operaciones de grupo de las trenzas tal como fueron estudiadas en el capítulo 3.
- Las operaciones sobre las trenzas descritas en el capítulo 4 para la solución al problema de conjugación también están implementadas.
- Se pueden generar trenzas aleatoriamente con fines de pruebas

La implementación de los grupos de trenzas es la base de los sistemas criptográficos desarrollados.

Obtención de llave

Para obtener la llave que se usa en el sistema criptográfico simétrico se utilizan los sistemas basados en la teoría de grupos de trenzas desarrollados en el capítulo 4. La implementación consiste en:

- Construcción de elementos públicos y privados para cada una de las partes.
- Obtención de las llaves comunes y verificación de igualdad entre las mismas.
- Conversión de las llaves de su representación en trenzas a una forma que puede ser utilizada como entrada del sistema criptográfico simétrico.

A continuación se describen a detalle los componentes más relevantes del sistema.

5.1.1. Criptografía

Se implementaron los sistemas criptográficos de Anshel-Anshel-Golfeld y el sistema basado en Diffie-Hellman, desarrollados en las secciones 4.2.1 y 4.2.2, respectivamente.

Esquema de Anshel-Anshel-Golfeld

La implementación del sistema de Anshel-Anshel-Golfed consiste esencialmente en:

- Definición de los conjuntos públicos de trenzas como listas de trenzas y funciones para generar estos conjuntos de manera aleatoria, para hacer pruebas.
- Implementación de la clase Palabra que representa palabras de un grupo libre genérico. Así como funciones para la construcción de palabras aleatorias de cierta longitud.
- Construcción de los elementos a transmitir entre las dos partes y las llaves comunes. Verificación de la igualdad de las dos llaves comunes.

La figura 5.5 muestra el diagrama UML de la clase Palabra. Esta clase se utiliza únicamente como auxiliar en la evaluación de palabras en conjuntos de trenzas específicos por lo que no se incluyen elementos para operar con otras palabras. El atributo *lista* es una serie de parejas de enteros (representadas a la vez como listas) que describen a los

elementos que forman las palabras. El primer elemento de cada pareja es un número que indica la letra y el segundo es el exponente que tiene esa letra (positivo o negativo). El método *evalua* recibe una lista de trenzas(con el mismo número de elementos que letras en el alfabeto) y sustituye los elementos de las trenzas en la palabra, concatenándolos y regresando la trenza resultante.

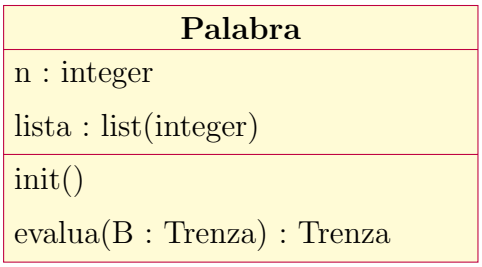


Figura 5.5: Diagrama de la clase Palabra

Esquema inspirado en Diffie-Hellman

El esquema de intercambio de llaves inspirado en Diffie-Hellman consiste en un protocolo más sencillo que el de Anshel-Anshel-Golfeld pero tiene algunas particularidades que lo hacen interesante, como los subconjuntos de trenzas UB y LB cuyas trenzas conmutan entre sí. La implementación consistió en:

- Construcción de la trenza pública para el sistema. En el caso de las pruebas esta trenza es generada aleatoriamente.
- Definición de las trenzas privadas de A y B, que son elementos de UB_n y LB_n. Así como generadores de trenzas en cada conjunto para fines de pruebas.
- Obtención de los elementos a transmitir y las llaves comunes. Verificación de igualdad de las llaves.

Los protocolos usados para el intercambio pueden consultarse de manera precisa en el capítulo 4. La implementación fue realizada en específico para las pruebas y no como una librería, en caso de querer usarse como parte de un sistema es necesario extender el programa realizado.

5.1.2. Trenzas

Para representar los elementos del grupo de trenzas se utilizó la forma canónica de Garside, la cual se desarrolló en la sección 3.4. Se definió primero una clase *FactorCanonico* para representar a una trenza de permutación. Uno de los atributos de la clase es una lista (perm) de números enteros representando a la permutación que determina al factor canónico, el segundo atributo es un número entero (n) que representa la cantidad de hilos en el grupo de trenzas actual.

Como métodos de la clase se incluyen uno booleano *compara*, para comparar dos factores canónicos, *tau* que aplica el automorfismo τ al factor canónico, como fue definido en la sección 3.2, *inverso* que obtiene el inverso de la permutación perm e *infimo* que calcula el ínfimo de dos factores canónicos.

Después se define la clase *Trenza* para representar un elemento del grupo de trenzas. Para representar una trenza se utiliza una factorización similar a la forma canónica de Garside, como fue descrita en 3.4, aunque se admite que la factorización no sea ponderada por la izquierda. Dicha representación no es única, de tal forma que para comparar dos trenzas es necesario convertirlas a la forma canónica de Garside, por medio del método *formacanonica*. Los atributos más significativos de la clase son la lista de factores canónicos ($fact$) que forman la factorización de la trenza y el exponente (r) que tiene la trenza fundamental en tal factorización. Se incluyen métodos *compara*, *inverso*, *producto*, *ciclar* e *inv_ciclar* que sirven para operar a los elementos en el grupo de trenzas. Los detalles de dicha implementación se presentarán en la siguiente sección.

5.1.3. Pruebas

Se diseñó una serie de pruebas para probar la consistencia de la implementación, los resultados de las pruebas se reportan en la sección de resultados junto con los comentarios acerca de las mismas.

Para probar la ejecución correcta del método *formacanonica* de la clase Trenza se obtiene una trenza generada aleatoriamente A y se calcula el producto AA^{-1}, posteriormente se ejecuta *formacanonica* para tener así la forma canónica de este producto y se verifica que sea igual que la trenza vacía. Esta prueba se realizó para diferente número de hilos en el grupo de trenzas y diferente cantidad de factores canónicos para la trenza inicial.

Para probar la consistencia de los sistemas criptográficos se hicieron pruebas con distintos números de hilos en el grupo de trenzas y distinta cantidad de factores canónicos en cada una de las trenzas. El sistema de Anshel-Anshel-Golfeld se probó con distinto número de elementos en los conjuntos públicos.
Finalmente, para probar el funcionamiento del algoritmo que resuelve el problema de la conjugación se hicieron pruebas, generando trenzas conjugadas y encontrando el conjugador con el algoritmo y probando, que en realidad, es un conjugador para las dos trenzas. Las pruebas se hicieron variando el número de hilos en el grupo de trenzas y la cantidad de factores canónicos en las trenzas conjugadas.

5.2. Implementación

Para la implementación de los sistemas criptográficos se utilizó el lenguaje de programación Python, versión 2.7. Los derechos de autor de Python pertenecen a la Python Software Fundation[1], una organización sin fines de lucro formada para promover el desarrollo del lenguaje. Python tiene implementaciones para los sistemas operativos mayores y para las máquinas virtuales JAVA y .NET. La licencia de uso de Python permite el desarrollo de proyectos de manera libre, incluyendo el uso comercial. Además, se seleccionó el lenguaje de programación Python para el proyecto debido a su sintaxis sencilla, así como al hecho de que tiene soporte nativo para estructuras de datos como listas y a la gran cantidad de librerías externas con las que cuenta, algunas de las cuales se utilizaron en el proyecto.
Python es un lenguaje de programación multi-paradigma, lo que permite la combinación de los paradigmas de programación orientada a objetos con la programación procedural. Así, el código puede dividirse en las clases que representan abstracciones de ciertas estructuras(como son los factores canónicos y los elementos del grupo de trenzas) y código secuencial procedural que describe el flujo del programa.
Primero se mostrarán los detalles de la implementación de las clases descritas en la sección anterior, comenzando con las clases encargadas de implementar los elementos del grupo de trenzas. La implementación está basada en [4], el código de la implementación

[1] *www.python.org*(liga activa al 23 de Octubre de 2011)

puede descargarse de la dirección `http://min.us/lbphmijfyWcFHC`[2]. En el Apéndice A puede consultarse una referencia de uso de la implementación.
A diferencia de la sección de diseño, aquí se comienza con la definición de las trenzas y posteriormente se detallan los otros componentes.

5.2.1. Trenzas

La clase FactorCanonico

Como se mencionó en la sección de diseño, para representar a una trenza de permutación (o factor canónico) en el grupo de trenzas B_n se utiliza una lista de n enteros que determina a la trenza en el grupo de permutación de n elementos S_n. Además de la lista que representa a la permutación, la clase también tiene como atributo un número entero que corresponde al número de hilos en el grupo de trenzas B_n. Así, si se considera el factor canónico $A = \sigma_1\sigma_3\sigma_2 \in B_4$, que tiene la permutación

$$\pi_A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix},$$

entonces tendrá la siguiente representación como objeto de la clase FactorCanonico:

```
Objeto FactorCanonico A
n = 4
perm = [3,1,4,2]
```

Para tener consistencia con la representación, a partir de ahora cuando se mencione una permutación solo se mostrará la parte correspondiente a la imagen de los elementos $1, \ldots, n$ bajo la misma. En la figura 5.6 se muestra el diagrama UML de la clase FactorCanonico.
En Python no existe el concepto de encapsulamiento estricto de información en una clase, es decir, no existen atributos o métodos privados en las clases. Como convención, los programadores de Python incluyen el prefijo _ a los atributos y métodos de uso interno de las clases, en el entendido de que éstos no deberán ser llamados de manera externa, es decir, son los miembros de la clase que serán definidos como privados. Por otro lado, los métodos que tienen el prefijo y sufijo __ son métodos definidos por el

[2]Verificado como activo al 23 de Octubre 2011

FactorCanonico
n : integer perm : list(integer)
__init__() compara(B : FactorCanonico) : boolean inverso() : integer _inverso() : FactorCanonico producto(B : FactorCanonico) : FactorCanonico tau(n : integer) : FactorCanonico _tau() : FactorCanonico _ordena_C(...) infimo(B : FactorCanonico) : FactorCanonico __eq__(B : FactorCanonico) : FactorCanonico __mul__(B : FactorCanonico) : FactorCanonico __getitem__(n : integer) : integer

Figura 5.6: El diagrama completo de la clase FactorCanonico

lenguaje para funciones específicas, así __init__ representa al constructor de la clase y __eq__ sirve para sobrecargar el operador de igualdad.

Para comparar dos factores canónicos se revisa elemento a elemento la imagen de cada una de las permutaciones de los dos factores canónicos, resolviendo que son iguales si ninguno de los elementos de la imagen es diferente y falso en caso contrario. Este proceso se realiza recorriendo cada una de las listas dentro de los factores canónicos y verificando que los enteros que representan las imágenes de las permutaciones son todos iguales. Siguiendo con el ejemplo, si se compara con el objeto de la clase factor canónico que tiene la representación:

```
Objeto FactorCanonico B
n = 4
perm = [3,1,2,4]
```

el resultado de la operación *compara* será falso, ya que los últimos dos elementos de la lista que representa la permutación de A son diferentes a los de B.

Como todos los factores canónicos son trenzas positivas, el inverso de un factor canónico no puede ser un factor canónico (todos sus generadores están elevados a potencias negativas). Sin embargo, es cierto que la permutación que corresponde al inverso de un factor canónico es el inverso de la permutación del factor canónico. Por lo anterior, el método inverso devuelve la permutación inversa a la permutación del factor canónico. El método de uso interno _inverso devuelve un factor canónico con la permutación inversa, esto se utiliza únicamente para realizar operaciones con facilidad y no debe considerarse conceptualmente correcto.

Para obtener el inverso de una permutación representada como lista de números enteros basta construir una nueva lista y poner el número i en la posición `perm[i]` de la nueva lista. En el ejemplo, el resultado del método *inverso* a la trenza A será la lista

```
[2,4,1,3]
```

que es, de hecho, la permutación del inverso de la trenza A, i.e. la trenza $\sigma_2^{-1}\sigma_3^{-1}\sigma_1^{-1}$, la cual no es un factor canónico.

El producto de dos factores canónicos A y B, en caso de que sea factor canónico, tiene como permutación la resultante de componer las permutaciones de A y B. Para calcular la composición de las dos permutaciones se crea una nueva lista y se calcula el elemento de la posición i de esa lista como `b.perm[A.perm[i]]`. Si se considera la trenza σ_3 que tiene la representación:

```
Objeto FactorCanonico B
n = 4
perm = [1,2,4,3]
```

El producto de los factores canónicos A y B es el factor canónico que tiene la representación:

```
Objeto FactorCanonico C
n = 4
perm = [4,1,3,2]
```

El método *tau* obtiene el resultado de aplicarle el automorfimo τ^n al factor canónico, el cual se explica en la sección 3.2. Como se vio en la sección anterior, para una elección par de n, el autormorfismo τ^n funciona como la identidad (por las propiedades de la

trenza fundamental) y para elecciones impares de n solamente se tiene que calcular τ^1, operación que se realiza en la función de uso interno _tau calculándolo directamente de la expresión $\Delta A \Delta^{-1}$, donde A es el factor canónico. Para realizar esta operación primero se obtienen representaciones para Δ_n y Δ_n^{-1}, en el caso del ejemplo se tienen:

```
Objeto FactorCanonico Delta
n = 4
perm = [4,3,2,1]


Objeto FactorCanonico iDelta
n = 4
perm = [4,3,2,1]
```

se observará que Δ_n^{-1} fue representada como un factor canónico a pesar de no serlo, esto se explica porque los factores canónicos son invariantes bajo τ, entonces como se sabe que el resultado será un factor canónico, se puede hacer esta concesión para facilitar las operaciones. También se observa que la representación para Δ_n y Δ_n^{-1} es esencialmente la misma, por lo que en la práctica no es necesario calcular ambas. En el ejemplo el resultado es el factor canónico con la representación:

```
Objeto FactorCanonico tauA
n = 4
perm = [3,1,4,2]
```

el cual, resultó ser igual con A. Esto se explica por lo siguiente:

$$\tau(A) = \tau(\sigma_1)\tau(\sigma_3)\tau(\sigma_2) = \sigma_3\sigma_1\sigma_2 = \sigma_1\sigma_3\sigma_2 = A.$$

El método ínfimo se utilizará posteriormente para el cálculo de la forma canónica de una trenza. El ínfimo de dos trenzas A y B es el elemento más grande de entre todos los elementos menores a A y B, es decir, si C es el ínfimo de A y B, entonces $C \leq A$, $C \leq B$ y si D es tal que $D \leq A$, $D \leq B$ se tiene que $D \leq C$. El algoritmo que se utilizó para calcular el ínfimo de dos trenzas está basado en [4], con pequeñas modificaciones para adecuarlo al lenguaje Python. Sean A y B factores canónicos, C es el ínfimo de A y B y π_C es la permutación que corresponde a C. El algoritmo ordena los enteros

$1, \ldots, n$ considerando que $x < y$ si y solo si $\pi_c[x] < \pi_c[y]$. El resultado del proceso es la permutación del inverso de C, con lo que solo es necesario obtener la permutación inversa para obtener el resultado deseado, para más detalles se puede consultar la fuente antes mencionada.
Finalmente, los métodos *__eq__*,*__mul__* sirven para sobrecargar el operador $=$ de igualdad y $*$ de producto, respectivamente. El método *__getitem__* declara a los objetos de la clase como objetos iterables, al llamar a un objeto de la clase con cierto índice i devolverá el elemento de la permutación que se encuentra en la posición i.

La clase Trenza

Los objetos de la clase Trenza representan a los elementos del grupo de trenzas B_n en una factorización, similar a la factorización canónica, formada por la trenza fundamental elevada a una potencia $r \in \mathbb{Z}$ y un conjunto de k factores canónicos concatenados después de la misma. La figura 5.7 muestra el diagrama UML de la clase Trenza.
Para este caso, se utilizará como ejemplo una trenza que tiene un exponente $r = 4$, dos factores canónicos que tiene la representación:

```
Objeto Trenza A
n = 4
r = 4
k = 2
fact = [[3,1,4,2],[4,2,1,3]]
```

Para fines de claridad, solo se escribe la permutación de cada uno de los factores canónicos.
El método *producto* calcula el producto de dos trenzas A y B, si A tiene la factorización $\Delta_n^r A_1 \ldots A_k$ y B tiene la factorización $\Delta_n^{r'} B_1 \ldots B_{k'}$, entonces el producto de A y B es la trenza con la factorización:

$$\Delta_n^{r+r'} \tau^{r'}(A_1) \ldots \tau^{r'} A_k B_1 \ldots B_{k'},$$

en general, esta trenza resultante no se encuentra en la forma canónica. Para calcular esta expresión se crea un nuevo objeto Trenza con exponente igual a $r+r'$ y una lista de factores de $k + k'$ elementos, los primeros k elementos se obtienen aplicando el método

Trenza
n : integer
r : integer
k : integer
fact : list(FactorCanonico)
canonica : boolean
__init__()
producto(B : Trenza) : Trenza
inverso() : Trenza
conjugar(B : Trenza) : Trenza
_cabeza(A : Trenza, B : Trenza) : Trenza
formacanonica() :
compara (B : Trenza) : boolean
ciclar() : Trenza
inv_ciclar() : Trenza
inf() : integer
sup() : integer
__eq__(B : Trenza) : boolean
__mul__(B : Trenza) : Trenza
__neg__() : Trenza

Figura 5.7: El diagrama completo de la clase Trenza

`tau(r)` a los k factores de A y los últimos k' elementos se obtienen directamente de la lista de factores de B.

Se considera la trenza B con la siguiente representación:

```
Objeto Trenza A
n = 4
r = 1
k = 1
fact = [[4,1,3,2]]
```

entonces el resultado del producto de A por B es la trenza con la representación

```
Objeto Trenza C
n = 4
r = 5
k = 2
fact = [[3,1,4,2],[2,4,3,1],[4,1,3,2]]
```

Se puede observar la aplicación de τ a los factores canónicos de A, mientas que el factor canónico de B permanece igual que antes.

Para calcular el inverso de una trenza A que tiene la factorización $\Delta_n^r A_1 \dots A_k$ se considera lo siguiente:

$$\begin{aligned} A^{-1} &= (\Delta_n^r A_1 \dots A_k)^{-1} \\ &= A_k^{-1} \dots A_1^{-1} \Delta_n^{-r} \\ &= A_k^{-1} \Delta_n \Delta_n^{-1} \dots A_1^{-1} \Delta_n \Delta_n^{-1} \Delta_n^{-r} \\ &= (A_k^{-1} \Delta_n) \Delta_n^{-1} \dots (A_1^{-1} \Delta_n) \Delta_n^{-r-1} \\ &= (A_k^{-1} \Delta_n) \Delta_n^{-1} \dots \Delta_n^{-r-1} \tau^{-r-1}(A_1^{-1} \Delta_n) \\ &= \Delta_n^{-r-k} \tau^{-r-k}(A_k^{-1} \Delta_n) \dots \tau^{-r-1}(A_1^{-1} \Delta_n) \\ &= \Delta_n^{-r-k} \tau^{-r-k}(B_k) \dots \tau^{-r-1}(B_1), \end{aligned}$$

donde $B_i = A_i^{-1} \Delta_n$.

Una vez más, esta trenza, en general, no se encuentra en su forma canónica. Para calcular el inverso se crea un objeto Trenza que tiene como exponente $-r-k$ y con una lista de factores que tiene en cada posición el factor $\tau^{r-i}(B_i)$, que se calcula usando los métodos de la clase FactorCanónico. Para calcular cada $\tau^{r-i} B_i$ se toma el elemento A_i de la lista de factores de la trenza, luego se invierte usando el método _inverso y obteniendo el producto con un factor que represente a la trenza fundamental, posteriormente se aplica el método `tau(r-i)` y se obtiene la trenza buscada.

Como el automorfimo τ se calcula en tiempo lineal sobre el número de hilos n, la complejidad de calcular inversos o productos es $\mathcal{O}(kn)$. La conjugación de trenzas se reduce a aplicar dos productos e invertir una trenza, por lo que su complejidad resulta ser la misma que las operaciones de producto e inversión. En el ejemplo, el resultado de invertir la trenza A es la trenza con la representación:

```
Objeto Trenza iA
n = 4
r = -6
k = 2
fact = [[2,3,1,4],[3,1,4,2]]
```

Para calcular la forma canónica de una trenza se utiliza el algoritmo presentado en [4]. Se presenta aquí una descripción general del algoritmo, para lo cual se define la cabeza maximal de una trenza positiva.

Definición. Si $P \in B_n^+$ tiene forma canónica $P = A_1 A_2 \dots A_k$, el primer factor canónico de dicha factorización se conoce como *cabeza máximal* de P. Como la forma canónica es única, la cabeza maximal de P es única.

Si la trenza A tiene la factorización $\Delta_n^r A_1 \dots A_k$, se define $P = A_1 \dots A_k$, entonces el algoritmo consiste en calcular la cabeza máximal B de $A_{k-1}A_k$ y después sustituir A_{k-1} por B y A_k por $B^{-1}A_k$. De esta manera se tiene que $A_{k-1}A_k$ se encuentra en forma canónica. El algoritmo continua aplicando el mismo proceso sucesivamente a $A_{k-2} \dots A_k$, $A_{k-3} \dots A_k$ y $A_1 \dots A_k = P$ hasta obtener la forma canónica de P. Finalmente se eliminan los elementos Δ_n del inicio de la factorización de P y se suma la misma cantidad de factores eliminados a la potencia r de la trenza.
Para calcular la cabeza maximal se utilizan las siguiente propiedades:

- Para trenzas positivas A y P arbitrarias, la cabeza máximal de AP es igual al producto de la cabeza maximal de A y la cabeza máximal de P.
- La cabeza maximal de dos factores canónicos A y B es igual a $A((A^{-1}D)) \wedge B$

donde $\wedge$ significa el ínfimo de los factores canónicos. En el artículo [4] se enuncia una forma distinta de calcular la cabeza maximal, pero en la implementación se descubrió que no daba los resultados esperados y se obtuvo la fórmula precedente. En el mismo artículo los autores afirman que la complejidad de obtener la forma canónica es de $\mathcal{O}(k^2 nlog(n))$, la cual se podrá verificar en la sección de resultados.
El método *formacanonica* es un método destructivo que altera la estructura del objeto de la clase Trenza, esto se hace porque una vez que se tiene la forma canónica de una trenza no es necesario regresar a la forma anterior.

Para la trenza del ejemplo, se tiene que su forma canónica es la trenza con la representación:

```
Objeto Trenza iA
n = 4
r = 4
k = 2
fact = [[3,2,4,1],[2,4,1,3]]
```

Se puede observar que el número de factores canónicos en la presentación no cambió, sin embargo se tienen factores canónicos diferentes que permiten que se cumpla la propiedad de ser una factorización ponderada por la izquierda.

Para comparar dos trenzas A y B, primero se convierten las dos trenzas en su forma canónica y después se revisa la igualdad de los exponentes de la trenza fundamental, la cantidad de factores canónicos y cada uno de los mismos, verificando las listas de factores elemento por elemento, en caso de encontrar diferencias en alguno de ellos las trenzas son diferentes, de otra forma son iguales.

Los métodos *inf* y *sup* regresan los valores de r y $r + k$ que corresponden al ínfimo y supremo de la trenza, como se describió en el teorema 28, de la sección 3.4. En el ejemplo, el valor de *inf* es 4 y el de *sup* es 6.

El método *conjugar* obtiene, como su nombre lo indica, el resultado de conjugar una-trenza representada con otra trenza que se pasa como parámetro. El resultado de la conjugación de dos trenzas A y B se calcula obteniendo el inverso de B y luego calculando el producto de B con A y después el producto del objeto Trenza resultante con el inverso de B.

Los métodos *ciclar* e *inv_ciclar* se usan para calcular el resultado de ciclar o ciclar inversamente la trenza actual, como se describió en el capítulo 4. Para calcular el resultado del método *ciclar* se hace un desplazamiento a la izquierda de la lista de factores de la trenza y se sustituye al último elemento de la lista por el resultado de aplicar el método `tau(r)` al primer factor de la trenza original. En el ejemplo se tiene el siguiente resultado:

```
Objeto Trenza A.ciclar
n = 4
r = 4
k = 2
fact = [[2, 4, 1, 3] , [3, 2, 4, 1]]
```

Por otro lado, para calcular el resultado de la operación de ciclado inverso, el método `inv_ciclar`, se debe hacer un desplazamiento a la derecha de la lista de factores y reemplazar el primer factor de la lista por el objeto de la clase FactorCanonico que resulta de aplicar el método `tau(r)` al objeto FactorCanonico que estaba en la última posición de la trenza original. El resultado de aplicar la operación de ciclado inverso a la trenza del ejemplo es el siguiente:

```
Objeto Trenza A.ciclar
n = 4
r = 4
k = 2
fact = [[2, 4, 1, 3] , [3, 2, 4, 1]]
```

Se observará que las dos trenzas son iguales, esto se debe a que el exponente de la trenza es 4 y la operación `tau` cuando recibe como parámetro un número entero par no realiza ninguna operación sobre la trenza.

Para generar factores canónicos aleatorios se crean permutaciones aleatorias con el paquete *random* de Python. Esto se hace para fines de prueba, ya que los autores del paquete indican que la generación de números aleatorios del mismo no es suficientemente segura como para usarse con fines criptográficos. Para generar Trenzas aleatorias, simplemente se genera aleatoriamente el número deseado de factores canónicos que se usan en la descripción de la misma y se crea un nuevo objeto Trenza con esos factores. En la implementación esto se hace con las funciones *genera_factor* y *genera_trenza*.

El código fuente de las clases FactorCanonico y Trenzas puede consultarse en el Apéndice B.

5.2.2. Criptografía

Los sistemas criptográficos implementados son esquemas de intercambio de llaves, por lo que se necesita un sistema criptográfico simétrico para hacer una implementación completa. Para esto se utilizó la libreria PyCrypto[3], un proyecto que implementa una variedad de sistemas criptográficos en Python. En particular se utilizó la implementación de AES del mismo, con una llave de 256 bits. Para generar esa llave se usó la función hash SHA, que tiene como salida cadenas de 256 bits, a la llave común obtenida por los esquemas de intercambio de llaves, esta función también forma parte de la librería PyCrypto.

Esquema de Anshel-Anshel-Golfeld

Los elementos públicos de A y B se implementaron como listas de objetos de la clase Trenza. Para generar estos conjuntos se utilizó la función *genera_trenza* para generar trenzas aleatorias como elementos de una lista. Los elementos privados del sistema son palabras de un conjunto genérico, que se implementaron con la clase Palabra.
Como se mencionó antes, la clase Palabra se usa para representar palabras de un grupo libre abstracto. Se considera el conjunto formal $X = \{x_1, x_2, \ldots, x_n\}$ como la base del grupo libre. Un elemento del grupo libre $F(X)$ puede escribirse como una sucesión de elementos de X elevados a una potencia positiva o negativa. Se considera a manera de ejemplo que X es un conjunto de 5 elementos y al elemento $x = x_1 x_4^{-1} x_2 x_3^{-2}$ de $F(X)$. Este elemento se representa de la siguiente forma:

```
Objeto Palabra x
n = 5
lista = [[1,1],[4,-1],[2,1],[3,-2]]
```

Se observa que dentro de la lista, se tienen parejas de elementos que representan a cada uno de los elementos de x. El primer elemento de cada pareja indica cuál es el elemento de X que corresponde a esa posición y el segundo es el exponente al cual está elevado dicho elemento.

[3]The Python Cryptography Toolkit *www.dlitz.net/software/pycrypto/*(activa al 23 de Octubre de 2011)

La operación esencial de un objeto de la clase Palabra es la evaluación de conjuntos de trenzas, tales conjuntos de trenzas deben tener la misma cantidad de elementos que X y el orden en que se encuentran en la lista es el orden en el que se evalúan, que tiene como resultado un elemento de la clase Trenza.

Si se tiene un conjunto de trenzas P representado por una lista y una Palabra x, se calcula el resultado de evaluar x en P creando una lista de trenzas igual a la longitud de x, y donde la i Trenza se obtiene verificando cuál es la trenza de P en la posición representada por el primer elemento de la pareja en la posición i de la Palabra x y se eleva a la potencia dada por el segundo elemento de dicha pareja, usando los métodos de la clase Trenza. Finalmente se calcula el producto de todas las trenzas en la lista.

Así, el conjunto $P = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$, con la palabra del ejemplo anterior, cuando se aplica el método *evalua* da como resultado la trenza:

$$\sigma_1\sigma_4^{-1}\sigma_2\sigma_3^{-2}.$$

Por otro lado el conjunto $Q = \{\sigma_3, \sigma_4, \sigma_5, \sigma_2, \sigma_1\}$ da como resultado:

$$\sigma_3\sigma_2^{-1}\sigma_4\sigma_5^{-2},$$

por lo que es importante observar el orden de los elementos de los conjuntos a evaluar en una palabra.

La función *genera_palabra* crea una palabra aleatoria de cierta longitud l de un alfabeto de n letras. Esto se hace calculando l números aleatorios entre 1 y n y l exponentes que pueden tomar los valores 1 o -1, con estos valores se construyen listas de dos elementos (las parejas) y la lista de la palabra.

Se usa la misma notación que se usó en la descripción del protocolo en el capítulo 4, donde u es la palabra privada de A y v la de B y P y Q son los conjuntos públicos de trenzas de A y B, respectivamente. Se calculan primero los elementos privados s y r usando el método *evalua* de las palabras u y v con los conjuntos P y Q, respectivamente. Luego se calculan los elementos a transmitir q_i' y p_i' a partir de los elementos q_i y p_i utilizando el método *conjugar* de la clase Trenza. Para calcular las llaves comunes, se evalúan las palabras u y v en los conjuntos transmitidos y se calculan directamente de las definiciones de las llaves comunes usando operaciones de la clase Trenza. La verificación de las llaves se hace comparando los objetos Trenza que representan a las llaves calculadas para A y B.

Esquema inspirado en Diffie-Hellman

Primero se implementaron funciones que generan trenzas de LB_n y UB_n, respectivamente. Para generar trenzas de LB_n o UB_n se necesitan generar factores canónicos en estos conjuntos, para esto se hacen listas de números enteros de 1 a n y se desordenan los primeros $\lfloor n/2 \rfloor - 1$ o $\lfloor n/2 \rfloor + 1$ para generar elementos de LB_n o UB_n, respectivamente. El elemento público del sistema es un objeto trenza generado con la función *genera_trenza*. Los elementos privados de A y B se generan usando los métodos mencionados antes para generar trenzas en LB_n y UB_n. Los elementos a transmitir y las llaves comunes se calculan utilizando el método *conjugar* de la clase Trenza. La comparación de las llaves comunes se hace directamente utilizando el método *compara* de la clase Trenza.

Solución al problema de conjugación

Se implementó también el algoritmo para resolver el problema de conjugación en B_n, descrito en 4.3.1, sin ninguna modificación. Los pasos para la implementación del algoritmo se enuncian a continuación.

1. Se implementaron métodos para calcular el valor máximo de ínfimo y el mínimo de supremo entre los conjugados de una trenza. Esto se realizó utilizando los métodos *ciclar* e *inv_ciclar* de la clase Trenza.

2. Se implementó una función que dada una trenza $A \in B_n$, encuentra un elemento de $SSS(A)$, utilizando los métodos *ciclar* y *inv_ciclar* de la clase Trenza. La función también encuentra el conjugador la trenza A y la trenza de $SSS(A)$ obtenida.

3. Se calcula el conjunto completo $SSS(A)$ a partir de una trenza $a \in SSS(A)$. Para hacer esto se inicializa una lista con la Trenza a, posteriormente se multiplica ese elemento por todos los elementos de $[0, 1]$, obtenidos generando todas las permutaciones de n, se verifica cuáles de los elementos resultantes pertenecen a $SSS(A)$ y se añaden a la lista y se continúa este proceso hasta que no se generan nuevos elementos de $SSS(A)$. Debido a que el número de elementos de $[0, 1]$ es $n!$, se tiene que la complejidad de este algoritmo es $\mathcal{O}(n!kn)$.

El código fuente del método que obtiene el SSS completo se puede consultar en el Apéndice B.

5.3. Resultados

Todas las pruebas fueron realizadas en una computadora con un procesador Intel Core 2 Quad Q8300 a 2.5 Ghz con 6 GB de memoria RAM con el sistema operativo Windows 7 de 64 bits.

Forma Canónica

Primero se presentan los resultados de las pruebas diseñadas para verificar la consistencia y el desempeño de la conversión a factor canónico. Como se mencionó antes, se generan trenzas aleatorias con diferente número n de hilos en el grupo de trenzas y diferente número k de factores canónicos, posteriormente se obtiene el inverso de dicha trenza y se calcula el producto de la trenza con su inverso. Se calcula la forma canónica de ese producto y se verifica que sea igual a la trenza vacía. Se realizaron pruebas para $n \in \{2, 3, \ldots, 14\}$ y $k \in \{5, 10, 25, 50\}$, donde n es el número de hilos en el grupo de trenzas usado y k el número de factores canónicos en las trenzas, para cada combinación se realizaron 100 pruebas, con lo que se calculó el tiempo promedio. Los tiempos de ejecución se presentan en la tabla 5.1, todas las ejecuciones de la prueba fueron correctas.

Es importante mencionar que por la construcción de la prueba, para cada elección de k se construye una trenza A con k factores canónicos pero se obtiene la forma canónica de AA^{-1} que tiene $2k$ factores canónicos.

Se mencionó en la sección de implementación que el algoritmo para obtener la forma canónica tiene una complejidad de $\mathcal{O}(k^2 nlog(n))$. Para visualizar el efecto de los cambios en cada uno de los parámetros se elaboraron gráficas de los tiempos de ejecución. La figura 5.8 muestra los cambios en el tiempo de ejecución para valores de número de hilos fijos cuando cambia la cantidad de factores canónicos en las trenzas usadas para las pruebas; se puede apreciar un comportamiento cuadrático, como era esperado.

Por otra parte, en la figura 5.9 se pueden observar los tiempos de ejecución para trenzas de un número fijo de factores canónicos pero cada una de ellas con diferente número de

n/k	5	10	25	50
2	0.01	0.02	0.13	0.50
3	0.01	0.04	0.23	0.89
4	0.02	0.05	0.32	1.36
5	0.02	0.07	0.44	1.70
6	0.02	0.09	0.55	2.14
7	0.03	0.09	0.65	2.58
8	0.03	0.13	0.77	3.00
9	0.04	0.15	0.90	3.52
10	0.05	0.17	1.03	4.04
11	0.05	0.19	1.16	4.58
12	0.05	0.22	1.29	5.08
13	0.06	0.27	1.42	5.59
14	0.07	0.25	1.55	6.11

Tabla 5.1: Tiempo de ejecución de la conversión a forma canónica(en segundos)

hilos. El comportamiento de la curva de tiempo parece ser mayor al lineal pero sin ser cuadrático, para los valores usados en la prueba, que es consistente con el comportamiento de $o(nlog(n))$ para valores pequeños de n. Como al hacer operaciones con trenzas el número de factores canónicos puede aumentar rápidamente, el tiempo de ejecución de algoritmos que usen intensivamente las operaciones elementales puede convertirse en un problema al momento de calcular la forma canónica.

Sistema inspirado en Diffie-Hellman

Para probar el criptosistema inspirado en el esquema de Diffie-Hellman se simuló un intercambio de mensajes entre dos partes. Se genera al azar un elemento común para las dos partes, así como elementos privados y se simulan los intercambios de manera local para generar la llave común de las dos partes. Posteriormente se hace una comparación para mostrar que la llave común es igual para las dos partes y se utiliza la llave para cifrar y descifrar la cadena “Esta es una prueba del sistema DH”, finalmente se compara el texto descifrado con el texto plano original y se concluye la prueba. Los tiempos de ejecución de la prueba se muestran en la tabla 5.2, todas las pruebas concluyeron

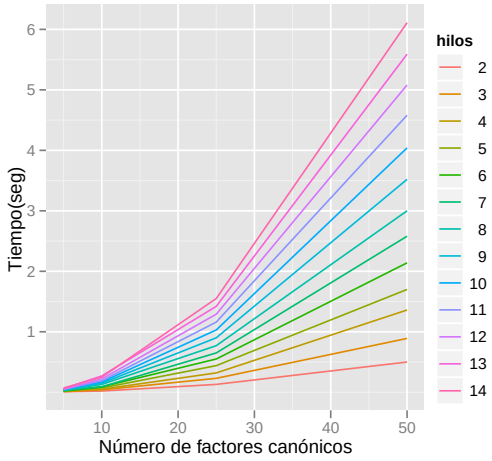


Figura 5.8: Gráfica de tiempo por número de factores canónicos

correctamente.

n/k	8	12	18	24
20	4	9	20	36
40	9	22	48	87
80	22	50	110	198
100	28	65	145	257

Tabla 5.2: Resultados de la prueba del sistema DH(en segundos)

En este sistema se hacen únicamente operaciones de conjugación, que se reduce simplemente a la operación de producto aplicada dos veces, la cual es una operación de complejidad lineal al número de factores canónicos de las trenzas. Para la comparación se tiene que convertir primero la pareja de trenzas que representan las llaves comunes en su forma canónica, de donde los tiempos de ejecución dependen esencialmente de la conversión a forma canónica. La figura 5.10 muestra la curva de tiempo de ejecución para cierto número de hilos en el grupo de trenzas cuando cambia el número de factores canónicos en los elementos públicos y privados.

Se observa un comportamiento ligeramente cuadrático en el tiempo con respecto al número de factores canónicos, como era de esperarse por lo mencionado anteriormente. Por otro lado, la figura 5.11 muestra la curva de tiempo cuando se utilizan trenzas de longitud fija y se prueba con diferente número de hilos en el grupo de trenzas.

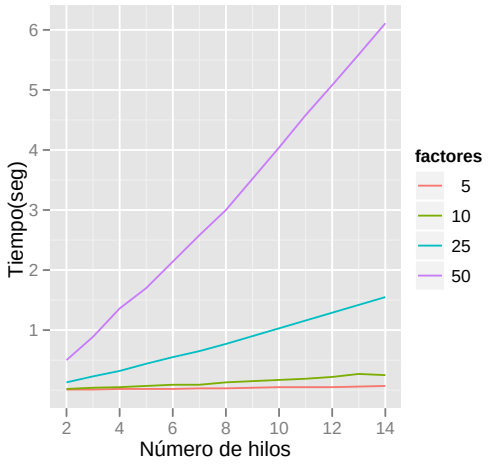


Figura 5.9: Gráfica de tiempo por número de hilos

El comportamiento se observa mayor al lineal, como era de esperarse en la discusión precedente.

Para el valor sugerido de B_{80} con trenzas de, al menos, 12 factores canónicos se tiene que el tiempo de ejecución aproximadamente se encuentra entre 1 y 3 minutos, los cuales son adecuados para la práctica.

El criptosistema de Anshel-Anshel-Golfeld

El sistema de Anshel-Anshel-Golfeld se probó de la misma forma que el inspirado en Diffie-Hellman. Para todas las pruebas se utilizaron conjuntos de 20 trenzas en B_{80}, parámetro sugerido por los autores, con 5 y 10 factores canónicos(k) y con palabras de diferente longitud(la longintud l de una palabra es el número de letras que la forman). En la tabla 5.3 se presentan los resultados de las pruebas.

Se observa un aumento considerable en el tiempo de ejecución al incrementar la longitud de la palabra usada, casi duplicándose con la adición de una unidad. Para explicar este aumento se puede calcular el número de factores canónicos que tendrían las trenzas comunes t_A o t_B antes de calcular la forma canónica para compararlas, se usa la misma notación que se usó en el capítulo 4 para explicar el sistema.

Para calcular la trenza s o r se necesita evaluar las palabras u y v en los conjuntos de trenzas de A y B; como cada trenza tiene k factores, las trenzas s y v tienen $k * l$ factores. Cada una de las trenzas p'_i y q'_i tienen $2lk + k$ factores, por lo que las palabras

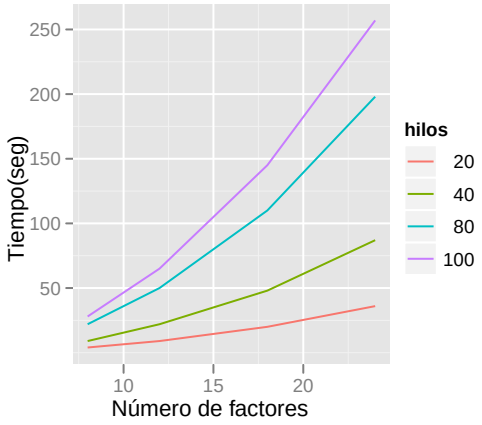


Figura 5.10: Gráfica de tiempo por número de factores canónicos

l/k	5	10
3	131	460
4	441	1360
5	904	3161
6	1765	6369
7	3166	12340
8	5327	19491
9	9077	29722
10	12384	44505

Tabla 5.3: Tiempos de ejecución para B_{80} en segundos

u y v evaluadas en los conjuntos de p'_i y q'_i están formados de $2lk(l+1)$ factores y finalmente t_A y t_B constan de $lk(3l+1)$ factores canónicos.

Entonces, un aumento de la longitud de l en un elemento implica un aumento de $k(6l+4)$ factores canónicos en t_A, lo que resulta en un aumento considerable en el tiempo de ejecución, como se observó en los resultados de la prueba de conversión a la forma canónica. La gráfica 5.12 muestra el aumento en tiempo de ejecución cuando aumenta la longitud en las palabras usadas.

En comparación con el sistema inspirado en Diffie-Hellman los tiempos de ejecución de este sistema son mucho mayores. Para el parámetro $k = 5$, con una longitud de palabra $l = 7$ el tiempo de ejecución es cercano a una hora y para $k = 10$ basta usar palabras

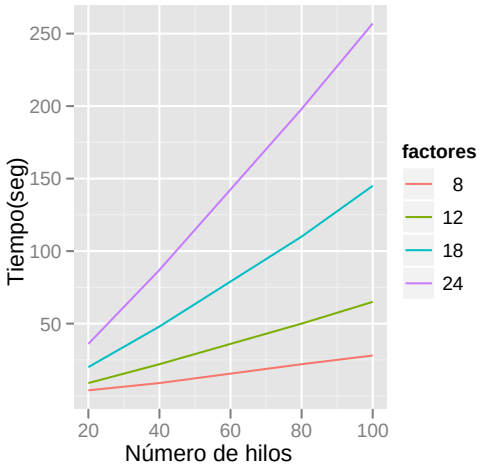


Figura 5.11: Gráfica de tiempo por número de factores e hilos

con longitud $l = 5$ para que el tiempo de ejecución sea cercano a una hora, mientras que en el sistema Diffie-Hellman usando trenzas en B_{80} con $k = 24$ factores canónicos el tiempo de ejecución no excede los 5 minutos.

La solución al problema de conjugación

Las pruebas que fueron realizadas para el problema de conjugación merecen especial atención por los problemas que se encontraron durante su ejecución. Para una cantidad de hilos en el grupo de trenzas n dada y para cada elemento conocido del SSS de una trenza, es necesario calcular el producto de ese elemento con los $n!$ elementos de $[0, 1]$, debido a esto solamente se pudieron hacer pruebas para $n \leq 6$ antes de encontrar problemas de memoria. Para fines de la prueba se fijó $k = 6$, el número de factores canónicos en las trenzas conjugadas, y se realizaron 10 pruebas para cada combinación de parámetros. El promedio de los tiempos de ejecución así como el número promedio de elementos en el conjunto SSS se presentan en la tabla 5.4.

Se observa primero el gran aumento en tiempo de ejecución y el número de elementos de SSS cuando n, el número de hilos en las trenzas, pasa de 4 a 5. Además del número de elementos de $[0, 1]$ se tiene también una variación considerable en el número de elementos de SSS para trenzas del mismo número de hilos y factores canónicos, esto se puede observar en la figura 5.13, en la cual se muestra el histograma del tamaño del

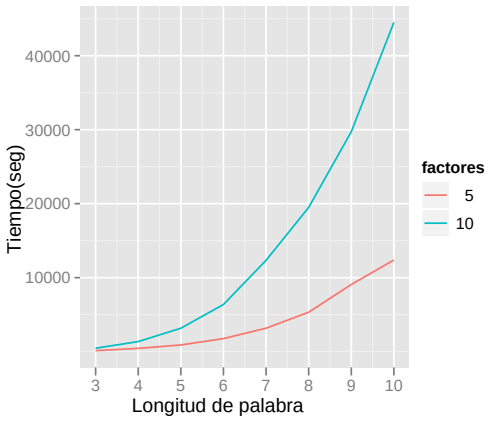


Figura 5.12: Gráfica de tiempo por número de factores y longitud de palabra

n	tiempo	SSS
3	1	3
4	3	24
5	1250	125
6	3351	457

Tabla 5.4: Tiempos de ejecución y tamaño de conjunto SSS

SSS para 100 trenzas de B_5 con $k = 6$ factores canónicos.

Se puede ver una gran variabilidad en el tamaño del SSS con características similares, lo que también se reflejaría en una variabilidad considerable en el tiempo de ejecución del algoritmo por las observaciones que se hicieron antes. Para los parámetros recomendados de los sistemas criptográficos, en particular el uso de trenzas en B_{80}, usar el algoritmo de solución al problema de conjugación para obtener la llave pública es una labor imposible con los recursos computaciones disponibles.

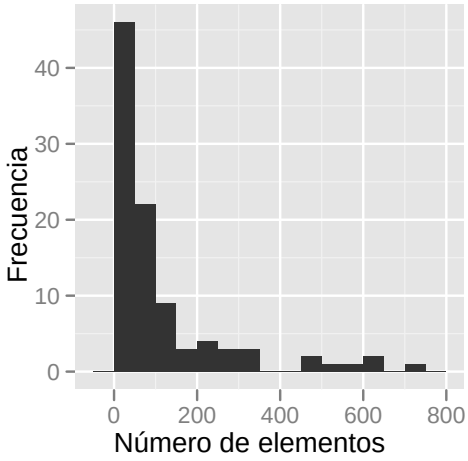


Figura 5.13: Histograma del tamaño de SSS para trenzas en B_5

Capítulo 6

Conclusiones

La criptografía es una área de estudio que avanza muy rápido, constantemente se exploran nuevas ideas para desarrollar sistemas y para tratar de hacer obsoletos a los ya existentes. Esas ideas provienen de distintas áreas de las matemáticas y las ciencias computaciones y surgen de investigaciones que normalmente no tienen como fin la criptografía. El descubrimiento de un problema algorítmico puede dar lugar, con un poco de ingenio, a un sistema criptográfico nuevo.

Así sucede con los grupos de trenzas, cuyo estudio se inicia en la topología y encuentra aplicaciones en distintas áreas de las matemáticas y la física teórica, que fueron definidos al inicio del siglo pasado pero sus aplicaciones a la criptografía no tienen más que unos años. La aparente dificultad del problema de conjugación en el grupo de trenzas da lugar a los sistemas presentados en este trabajo. El uso de la palabra *aparente* se hizo de manera deliberada, ya que lo que parece hoy ser un problema complicado podría convertirse en un problema sencillo el día de mañana.

Ya existen trabajos que proponen mejoras a los algoritmos para la solución al problema de conjugación, en [8] los autores prueban que se puede usar un conjunto más pequeño de trenzas para calcular el SSS, en lugar de usar el conjunto completo $[0, 1]$, lo que se traduciría en mejoras a la eficiencia del algoritmo; por otro lado, en [11] los autores proponen utilizar un subconjunto del SSS, que llaman ultra conjunto cumbre(USS, por su nombre en inglés Ultra Summit Set) y que es calculado de manera eficiente. A pesar de que queda mucho por demostrar todavía, podría ser que el problema de conjugación en el grupo de trenzas deje de ser un problema computacionalmente complejo y los sistemas criptográficos que en este problema se sustentan dejarían de ser viables.

Este riesgo se corre con cualquier esquema criptográfico, incluyendo los que se utilizan diariamente en los grandes sistemas y se consideran seguros. Debido a esto no se debe subestimar la importancia del estudio y el desarrollo de nuevos protocolos criptográficos. En el caso del grupo de trenzas existen otros problemas algorítmicos que comienzan a cobrar importancia debido a los desarrollos para la solución del problema de conjugación, y otro enfoque es el de buscar nuevos grupos donde los sistemas criptográficos definidos puedan funcionar y cuenten con un problema de conjugación aparentemente más difícil de resolver.

La implementación de los sistemas criptográficos funcionó correctamente y se cumplió con los objetivos de la misma, sin embargo quedan cosas por hacer para mejorarla. La eficiencia de la implementación era algo que siempre se consideró vendría como resultado de una programación clara y sencilla, pero no se consideraron los problemas que se encontrarían durante la implementación del sistema.

El lenguaje de programación Python proporciona una sintaxis de código claro y permite el desarrollo veloz de aplicaciones, pero al ser un lenguaje interpretado su rendimiento no se compara con aplicaciones desarrolladas en lenguajes donde el código es compilado, como C o C++. Aún así, se pueden realizar optimizaciones al programa en su estado actual, comenzando por preparar, en la medida que sea posible, los algoritmos para que funcionen en sistemas multihilos, los cuales ya se pueden encontrar con mucha facilidad en toda una serie de dispositivos.

También se puede optimizar el uso de memoria del sistema, en [7] se sugiere tener instancias únicas de los factores canónicos y que los objetos sólo contengan referencias a estos objetos únicos, se sugiere también tener una tabla de operaciones donde se indique el resultado de operar los factores, tabla que se consulta cada vez que se realiza una operación entre los factores.

Asimismo, queda pendiente la incorporación de los sistemas criptográficos en un sistema real, donde se realice una transmisión real de información entre dos partes de manera remota. Seguramente al hacer esta extensión de la implementación se harán observaciones de la misma que en la implementación actual no podían verse.

La seguridad de la información es un tema que ha tomado especial importancia en los últimos años. Los ataques a sitios de Internet y centros de información se han vuelto más comunes y la información acerca de los mismos se difunde en mayor medida que antes.

En Junio del 2011 se reveló la noticia de que la información de más de 200 mil cuentas de clientes de Citibank fueron filtradas de sus servidores por criminales cibernéticos[1]; se teme que la información obtenida se utilice en delitos de fraude y robo de identidad. En cualquier sistema, en especial si el sistema maneja información crítica de los usuarios, deben considerarse todas las posibles *vulnerabilidades* de cada una de las partes del mismo. Una falla en uno de los componentes puede crear una vulnerabilidad que puede ser explotada por algún usuario malicioso, lo que le permitiría tener acceso a la información o tomar el control del sistema.

A pesar de que sólo una parte de los ataques exitosos tiene que ver con debilidades de los sistemas criptográficos, es importante que cuando se desarrollen estos sistemas se realicen las pruebas necesarias para validarlos y se haga un juicio honesto sobre la seguridad que ofrecen antes de utilizarlos en un sistema real. Parte de la responsabilidad de los involucrados en el desarrollo de estos sistemas consiste en no apresurar este proceso y en no permitir que un sistema que no sea seguro sea utilizado en aplicaciones reales.

[1]BBC,`www.bbc.co.uk/news/technology-13711528 (verificado al 12 de Noviembre)`

Apéndices

Apéndice A

Referencia de uso de la implementación

El propósito de este apéndice es dar una referencia para el uso de la implementación de los grupos de trenzas y la criptografía con los mismos. Para usar la implementación se necesita descargar el archivo `http://min.us/lbphmijfyWcFHC` que contiene los archivos fuente Python, así como tener instalada una distribución reciente de Python, compatible con la versión 2.7. Una vez descargado el archivo, se debe descomprimir en alguna carpeta e iniciar una sesión interactiva de Python en dicha carpeta. Las siguientes secciones detallan cómo se utilizan los módulos de la implementación.

A.1. El grupo de trenzas

En el apéndice se mostrarán tanto entradas como salidas de la consola de Python, las entradas se denotarán usando los símbolos que aparecen en la línea de comandos de la sesión de Python junto con los comandos que se irán a ejecutar en la misma:

```
>>> "hola mundo"
```

las salidas se reproducirán directamente de los resultados de la sesión de Python:

```
'hola mundo'
```

El archivo fuente `trenzas.py` contiene las clases FactorCanónico y Trenza que fueron descritas en el capítulo 5. Para poder utilizar las clases en la sesión actual es necesario importar el archivo con la instrucción:

```
>>> from trenzas import *
```

Una vez importado el archivo se puede comenzar a trabajar con objetos del grupo de trenzas. Para simplificar los ejemplos y facilitar la visualización se utilizarán únicamente trenzas del grupo B_5 y se usarán trenzas con un pequeño número de factores canónicos. Para crear objetos de la clase FactorCanónico se necesita indicar la permutación que corresponde al factor canónico que queremos crear:

```
>>> a = FactorCanonico(5,[5,1,2,3,4])
>>> a
Factor Canonico
Hilos:  5
[5, 1, 2, 3, 4]
```

Como se observa en la salida, al evaluar el factor canónico, en la línea de comandos se escribe una representación del mismo. Ahora con el factor canónico se pueden hacer las operaciones que se describieron en el capítulo 5.

```
>>> a.tau(1)
Factor Canonico
Hilos:  5
[2, 3, 4, 5, 1]
>>> a.tau(2)
Factor Canonico
Hilos:  5
[5, 1, 2, 3, 4]
>>> a.inverso()
[2, 3, 4, 5, 1]
```

Por último, para las operaciones binarias de factores canónicos se pueden usar los operadores sobrecargados para la comparación y el producto:

```
>>> b = FactorCanonico(5,[1,4,2,3,5])
>>> a*b
Factor Canonico
Hilos:  5
```

```
[5, 1, 4, 2, 3]
>>> a == b
False
>>> a == a
True
```

Para crear un objeto tipo Trenza, es necesario indicar en una lista los factores canónicos que la forman, así como el exponente de la trenza fundamental que aparece al inicio de la trenza:

```
>>> A = Trenza(5,3,[a,b,a])
>>> A
Trenza
Hilos: 5
Exponente: 3 Longitud: 3
[5, 1, 2, 3, 4] , [1, 4, 2, 3, 5] , [5, 1, 2, 3, 4]
```

De la misma forma que con el factor canónico, ahora se pueden utilizar los métodos de la clase Trenza para el objeto que se acaba de crear:

```
>>> A.sup()
5
>>> A.inf()
3
>>> A.forma_canonica()
>>> A
Trenza
Hilos: 5
Exponente: 3 Longitud: 2
[5, 4, 3, 1, 2] , [1, 2, 3, 5, 4]
```

Se pueden generar trenzas y factores canónicos aleatorios utilizando los procedimientos *genera_trenza* y *genera_factor*:

```
>>> c = genera_factor(5)
>>> c
```

```
Factor Canonico
Hilos:  5
[4, 3, 2, 5, 1]
>>> B = genera_trenza(5,3)
>>> B
Trenza
Hilos: 5
Exponente: 0 Longitud: 3
[1, 5, 3, 2, 4] , [2, 5, 3, 1, 4] , [5, 2, 1, 4, 3]
```

Se recomienda también usar los operadores sobrecargados para realizar operaciones con trenzas, ya que facilitan la lectura y comprensión de las instrucciones:

```
>>> -A
Trenza
Hilos: 5
Exponente: -5 Longitud: 2
[4, 5, 3, 2, 1] , [2, 1, 3, 4, 5]
>>> C = A*B
>>> D = B*A
>>> C == D
False
>>> C
Trenza
Hilos: 5
Exponente: -5 Longitud: 5
[4, 5, 3, 2, 1] , [2, 1, 3, 4, 5] , [1, 5, 3, 2, 4] ,
[2, 5, 3, 1, 4] , [5, 2, 1, 4, 3]
>>> D
Trenza
Hilos: 5
Exponente: 4 Longitud: 3
[2, 5, 4, 3, 1] , [5, 1, 4, 2, 3] , [1, 2, 5, 3, 4]
```

Usando las clases FactorCanónico y Trenza se pueden programar y extender para programar todo tipo de algoritmos. Como un ejemplo simple se reproduce un método que resuelve la ecuación $Ax = B$, que tiene solución en todo grupo:

```
def resuelve(A,B):
    return (-A)*B


>>> C = resuelve(A,B)
>>> A*C == B
```

A.2. Criptografía

Para utilizar el módulo de criptografía es necesario tener instalada la librería PyCrypto, que se puede descargar de `www.dlitz.net/software/pycrypto/`. El archivo fuente que contiene los procedimientos para los sistemas criptográficos es `cripto.py` y se importa de la misma forma que `trenzas.py`.

```
>>> from cripto import *
```

Los métodos AAG y DH ejemplifican un intercambio de llave entre dos partes de manera local y su resultado es una llave común que puede utilizarse para cifrar:

```
>>> llaveA = AAG(5,5)
Criptosistema Anshel-Anshel-Golfeld
Elementos privados de A y B generados
Elementos públicos de A y B generados
Elemento común generado
Son iguales:
True
Llave común generada
>>> llaveB = DH(10,5)
Criptosistema DH
Elementos privados de A y B generados
Elementos públicos de A y B generados
Elemento común generado
```

```
Son iguales:
True
Llave común generada
True
>>> llaveA
"\xf7yF\xa7\x93x\x8f\x80\xab$'\xf1\r\xc6C_\xe7
\xae\xb2\xd4<\x15\xbb\xbd\x0b\x99\x98\x993\x02=/"
>>> llaveB
'\xf3\x05\x8d\xa8\x89\xfb\xe0\xfc\xa7\x963\xd5
\x07R+%\n\xe9\x16\xbb\xed\x19"\xf5%\'\xd1,[dlp'
```

Las llaves son cadenas de 256 bits que son el resultado de la función Hash SHA256, que luego se utilizan para cifrar con el criptosistema AES.

```
>>> cadena="El perro es el mejor amigo del hombre"
>>> cipher = AES.new(llaveA)
>>> cipher
<AES object at 0x023941C8>
>>> cifrado= EncodeAES(cipher,cadena)
>>> cifrado
'ZInSzDyKxP7jJEcoub6+ChpFocvzFoTWmHMsTU
9i/kwqGA30eyDAg2rSHMu04/17PuHzME+ML3aIHPA0lWd68A=='
>>> descifrado = DecodeAES(cipher,cifrado)
>>> descifrado
'El perro es el mejor amigo del hombre'
```

El código puede ampliarse para que se realice un intercambio propio de llaves, por ejemplo a través de una red de computadoras, pero dicha implementación va más allá del alcance del trabajo presente.
Los procedimientos relacionados con la solución al problema de conjugación están en el archivo fuente `conjugacion.py`. La manera de importar el archivo es idéntica a la que se ha usado antes.

```
>>> from conjugacion import *
```

Para encontrar el conjugador de dos elementos del grupo de trenzas se utiliza el procedimiento *obtener_conjugador*:

```
>>> A = genera_trenza(3,5)
>>> C = genera_trenza(3,2)
>>> B = A.conjugar(C)
>>> D = obtener_conjugador(A,B)
>>> A.conjugar(D) == B
True
>>> D == C
True
```

Para ver los detalles del algoritmo y la implementación se recomienda revisar los capítulos 4 y 5, así como los códigos fuente de la implementación.

Apéndice B

Código fuente

Se incluyen en el presente apéndice el código fuente de Python de las clases Factor Canónico, así como algunos de los métodos de la solución del problema de conjugación en el grupo de trenzas. Se utilizaron listas con índice inicial 1 para aumentar la claridad en los algoritmos.
Primero se presenta la clase factor canónico, se omiten los métodos que sirven para sobrecargar los operadores así como el método que escribe la presentación de la trenza en la línea de comandos de Python.

```
class FactorCanonico:
    """Clase que representa un factor canónico """

    #contructoras

    def __init__(self,n, perm):
        self.n = n
        self.perm = n_based_list(1,perm)

    def compara(self,y):
        """Compara dos factores canónicos
        por medio de la comparación de sus
        permutaciones elemento a elemento"""
```

```
        res = True
        if self.n != y.n:
            res= False
        else:
            for i in range(1,self.n+1):
                if y.perm[i] != self.perm[i]:
                    res = False
        return res

    def inverso(self):
        """Obtiene el inverso de un factor canónico
        regresando la permutación inversa"""
        li = n_based_list(1,range(1,self.n+1))
        for i in range(1,self.n+1):
            li[self.perm[i]] = i
        return li

    def _inverso(self):
        """Solo para uso interno, regresa un factor
        canónico que tiene como permutación la
        permutación inversa(Nota: no es el inverso)"""
        inv = self.inverso()
        res = FactorCanonico(self.n,res)
        return res


    def producto(self,b):
        """obtiene el producto de dos factores canónicos
        regresando el factor canónico con la permutación
        igual al producto de sus permutaciones"""
        li = n_based_list(1,range(1,self.n+1))
        for i in range(1,self.n+1):
```

```
            """li[i] = self.perm[b.perm[i]]"""
            li[i] = b.perm[self.perm[i]]
        prod = FactorCanonico(self.n,li)
        return prod

    def _tau(self):
        """obtiene el valor de aplicar el automorfismo
        tau a la trenza, multiplicando por la izquierda
        por el inverso de la trenza fundamental y por
        la derecha por la trenza fundamental"""
        li = n_based_list(1,range(1,self.n+1))
        li.reverse()
        delta = FactorCanonico(self.n,li)
        inv = delta.inverso()
        delta_inv =  FactorCanonico(self.n,inv)
        tau = delta_inv.producto(self)
        tau = tau.producto(delta)
        return tau

    def tau(self,n):
        """obtiene el resultado de aplicar el
        automorfimso tau elevando a la potencia
        n, dependiendo de la paridad de n se
        reduce a regresar el mismo factor
        o una aplicación normal de tau"""
        tau = self
        if n % 2 == 1:
            tau = self._tau()
        return tau


    def _ordena_C(self, C, ss, tt, A, B, U, V, W):
```

```
"""Ordena el arreglo C de acuerdo con
las permutaciones A,B"""
if tt > ss:
    mm = int(math.floor( (ss+tt)/2))
    self._ordena_C(C, ss, mm, A, B, U, V, W)
    self._ordena_C(C, mm+1, tt, A, B, U, V, W)
    U[mm] = A[C[mm]]
    V[mm] = B[C[mm]]
    if ss < mm:
        ran = range(ss,mm)
        ran.reverse()
        for i in ran:
            U[i] = min(A[C[i]],U[i+1])
            V[i] = min(B[C[i]],V[i+1])
    U[mm+1] = A[C[mm+1]]
    V[mm+1] = B[C[mm+1]]
    if tt > (mm+1):
        for i in range(mm+2,tt+1):
            U[i] = max(A[C[i]],U[i-1])
            V[i] = max(B[C[i]],V[i-1])
    ll = ss
    rr = mm+1
    for i in range(ss,tt+1):
        if ((ll > mm) or ((rr <= tt) and (U[ll] > U[rr])
                    and (V[ll]> V[rr]))):
            W[i] = C[rr]
            rr = rr + 1
        else:
            W[i] = C[ll]
            ll = ll + 1
    for i in range(ss,tt+1):
        C[i] = W[i]
```

```
    def infimo(self, T):
        """Obtiene el infimo de dos trenzas, la actual
        y T"""
        A = self.perm
        B = T.perm
        C = n_based_list(1,range(1, self.n + 1))
        U = n_based_list(1,map(lambda x: x*0 ,range(1, self.n + 1)))
        V = n_based_list(1,map(lambda x: x*0 ,range(1, self.n + 1)))
        W = n_based_list(1,map(lambda x: x*0 ,range(1, self.n + 1)))
        self._ordena_C(C, 1, self.n, A,B,U,V,W)
        res = FactorCanonico(self.n,C)
        inf_inv = res.inverso()
        inf = FactorCanonico(self.n,inf_inv)
        return inf
```

A continuación se presenta el código fuente de la clase Trenza, de la cual se omiten algunos métodos de la clase con funciones no esenciales.

```
class Trenza:
    """Clase que representa una trenza por medio de factores canónicos"""

    def __init__(self, n, s, factores):
        """ Constructor donde n es el número de hilos en el grupo de
        trenzas, r = s es el exponente de la trenza fundamental y
        factores es una lista de factores canónicos"""
        self.n = n
        self.fact = n_based_list(1,factores)
        self.r = s
        self.k = len(self.fact)
```

```
        self._can = False

    def __repr__(self):
        cad = "Trenza" + "\n"
        cad = cad + "Hilos: " + str(self.n) + "\n"
        cad = cad + "Exponente: " + str(self.r) + " "
        cad = cad + "Longitud: " + str(self.k) + "\n"
        if self.k > 0:
            facts = map( lambda x: str(x.perm), self.fact)
            facts = reduce( lambda x,y: x + " , " + y, facts)
            cad = cad + facts
        else:
            cad = cad + "[]"
        return cad

    def producto(self, B):
        """ Devuelve el resultado de multiplicar la trenza actual por
        B, una operación de grupo"""
        ind = self.r + B.r
        la = self.fact
        lb = B.fact
        la = map( lambda fc : fc.tau(B.r) , la)
        la = n_based_list(1,la + lb)
        res = Trenza(self.n, ind, la)
        return res

    def inverso(self):
        """ Devuelve el inverso de la trenza actual, operación de grupo"""
        exponente = -(self.r + self.k)
        li = map(lambda x:x ,self.fact)
        li.reverse()
        li = map(lambda fc : fc.inverso(), li)
```

```
        li = map(lambda inv: FactorCanonico(self.n,inv), li)
        perm = range(1,self.n + 1)
        perm.reverse()
        fund = FactorCanonico(self.n, perm)
        li = n_based_list(1,map(lambda fc : fc.producto(fund), li))
        for i in range(1,self.k+1):
            li[i] = li[i].tau(-(self.r + (self.k +1 - i)) )
        li = n_based_list(1,li)
        inv = Trenza(self.n, exponente ,li)
        return inv

    def conjugar(self,B):
        inv_B = B.inverso()
        res = B.producto(self)
        res = res.producto(inv_B)
        return res

    def _cabeza(self , A, B):
        perm = range(1, self.n +1)
        perm.reverse()
        fund = FactorCanonico(self.n, perm)
        inv_a = FactorCanonico(self.n, A.inverso())
        """aux = fund.producto(inv_a)"""
        aux = inv_a.producto(fund)
        cab = aux.infimo(B)
        return cab

    def _trivial(self, A):
        perm = range(1,self.n+1)
        iden = FactorCanonico(self.n, perm)
        res = iden.comparar(A)
        return res
```

```
    def _fundamental(self,A):
        perm = range(1,self.n+1)
        perm.reverse()
        fund = FactorCanonico(self.n, perm)
        res = fund.comparar(A)
        return res

    def forma_canonica(self):
        """Obtiene la forma canónica de la trenza actual, modifica
        la estructura de la trenza y marca la bandera _can como
        verdadera indicando que está en su forma canónica"""
        i = 1
        ll = self.k
        while i < ll :
            tt = ll
            ran = range(i,ll)
            ran.reverse()
            for j in ran:
                B = self._cabeza(self.fact[j],self.fact[j+1])
                if not self._trivial(B):
                    tt = j
                    self.fact[j] = self.fact[j].producto(B)
                    inv_b = FactorCanonico(self.n,B.inverso())
                    self.fact[j+1] = inv_b.producto(self.fact[j+1])
            i = tt + 1
        while ( ll > 0  and self._fundamental(self.fact[1])):
            self.fact.pop(1)
            self.k = self.k - 1
            self.r = self.r + 1
            ll = ll - 1
        while (ll > 0 and self._trivial(self.fact[ll])):
```

```
            self.fact.pop(ll)
            self.k = self.k - 1
            ll = ll - 1
        self._can = True

    def compara(self, B):
        """Compara dos trenzas, la actual y B, comparando primero
        los exponentes de la trenza fundamental y después elemento
        a elemento de sus listas de factores"""
        res = True
        if self._can == False:
            self.forma_canonica()
        if B._can == False:
            B.forma_canonica()
        if self.k != B.k or self.r != B.r or self.n != B.n:
            res = False
        if res:
            li = []
            for i in range(1, self.k+1):
                li.append( self.fact[i-1] == B.fact[i-1] )
                res = reduce( lambda x,y : x and y, li)
        return res


    def ciclar(self):
        if self._can == False:
            self.forma_canonica()
        if len(self.fact) == 0:
            return Trenza(self.n,self.r,[])
        li = n_based_list(1,self.fact)
        r = self.r
        k = self.k
```

```
        fac = li[1]
        for i in range(2,k+1):
            li[i-1] = li[i]
        li[k] = fac.tau(r)
        tren = Trenza(self.n,r,li)
        return tren

    def inv_ciclar(self):
        if self._can == False:
            self.forma_canonica()
        if len(self.fact) == 0:
            return Trenza(self.n,self.r,[])
        li = n_based_list(1,self.fact)
        r = self.r
        k = self.k
        fac = li[k]
        ran = range(1,k)
        ran.reverse()
        for i in ran:
            li[i+1] = li[i]
        li[1] = fac.tau(r)
        tren = Trenza(self.n,r,li)
        return tren
```

Finalmente también se incluye la función recursiva que dada una trenza del conjunto $SSS(A)$ de una trenza A obtiene una lista de objetos tipo Trenza que representan el conjunto $SSS(A)$ completo. Los parámetros de la función, en el orden en que son llamados, son el elemento de SSS de alguna trenza, obtenido por la operación de ciclado y ciclado inverso; una lista que incluye a los elementos de SSS conocidos, el ínfimo y supremo de los elementos del SSS de la trenza, obtenidos también por operaciones de ciclado; una lista, llamada *mapa*, que tiene en su posición i una indicación de los elementos de $[0, 1]$ por los que debe conjugarse la trenza de SSS inicial para obtener

la trenza en la posición i de de la lista que representa al conjunto SSS. Así, si en la posición j de *mapa* se tiene la cadena 0-1-17 se tiene por entendido que para obtener la trenza del SSS en la posición j debe conjugarse la trenza que se usó para generar el SSS primero por la trenza de $[0, 1]$ en la posición 1 y posteriormente por la que tiene la posición 17 en el orden que fueron generados. Finalmente el parámetro *ant* es una cadena que indica, a partir de la trenza inicial, el proceso de conjugación de la trenza del SSS con la que fue llamada la función; el valor inicial de *ant* es la cadena 0, que indica que no debe realizarse ninguna operación de conjugación a la trenza inicial para obtener la trenza inicial.

```
def aux_sss(el, lista,inf,sup,mapa,ant):
    lis = list(lista)
    #regresa una lista con los generadores
    print("li: " + str(len(lis)))
    gens = permutaciones(el.n)
    lig = map(lambda x : el.conjugar(x), gens)
    nuevo = map(lambda x : x.inf() == inf and x.sup() == sup
                and (not miembro(x,lis)), lig)
    sigs = []
    sigsm = []
    for i in range(0,len(nuevo)):
        if(nuevo[i] == True):
            sigs.append(lig[i])
            aux = list(ant)
            aux.append(i)
            sigsm.append(aux)
    print(str(len(sigs)))
    lis.extend(sigs)
    mapa.extend(sigsm)
    for i in range(len(sigs)):
        res = aux_sss(sigs[i],list(lis),inf,sup,mapa,sigsm[i])
        lis = res[0]
```

```
        mapa = res[1]
        print("-")
    print("+")
    res = [lis,mapa]
    return res
```

Bibliografía

[1] Anshel, I., Anshel, M., Goldfel, D. An algebraic method for public key criptography. *Math. Research letters 6* (1999).

[2] Anshel, I., Anshel, M., Goldfel, D., Fisher, B. New key agreement protocols in braid group criptography. *Springer Lec. Notes. in Comp. Sci.* (2001).

[3] Artin, E. *Theorie der Zöpfe.* Abh. Math. Sem. Univ. Hamburg, 1925.

[4] Choon, C. J., Ko, K. H., Lee, S. J., Han, J. W., Hee, J. An efficient implementation of braid groups. *Advances in criptography ASIACRIPT 2248* (2001), 144–156.

[5] Dehornoy, P. Braid-based criptography. *Contemp. Math 350* (2004), 5–33.

[6] Diffie, W., Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory* (1976).

[7] Elrifai, E. Algorithms in positive braids. *Oxford Quarterly Journal of Mathematics* (1994).

[8] Franco, N., Gonzales-Meneses, J. Conjugacy problem for braid groups and garside groups. *J. Algebra 266* (2003), 21.

[9] Garber, D. Braid group criptography. *Lecture notes of Tutorials given at Braids PRIMA Summer School at Singapore* (2008).

[10] Garside, F. The braid group and other groups. *Oxford Quarterly Journal of Mathematics* (1969).

[11] Gebhardt, V. A new approach to the conjugacy problem in garside groups. *Journal of Algebra 292* (2003), 282–302.

[12] Hall, M. *The Theory of Groups*. The Macmillan Company, 1999.

[13] Hungerford, T. W. *Algebra*. Graduate Texts in Mathematics. Springer, 2003.

[14] Ko, K., Lee, S., Cheon, J., Han, J., Kang, J., Park, C. New public key criptosystems using baid groups. *Crypto 2000* (2000).

[15] Kurosh, A. G. *The Theory of Groups*. Chelsea Publishing Company, 1960.

[16] Mahlburg, K. An overview of braid group criptography. *Group* (2004), 1–13.

[17] Menezes, A. J., van Oorschot, P. C., Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 2001.

[18] Myasnikov, A., Shpilrain, V., Ushakov, A. *Group-based Cryptography*. Advanced Courses in Mathematics CRM Barcelona. Birkhäuser Verlag, 2008.

[19] National Institute of Standards and Technology (NIST). Advanced encryption standard(aes). *Federal information, Processing standards publication 197* (2001).

[20] Rivest, R., Shamir, A., Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21* (1978), 120–126.

[21] Rotman, J. J. *An Introduction to the Theory of Groups*. Springer-Verlag, 1995.