Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Computación

Ocultamiento de información en documentos de formato abierto

Tesis que presenta

Michel Ruiz Tejeida

para obtener el grado de

Maestro en Ciencias en Computación

Director de Tesis

Dr. Guillermo Benito Morales-Luna

México, Distrito Federal

Noviembre 2013

Esta tesis se la dedico a mis padres Miguel Ruiz y Amada Tejeida a quienes les debo mi existencia y todo lo que he logrado; a mi esposa Michelle Herrera por apoyarme y convertir cada momento a mi lado en un recuerdo inolvidable; a mis hermanos Yoset, Areli y mi sobrina Melany por su apoyo y cariño. Y especialmente a mi hijo Miguel Ruiz Herrera por ser mi motivo de superación.

RESUMEN

El éxito de Internet, su bajo costo, la promesa de mayor ancho de banda y calidad en los servicios, tanto para redes cableadas como inalámbricas, ha permitido crear, reproducir, transmitir y distribuir contenido digital al menor esfuerzo. Privacidad en las comunicaciones digitales se desea cuando información confidencial se comparte entre dos entidades. La criptografía ofrece soluciones para impedir que la información pueda ser entendida por alguien ajeno al destinatario; pero es posible identificar los archivos cifrados, conocer las entidades que se comunican y ser blanco de ataque por usuarios maliciosos.

Para evitar este problema de seguridad, técnicas para ocultar información como la *esteganografía* son la solución. La comunicación a través de la esteganografía hace difícil su rastreo y para quienes deseen obtener información es todo un reto, porque no es posible identificar de forma eficiente cuales archivos tienen oculto un mensaje. Aún cuando la esteganografía ofrece este grado de privacidad, la seguridad para impedir que alguien obtenga el mensaje secreto, en posesión del archivo alterado, radica en el proceso esteganográfico utilizado para ocultarlo.

En esta tesis, se introduce un nuevo protocolo para el ocultamiento de información en documentos de texto dispuestos en formato abierto y estandarizado por la ISO. Además, mostramos una herramienta de software creada para probar la robustez del protocolo en diferentes entornos de trabajo.

ABSTRACT

The success of the Internet, cost-effective and the promise of higher bandwidth and quality of services for both wired and wireless networks has made it possible to create, replicate, transmit, and distribute digital content in an effortless way. Privacy in digital communications is desired when confidential information is shared between two entities. Cryptography offers solutions for preventing information can be understood by someone outside the recipient, but it is possible to identify encrypted files, know the entities want to communicate and be targeted for attack by malicious users.

To avoid this security issue, information hiding techniques are the solution. Communication through steganography makes it difficult to trace and seeking information is a challenge, because it is not possible to efficiently identify which files have been modified. Although, steganography offers privacy, security to prevent anyone get the secret message, in possession of the modified file lies in the process used to hide it.

In this thesis, we introduce and information hiding protocol for textual documents arranged in open formats which are open and standards by the ISO. We show a software tool created to test the protocol's robustness on serveral work-group enviroments.

AGRADECIMIENTOS

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [22]

En primer lugar, agradezco a mi familia por su amor incondicional, apoyo, comprensión y consejos brindados para realizar y concluir mis estudios de maestría.

Doy las gracias a mi asesor el Dr. Guillermo Morales Luna por aceptarme como su estudiante y por su orientación a través de este proyecto. A los Drs. Debrup Chakraborty y Feliú Sagols Troncoso por el tiempo dedicado en revisar mi tesis.

También quiero dar las gracias al CINVESTAV por la oportunidad de ser parte de esta comunidad científica, por el apoyo económico para asistir a los congresos nacionales e internaciones, especialmente al Departamento de Computación, incluyendo los profesores, secretarias y compañeros que de una u otra forma me ayudaron durante mi estancia en el CINVESTAV.

Finalmente, quiero agradecer al CONACyT, por el apoyo económico brindado durante los primeros dos años de la maestría.

ÍNDICE GENERAL

Algoritmos xi

Figuras xii

Tablas xiii

Acrónimos xiv

1 INTRODUCCIÓN 1

2 OCULTAMIENTO DE LA INFORMACIÓN 3
- 2.1 Introducción . . . 3
- 2.2 Sistema genérico . . . 4
- 2.3 Esteganografía . . . 4
  - 2.3.1 Antecedentes . . . 5
  - 2.3.2 Sistema esteganográfico o estegosistema . . . 5
- 2.4 Marcas de agua digitales . . . 6
  - 2.4.1 Antecedentes . . . 6
  - 2.4.2 Sistema de marcas de agua digitales . . . 7

3 ALGORITMOS ESTEGANOGRÁFICOS 9
- 3.1 Algoritmos y estegosistemas para imágenes . . . 9
  - 3.1.1 Bit Menos Significativo . . . 10
  - 3.1.2 Bit Menos Significativo Aleatorio . . . 12
  - 3.1.3 Transformación de Karhunen-Loève . . . 13
  - 3.1.4 F5 . . . 15
- 3.2 Algoritmos y estegosistemas para documentos . . . 17
  - 3.2.1 Modificación de los elementos del texto . . . 17
  - 3.2.2 Espacios blancos . . . 18
  - 3.2.3 Lingüísticos . . . 19
  - 3.2.4 Ordenamiento de datos . . . 19

4 FORMATOS ESTÁNDARES PARA DOCUMENTOS 23
- 4.1 El Formato de Documento Abierto para Aplicaciones Ofimáticas . . . 23
  - 4.1.1 Estructura del archivo . . . 24
  - 4.1.2 Convenciones comunes del XML . . . 30
  - 4.1.3 Documentos y sus extensiones . . . 30
  - 4.1.4 El documento de texto . . . 30
- 4.2 El Formato de Documento Portátil . . . 31
  - 4.2.1 Propiedades . . . 32
  - 4.2.2 Sintaxis . . . 33

4.2.3 Estructura del archivo 35

5 DISEÑO DEL ESQUEMA ESTEGANOGRÁFICO 43
5.1 Objeto anfitrión 43
5.2 Enumeración de permutaciones 45
5.3 Definición, notación y terminología 48
5.4 Algoritmo para generación de claves 49
5.5 Algoritmo de incrustación 50
5.6 Algoritmo de extracción 51
5.7 Seguridad del estegosistema 51

6 LA IMPLEMENTACIÓN DEL ESQUEMA 53
6.1 Una breve descripción del sistema 53
6.2 El paquete esteganográfico 55
6.2.1 Manipulación de archivos 55
6.2.2 Cómputo de permutaciones 57
6.2.3 Generador de claves esteganográficas 58
6.2.4 Incrustación del mensaje 59
6.2.5 Extracción del mensaje 59
6.3 Aplicación multiplataforma 59
6.3.1 Creación de la interfaz de usuario 59
6.3.2 Capa de negocio 60
6.4 Pruebas 60
6.4.1 Cálculos matemáticos 61
6.4.2 Validación de archivos 61
6.4.3 Funcionales 61

7 APLICACIONES 63
7.1 Autenticación de documentos 63
7.2 Seguimiento de documentos 64

8 CONCLUSIONES Y PERSPECTIVAS 67
8.1 Conclusiones 67
8.2 Perspectivas 67

A EL PAQUETE DE DESARROLLO 69

BIBLIOGRAFÍA 75

LISTA DE ALGORITMOS

3.1 Incrustación por el método LSB 10
3.2 Incrustación por el método Aleatorio LSB 12
3.3 Incrustación por el método F5 17

5.1 Cambio de base decimal a factorial 46
5.2 Obtener la i-ésima permutación 47
5.3 Obtener el índice de una permutación 48
5.4 Cambio de base factorial a decimal 48
5.5 Generar un grupo cíclico y un elemento generador 50
5.6 Algoritmo de incrustación 51
5.7 Algoritmo de extracción 52

ÍNDICE DE FIGURAS

Figura 2.1 Esquema de un sistema genérico para ocultar información 4
Figura 2.2 Esquema de un sistema esteganográfico 6
Figura 2.3 Esquema de un sistema de marcas de agua digitales 8
Figura 3.1 Ejemplo de un archivo PostScript. 20
Figura 3.2 Ejemplo de un archivo PostScript con comandos de posiciones explícitas. 20
Figura 3.3 Ejemplo de un archivo PostScript permutado aleatoriamente. 21
Figura 3.4 Ejemplos de esteganografía usando permutaciones. 21
Figura 4.1 Ejemplo de una tabla de referencias cruzadas del formato PDF 37
Figura 5.1 Atributos permutables del formato ODF 44
Figura 5.2 La tabla de referencias cruzadas del formato PDF 44
Figura 5.3 Creación del elemento *F* en el esquema esteganográfico 45
Figura 6.1 Cadenas para identificar el tipo de archivo en el encabezado 55
Figura 6.2 Permutación inicial del objeto anfitrión 56
Figura 6.3 Pantalla principal del sistema, en un dispositivo iPhone 60
Figura 7.1 Esquema de incrustación para autenticación de documento 64
Figura 7.2 Esquema de extracción para verificación de integridad de documento 65

ÍNDICE DE TABLAS

Tabla 3.1 Tabla comparativa de los estegosistemas para imagen. 9
Tabla 3.2 Tabla comparativa de los estegosistemas para documentos de texto. . 17
Tabla 4.1 Extensiones empleadas para los tipos de documentos del formato ODF. 30
Tabla 4.2 Caracteres de espacio en blanco 35
Tabla 4.3 Las entradas en el diccionario *trailer* 38
Tabla 5.1 Enumeración de todas las permutaciones para $n = 3$ 46
Tabla 5.2 Ejemplo del sistema de números factoriales 47
Tabla 6.1 Contenido de un empaquetado ODF 57

ACRÓNIMOS

AES	Estándar Avanzado de Cifrado (siglas del inglés *Advanced Encryption Standard*)
ASCII	Código Estándar Estadounidense para el Intercambio de Información (acrónimo inglés de *American Standard Code for Information Interchange*)
API	Interfaz de Programación de Aplicaciones (siglas del inglés *Application Programming Interface*)
BMP	Mapa de Bits de *Windows* (siglas del inglés *Windows bitmap*)
CR	Retorno de carro (del inglés *Carriage Return*)
DCT	Transformada de Coseno Discreta (del inglés *Discrete Cosine Transform*)
DES	Estándar de Cifrado de Datos (siglas del inglés *Data Encryption Standard*)
DIIT	*Digital Invisible Ink Toolkit*
EOL	Fin De Línea (del inglés *End-Of-Line*)
EOF	Fin De Archivo (del inglés *End-Of-File*)
FF	Salto de página (del inglés *Form Feed*)
GIF	Formato de Intercambio de Gráficos (siglas del inglés *Graphics Interchange Format*)
GNU/GPL	Licencia Pública General de GNU (siglas del inglés *General Public License*)
HT	Tabulador
HTML	Lenguaje de Marcado de HiperTexto (siglas de inglés *HyperText Markup Language*)
IEC	Comisión Internacional de Electrotécnica (siglas del inglés *International Electrotechnical Commission*)
ISO	Organización Internacional de Normalización (siglas del inglés *International Organization for Standardization*)
JAI	Procesamiento Avanzado de Imágenes de Java (siglas del inglés *Java Advanced Imaging*)
JAR	Archivo Java (siglas del inglés *Java Archive*)

JPEG	Grupo de Expertos en Fotografía (siglas en inglés de *Joint Photographic Experts Group*)
KLT	Transformación de Karhunen-Loève (siglas del inglés *Karhunen–Loève transform*)
LF	Salto de línea (del inglés *Line Feed*)
LSB	Bit Menos Significativo (siglas del inglés *Least Significant Bit*)
MAC	Código de Autenticación de Mensaje (siglas del inglés *Message Authentication Code*)
MD5	Algoritmo de Resumen del Mensaje 5 (abreviatura del inglés *Message-Digest Algorithm 5*)
MIME	Extensiones Multipropósito de Correo de Internet (siglas del inglés *Multipurpose Internet Mail Extensions*)
OASIS	Organización para el Avance de los Estándares de Información Estructurada (en inglés *Organization for the Advancement of Structured Information Standards*)
ODF	Formato de Documento Abierto para Aplicaciones Ofimáticas de OASIS (en inglés, *OASIS Open Document Format for Office Applications*)
OTP	Clave de un solo uso (en inglés *One-time Pad*)
PDF	Formato de Documento Portátil (siglas del inglés *Portable Document Format*)
PKI	Infraestructura de Clave Pública (siglas del inglés *Public-Key Infrastructure*
PNG	Gráficos de Red Portátiles (siglas del inglés *Portable Network Graphics*)
RELAX NG	Lenguaje Regular para la Siguiente Generación de XML (del inglés *REgular LAnguage for XML Next Generation*)
SP	Espacio (del inglés *Space*)
TC	Comité Técnico (del inglés *Technical Committee*)
VSL	*Virtual Steganographic Laboratory for Digital Images*
XML	Lenguaje de Marcado eXtensible (siglas del inglés *eXtensible Markup Language*)
XTSP	Problema del viaje Exacto del Agente Viajero (siglas del inglés *Exact Travel of Salesman Problem*)

1

INTRODUCCIÓN

La *esteganografía* digital es el arte de ocultar, discretamente, datos dentro de los datos. El objetivo, en general, es ocultar suficientemente bien los datos de tal forma que los destinatarios no sospechen del medio en que se comunican. Esteganografía es también un medio de almacenamiento de información basada en la idea de mantener oculta su existencia, que junto con los métodos de comunicación existentes se puede utilizar para llevar a cabo intercambios ocultos.

Los gobiernos están interesados en éste tipo de comunicación, específicamente cuando se emplea para los siguientes propósitos: los que apoyan la seguridad nacional y los que no lo hacen. La esteganografía ofrece un enorme potencial para ambos tipos. Las empresas pueden tener problemas similares con respecto a los secretos comerciales o información de nuevos productos. Establecer una comunicación a través del ocultamiento de información reduce en gran medida el riesgo de fuga de ésta en tránsito. La esteganografía también puede mejorar la privacidad individual. Aunque no es un sustituto para la criptografía, la esteganografía proporciona un medio de comunicación privado. Por supuesto, esto sólo es eficaz si no se detecta la comunicación oculta.

Hay muchas técnicas disponibles para el ocultamiento de información. La técnica más común es la de explotar las limitaciones leves de los formatos de archivos populares. Muchos paquetes de software disponibles públicamente utilizan esta técnica en una variedad de medios de comunicación.

El medio más utilizado son las imágenes digitales. Mientras más detallada es una imagen, menos limitaciones existen en la cantidad de datos que puede ocultar antes de que sea sospechosa. Esto se logra al modificar insignificantes partes de la imagen tratando de no afectarla visualmente. Algunas modificaciones utilizadas radican en alterar los bits menos significativos de la imagen, o el mapa de colores, y en la mayoría de las veces se logra mantener la imagen intacta, a simple vista.

Otros paquetes utilizan archivos de audio digital como medio de almacenamiento, que no solo esconden información de forma arbitraria, sino que han mostrado ser útiles para crear marcas de propiedad intelectual, mejor conocidas como *marcas de agua* digitales. Para este tipo de medio el objetivo es proporcionar un archivo marcado que no muestre alteración sonora de su contenido.

Por último tenemos los archivos de texto, un medio digital menos utilizado para ocultar información debido a sus características, como la facilidad de modificarlo o lo sencillo del formato. Una de las técnicas se basa en utilizar símbolos no visibles, como espacios en blanco o tabuladores, logrando ocultar información al asignarle un valor dependiendo del tipo de espacio utilizado; otros modifican ligeramente los signos de puntuación para utilizarlos como marcas de agua, e incluso establecen un espacio definido entre caracteres, líneas o párrafos, y con esto identificar si un archivo fue marcado.

Para todos los medios de almacenamiento descritos, el espacio de trabajo en donde se puede ocultar la información depende del formato, en este trabajo nos enfocamos en dos formatos específicos, el formato de documento abierto para aplicaciones ofimáticas y el formato de documento portátil. Estos formatos tienen elementos de meta-información que pueden ser permutados sin alterar de manera sensible el documento. Una técnica que se basa en la idea del ordenamiento de los datos, fue presentada por Artz [2]. Cuando los datos no tienen un orden estricto dentro de un formato, éste puede ser utilizado para ocultar información. A cada permutación de un conjunto de objetos se le puede asignar un número entero positivo. Esta asignación se puede entonces utilizar para codificar los datos ocultos mediante la alteración del orden de los objetos, que no se consideran ordenados por el medio portador.

La dificultad computacional de generar todas las permutaciones de un conjunto es la principal motivación para seleccionar este tipo de técnica y conformar un esquema de ocultamiento de información para proveer, un sistema de comunicación seguro basado en el ocultamiento de secretos.

La organización de esta tesis es como sigue. En el Capítulo 2 discutimos los conceptos básicos sobre las técnicas para el ocultamiento de información, a partir de sus diferencias filosóficas. En el Capítulo 3 se incluyen métodos que utilizan tanto imágenes como documentos de texto para el objeto anfitrión. En el Capítulo 4 presentamos la especificación técnica de los formatos utilizados por el esquema esteganográfico.

El Capítulo 5 presenta el modelo matemático y computacional de un nuevo esquema esteganográfico que permite ocultar información en formatos de archivo estándar, sin alterar su contenido o tamaño. En el Capítulo 6 presentamos los detalles de implementación del protocolo de ocultamiento y algunas pruebas realizadas para verificar la correcta definición del estegosistema. En el Capítulo 7 proponemos aplicaciones de nuestro esquema que muestran su potencial. Finalmente en el Capítulo 8 se hace un resumen de los resultados más importantes en todo el trabajo. Asimismo, se proponen líneas futuras de investigación.

2

OCULTAMIENTO DE LA INFORMACIÓN

El ocultamiento de la información, la esteganografía y las marcas de agua son tres campos estrechamente relacionados, que tienen un alto grado de superposición y comparten muchos enfoques técnicos. Sin embargo, existen diferencias filosóficas fundamentales que afectan los requerimientos, presentadas en la sección 2.1 y por lo tanto el diseño de una solución técnica, es decir un sistema para el ocultamiento de información tema descrito en la sección 2.2, los detalles particulares de cada técnica de ocultamiento se presentan en las secciones 2.3 y 2.4 para la esteganografía y marcas de agua, respectivamente.

2.1 INTRODUCCIÓN

Ocultar información es un término general que abarca una amplia gama de problemas más allá de integrar mensajes. La palabra *ocultar* se puede referir a hacer que determinada información sea imperceptible (marcas de agua) o bien mantener la existencia de la información en secreto (esteganografía).

El inventor de la palabra *esteganografía* es Trithemius [9], el autor de las primeras publicaciones sobre Criptografía: *Polygraphia* y *Steganographia*. El término técnico en sí se deriva de las palabras griegas *steganos*, que significa "oculto", y *graphia* "escritura". La esteganografía es el arte de la comunicación oculta. En donde la existencia del mensaje es en sí, un secreto.

Además de la tinta invisible, un ejemplo muy citado de la esteganografía es una antigua historia de Heródoto [10], que habla de un esclavo enviado por su amo, Histieo, a la ciudad jonia de Mileto, con un mensaje secreto tatuado en el cuero cabelludo. Una vez tatuado, el esclavo se dejó crecer el pelo con el objetivo de ocultar el mensaje. Luego viajó a Mileto y, al llegar, se afeitó la cabeza para revelar el mensaje al regente de la ciudad, Aristágoras. El mensaje anima a Aristágoras, para iniciar una revuelta contra el rey persa. En este escenario, el mensaje es primordial para Histieo y el esclavo es simplemente el portador del mensaje.

Podemos utilizar este ejemplo para destacar la diferencia entre esteganografía y marcas de agua. Supongamos que el mensaje tatuado en el cuero cabelludo del esclavo es "Este esclavo es propiedad de Histieo". De esta forma el mensaje se refiere al esclavo, y el único propósito de grabarlo en el cuero cabelludo es por cuestión de estética. En caso de que alguien reclame la posesión del esclavo, Histieo solo tiene que rapar al esclavo para demostrar que le pertenece. En este escenario, el esclavo es primordial para Histieo, y el mensaje sólo proporciona información sobre el esclavo, idea fundamental de las *marcas de agua*.

Cox et al. [9], presentan las siguientes definiciones a partir de esta diferencia filosófica.

> Se define *marcas de agua* como la práctica de alterar de forma imperceptible un objeto al incrustarle un mensaje acerca de éste.
>
> Se define *esteganografía* como la práctica de alterar de forma indetectable un objeto para incrustar un mensaje secreto.

2.2 SISTEMA GENÉRICO

A pesar de que los objetivos de las marcas de agua y esteganografía son muy diferentes, en aplicaciones comparten ciertos elementos de alto nivel. Ambos sistemas consisten de un algoritmo de *inscrustación* y uno de *detección*, como se ilustra en la Figura 2.1. El algoritmo de incrustación recibe dos parámetros de entrada. Uno de ellos es la información que queremos insertar (por ejemplo, ya sea la marca de agua o el mensaje secreto), y el otro es el objeto anfitrión que ocultará la información. La salida del algoritmo de incrustación usualmente es transmitido o almacenado. Posteriomente, este nuevo objeto (o algún otro objeto que no ha sido modificado por el algoritmo de incrustación) se utiliza como elemento de entrada al algoritmo de detección. La mayoría de los algoritmos de detección intentan determinar si existe información oculta, y en caso de éxito, recuperarla.

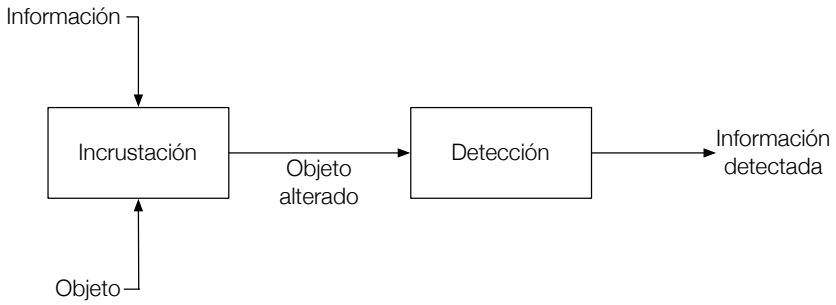

Figura 2.1: Esquema de un sistema genérico para ocultar información

Los sistemas para insertar mensajes en objetos pueden dividirse en sistemas de marcas de agua, en donde el mensaje esta relacionado con el objeto, y en sistemas sin marcas de agua, en donde el mensaje no tiene relación con el objeto. También se pueden dividir en forma independiente en sistemas esteganográficos, en el que la existencia misma del mensaje se mantiene en secreto, y en sistemas no esteganográficos, cuando no es necesario mantener el mensaje en secreto.

2.3 ESTEGANOGRAFÍA

La esteganografía es el acto de las comunicaciones secretas, lo que significa que sólo el emisor, Alicia, y el receptor, Beto, son conscientes de la comunicación secreta. Para lograr esto, el mensaje secreto, se oculta dentro de un objeto común con el propósito de no levantar sospecha de su contenido. Para un adversario, Isabel, está claro que Alicia y Beto se comunican, pero el objeto que oculta el mensaje parece ser inocuo.

El principal requisito de la esteganografía es que el mensaje secreto dentro del objeto alterado sea *indetectable*, que, vagamente definido, significa que no existe ningún algoritmo que puede determinar si un objeto contiene un mensaje oculto. El análisis esteganográfico es el proceso de detección de las comunicaciones esteganográficas. Y puesto que la esteganografía y el análisis esteganográfico están estrechamente entrelazados, parte de la discusión que sigue oscilará entre uno y otro.

La primer parte de esta sección, el tema 2.3.1 es una descripción breve de los antecedentes de la esteganografía. Posteriormente en el tema 2.3.2 se introducen las notaciones y terminología de un sistema para ocultar información.

2.3.1 *Antecedentes*

La primera evidencia escrita sobre la esteganografía, utilizada para enviar mensajes es la historia de Heródoto descrita al principio del capítulo. Heródoto también documentó la historia de Demerato, quien alertó a Esparta sobre el plan de invasión de Grecia por el persa Jerjes. Demerato tomó un cuadernillo de dos hojas o tablillas; rayó bien la cera que las cubría y en la madera misma grabó el mensaje y lo volvió a cubrir con cera regular. Esto permitió enviar el cuadernillo a Esparta sin levantar sospecha sobre su contenido.

Aeneas Tacticus propuso muchas técnicas esteganográficas que podrían considerarse el "estado del arte" de su tiempo [9], como ocultar mensajes en los aretes de las mujeres o los mensajes transportados por palomas. Más adelante, tenemos la esteganografía lingüística, también conocida como *acróstico*, que pasó a ser uno de los métodos antiguos más populares para ocultar información. Mensajes secretos eran codificados utilizando las letras iniciales de oraciones o tercetos sucesivas en un poema. Uno de los ejemplos más famosos es la *Amorosa visión* por Giovanni Boccaccio [3].

Versiones más avanzadas de esta esteganografía lingüística, originalmente concebida en China y reinventado por Cardano hacia 1550, es la famosa rejilla de Cardano. Las letras del mensaje secreto no forman una estructura regular, son más bien un patrón aleatorio. El mensaje se lee colocando una máscara sobre el texto. La máscara es un ejemplo temprano de un secreto (*stego*) clave que tenía que ser compartido entre las partes que se comunican. El acróstico también fue utilizado en la Primera Guerra Mundial por los alemanes y los aliados.

Durante la Segunda Guerra Mundial se usaron los microfilmes, en los puntos de las *i* o en signos de puntuación para enviar mensajes. Los prisioneros usan *i, j, t* y *f* para ocultar mensajes en código morse. Pero uno de los sistemas más ingeniosos se conoce con el nombre de *Null Cipher* [8]. Este último consiste en enviar un mensaje, de lo más común posible, y elegir cierta parte de él para ocultar el mensaje.

Aunque existen registros de muchas aplicaciones esteganográficas a lo largo de la historia, el verdadero auge para la esteganografía y las marcas de agua surgió a partir de la popularidad de la Internet [29, 9].

2.3.2 *Sistema esteganográfico o estegosistema*

Un sistema esteganográfico o estegosistema consiste en una función de incrustación (*embedding function*) y su contraparte una función de extracción *extraction function*), como se muestra en la Figura 2.2. La función de incrustación toma como entradas, el medio digital anfitrión (*cover work*), una clave esteganográfica (*stego key*) y un mensaje a ocultar (*message*). La salida del proceso, es decir el objeto con el mensaje incrustado se conoce como el objeto alterado (*stego work*), comúnmente se hace referencia al objeto alterado como el canal de comunicación, mismo que está en constante monitoreo por el adversario en busca de objetos esteganográficos. La función que permite recuperar el mensaje se conoce como función de ex-

tracción (*extraction function*) la cual a partir de un objeto alterado y una clave esteganográfica permite obtener el mensaje oculto.

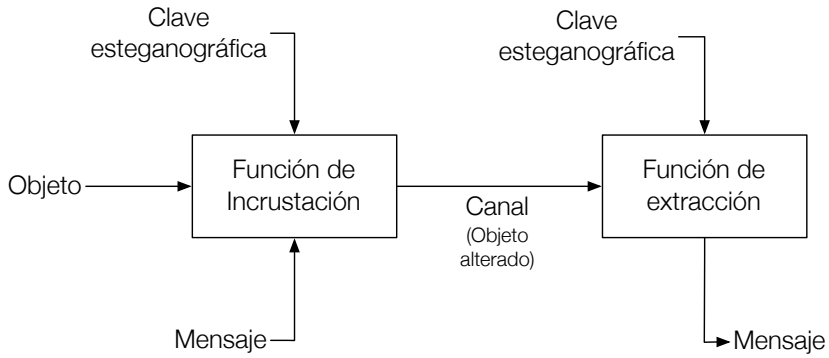

Figura 2.2: Esquema de un sistema esteganográfico

Un estegosistema tiene las siguientes propiedades:

- *Capacidad de incrustación*. Es el número máximo de bits que se pueden ocultar en un objeto anfitrión seleccionado
- *Eficiencia de la incrustación*. Que se define como el ńumero de bits posibles a ocultar de forma segura por unidad de distorsión.

2.4 MARCAS DE AGUA DIGITALES

La ténica de marcas de agua digitales es la otra para ocultar información. En la sección 2.3 se expresó que la Esteganografía tiene como propósito impedir que un adversario conozca la existencia de un mensaje secreto, en otras palabras la información debe ser *indetectable*, en contraste la técnica de marcas de agua digitales debe estar diseñada para evitar la *extracción* de la marca, de hecho una marca de agua puede ser visible para cualquiera pero ningún adversario debe ser capaz de removerla.

Iniciamos este capítulo con los antecedentes de las marcas de agua correspondientes al tema 2.4.1, después en el tema 2.4.2 se describen los elementos y funcionamiento de un sistema para marcado y detección de objetos digitales.

2.4.1 *Antecedentes*

Aunque el arte de la fabricación de papel se inventó en China hace más de mil años, las marcas de agua en papel aparecieron hasta 1282, en Italia [17]. Las marcas fueron hechas mediante la adición de patrones finos de alambre para los moldes de papel. El documento sería un poco más delgado que el cable y por lo tanto más transparente. El significado y propósito de las primeras marcas de agua son inciertos. Pudieron ser utilizados para funciones prácticas como la identificación de moldes en las hojas de papel, o como marcas para identificar al fabricante.

Por otra parte, pudieron haber representado señales místicas, o simplemente podrían haber servido como decoración.

Con el tiempo, estas técnicas se extendieron por toda Europa y los patrones adquirieron un propósito pragmático; para el siglo XVIII, se utilizaban para registrar marcas, la fecha de fabricación, el tamaño de la hoja original, etc. Durante este siglo, otros dos eventos importantes ocurrieron en el desarrollo de las marcas de agua. El término *Wassermarke* fue acuñado por primera vez en Alemania y las marcas de agua comenzaron a ser utilizadas como una medida contra la falsificación de documentos, especialmente en el papel moneda.

El primer uso moderno de una técnica de marca de agua para proteger los derechos de autor fue desarrollado por Emil Hembrooke en 1954. Se considera "moderno" porque las marcas de agua anteriores, por lo general consistían en la alteración de los objetos físicos. Por el contrario, Hembrooke introdujo un mensaje en una señal analógica. Para ello, se introduce un mensaje codificado en clave Morse, en la banda de 1 kHz de un disco de música. Esto proporcionó una nota de copyright imperceptible, usada por la corporación Muzak hasta alrededor de 1984.

La primer marca de agua digital fue descrita en 1979 por Szepanski [34], incluso Holt [14] describió una aplicación para identificar mensajes en una señal de audio en 1988. Sin embargo, el crédito para el primer uso del término *marca de agua digital* es otorgado a Komatsu y Tominaga también en 1988 [9]. Desde aquellos años, la investigación científica en estos rubros ha estado creciendo; muy despacio al principio y vigorosamente después. Por ejemplo, el primer *Information Hiding Workshop* que incluyó las marcas de agua digitales dentro de los temas principales, se celebró en 1996.

2.4.2 *Sistema de marcas de agua digitales*

Un sistema de marcas de agua digitales consiste en un marcador (*embedder*) y un detector (*detector*), como se muestra en la Figura 2.3. El marcador recibe como entradas, el medio digital original (*cover work*), el cual puede ser texto, imágenes, audio o video, y la marca de agua (*watermark*) cifrada con la clave (*watermark key*) seleccionada. La salida del proceso es el medio digital marcado (*work*). El detector recibe como entrada un objeto tentativamente marcado, y trata de identificar si contiene una marca de agua, una vez recuperada la marca de agua, se utiliza la misma clave para decifrar y obtener la información oculta. El rol de la clave es asegurar que la información de la marca de agua sea obtenida solamente por usuarios autorizados.

De acuerdo a [25], un sistema de marca de agua digital tiene tres características básicas.

- *Imperceptibilidad*. Es una condición esencial para la marca de agua digital, es decir, la marca de agua incrustada debe ser imperceptible y la calidad visual del objeto marcado debe ser similar a la del objeto original.
- *Robustez*. Significa que la marca de agua no puede ser destruida a menos que el objeto sea alterado por completo. Bajo la condición de imperceptibilidad, la implementación de un sistema de marcas de agua digitales que soporté manipulaciones del objeto tanto como sea posible es siempre una tarea complicada. Por lo tanto, la robustez es una característica importante para evaluar el rendimiento de un esquema de marca de agua.

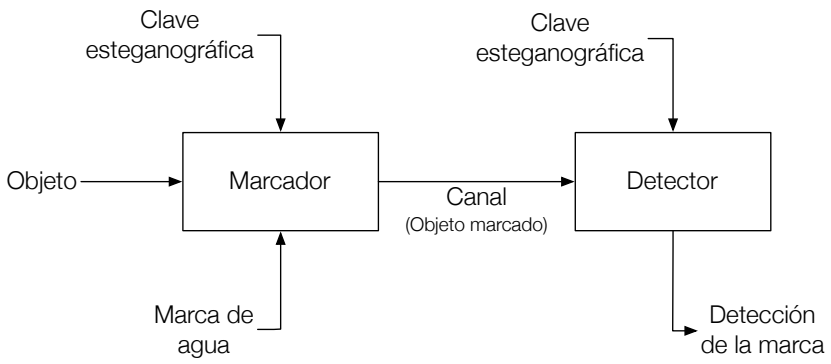

Figura 2.3: Esquema de un sistema de marcas de agua digitales

- *Capacidad*. La capacidad es también una característica importante para evaluar el rendimiento de un sistema de marcas de agua. Bajo la condición de imperceptibilidad, así como los requisitos de robustez, mientras más información se desea incrustar con la marca de agua, más difícil la implementación del esquema. En varios artículos [1, 13, 39], los autores muestran crear sistemas cada vez más robustos, de acuerdo a sus resultados experimentales. Pero, en sus experimentos, la imagen incorporada por lo general tiene sólo 1-bit de información, diseños conocidos como de 0-bit de información, ya que sólo se puede decir "si" o "no" a la marca de agua sin más información adicional. Es importante mencionar, éstos métodos propuestos perderán su robustez dramáticamente si se le agrega más información a la marca de agua.

Como se mencionó anteriormente las características de imperceptibilidad, robustez y capacidad se limitan entre sí. En otras palabras, para cualquier esquema de marcas de agua, es imposible cubrir las tres características simultáneamente. Otras características importantes de los sistemas de marcas de agua dependen de su aplicación.

- Cuando se desea proteger los derechos de autor, se espera que la marca de agua sea inseparable del objeto. Para evitar que personas no autorizadas inserten su propia marca.
- En la distribución de objetos por Internet es necesario que la marca de agua pueda ser removida, o bien se permita insertar otras marcas, para identificar los nuevos distribuidores del objeto.

3

ALGORITMOS ESTEGANOGRÁFICOS

En el capítulo 2 se mencionaron algunos métodos que posiblemente fueron utilizados en la antigüedad para ocultar mensajes en objetos de uso común, la mayoría de los cuales dejaron de ser utilizados, en su lugar nuevas técnicas se han diseñado especialmente para los archivos digitales.

El aumento generalizado en el uso de las computadoras y las telecomunicaciones, ha sugerido utilizar los archivos digitales como medio anfitrión de mensajes, por su fácil creación, modificación y transmisión.

En las siguientes secciones se describen los algoritmos más utilizados para enmascarar información, clasificados en dos categorías: en la sección 3.1 se describen los algoritmos para imágenes digitales y en la sección 3.2 los métodos que utilizan documentos de texto, como objeto anfitrión.

3.1 ALGORITMOS Y ESTEGOSISTEMAS PARA IMÁGENES

Las imágenes digitales tienen algunas características que les permiten ser el medio anfitrión más utilizado, cuando se trata de ocultamiento de la información, en donde el propósito es incrustar información de forma casi arbitraria sin afectar visualmente la imágen. Dos son los procesos requeridos para incrustar datos en las imágenes, un mecanismo de incrustación y uno de extracción o detección [24].

En los siguientes apartados se presentan y describen los principales métodos, se muestra el algoritmo y algunas aplicaciones que lo implementan, las cuales se pueden localizar fácilmente en Internet, todas de código abierto y bajo la licencia GNU/GPL, listadas en la Tabla 3.1.

NOMBRE	ALGORITMO	LENGUAJE	ACTUALIZACIÓN
OpenStego	LSB, RandomLSB	Java	20/09/2012
DIIT	LSB	Java	10/09/2007
Hide and Reveal	LSB	Java	19/05/2010
VSL	LSB, KLT, F5	Java	07/02/2011
F5 Steganography	F5	Java	01/05/2011

Tabla 3.1: Tabla comparativa de los estegosistemas para imagen.

3.1.1 *Bit Menos Significativo*

El algoritmo Bit Menos Significativo (siglas del inglés *Least Significant Bit*) (LSB) es un método escalar para marcar, comúnmente imágenes, donde cada pixel es representado como un valor de 8-bits. En su forma básica, para cada pixel de la imagen se modifica un solo bit de información, el menos significativo.

Por ejemplo, para insertar un bit cero a un pixel con valor en escala de grises 51, primero se escribe el valor como un número binario $(51)_2$ = 00110011, en este caso, la representación binaria se modifica para 00110010 = $(50)_2$. Obviamente, si tuviéramos que insertar un 1, no se necesita realizar cambios. Como resultado de la incrustación, el valor 51 cambió por 50. Es fácil ver que los valores 50 y 51 forman un par, de tal manera que durante la incrustación uno de estos valores puede cambiar en el otro, pero nunca a ningún otro valor. Así, el conjunto de todos los posibles valores que pueden tomar los pixeles en una imagen, se pueden dividir en pares disjuntos de valores $(2i, 2i+1); i = 0, \ldots, 127$, que se pueden cambiar uno con otro durante la incrustación.

El proceso de incrustación modifica a lo más 1 bit por cada pixel en la imagen, de tal forma el impacto visual es mínimo. Por otra parte, si la información que se está incrustado es independiente de los valores de los bits menos significativos del pixel, en promedio, sólo el 50 % de los píxeles serán alterados [9].

Algoritmo 3.1 Incrustación por el método LSB

ENTRADA: Imagen anfitriona $C \in \{0,1\}^n$, marca $m \in \{0,1\}^k$
SALIDA: Imagen marcada $S \in \{0,1\}^n$

```
1:  if k ≤ n/8 then
2:      S ← C
3:      j ← 8
4:      for i = 1 to k do
5:          if S_j ≠ m_j then
6:              S_j ←~ S_j
7:          end if
8:          j ← j + 8
9:      end for
10: end if
```

Digital Invisible Ink Toolkit

El algoritmo de marcado LSB es uno de los más utilizados en los estegosistemas, por las ventajas que ofrece y la gran variedad en que puede ser implementado para mejorar la seguridad. Herramientas como *Digital Invisible Ink Toolkit* (DIIT) decidieron aprovechar esta cualidad. Escrito en `Java`, DIIT un proyecto creado en el Departamento de Computación de la Universidad de Waikato, que al igual que otros proyectos está alojado en los servidores de `sourceforge`, utiliza el método simple LSB para marcar imágenes de 24 bits, con la característica que permite seleccionar el orden en que los pixeles de la imagen anfitriona son alterados, a estas variaciones en la incrustación el autor de DIIT las nombró como sigue:

- *BlindHide* O esconder "a ciegas", el proceso inicia en la esquina superior izquierda de la imagen y continua línea a línea hacia abajo, modificando pixel a pixel para que coincidan con el mensaje. Esto no es muy seguro, porque es fácil de leer los bits menos significativos de una imagen. Tampoco inteligente, si el mensaje no llena completamente el espacio posible, entonces solo la parte superior de la imagen se degrada pero el fondo se mantiene sin cambios, por lo que es fácil identificar los cambios.
- *HideSeek* En esta forma el mensaje trata de distribuirse aleatoriamente en el espacio de la imagen. Se utiliza una contraseña que sirve para generar un número pseudo-aleatorio, el cual indica la posición inicial del mensaje, se continúa generando números pseudo-aleatorios para descubrir la siguiente posición de bits en la imagen. Este proceso brinda mayor seguridad ya que para recuperar el mensaje es necesario probar todas las combinaciones posibles. Sin embargo, es un proceso que puede realizarse en un tiempo razonable.
- *FilterFirst* Esta técnica cambia la imagen usando los filtros incorporados para modificar los bits más significativos y poder usar los menos significativos para ocultar el mensaje "a ciegas", de esta forma es más dificil distinguir los pixeles modificados.
- *BattleSteg* Para mejorar la forma en que se puede incrustar un mensaje utilizando el algoritmo LSB, se requiere filtrar la imagen modificando los bits más significativos y seleccionar de forma aleatoria las posiciones en donde disfraza el mensaje, una contraseña se requiere para incrustar y recuperar el mensaje, como se puede apreciar es una combinación entre *HideSeek* y *FilterFirst*

Este proyecto no reporta actualizaciones desde septiembre del 2007; sin embargo, fue utilizado para incorporar otro algoritmo, que pretende ser una mejora al método convencional LSB, seleccionando los pixeles a modificar y posteriormente cambiar los bits menos significativos por los del mensaje, el proyecto que tiene esta mejora lleva por nombre Steganography Studio.

Hide and Reveal

Hide and Reveal es el proyecto que obtuvo el tercer lugar en el primer concurso *"Etoiles du libre"* (algo así como "Las estrellas libres") organizado por la Asociación de Estudiantes de la Universidad de Tecnología de Belfort-Montbéliard en 2009. Escrito en `Java` y con ambiente gráfico, implementa el método LSB variando el número de bits por pixel a modificar y de forma similar a DIIT permite escoger el orden en que serán seleccionados los pixeles de la imagen anfitrión.

Es posible seleccionar dos esquemas de modulación y algunas variaciones en estos. La negación del secreto, es decir, negar todos los bytes de la marca, negar un byte de cada dos, realizar la negación del byte dependiendo de la condición del byte anterior. O bien, realizar la operación or-exclusivo de los bytes del secreto con los bytes de una contraseña proporcionada por el usuario.

Las variaciones en que se disfraza el mensaje utilizando el algoritmo LSB son, LSB doble en donde cada byte de la marca se oculta en dos pixeles, seis bits son ocultos en el primer pixel y los dos bits restantes en el siguiente, el tamaño mínimo que puede tener la imagen anfitrión se

puede expresar como $T_{min} = 8 \times (t + 4)$; y LSB triple en donde cada byte de la marca se oculta en un solo pixel, el tamaño mínimo de la imagen para esta variación es $T_{min} = 4 \times (t + 4)$.

3.1.2 *Bit Menos Significativo Aleatorio*

El método LSB 3.1.1, tiene como prioridad marcar una imagen causando el menor impacto visual posible. De hecho, la imagen marcada en la mayoría de los casos es indistinguible de la original a simple vista; sin embargo, si comparamos los pixeles de ambas imágenes utilizando un editor binario, observaremos que los primeros pixeles de la imagen marcada son diferentes en su bit menos significativo, esto se debe a que la marca es incrustada en un orden secuencial iniciando por el primer pixel de la imagen anfitriona. Una forma en que se puede solucionar este problema es utilizando el método aleatorio.

En el método de incrustación aleatorio de LSB, a diferencia del método simple, el conjunto de datos de la marca se distribuye entre los datos de la imagen de manera aparentemente aleatoria utilizando una clave, previamente compartida entre el remitente y el receptor. Con esta clave se generan números pseudoaleatorios, los números generados definen el lugar y el orden en que se colocan los bits de la marca en la imagen anfitriona [16]. Ahora los bits de la marca están dispersos en la imagen al azar y al realizar una comparación pixel a pixel entre ambas imágenes da la impresión de ser ruido estadístico [16].

Algoritmo 3.2 Incrustación por el método Aleatorio LSB

ENTRADA: Imagen anfitriona $C \in \{0,1\}^n$, marca $m \in \{0,1\}^k$
SALIDA: Imagen marcada $S \in \{0,1\}^n$

1: $k \xleftarrow{\$} \{0,1\}^m$
2: $n \leftarrow k_1$
3: **for** $i = 1$ to k **do**
4: $\quad S_n \leftarrow \mathrm{LSB}(C_n) = m_i$
5: $\quad n \leftarrow n + k_i$
6: **end for**

OpenStego

Proyecto de código abierto, publicado en `sourceforge`, recientemente actualizado y de gran popularidad entre los cibernautas, tiene por nombre OpenStego. Desarrollado en el lenguaje de programación `Java`, que implementa el algoritmo básico de esteganografía para imágenes, LSB y RandomLSB (versión pseudo-aleatoria de LSB), como su nombre lo indica utiliza los bits menos significativos de la imagen para incrustar la marca. El modo de funcionamiento de OpenStego es: se solicita al usuario una imagen en la cual se ocultará un archivo binario (marca), puede o no ingresar una contraseña, en donde el picadillo de la contraseña generado por MD5 será la clave para el algoritmo de cifrado DES para utilizarlo sobre la marca antes de incorporarlo a la imagen. En caso de no ingresar una contraseña la marca se incrusta sin modificación alguna.

El proceso de detección recibe como entrada la imagen tentativamente marcada y recupera, dado el caso, la marca. Si se utilizó una contraseña para cifrar la marca, se pide al usuario se ingrese la contraseña para poder decifrarlo. Si bien además de ocultar la imagen provee la capacidad para cifrar el archivo a ocultar, se podría utilizar otros algoritmos tanto para realizar la función de picadillo (*hashing*) como el algoritmo de cifrado y decifrado, que brinden mayor seguridad.

3.1.3 *Transformación de Karhunen-Loève*

La transformación de Karhunen-Loève (KLT) es un operador que dado un conjunto de puntos en el espacio $\mathbb{R}^n$, de dimensión n, lo proyecta en un subespacio, de dimensión m, con $m \leq n$, de manera que "se conserven en lo posible las posiciones relativas entre sus puntos", es decir, se minimice el error de mínimos cuadrados.

Para presentar la KLT veamos algunas nociones geométricas involucradas en ella.

Descomposición espectral

Sea A una matriz real de orden $n \times n$. Ella determina una aplicación lineal $\mathbb{R}^n \to \mathbb{R}^n$. Un número real $\lambda \in \mathbb{R}$ es un *valor propio*, o *eigenvalor*, de A si existe un vector no-nulo $\mathbf{z} \in \mathbb{R}^n - \{\mathbf{o}\}$ tal que $A\mathbf{z} = \lambda\mathbf{z}$, es decir, si existe una dirección en $\mathbb{R}^n$ sobre la cual la acción de A es una mera elongación de razón λ. En tal caso, el vector $\mathbf{z}$ se dice ser un *vector propio*, o *eigenvector*, correspondiente al valor propio λ. La pareja $(\lambda, \mathbf{z})$ se dice ser *propia*.

Se tiene que toda matriz de orden $n \times n$ tiene a lo sumo n valores propios. Además, si la matriz A fuese simétrica, se ha de tener que valen dos condiciones importantes:

- A tiene exactamente n valores propios reales $\lambda_0, \ldots, \lambda_{n-1}$, por tanto se les puede suponer ordenados de manera no-creciente, $[i < j \Rightarrow \lambda_i \geq \lambda_j]$, y
- los correspondientes eigenvectors son ortogonales a pares: $\mathbf{z}_i \cdot \mathbf{z}_j = \delta_{ij}$: la delta de Kronecker ($\delta_{ij} = 1$ si $i = j$ pero $\delta_{ij} = 0$ si $i \neq j$).

En este caso, la colección $\big((\lambda_j, \mathbf{z}_j)\big)_{j=0}^{n-1}$ es un *sistema propio*, o *eigensistema*, de la matriz A.

Si se elige a las primeras k parejas en el eigensistema, se está eligiendo a las direcciones donde "A actúa más": son las direcciones con mayor variación bajo la acción de A.

Proyecciones ortogonales

Sea $H \subset \mathbb{R}^n$ un subespacio lineal, digamos de dimensión k. Si $\{\mathbf{h}_0, \ldots, \mathbf{h}_{k-1}\}$ es una base de H entonces todo vector $\mathbf{h} \in H$ se escribe de manera única como una combinación lineal de los elementos en la base, es decir, existen $a_0, \ldots, a_{k-1}$ tales que $\mathbf{h} = \sum_{j=0}^{k-1} a_j \mathbf{h}_j$. El espacio H es pues la imagen de la aplicación lineal $H : \mathbb{R}^k \to \mathbb{R}^n$, $\mathbf{a} \mapsto \sum_{j=0}^{k-1} a_j \mathbf{h}_j$, la cual a su vez se representa por la matriz $H = [\mathbf{h}_0 \cdots \mathbf{h}_{k-1}]$, de orden $n \times k$, que tiene como columnas a los elementos en la base. Obsérvese que hemos utilizado el mismo símbolo, H, para denotar al subespacio, a la aplicación lineal y a la matriz, pero tal confusión voluntaria se debe a lo cercano de esos conceptos.

La *proyección ortogonal* sobre el espacio H es la aplicación $\pi_H : \mathbb{R}^n \to H$ que a cada vector $\mathbf{x} \in \mathbb{R}^n$ le asocia el vector $\mathbf{y} = \pi_H(\mathbf{x}) \in H$ tal que el segmento $\overline{\mathbf{xy}}$ es perpendicular al subespacio H. Puede verse que la proyección ortogonal está representada por la matriz $H(H^TH)^{-1}H^T$, es decir:

$$\forall \mathbf{x} \in \mathbb{R}^n : \ \pi_H(\mathbf{x}) = H(H^TH)^{-1}H^T\mathbf{x} \tag{3.1}$$

En efecto, al ser el espacio H de dimensión k en $\mathbb{R}^n$, la matriz $H \in \mathbb{R}^{n\times k}$ posee rango k por tanto la matriz, simétrica, $H^TH \in \mathbb{R}^{k\times k}$ es invertible. Con esto en mente, de la relación 3.1 se verifica directamente que $0 - \pi_H(\mathbf{x})$ y $\pi_H(\mathbf{x})$ son ortogonales. es decir $\langle \mathbf{x} - \pi_{\mathbf{H}}(\mathbf{x}) | \pi_{\mathbf{H}}(\mathbf{x}) \rangle = \mathbf{0}$, y presuponiendo la acción de H más a la izquierda:

$$\text{Cada } \mathbf{x} \in \mathbb{R}^n \text{ puede representarse por el vector } \zeta(\mathbf{x}) = (H^TH)^{-1}H^T\mathbf{x} \in \mathbb{R}^k. \tag{3.2}$$

Así, mediante la proyección ortogonal en H, a un vector se le puede representar por uno con un menor número de dimensiones.

Karhunen-Loève

Sea $X = \{\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}\} \subset \mathbb{R}^n$ un conjunto de m puntos en $\mathbb{R}^n$. El promedio de los vectores es $\mathbf{x}_M = \frac{1}{m}\sum_{j=0}^{m-1}\mathbf{x}_j$, y acaso remplazando a X por su traslación $X - \mathbf{x}_M$, se puede suponer que el promedio es el vector nulo $\mathbf{0} \in \mathbb{R}^n$.

A cada vector $\mathbf{x}_j$ se le ve como un vector columna y al conjunto X como una matriz X de orden $n\times m$. La matriz producto $C = X\cdot X^T$ es de orden $n\times n$ y es simétrica. Se dice ser la *matriz de covariancia* o de dispersión [5], del arreglo X y puede verse como la transformación lineal $\mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{x} \mapsto C\mathbf{x}$. De acuerdo con la descomposición espectral localizada en la página 13, al calcular un eigensistema de C, se distingue las direcciones con "mayores acciones" de C, lo que corresponde a aquellas que "dan una mejor perspectiva", en dimensiones menores, del conglomerado de puntos X. La noción de "mejor perspectiva" es pues de visualizarlo desde las direcciones principales, las distinguidas por un eigensistema de la matriz de covariancia.

Dado $X = \{\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}\} \subset \mathbb{R}^n$ de promedio nulo, y un parámetro $k \in \{1, \ldots, n\}$, su KLT se calcula como sigue:

A. sea $C := X \cdot X^T$ la matriz de covariancia ;

B. calcúlese un eigensistema $\big((\lambda_j, \mathbf{z}_j)\big)_{j=0}^{n-1}$ de C ;

C. sea $H = [\mathbf{z}_0 \cdots \mathbf{z}_{k-1}]$ la matriz de orden $n \times k$ cuyas columnas son los primeros k eigenvectores ;

D. para cada $j = 0, \ldots, m-1$ sea $\mathbf{y}_j := \zeta(\mathbf{x}_j)$ calculada como en (3.2) ;

E. dése como resultado $Y = \{\mathbf{y}_0, \ldots, \mathbf{y}_{m-1}\} \subset \mathbb{R}^k$.

En este caso, el arreglo $Y = KLT(X)$ es el resultado de aplicar la KLT al arreglo X.

Supóngase que se tiene imágenes con 2^n colores ($n = 8$ para 256 colores) y que se quiere utilizar menos colores, digamos 2^k, aún a costa de perder "resolución". Un *pixel* es pues un vector $\mathbf{x}$ de dimensión n, y una imagen es una colección X de m pixeles. Se tiene que $Y = KLT(X)$ es una representación de la misma imagen con 2^k colores. Como requiere de menos espacio, es una compresión de la imagen original.

Virtual Steganographic Laboratory for Digital Images

Dentro de la categoría de estegosistemas para imágenes presentados en este documento *Virtual Steganographic Laboratory for Digital Images* (VSL) es la herramienta escrita en Java que implementa más algoritmos esteganográficos, el simple método LSB descrito en secciones anteriores, la KLT y el algoritmo F5; estos dos últimos utilizan la transformación DCT para imágenes con formato JPEG. Además de los métodos de marcado provee la implementación del algoritmo RS para análisis y un grupo de funciones para modificaciones geométricas.

Todas las características anteriores presentadas en una interfaz gráfica de usuario, en donde se puede seleccionar mediante la acción presionar y arrastrar el proceso que sufrirá la imagen anfitrión y la marca, en ambos procesos de incrustación y detección. Incluso es posible realizar dos procesos no dependientes al mismo tiempo utilizando hilos de procesamiento, lo que genera cómputo paralelo [12].

3.1.4 *F5*

Muchos sistemas esteganográficos son débiles contra ataques visuales y estadísticos; y aquellos sin estas deficiencias ofrecen una capacidad muy pequeña. El algoritmo F5 por el contrario, resiste los ataques visuales y estadísticos, incluso ofrece una capacidad grande. F5 implementa una codificación de matriz, para mejorar significativamente el proceso de incrustación. Reduce el número de cambios necesarios y emplea un procedimiento a horcajadas (*permutative straddling*) para repartir uniformemente el mensaje dentro de la imagen.

Permutación a horcajadas

El mecanismo a horcajadas que utiliza el algoritmo F5, revuelve todos los coeficientes obtenidos de la transformada de coseno discreta, utilizando una permutación la cual depende de una clave derivada de una contraseña. Enseguida incrusta los datos en la secuencia permutada. La contracción no cambia el número de coeficientes, solamente modifica sus valores. El algoritmo proporciona los coeficientes resultantes al codificador de Huffman. Con la llave correcta, el receptor es capaz de invertir la permutación. El proceso de permutación tiene complejidad lineal $O(n)$ [35].

Codificación por matriz

Supongamos que tenemos un mensaje secreto distribuido uniformemente y valores uniformemente distribuidos en las posiciones a cambiar. Una mitad del mensaje producirá cambios, la otra mitad no. Sin la codificación por matriz, tenemos una eficiencia de incrustación de 2 bits por cada cambio.

Por ejemplo, si incrustamos un mensaje corto que contiene sólo 217 bytes (1736 bits), utilizando la codificación por matriz, solamente se realizan 459 cambios, con una eficiencia de incorporación de 3,8 bits por cambio.

El siguiente ejemplo ilustra como funciona la codificación por matriz. Se desea incrustar dos bits x_1, x_2 en una posición con tres bits modificables a_1, a_2, a_3, cambiando a lo más un lugar. Los cuatro casos son:

$$x_1 = a_1 \oplus a_3, x_2 = a_2 \oplus a_3 \Rightarrow \texttt{sin cambio}$$
$$x_1 \neq a_1 \oplus a_3, x_2 = a_2 \oplus a_3 \Rightarrow \texttt{cambia } a_1$$
$$x_1 = a_1 \oplus a_3, x_2 \neq a_2 \oplus a_3 \Rightarrow \texttt{cambia } a_2$$
$$x_1 \neq a_1 \oplus a_3, x_2 \neq a_2 \oplus a_3 \Rightarrow \texttt{cambia } a_3$$

De los cuatro casos solo cambia un bit. En general, tenemos una palabra a codificar a con n bits modificables para los x bits del mensaje secreto k. Sea f la función de picadillo que extrae k bits de la palabra a codificar. La codificación por matriz permite encontrar una palabra a codificar a' tal que, para cualquier a y x con $x = f(a')$, la distancia de Hamming es

$$d(a, a') \leq d_{max} \tag{3.3}$$

Denotamos esta ecuación por una tripleta ordenada (d_{max}, n, k): un código con n lugares que cambiará en no más de d_{max} lugares para incrustar k bits. F5 implementa la codificación por matriz solo para cuando $d_{max} = 1$. Para $(1, n, k)$, los códigos tiene una longitud definida por $n = 2^k - 1$. Obtenemos una densidad de cambio

$$D(k) = \frac{1}{n+1} = \frac{1}{2^k} \tag{3.4}$$

y una tasa de incrustación

$$R(k) = \frac{k}{n} = \frac{1}{n}\mathrm{ld}(n+1) = \frac{k}{2^k - 1} \tag{3.5}$$

Usando la densidad de cambio y la tasa de incrustación, podemos definir la eficiencia de incrustación $W(k)$, indica el número promedio de bits que se pueden insertar por el cambio

$$W(k) = \frac{R(k)}{D(k)} = \frac{2^k}{2^k - 1}k \tag{3.6}$$

La eficiencia de incrustación para el código $(1, n, k)$ es siempre mayor a k

F5 Steganography

El último estegosistema dentro de la categoría de imágenes lleva por título el nombre del algoritmo que implementa F5 (descrito en la página 15), escrito en `Java` basado en línea de comando, permite incrustar cualquier tipo de archivos binarios en imágenes JPEG. Permite ingresar una contraseña utilizada para incrustar y recuperar la marca. Dado el diseño del algoritmo F5 permite comprimir la imagen resultante sin perder datos de la marca y mantener la calidad de la imagen original lo suficiente para pasar la prueba visual.

Algoritmo 3.3 Incrustación por el método F5

ENTRADA: Imagen C, Mensaje m, Contraseña q
SALIDA: Imagen con mensaje incrustado S

1: Compresión JPEG. Detener después de la cuantización.
2: Utilizando la contraseña inicializar un generador de números aleatorios
3: Crear instancia de permutación con el generador y el número de coeficientes de la compresión como parámetros.
4: Determinar el parámetro k de la capacidad de la imagen y la longitud del mensaje secreto
5: Calcular la longitud de la palabra código n = 2k -1
6: Insertar el mensaje secreto usando la codificación de matriz y los parámetros (1, N, K).
7: Continuar compresión JPEG [35].

NOMBRE	ALGORITMO	LENGUAJE	ACTUALIZACIÓN
Snow	Espacios blanco	Java	13/02/2006
Texto	Lingüístico	C	28/02/1995
Stego!	Lingüístico	JavaScript	20/12/2005

Tabla 3.2: Tabla comparativa de los estegosistemas para documentos de texto.

3.2 ALGORITMOS Y ESTEGOSISTEMAS PARA DOCUMENTOS

Cada copia de un documento de texto puede cambiarse de forma casi invisible mediante el reposicionamiento o al modificar la apariencia de los elementos de texto, como las líneas, las palabras o los caracteres. Una copia única puede ser registrada con su autor o destinatario, así las posteriores copias no autorizadas que se recuperen crearán un rastro al dueño original [4]. Algunos algoritmos se han diseñado con el propósito de proteger la propiedad intelectual en los documentos de texto digitales.

En esta sección se presentan algunos métodos utilizados para ocultar información en archivos de texto, en la Tabla 3.2 se listan algunas aplicaciones analizadas para conocer las técnicas de implementación.

3.2.1 *Modificación de los elementos del texto*

Los elementos del texto son todos los caracteres permitidos en un documento, como las letras de un lenguaje, el caractér especial de salto de línea, el de retorno de carro, el de fin de archivo, incluso el espacio entre palabras, entre líneas o párrafos. Todos estos elementos son válidos para un documento y son interpretados de la misma forma en todas las aplicaciones, así es posible utilizarlos para ocultar información.

Brassil et al. [4], describen varias formas de modificar o cambiar de posición algunos elementos, con el propósito de incrustar una marca y con esta identificar si el documento ha sufrido cambios no autorizados, una técnica para además de ocultar información en un documento brindar integridad de algunos datos. Las técnicas se describen a continuación:

- *Desplazamiento de línea*, mover ligeramente una línea de texto dentro de un parrafo puede ser utilizado para interpretar una marca oculta, imperceptible a la vista pero fácilmente detectable por un algoritmo para identificar si la marca existe.
- *Desplazamiento de palabra*, incrustar una marca por desplazamiento horizontal de la ubicación de una palabra dentro de una línea de texto, el desplazamiento puede ser tan insignificante como para pasar desapercibido y al mismo tiempo el necesario para poder recuperarlo.
- *Codificación de caractér* es una clase de técnica que incrusta una marca alterando una característica en particular de un caractér.

Existen varias definiciones de esquemas utilizando las técnicas mostradas arriba, incluso los mismos autores en [4] proponen el uso del desplazamiento de línea en archivos de tipo `PostScript`

3.2.2 *Espacios blancos*

Esta técnica de espacios blancos, se refiere al uso de los caracteres dentro de un archivo de texto que representan visualmente un "espacio en blanco", como el retorno de línea, fin de linea, fin de archivo, tabuladores, y espacios en blanco normales, para ocultar información, no se tiene un esquema exacto de como utilizar los espacios, por ese motivo se describe una aplicación de código abierto que la implementa.

Snow

El primer estegosistema seleccionado dentro de la categoría de texto es Snow, el nombre de esta herramienta significa la naturaleza esteganográfica del espacio en blanco (en inglés *s*teganographic *n*ature *o*f *w*hitespace). Escrito en `Java` diseñado para trabajar desde la línea de comando y con soporte para web en forma de *applet*, permite ocultar secretos (marcas) en archivos de texto ASCII al concatenar espacios en blanco (*whitespaces*) al final de la línea. Como los espacios sencillos y los tabuladores generalmente no se muestran en editores de texto, entonces la marca permanece oculta a simple vista. Si la marca se cifra antes de ocultarse, no puede ser entendida aún si es detectada, motivo por el que Snow utiliza el algoritmo ICE para cifrar la marca.

Otra característica en el proceso de marcado y detección, es un método de compresión. De forma opcional se puede comprimir la marca utilizando el esquema básico del codificador de Huffman para reducir entre un 25 y 40 % su longitud y con esto permitir ocultar más información.

El proceso de incrustación de Snow, realiza lo siguiente, se inserta un tabulador al final de la primer línea de texto con espacio suficiente, en el proceso contrario, de detección, se busca este tabulador para saber si el documento contiene una marca. Los datos se escriben en conjuntos de 3 bits, en codificación octal. En caso de que la longitud de la marca no sea múltiplo de 3 se rellena con ceros. En la detección estos bits son ignorados. Regularmente los 3 bits son codificados en 8 columnas de texto, como una línea tiene por defecto 80 caracteres de longitud, es posible almacenar 30 bits en los espacios en blanco de una línea de texto. El

tabulador no se agrega al final de la línea a menos que los últimos 3 bits almacenados sean ceros, esto se hace para identificar que el grupo de ceros pertenece a la marca.

Finalmente si la marca no puede ser almacenada en los espacios del texto anfitrión, se agregan líneas en blanco al final del documento para ocultar el texto restante.

3.2.3 *Lingüísticos*

El método lingüístico incluye técnicas que hacen uso del lenguaje utilizado dentro del documento de texto para modificar palabras por sinónimos, o tratar de formar frases en el lenguaje, en donde las palabras desordenadas contiene algun caractér de importancia, con el cual se puede recupear el mensaje, la principal técnica conocida son los acrósticos.

Dos herramientas que implementan esta técnica se describen en los siguientes apartados.

Texto

Texto es un estegosistema de tipo lingüistico escrito en lenguaje `C`, transforma valores ASCII pseudoaleatorios como `pgp` o `uuencode` a sentencias escritas en inglés. El resultado aparenta ser una carta mal redactada. Texto funciona igual que un cifrador de sustitución simple, cada uno de los 64 símbolos ASCII usados por `pgp` o `uuencode` se sustituye por una palabra del inglés. Cuenta con un diccionario de sustantivos, verbos, adjetivos y adverbios, que son utilizados para rellenar estructuras de frases predefinidas, de esta forma las palabras usadas para conectar, como artículos, así como signos de puntuación son ignorados en el proceso de detección. Para obtener el código original Texto utiliza el diccionario para saber la letra que fue reemplazada por la palabra en el proceso de codificación.

Stego!

Mostrando el potencial que tiene `JavaScript` como lenguaje interpretado sobre un navegador web, surge Stego! un estegosistema lingüistico de reciente creación, convierte una secuencia de caracteres pseudoaleatorios, producto de un texto cifrado, en un texto en inglés sin sentido. El texto resultante consta de palabras elegidas de un diccionario de 2^{16} palabras, cada codificación de 2 bytes del mensaje representa la posición de la palabra en el diccionario que será utilizada en el texto final. De forma aleatoria se agregan signos de puntuación al texto resultante para hacerlo ver más plausible.

En el proceso de detección y de recuperación del mensaje, los espacios en blanco y signos de puntuación son ignorados, mediante la tabla que relaciona la palabra del diccionario con la codificación de 2 bytes en formato `Base64` se obtiene el texto cifrado original.

3.2.4 *Ordenamiento de datos*

Existen diferentes formatos que requieren de un lenguaje para representar de forma gráfica su contenido, como las páginas de Internet o los documentos de `Word`. Cuando no existe una regla para el orden de las sentencias en el lenguaje, es posible dar una importancia al orden y utilizarlo para ocultar datos.

Zhu et al. [40] proponen un método de autenticación de documentos, el cual codifica datos ocultos en la capa de diseño de los documentos con formato, mediante la modificación de las secuencias de visualización de las palabras o caracteres, sin cambiar el contenido o el aspecto del documento. La codificación se consigue mediante una permutación específica de la secuancia de visualización y la permutación específica está fuertemente ligada a la gira de Hamilton **T**, en el Problema del viaje Exacto del Agente Viajero (siglas del inglés *Exact Travel of Salesman Problem*) (XTSP).

Básicamente se utilizan documentos de lenguaje PostScript basado en tercias (carácter, letra, posición) para aplicar su esquema. En la Figura 3.1 se muestra un ejemplo de un documento en formato PostScript tomado del método propuesto por Zhu et al. [40].

```
/Times-Roman findfont      %Font
12 scalefont               %Font
setfont                    %Font
newpath
100 200 moveto             %Positioning
(This is a sentence.) show %Characters
showpage
```

Figura 3.1: Ejemplo de un archivo PostScript.

Las primeras tres líneas definen el tipo de letra como Times-Roman de 12 puntos, la línea 5 indica la posición donde se trazara la primer letra de la sentencia en este caso la letra "T" y a partir de ahí se pinta en la horizontal lo que sigue. De acuerdo al método, se permutan posiciones de la secuencia, si primero re-escribimos el archivo como se muestra en la Figura 3.2.

```
/Times-Roman findfont      %Font
12 scalefont               %Font
setfont                    %Font
newpath
100 200 moveto             %Positioning
(This) show                %Characters
124 200 moveto             %Positioning
(is) show                  %Characters
135 200 moveto             %Positioning
(a) show                   %Characters
144 200 moveto             %Positioning
(sentence) show            %Characters
185 200 moveto             %Positioning
(.) show                   %Characters
showpage
```

Figura 3.2: Ejemplo de un archivo PostScript con comandos de posiciones explícitas.

Y con esto poder permutar el contenido del documento, como se ilustra en la Figura 3.3.

En resultado al procesar el archivo de la Figura 3.2 y 3.3 será el mismo, porque aunque en el lenguaje de descripción las sentencias se modificaron, en el visor la posición de los caracteres sigue siendo el mismo.

```
/Times-Roman findfont       %Font
12 scalefont                %Font
setfont                     %Font
newpath
124 200 moveto              %Positioning
(is) show                   %Characters
144 200 moveto              %Positioning
(sentence) show             %Characters
100 200 moveto              %Positioning
(This) show                 %Characters
185 200 moveto              %Positioning
(.) show                    %Characters
135 200 moveto              %Positioning
(a) show                    %Characters
showpage
```

Figura 3.3: Ejemplo de un archivo PostScript permutado aleatoriamente.

También Artz [2], explica una forma de hacer esteganografía con permutaciones. Cuando los datos no tienen un orden estricto dentro de un formato, éste puede ser utilizado para ocultar información. A cada permutación de un conjunto de objetos se le puede asignar un número entero positivo. Esta asignación se puede entonces utilizar para codificar los datos ocultos mediante la alteración del orden de los objetos, que no se consideran ordenados por el medio portador, un ejemplo se muestra en la Figura 3.4.

```
Perl                                E-mail
$d = $a + $b + $c;                  to: mruiz@cs.cinvestav.mx,
$d = $a + $c + $b;                      miruizt@ipn.mx
$d = $b + $a + $c;                  to: miruizt@ipn.mx,
. . .                                   mruiz@cs.cinvestav.mx
```

Figura 3.4: Ejemplos de esteganografía usando permutaciones.

Si representamos las variables del lenguaje `Perl` como ceros o unos, al cambio de posición tenemos diferentes secuencias de bits que se utiliza para representar información oculta.

4

FORMATOS ESTÁNDARES PARA DOCUMENTOS

Durante varios años, cuando se utilizaba una aplicación, los archivos se almacenaban en un formato que se podía interpretar completa y adecuadamente sólo por la aplicación que los generaba. `Pages` creaba los archivos `Pages`, `Microsoft Word` generaba los archivos `Word`, y así sucesivamente.

Eventualmente los programadores incluían módulos para dar soporte a los formatos de sus competidores, en ocasiones podía ser un éxito, al menos hasta la siguiente actualización del formato por parte de los propietarios; lo mismo para presentaciones, hojas de cálculo y otros. El intercambio de archivos entre diferentes aplicaciones podría ser todo un reto para los usuarios, tal así que copiar el contenido y adecuar el formato era el procedimiento realizado.

Los formatos estandarizados como el Formato de Documento Abierto para Aplicaciones Ofimáticas de OASIS (en inglés, *OASIS Open Document Format for Office Applications*) (ODF) y el Formato de Documento Portátil (siglas del inglés *Portable Document Format*) (PDF) llegan a solucionar el problema de almacenar e intercambiar archivos entre aplicaciones y sistemas. Un archivo puede ser creado en `Windows` utilizando `Microsoft Word` y guardado con alguno de los formatos mencionados, al compartirlo a otro equipo con `Linux`, por ejemplo, cualquier aplicación que soporte estos formatos va a desplegar de la misma forma su contenido.

El presente capítulo se divide en dos secciones, cada una recapitula las especificaciones técnicas del formato, empezando por los documentos ODF.

4.1 EL FORMATO DE DOCUMENTO ABIERTO PARA APLICACIONES OFIMÁTICAS

El actual estándar de la Organización Internacional de Normalización (siglas del inglés *International Organization for Standardization*) (ISO) y la Comisión Internacional de Electrotécnica (siglas del inglés *International Electrotechnical Commission*) (IEC), fue inicialmente presentado por Sun Microsystems en la aplicación `OpenOffice`, posteriormente y hasta la fecha es mantenido por el Comité Técnico (del inglés *Technical Committee*) (TC) de OASIS; está basado en Lenguaje de Marcado eXtensible (siglas del inglés *eXtensible Markup Language*) (XML) para almacenar e intercambiar documentos entre aplicaciones *ofimáticas*, incluyendo procesadores de texto, hojas de cálculo y presentaciones, es neutral en cuanto a la aplicación, plataforma y sistema operativo.

Para utilizar los documentos ODF fue necesario revisar cuidadosamente el estándar publicado por OASIS [33], en las siguientes secciones se presenta una recopilación de los puntos importantes utilizados en el presente trabajo. En la sección 4.1.1 se describe como un documento puede estar constituido como un paquete o como archivo único. En la sección 4.1.2 se encuentran las convenciones del lenguaje, las extenciones de los diferentes tipos de archivos que pueden crearse bajo el formato se pueden localizar en la sección 4.1.3. Por último, la sección 4.1.4 contiene los detalles específicos para los documentos de texto del formato.

4.1.1 *Estructura del archivo*

Un archivo ODF es almacenado en formato Archivo Java (siglas del inglés *Java Archive*) (JAR) [11], es decir un empaquetado zip que contiene un archivo adicional, utilizado para listar el resto de los archivos dentro del comprimido. Como ya se mencionó el formato ODF es válido si se crea como paquete o como archivo único; sin embargo, la reciente explicación sobre un archivo ODF contradice que pueda ser únicamente un archivo, puesto que todo documento ODF es en realidad un empaquetado.

Entonces la diferencia es la estructura del JAR. Un archivo ODF debe cumplir los siguientes requerimientos [33]:

A) Como paquete
   a) debe contener al menos uno de los archivos: `content.xml` o `styles.xml`. También puede incluir algunos archivos adicionales como `settings.xml` y `meta.xml`
   b) los archivos XML deben respetar las normas descritas en el estándar XML 1.0
   c) los elementos raíz para los archivos XML deben ser
      i. `<office:document-content>` para documentos de texto o `<math:math>` en caso de fórmulas matemáticas, ambos para el archivo `content.xml`
      ii. `<office:document-styles>` para `styles.xml`
      iii. `<office:document-meta>` para `meta.xml`
      iv. `<office:document-settings>` para el archivo `settings.xml`

B) Como archivo único
   a) el archivo XML debe respetar las normas descritas por el estándar de XML versión 1.0
   b) el elemento raíz del archivo XML debe ser `<office:document>`

Para ambos casos los archivos XML deben tener etiquetas bien formadas de acuerdo a la especificación del estándar XML 1.0. Los elementos que se pueden insertar en un documento de texto, tales como imágenes, u otros objetos MIME, también se almacenan dentro del comprimido JAR y comúnmente se organizan en carpetas dependiendo del tipo de objeto. Las mismas aplicaciones *ofimáticas* que utilizan el formato ODF pueden organizar e incluir documentos adicionales como mejor les convenga, siempre y cuando se respeten los requerimientos listados arriba. En seguida se muestra el contenido de un archivo ODF generado con la herramienta `LibreOffice`.

```
Archive:  Documento.odt
 Length   Method    Size  Cmpr Name
--------  ------ ------- ---- ----
      39  Stored      39   0% mimetype
   11950  Stored   11950   0% Thumbnails/thumbnail.png
      45  Defl:N      37  18% layout-cache
  807121  Defl:N  796945   1% Pictures/100000000967E8.png
```

```
   10403  Defl:N     2178  79% content.xml
   11784  Defl:N     2165  82% styles.xml
    9187  Defl:N     1424  85% settings.xml
    1053  Stored     1053   0% meta.xml
     899  Defl:N      261  71% manifest.rdf
       0  Defl:N        2   0% Configurations2/accel/current.xml
       0  Stored        0   0% Configurations2/toolpanel/
       0  Stored        0   0% Configurations2/statusbar/
       0  Stored        0   0% Configurations2/progressbar/
       0  Stored        0   0% Configurations2/toolbar/
       0  Stored        0   0% Configurations2/images/Bitmaps/
       0  Stored        0   0% Configurations2/popupmenu/
       0  Stored        0   0% Configurations2/floater/
       0  Stored        0   0% Configurations2/menubar/
    1299  Defl:N      333  74% META-INF/manifest.xml
--------          -------  --- -------
  853780           816387   4% 19 files
```

El formato OpenDocument ofrece una clara separación entre el contenido, la disposición de éste en el documento y los metadatos. Los componentes más notables del formato son los siguientes:

A) Como paquete

a) `manifest.rdf` o la tabla de contenidos del formato ODF, enumera todos los archivos que tiene algún tipo de MIME asociado (por ejemplo, `text/html` o `image/png`);

```
<rdf:Description rdf:about="styles.xml">
  <rdf:type rdf:resource="http://docs. \
       oasis-open.org/ns/office/1.2/meta \
       /odf#StylesFile"/>
</rdf:Description>
<rdf:Description rdf:about="">
  <ns0:hasPart xmlns:ns0="http://docs. \
   oasis-open.org/ns/office/1.2/meta/pkg#"
  rdf:resource="styles.xml"/>
</rdf:Description>
```

b) `meta.xml` describe los metadatos del documento, incluyendo los campos bibliográficos definidos por el conjunto de elementos *Dublin Core*, como autor, título, tema y lenguaje;

```
<meta:initial-creator>
  Michel Ruiz Tejeida
</meta:initial-creator>
<meta:creation-date>
```

```
    2013-01-15T18:59:34
  </meta:creation-date>
  <dc:date>
    2013-01-15T19:00:29
  </dc:date>
```

c) `styles.xml`, representa las definiciones del estilo del documento, como los atributos de texto que se debe utilizar para los encabezados, para las palabras en negritas, etcétera;

```
  <text:outline-style style:name="Outline">
  <text:outline-level-style text:level="1" \
   style:num-format="">
  <style:list-level-properties
text:list-level-position-and-space-mode= \
"label-alignment">
  <style:list-level-label-alignment \
   -followed-by="listtab"
text:list-tab-stop-position="0.3in"
fo:text-indent="-0.3in"
fo:margin-left="0.3in"/>
  </style:list-level-properties>
  </text:outline-level-style>
```

d) `settings.xml`, incluye propiedades relacionadas con el aspecto del visor del documento al momento de abrir

```
  <config:config-item config:name=" \
   ViewAreaHeight" config:type="long">
    24626
  </config:config-item>
  <config:config-item config:name=" \
   ShowRedlineChanges" config:type="boolean">
    true
  </config:config-item>
  <config:config-item config:name= \
   "InBrowseMode" config:type="boolean">
    false
  </config:config-item>
```

e) `content.xml`, almacena el contenido estructurado del documento, incluyendo párrafos, listas, tablas, cuadros e imágenes.

```
  <text:p text:style-name="P1">
  <text:span text:style-name="T1">
    Aquí se encuentra la información
```

```
  dentro de una etiqueta de párrafo.
</text:span>
</text:p>
```

B) Como archivo único

a) `content.xml` es el único archivo XML del formato, describe los metadatos del documento, los estilos automáticos y definidos por el usuario, así como el contenido estructurado del documento, en otras palabras, el contenido de los archivos describos anteriormente en un solo archivo.

```
<?xml version="1.0" encoding="utf-8"?>
<office:document office:version="1.2"
  xmlns:office="urn:oasis:names:tc: \
   opendocument:xmlns:office:1.0"
  xmlns:text="urn:oasis:names:tc: \
   opendocument:xmlns:text:1.0"
  xmlns:meta="urn:oasis:names:tc: \
   opendocument:xmlns:meta:1.0"
  xmlns:style="urn:oasis:names:tc: \
   opendocument:xmlns:style:1.0"
  xmlns:fo="urn:oasis:names:tc: \
   opendocument:xmlns:xsl-fo-compatible:1.0"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <office:meta>
     <meta:generator>Creado por \
      Michel Ruiz Tejeida</meta:generator>
     <dc:date>2013-01-25T16:08:26</dc:date>
     <meta:creation-date>2013-01-23T15:49:32
     </meta:creation-date>
     <meta:document-statistic
      meta:word-count="161"
      meta:image-count="0"
      meta:object-count="0"
      meta:page-count="1"
      meta:character-count="1040"
      meta:paragraph-count="2"
      meta:table-count="0"/>
     <dc:language>en-US</dc:language>
  </office:meta>
  <office:styles>
    <style:default-style>
      <style:paragraph-properties
      style:writing-mode="page"
       style:text-autospace="ideograph-alpha"
```

```
    style:tab-stop-distance="0.4925in"
    style:punctuation-wrap="hanging"
    fo:hyphenation-ladder-count="no-limit"
    style:line-break="strict"/>
   <style:text-properties
    style:use-window-font-color="true"
    fo:country="US" fo:language="en"
    fo:hyphenation-push-char-count="2"
    style:font-name="Times New Roman"
    fo:font-size="12pt"
    fo:hyphenation-remain-char-count="2"
    fo:hyphenate="false"/>
 </style:default-style>
 <style:style style:class="text"
        style:family="paragraph"
style:name="Text_20_body"
style:display-name="Text body"
style:parent-style-name="Standard"
style:master-page-name="">
   <style:text-properties fo:font-size="13pt"/>
   <style:paragraph-properties
    fo:margin-top="0in"
    fo:background-color="transparent"
    fo:margin-bottom="0.0835in"
    style:shadow="none"/>
 </style:style>
 <style:style style:name="Heading_20_2"
   style:next-style-name="Heading_20_1"
   style:display-name="Heading 2"
   style:default-outline-livel="2
   style:class="text"
   style:family="paragraph"
   style:parent-style-name="Heading">
   <style:text-properties
    fo:font-weight="bold" />
 </style:style>
 <style:style style:name="Heading_20_1"
    style:next-style-name="Text_20_body"
    style:default-outline-level="1"
    style:class="text"
    style:family="paragraph"
    style:parent-style-name="Heading"
    style:display-name="Heading 1">
   <style:text-properties
```

```
          fo:font-weight="normal"
            fo:font-size="20pt"/>
      </style:style>
    </office:styles>
    <office:automatic-styles>
      <style:style style:name="P1"
         style:parent-style-name="Text_20_body"
         style:family="paragraph">
        <style:paragraph-properties
       fo:margin-left="0in"
       fo:margin-right="0.9272in"
           fo:text-indent="0in"
           style:auto-text-indent="false"/>
      </style:style>
      <style:style style:name="T1"
       style:family="text">
        <style:text-properties
        fo:font-style="italic"/>
      </style:style>
      <style:style style:name="T2"
      style:family="text">
        <style:text-properties
        fo:font-weight="bold"/>
      </style:style>
    </office:automatic-styles>
    <office:body>
      <office:text>
        <text:h text:outline-level="2"
         text:style-name="Heading_20_1">
        Capítulo Primero
        </text:h>
        <text:h text:outline-level="1"
         text:style-name="Heading_20_2">
        Que trata de la condición y
        ejercicio del famoso hidalgo
        D. Quijote de la Mancha
        </text:h>
        <text:p text:style-name="P1">
    En un lugar de la Mancha, de cuyo nombre
    no quiero acordarme...
        </text:p>
      </office:text>
    </office:body>
  </office:document>
```

TIPO	EXTENSIÓN	MIME
Texto	odt	text
Hoja de cálculo	ods	spreadsheet
Presentación	odp	presentation
Dibujo	odg	graphics
Gráfica	odc	chart
Fórmula	odf	formula
Base de datos	odb	database
Imagen	odi	image
Maestro	odm	text-master

Tabla 4.1: Extensiones empleadas para los tipos de documentos del formato ODF.

4.1.2 *Convenciones comunes del XML*

El grupo de trabajo que definió el formato ODF se basó en el lenguaje de esquema Lenguaje Regular para la Siguiente Generación de XML (del inglés *REgular LAnguage for XML Next Generation*) (RELAX NG) [28]. Por lo cual utiliza espacios de nombres XML para colocar elementos en grupos funcionales. El formato recomienda usar prefijos específicos para cada espacio de nombres, aunque, por supuesto, las aplicaciones son libres de usar cualquier prefijo alternativo. Un espacio de nombres recomendado es:

```
urn:oasis:names:tc:opendocument:xmlns:text:1.2
```

el cual utiliza el prefijo recomendado `text`, para los elementos que contienen construcciones de texto. Como este otros prefijos recomendados son: `style`, `table`, `config`, etcétera.

El uso de estos prefijos se recomienda para su reutilización en diferentes escenarios, en una hoja de cálculo, por decir algo, se puede utilizar el espacio de nombres con el prefijo `text` para el contenido de una celda, este sería el mismo en un documento de texto, para indicar el contenido de una tabla, o en una presentación para las etiquetas.

4.1.3 *Documentos y sus extensiones*

Para cada uno de los diferentes tipos de archivos que se pueden generar con el formato ODF, se ha registrado una extensión única, la cual se identifica dentro del JAR por el tipo MIME como sigue, `odt` para documentos de texto, `ods` para hojas de cálculo, `odp` para presentaciones, `odg` para gráficos y `odb` para bases de datos, la lista completa se muestra en la Tabla 4.1.

4.1.4 *El documento de texto*

Todo documento de texto del formato *OpenDocument* debe cubrir todos los requerimientos para ser un archivo ODF, además de los siguientes requerimientos adicionales:

A. El elemento `<office:document>` debe tener el atributo `office:mimetype` con algunos de estos valores:

- `application/vnd.oasis.opendocument.text`
- `application/vnd.oasis.opendocument.text-template`
- `application/vnd.oasis.opendocument.text-master`

B. Si el documento de texto es conformado como un paquete ODF debe contener un archivo `mimetype` y dentro alguna de las siguientes cadenas de texto

- `application/vnd.oasis.opendocument.text`
- `application/vnd.oasis.opendocument.text-template`
- `application/vnd.oasis.opendocument.text-master`

C. El elemento `<office:document>` debe tener el elemento hijo `<office:text>`.

En la sección 4.1.1 se muestra el contenido completo de un archivo de texto, presentado como archivo único dentro del formato ODF, como se puede ver el primer elemento después del cuerpo del documento (`<office:body>`), es el encabezado, definido por la etiqueta `text:h`. Este contenedor de texto señala una parte de la estructura que debe ser notable para la aplicación. La aplicación debe coleccionar todos los elementos `text:h` para crear de forma automática una tabla de contenidos.

El atributo `text:outline-level` determina el nivel de profundidad en el esquema de la estructura, `1` especifica el nivel más álto y es el valor que se asigna de forma predeterminada. Los atributos se especifican en el mismo espacio de nombres que el elemento al que pertenece. Como este, el formato ODF utiliza mucho atributos a los que se les conoce como *atributos universales*.

Otro elemento muy importante para cualquier documento de texto es el definido por el elemento `text:p`, en referencia a un párrafo. Este elemento al igual que muchos otros utilizan el estilo semántico del lenguaje HTML, para ilustrar, se puede especificar un estilo a un párrafo en particular si se escribe dentro del elemento `text:span`. Otro aspecto que está relacionado con el lenguaje HTML son los espacios en blanco, si se desea ingresar más espacios en blanco después de un punto es necesario utilizar la representación en código del carácter blanco `text:s`. Elementos similares al espacio en blanco son el tabulador regido por el elemento `text:tab` y el salto del línea por `text:line-break`. El salto de página también tiene un elemento específico `text:soft-page-break`.

4.2 EL FORMATO DE DOCUMENTO PORTÁTIL

El popular formato PDF es uno de archivo usado para representar documentos de manera independiente a la aplicación de software, hardware y de los sistemas operativos [20], creado y mantenido por Adobe. Un *documento PDF* consiste en una colección de objetos que juntos describen la apariencia de una o más páginas, posiblemente acompañado de otros elementos interactivos. Un *archivo PDF* contiene los objetos que lo crean junto a la información estructurada que despliega, todo se representa como una secuencia de bytes.

Las páginas de un documento, y otros elementos visuales, pueden contener cualquier combinación de texto, gráficos e imágenes. La apariencia de una página se describe a través de un *flujo de contenido*, el cual contiene una secuencia de *objetos* gráficos que serán pintados en la página.

Además, para describir el aspecto estático de las páginas, un documento PDF puede contener elementos interactivos que son posibles sólo en una representación electrónica. Se admiten anotaciones de muchos tipos de cosas, tales como texto, enlaces de hipertexto, de marcado, archivos adjuntos, sonidos y películas. Un documento puede incluso definir su propia interfaz de usuario; acciones del teclado y del ratón, las cuales pueden ser vinculadas con acciones descritas en objetos.

Incluso, en un documento PDF puede haber información de alto nivel que es útil para el intercambio de contenidos entre aplicaciones. Es decir, puede incluir la identificación y la información de la estructura lógica que le permite realizar búsquedas, editar, o extraer información para su reutilización en otros lugares. Las contraseñas y certificados son parte de esta información de alto nivel que especifica el formato.

Los detalles del formato se presentan en tres secciones, en la sección 4.2.1 se describen las propiedades, en la sección4.2.2 se localiza los detalles de la sintaxis y al final, en la sección 4.2.3 esta descrita la estructura del archivo.

4.2.1 *Propiedades*

El corazón del formato PDF es la habilidad de describir la apariencia de gráficos sofisticados y tipografía. Esta habilidad se logra a través del modelo de imágenes de Adobe. Aunque los programas de aplicación teóricamente pueden describir cualquier página como una matriz de pixeles, el archivo resultante sería voluminoso, dependiente del dispositivo, y poco práctico para mostrar gráficos de alta resolución. En cambio, un modelo de imagen de alto nivel permite que las aplicaciones, sean las que describan la apariencia de las páginas con texto, formas gráficas e imágenes, todos en términos de elementos gráficos abstractos en lugar de utilizar directamente los pixeles del dispositivo. Esta manera de realizar la descripción de las páginas es económica e independiente del dispositivo, y se puede utilizar para reproducir objetos de alta calidad en una amplia gama de impresoras, pantallas y otros dispositivos de salida.

4.2.1.1 *Lenguajes de descripción de página*

Como una de sus funciones, un PDF sirve como un lenguaje de descripción de páginas, un lenguaje para describir el aspecto gráfico de las páginas con respecto a un modelo de formación de imágenes. Un programa genera la salida mediante estos dos pasos:

A. La aplicación genera una descripción del dispositivo de salida deseada en el lenguaje de descripción de páginas.

B. Un programa controlando un dispositivo específico de salida, interpreta la descripción y lo crea para ese dispositivo.

Los pasos recien descritos se deben ejecutar en lugares y tiempos distintos. El lenguaje de descripción de páginas funge como un estándar para la transmisión y almacenamiento de documentos entre dispositivos independientes.

4.2.1.2 *Portabilidad*

Los archivos PDF se forman como secuencias de bytes (grupos de 8-bits). Es diseñado para ser portable hacia cualquier plataforma y sistema operativo. La representación binaria se pretende sea generada, transportada y consumida directamente; sin traducción entre el conjunto nativo de caracteres, representación del fin de archivo, y otras convenciones usadas en las diferentes plataformas.

Cualquier archivo PDF puede ser creado en una forma que solo utilice caracteres ASCII, es decir grupos de 7-bits. Sin importar el tipo de representación que se utilice para los caracters, los archivos PDF se transportan y almacenan como archivos binarios, no como archivos de texto.

4.2.1.3 *Acceso aleatorio*

Un archivo PDF puede ser visto como una estructura de datos que consiste en una colección de objetos que pueden referenciarse entre sí, de forma arbitraria. El orden de ocurrencia de los objetos en el formato no está relacionada con la semántica que lo rige. En general, cualquier aplicación debe procesar el PDF siguiendo las referencias de un objeto a otro, y no procesando los objetos de manera secuencial. Este acceso aleatorio en los objetos es posible gracias a una tabla de referencia que cada archivo contiene, y que se encuentra al final del documento, de esta forma las aplicaciones pueden identificarla fácilmente. El potencial de la tabla radica en la facilidad de poder encontrar un objeto en documentos que llegan a tener cientos de páginas.

4.2.1.4 *Seguridad*

Los PDF tienen dos atributos de seguridad que se pueden utilizar, de forma separada o juntos, en cualquier documento:

- El documento se puede *cifrar*, así solo usuarios autorizados pueden accesar a este. La autorización de acceso al documento crea dos roles, uno para el autor y otro para el resto de los usuarios, el segundo tiene permisos sobre operaciones del archivo como visualizar, mandar a imprimir, o editar contenido.
- El archivo puede estar *firmado* digitalmente para certificar su autenticidad. La firma digital puede tomar diversos valores, como el digesto o resumen de un documento que ha sido cifrado con una clave privada o pública, una firma biométrica como una huella dactilar, entre otros. Cualquier cambio posterior a la firma la invalida.

4.2.2 *Sintaxis*

La sintaxis de un archivo PDF se entiende mejor si se divide en cuatro partes.

- *Objetos*. Un documento PDF es una estructura de datos compuesta a partir de un pequeño conjunto de tipos básicos de objetos de datos.
- *Estructura del archivo*. La estructura de los archivos PDF determina cómo los objetos se almacenan en un archivo PDF, cómo se accede a ellos, y cómo se actualizan. Esta estructura es independiente de la semántica de los objetos.

- *Estructura del documento*. La estructura del documento PDF especifica los tipos de objetos que se han utilizado para representar los componentes de un documento, es decir, páginas, estilos de letra, anotaciones, etcétera.
- *Flujo de contenido*. Una secuencia de contenido PDF incluye una secuencia de instrucciones que describen la apariencia de una página o entidad gráfica. Estas instrucciones, a la vez que representan los objetos, son conceptualmente distintos de los objetos que represetan la estructura del documento y se describen por separado.

4.2.2.1 *Convenciones léxicas*

En el nivel más fundamental, un archivo PDF es una secuencia de bytes de 8 bits. Estos bytes se agrupan en *tokens* de acuerdo con las reglas de sintaxis que se describen a continuación. Uno o más símbolos se ensamblan para formar entidades sintácticas de mayor nivel, principalmente objetos, que son los datos básicos a partir de los cuales se construye un documento PDF.

Un PDF puede ser representado completamente usando valores de bytes correspondientes al subconjunto del juego de caracteres imprimibles de la tabla ASCII, además de caracteres vacío: como el espacio en blanco, retorno de carro, tabuladores y saltos de línea. ASCII es el código estándar americano para intercambio de información, una convención ampliamente utilizada para codificar un conjunto específico de 128 caracteres como números binarios. Sin embargo, un archivo PDF no se limita al conjunto de caracteres ASCII, sino que puede contener valores arbitrarios de 8 bits, sujetos a las siguientes consideraciones:

- Los símbolos que delimitan los objetos y que describen la estructura de un archivo PDF se escriben en el conjunto de caracteres ASCII, como lo son todas las palabras reservadas y los nombres usados como claves en diccionarios estándar.
- Pueden ser los valores de ciertos tipos de objetos, como cadenas de texto, los cuales no tienen por que ser escritos totalmente en ASCII. Considerando el resultado al momento de su visualización gráfica. Los datos que son naturalmente binarios, como las imágenes, se representa directamente in binario por eficiencia.
- Un archivo PDF que contiene los datos binarios debe ser transportado y almacenado por medios que preserven todos los bytes del archivo fielmente, es decir, como un archivo binario en lugar de un archivo de texto. Este tipo de archivo no es portable a ambientes que imponen reservados códigos de caracteres, la longitud máxima de la línea, las convenciones de fin de línea, u otras restricciones.

4.2.2.2 *Juego de caracteres*

El juego de caracteres en los documentos PDF se divide en tres clases, regulares, delimitadores, y los caracteres de espacio en blanco. Esta clasificación determina la agrupación de los caracteres en *tokens*, excepto dentro de las cadenas y comentarios; donde se aplican reglas diferentes en esos contextos.

Los caracteres de espacio (blancos), como se muestran en la Tabla 4.2 separan construcciones sintácticas, como nombres y números. Todos los caracteres de espacio en blanco son equivalentes, excepto en los comentarios y cadenas. En los demás casos, el PDF trata cualquier secuencia de espacios en blanco consecutivos como un carácter.

Los caracteres retorno de carro CR y el salto de línea LF, se tratan como una marca de fin de línea EOL. La combinación de retorno de carro CR seguido inmediatamente de un salto de línea LF se combinan para ser interpretados como una marca de fín de línea EOL.

DECIMAL	HEXADECIMAL	OCTAL	NOMBRE
0	00	000	Nulo (NUL)
9	09	011	Tabulador (HT)
10	0A	012	Salto de línea (LF)
12	0C	014	Salto de página (FF)
13	0D	015	Retorno de carro (CR)
32	20	040	Espacio (SP)

Tabla 4.2: Caracteres de espacio en blanco

4.2.3 *Estructura del archivo*

Un archivo PDF inicialmente se compone de cuatro elementos:

- Un *encabezado* de una sola línea que identifica la versión del formato PDF utilizado en el documento.
- Un *cuerpo* que contiene los objetos que forman el documento contenido en el archivo.
- Una *tabla de referencias cruzadas*, la cual contiene información acerca de los objetos indirectos en el archivo.
- Un *tráiler* indicando la posición de la tabla de referencias cruzada y de ciertos objetos especiales ubicados en el cuerpo del archivo.

Los *tokens* en un archivo PDF se acomodan en lineas, una línea se identifica al encontrar un carácter EOL, que puede ser un CR, LF, o ambos. Archivos que contengan información binaria tienen lineas de longitud arbitraria; sin embargo, para mejorar la compatibilidad entre aplicaciones se recomienda no se exceda de 255 caracteres por línea.

4.2.3.1 *Encabezado*

La primer línea de un PDF es un *encabezado* que identifica la versión del formato utilizado para la especificación del documento. Esta línea es como se muestra a continuación:

```
%PDF-1.7
```

En donde los últimos caracteres indican la versión del formato.

4.2.3.2 *Cuerpo*

El *cuerpo* de un archivo PDF consiste en una secuencia de objetos, los cuales representan el contenido del documento, como la tipografía, las páginas e imágenes. En versiones iguales o posteriores a la 1.5, el cuerpo puede contener flujos de objetos. Cada flujo de objetos contiene una secuencia de objetos indirectos.

4.2.3.3 *Tabla de referencias cruzadas*

En la penúltima sección de la estructura estándar de un archivo, se localiza la *tabla de referencias cruzadas*, la cual contiene información que permite el acceso aleatorio de los objetos descritos en el cuerpo, de esta forma no es necesario leer todo el documento para identificar un objeto en particular. La tabla esta diseñada para tener una entrada por línea, especificando la posición del objeto dentro del documento. La tabla de referencias cruzadas, es la única parte de todo el documento que tiene un formato fijo, que permite entradas de la tabla para tener acceso aleatorio. Inicialmente la tabla tiene una sola sección y conforme se forme el documento se agregan las demás entradas a la tabla.

Cada sección en la tabla inicia con una línea que contiene la palabra reservada `xref`, después de esta línea se pueden agregar una o más subsecciones, sin importar el orden en que se registren. Cada subsección de referencias cruzadas contiene entradas para un rango contiguo de objetos. Cada subsección inicia con una línea que contiene dos números separados por un espacio: el número del primer objeto de la subsección y la cantidad de entradas en la subsección. Por ejemplo, la línea

```
28 5
```

introduce un apartado que contiene cinco objetos, enumerados consecutivamente del 28 al 32. Seguida de esta línea, las consecutivas son referencias cruzadas. Cada entrada en la tabla tiene una longitud exacta de 20 bytes, incluyendo los caracteres de fin de línea. La primer entrada en la tabla debe indicar el objeto raíz, que posteriormente será señalado en el trailer, las siguientes entradas son una de dos tipos de referencias cruzadas: para los objetos que están en uso o para indicar los que han sido eliminados, se dice que quedan libres. Ambas entradas usan el mismo formato, y se pueden distinguir por la palabra clave `n` (para las que están en uso) y `f` (para las entradas libres). El formato de una entrada es:

```
nnnnnnnnnn ggggg n eol
```

donde

- `nnnnnnnnnn` es un elemento de 10 dígitos, indica la posición en bytes del objeto al que hace referencia
- `ggggg` es un número de 5 dígitos
- `n` es la literal que indica el estado del objeto
- `eol` es una secuencia de 2 caracteres para el fin de línea.

Un ejemplo se muestra en la Figura 4.1

4.2.3.4 *Tráiler*

El *tráiler* de un PDF permite que la aplicación de lectura encuentre fácilmente la tabla de referencias cruzadas. Regularmente una aplicación lee un documento PDF desde el final del archivo. La última línea del archivo contiene solo una marca de fin de archivo EOF. Antes de la marca de fin de archivo se encuentra el diccionario *tráiler*, de la siguiente forma:

```
xref
0 23
0000000000 65535 f
0000064425 00000 n
0000000019 00000 n
0000001668 00000 n
0000001689 00000 n
0000001866 00000 n
0000001906 00000 n
0000024415 00000 n
0000064568 00000 n
0000024437 00000 n
0000046596 00000 n
0000046619 00000 n
0000046819 00000 n
0000047307 00000 n
0000047648 00000 n
0000063346 00000 n
0000063369 00000 n
0000063584 00000 n
0000063985 00000 n
0000064257 00000 n
0000064300 00000 n
0000064667 00000 n
0000064764 00000 n
```

Figura 4.1: Ejemplo de una tabla de referencias cruzadas del formato PDF

```
trailer
    << clave[1] valor[1]
       clave[2] valor[2]
       ...
       clave[n] valor[n]
    >>
startxref
%%EOF
```

Las claves permitidas en el diccionario tráiler se muestran en la Tabla 4.3.

Clave	Tipo	Valor
Size	entero	(Obligatorio, no debe ser una referencia indirecta) El total de entradas en la tabla de referencias cruzadas, definido por la combinación de la sección original y todas las secciones de actualización. De manera equivalente, este valor es mayor en 1 que la numeración de los objetos utilizados en el archivo
Prev	entero	(Sólo si el archivo tiene más de una referencia cruzada, no debe ser una referencia indirecta). El desplazamiento de bytes desde el comienzo del archivo al principio de la anterior referencia cruzada.

Root	diccionario	(Obligatorio, debe ser una referencia indirecta) El diccionario del catálogo para el documento PDF.
Encrypt	diccionario	(Obligatorio, si el documento esta cifrado) El diccionario para el documento cifrado
Info	diccionario	(Opcional, debe ser una referencia indirecta) La información del diccionario del documento
ID	arreglo	(Opcional, pero se recomienda su uso) Un arreglo de dos cadenas de bytes que constituyen un identificador de archivo. Las dos cadenas de bytes deben ser objetos directos y deben estar sin cifrar.

Tabla 4.3: Las entradas en el diccionario *trailer*

Un ejemplo se muestra a continuación:

```
trailer
    << /Size 22
       /Root 2 0 R
       /Info 1 0 R
       /ID [ < 81b14aafa313db63dbd6f981e49f94f4 >
             < 81b14aafa313db63dbd6f981e49f94f4 >
           ]
    >>
startxref
18799
%%EOF
```

4.2.3.5 *Flujos de objeto*

A partir de la versión 1.5 del formato, se introduce un nuevo tipo de flujos, los *flujos de objeto*, que contienen una secuencia de objetos. El propósito de los flujos de objeto es permitir que un mayor número de objetos PDF sean comprimidos, reduciendo el tamaño de los archivos. Cualquier objeto puede aparecer en una secuencia de objetos, con las siguientes excepciones:

- Flujos de objeto
- Objetos con un número de generación diferente de cero
- El diccionario de cifrado de un documento
- Un objeto que represente el valor de la longitud de un flujo de objetos en el diccionario.

El creador del archivo PDF tiene la flexibilidad de determinar que objetos almacenar en un flujo de objetos. Por ejemplo, puede ser útil agrupar los objetos que compartan ciertas características, como las "fuentes en la página 1", o "comentarios del borrador 3". Estos objetos se conocen como colecciones. Es importante no sobrecargar la cantidad de objetos en cada flujo, de lo contrario sería lento el proceso de lectura.

En seguida se muestra una colección de objetos (2 fuentes y un descriptor de fuente) como se escribiría de acuerdo a la versión 1.4 del formato, junto a la tabla de referencias cruzadas:

```
  11 0 obj
     <<  /Type /Font
         /Subtype /TrueType
         ...otras entradas...
         /FontDescriptor 12 0 R
     >>
  endobj

  12 0 obj
     <<  /Type /FontDescriptor
         /Ascent 891
         ...otras entradas...
         /FontFile2 22 0 R
     >>
  endobj

  13 0 obj
     <<  /Type /Font
         /Subtype /Type0
         ...otras entradas...
         /ToUnicode 10 0 R
     >>
  endobj

  ...

  xref
  0 32
  0000000000 65535 f
  ...
  0000001434 00000 n
  0000001735 00000 n
  0000002155 00000 n
  ...
  trailer
     << /Size 32
        /Root ...
     >>
```

ahora el mismo ejemplo (al anterior) pero con el formato de la versión 1.5, en donde se introducen los flujos de objeto.

```
% El flujo de objetos
  15 0 obj
```

```
      <<  /Type /ObjStm
          /Length 1856

% El número de objetos en el flujo
          /N 3

% El desplazamiento del primer objeto (bytes)
          /First 24
      >>
  stream

% El número de objetos y los desplazamientos de
% éstos, en relación con el primero
     11 0 12 547 13 665
     <<  /Type /Font
         /Subtype /TrueType
         ...otras claves...
         /FontDescriptor 12 0 R
     >>

     <<  /Type /FontDescriptor
         /Ascent 891
         ...otras claves...
         /FontFile2 22 0 R
     >>
     <<  /Type /Font
         /Subtype /Type0
         ...otras claves...
         /ToUnicode 10 0 R
     >>
  ...
  endstream
  endobj

% El flujo de la referencia cruzada
  99 0 obj
      << /Type /XRef

% Esta sección tiene un apartado con 32 objetos
         /Index [0 32]

% Cada entrada tiene 3 campos: 1 y 2,
% con 2 bytes de ancho, cada uno
         /W [1 2 2]
```

```
% Para facilitar la lectura, en este ejemplo
         /Filter /ASCIIHexDecode
         /Size 32
         ...
      >>
  stream

% ‘‘0 65535 f’’ en la tabla de referencias cruzadas
      00 0000 FFFF
      ...

% La entrada para el objeto 11,
% la primera fuente
      02 000F 0000

% La entrada para el objeto 12,
% el descriptor de la fuente
      02 000F 0001

% La entrada para el objeto 11,
% la segunda fuente
      02 000F 0002
      ...

% La entrada para el objeto 15,
% el flujo de objetos
      01 BA5E 0000
      ...
  endstream
  endobj

  startxref

 % El desplazamiento de ‘‘99 0 obj’’
      54321
  %%EOF
```

5 DISEÑO DEL ESQUEMA ESTEGANOGRÁFICO

La primera definición informal de un esquema esteganográfico fue formulada por Simmons [31] como el problema de los presos. Dos presos, Alicia y Beto, desean planear su escape de la cárcel. Sin embargo, el guardián de la cárcel, Isabel, puede monitorear cualquier comunicación, si Isabel detecta algo inusual, los pone en régimen de aislamiento. De tal forma Alicia y Beto deben transmitir sus planes secretos sin que Isabel encuentre algo sospechoso en la comunicación.

Uno de los componentes más importantes en el diseño de un estegosistema es la selección del medio anfitrión. La creciente actividad de crear y distribuir documentos de texto con herramientas *ofimáticas* en línea, sistemas colaborativos o por editores en dispositivos móviles, entre otros tantos ejemplos, sugiere utilizar los documentos de texto como medio para enmascarar el mensaje.

En el capítulo 3, se muestra un análisis realizado a diferentes herramientas esteganográficas que utilizan archivos de texto e imágenes para ocultar información. Cada herramienta provee una característica que la hace única con el fin de mejorar la forma de ocultar la información; sin embargo las que utilizan archivos de texto, modifican la información, si un atacante se apodera del archivo marcado puede identificar que tienen información demás y sospechar.

En las siguientes secciones se presenta un esquema esteganográfico que utiliza los formatos estándares de archivos de texto PDF y ODF, para ocultar mensajes sin alterar el tamaño del archivo y la información que contiene. En la sección 5.1 se explica el objeto anfitrión del esquema y como es utilizado para ocultar la información. En la sección 5.2 se describe una forma de enumerar las permutaciones de n sin la necesitadad de computarlas. Posteriormente en la sección 5.3 se da la definición del esquema, notación y terminología utilizada en el resto del capítulo. Las siguientes tres secciones, describen los algoritmos del esquema esteganográfico, el de generación de claves 5.4, incrustación 5.5 y extracción 5.6. Finalmente la sección 5.7 presenta una descripción de la seguridad.

5.1 OBJETO ANFITRIÓN

En la definición de un sistema esteganográfico es primordial conocer a detalle el medio anfitrión que se utilizará para ocultar el mensaje, para nuestros propósitos los archivos PDF y documentos ODF son el objeto de interés. En el capítulo 4 se mostraron los detalles técnicos de cada uno, sin embargo en esta sección se explica como son utilizados para ocultar datos.

En el caso de los documentos ODF no se tienen requerimientos adicionales a los estipulados por el estándar mismo, pero para hacer uso de un archivo PDF es necesario contenga la tabla de referencias cruzadas, descrita en el apartado 4.2.3.3. Además de que en ningún caso los archivos deben estar cifrados.

Un documento de texto en el esquema, es visto como una entidad compuesta de dos partes:

A. la información que almacena el documento,

B. las instrucciones que indican la forma en que se debe mostrar la información.

A diferencia de otras herramientas y modelos presentados, nuestro esquema no altera la información almacenada en el documento, en su lugar aprovecha las características definidas en el formato para las aplicaciones que despliegan su contenido. En el archivo ODF es posible cambiar el orden de los atributos de una etiqueta, como se puede mostrar en la Figura 5.1, la parte de la izquierda muestra un fragmento del lenguaje del formato y en la derecha, si se es observador, es el mismo código pero en diferente orden. Para una aplicación como `OpenOffice` ambos códigos producen el mismo resultado en pantalla.

```
<meta:document-statistic
  meta:word-count="161"
  meta:image-count="0"
  meta:object-count="0"
  meta:page-count="1"
  meta:character-count="1040"
  meta:paragraph-count="2"
  meta:table-count="0"/>
```

```
<meta:document-statistic
  meta:page-count="1"
  meta:table-count="0"
  meta:object-count="0"
  meta:image-count="0"
  meta:character-count="1040"
  meta:word-count="161"
  meta:paragraph-count="2"/>
```

Figura 5.1: Atributos permutables del formato ODF

En el caso del formato PDF la tabla de referencias cruzadas contiene los elementos que se pueden permutar sin que se afecte la forma en que se muestra la información, un ejemplo de una tabla de referencias cruzadas se muestra en la Figura 5.2.

```
xref
0 14
0000000000 65535 f
0000064425 00000 n
0000000019 00000 n
0000001689 00000 n
0000001906 00000 n
0000024415 00000 n
0000064568 00000 n
0000024437 00000 n
0000046819 00000 n
0000047648 00000 n
0000063346 00000 n
0000063369 00000 n
0000063584 00000 n
0000064667 00000 n
```

```
xref
0 14
0000000000 65535 f
0000001689 00000 n
0000001906 00000 n
0000024415 00000 n
0000064568 00000 n
0000064667 00000 n
0000063369 00000 n
0000000019 00000 n
0000046819 00000 n
0000024437 00000 n
0000047648 00000 n
0000064425 00000 n
0000063346 00000 n
0000063584 00000 n
```

Figura 5.2: La tabla de referencias cruzadas del formato PDF

La diferencia entre los formatos es la cantidad de información que pueden ocultar, valor que está directamente relacionado con el número de elementos posibles a permutar. En el estándar de OASIS para documentos ODF, se listan todos los atributos que puede tener cada etiqueta, dependiendo de la versión del formato a utilizar, y con esto el límite de elementos por etiqueta. Mientras que en los archivos PDF el número de entradas de la tabla de referencias cruzadas depende de la información y de los objetos utilizados.

La ventaja de utilizar documentos ODF en nuestro esquema, es la posibilidad de determinar de antemano la cantidad de información a ocultar, seleccionando un determinado número de etiquetas y atributos. Un ejemplo es, si utilizamos solamente los atributos de la equiteta principal del cuerpo del documento, con un total de 34 atributos permutables, es posible almacenar 128 bits de información [1].

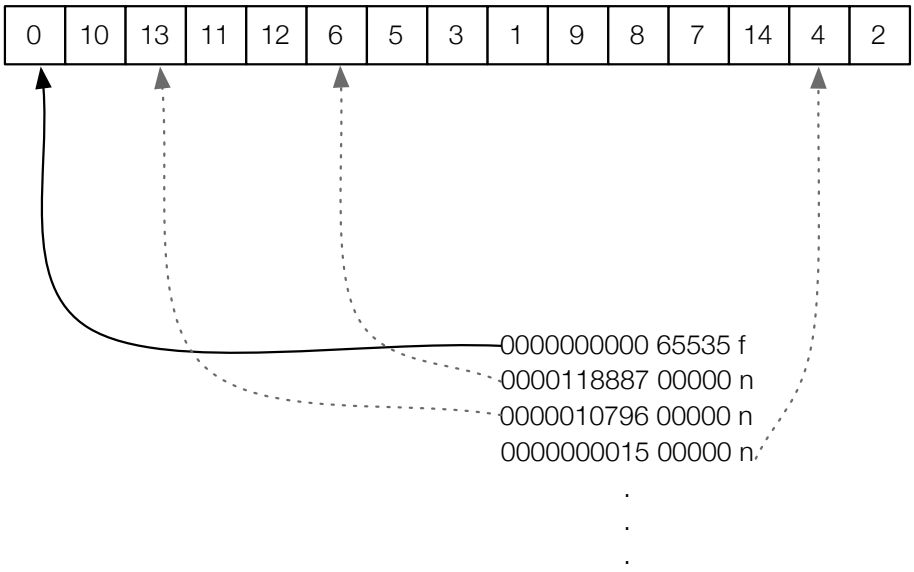

Figura 5.3: Creación del elemento F en el esquema esteganográfico

Como puede ser evidente solamente los elementos que se pueden permutar serán los utilizados para ocultar la información. En el capítulo 3 se mostraron dos métodos que utilizan permutaciones para ocultar información, si utilizamos el descrito por Artz [2], tendríamos que asignar un cero o uno a cada elemento para utilizarlo como una secuencia de bits, sin embargo en nuestra propuesta se representará de la forma:

$$F = (f_0, f_1, \cdots, f_{n-1}),$$

donde cada elemento f_i es un entero positivo en el rango $[0..n-1]$, un ejemplo de esta representación se muestra en la Figura 5.3.

5.2 ENUMERACIÓN DE PERMUTACIONES

Se llama permutación de un conjunto a una biyección de éste sobre sí. $\forall n \in \mathbb{N}$ designaremos por S_n el conjunto de todas las permutaciones del intervalo $[1, n]$ de $\mathbb{N}$ [27].

1 El número de bits se calcula: $\lceil \log_2 \lfloor n! \rfloor \rceil$, donde n es el número de elementos permutables

Una forma de representar los elementos del conjunto S_n es mediante un índice i con $0 < i \leq n!$ y algunos métodos convencionales para convertir el entero en una representación habitual de una permutación, como una secuencia de la forma

$$\pi = [a_1, a_2, \cdots, a_n],$$

donde cada a_i es un entero positivo menor a n. Una correspondencia entre n permutaciones y los primeros $n!$ números naturales asocia a cada permutación π su índice i y se escribe $\pi = \pi_i$, se dice pues que π_i es la i-ésima permutación.

Índice	Permutación
1	[123]
2	[132]
3	[213]
4	[231]
5	[312]
6	[321]

Tabla 5.1: Enumeración de todas las permutaciones para $n = 3$

En el esquema propuesto uno de los cálculos más importantes en el proceso de ocultamiento de la información es la correcta enumeración de permutaciones del conjunto S_n. Sedgewick [30] resume varios algoritmos para generar permutaciones, e identifica el algoritmo de cambio mínimo de *Heap*, que generalmente es el más rápido para calcular todas las permutaciones de una lista con n elementos [32].

Para un número pequeño de n es fácil computar todas las permutaciones del conjunto y enumerarlas, como se muestra en la Tabla 5.1. Sin embargo, esta solución no es práctica para cuando n es grande, en realidad no necesariamente muy grande, por la cantidad de elementos a generar y el espacio en memoria que se requiere. Buscando otra forma de solucionar el problema recordamos la definición de una permutación y la forma en que ésta se representa.

Algoritmo 5.1 Cambio de base decimal a factorial

ENTRADA: $i \in \mathbb{N}$
SALIDA: D: representación de i en base factorial

```
1: b ← 1
2: while i > 0 do
3:     r ≡ i  mód b
4:     i ← i  div b
5:     D[b] ← r
6:     b ← b + 1
7: end while
8: return D
```

Como ya se mencionó, una forma de representar una n-permutación es mediante su índice i con $0 < i \leq n!$ y algunos métodos convencionales para convertir el entero en una representación habitual de una permutación, como una secuencia. Para este propósito se utilizó

un sistema para cambio de base, estudiado originalmente por Witzschel and Cantor [36] y presentado por Knuth [23] como el "sistema de números factoriales", un sistema de cambio de base adaptado para enumerar permutaciones, en donde un factorial representa la posición de un dígito.

Base	8	7	6	5	4	3	2	1
Valor de posición	7!	6!	5!	4!	3!	2!	1!	0!
Valor de posición en decimal	5040	720	120	24	6	2	1	1

Tabla 5.2: Ejemplo del sistema de números factoriales

Mediante la conversión de un número menor que $n!$ a la representación factorial, se obtiene una secuencia de n dígitos $d_n, d_{n-1}, \cdots, d_2, d_1$, Algoritmo 5.1, en donde el i-ésimo dígito desde la derecha tiene base i, lo que significa que la cifra debe ser estrictamente menor que i, y que su valor será multiplicado por $(i-1)!$ (su valor de posición), un ejemplo se muestra en la Tabla 5.2.

El número factorial calculado, se pueden convertir a una permutación de n de una manera directa, usualmente se utilizaría una representación del código Lehmer o una tabla de inversión [21]. Sin embargo proponemos un nuevo método para generar la permutación a partir del número factorial, presentado en el Algoritmo 5.2.

Algoritmo 5.2 Obtener la i-ésima permutación

ENTRADA: Índice i en el intervalo $[1..n!]$, $n \in \mathbb{N}$
SALIDA: Permutación $\pi_i \in S_n$

```
1:  D ← Factorial(i)        // Algoritmo 5.1
2:  for j = 0 → n − 1 do
3:      π_1[j] ← j
4:      j ← j + 1
5:  end for
6:  for j = 0 → n − 1 do
7:      a ← D[j]
8:      π_i[j] ← π_1[a]
9:      Borrar π_1[a]
10: end for
11: return π_i
```

Dado que el número factorial representa la posición del elemento simplemente se compara este valor de posición con un arreglo ordenado, al que denominamos como permutación inicial o π_1 para así generar la permutación correspondiente a la posición, es decir, su índice.

En el proceso contrario, dada una permutación π_i y la primer permutación del conjunto $\pi_1 \in S_n$, se identifica el índice de la permutación en su representación en el sistema de numeración factorial. Lo anterior se puede realizar si se buscan los cambios en posición de los elementos de la permutación π_i con respecto a π_1, este proceso está descrito en el Algoritmo 5.3.

Algoritmo 5.3 Obtener el índice de una permutación

ENTRADA: $\pi_i \in S_n$, $n \in \mathbb{N}$
SALIDA: Índice i

1: **for** $j = 0 \to n-1$ **do**
2: $\pi_1[j] \leftarrow j$
3: $j \leftarrow j+1$
4: **end for**
5: **for** $j = 0 \to n-1$ **do**
6: $a \leftarrow \pi_i[j]$
7: $p \leftarrow$ Posicion(π_1, a) // *Regresar la posición del valor de a en* π_1
8: $D[j] \leftarrow p$
9: Borrar $\pi_1[p]$
10: **end for**
11: $i \leftarrow$ Decimal(D) // *Algoritmo 5.4*
12: **return** i

Finalmente la posición de la permutación π_i, es decir el número en su representación en el sistema de numeración factorial, se cambia de base para generar un número natural, esto es posible dado que el sistema de números factoriales proporciona una representación única para cada número natural, con la restricción indicada en los "'digitos" utilizados. Ningún número puede ser representado en más de una manera, porque la suma de los factoriales consecutivos multiplicado por el índice es siempre el siguiente factorial menos uno:

$$\sum_{i=0}^{n} i \cdot i! = (n+1)! - 1$$

El proceso para su cálculo se muestra en el Algoritmo 5.4.

Algoritmo 5.4 Cambio de base factorial a decimal

ENTRADA: D: representación de i en base factorial, $n \in \mathbb{N}$
SALIDA: $i \in \mathbb{N}$

1: $j \leftarrow n-1$
2: **for** $j = 0 \to n-1$ **do**
3: $a \leftarrow D[i] \times j!$
4: $i \leftarrow i + a$
5: $j \leftarrow j-1$
6: **end for**
7: **return** i

5.3 DEFINICIÓN, NOTACIÓN Y TERMINOLOGÍA

Se define matemáticamente el esquema esteganográfico. Sea k una clave esteganográfica extraída del conjunto, $\mathcal{K}$, de todas las claves secretas, $\mathcal{M}$ el conjunto de todos los mensajes

incrustables, $\mathcal{C}$ el conjunto de todos los objetos anfitriones y $\mathcal{W}$ el conjunto de los objetos marcados. El esquema está conformado por dos asignaciones, la de incrustación, I, y la de extracción, E:

$$I : \mathcal{C} \times \mathcal{K} \times \mathcal{M} \to \mathcal{W}$$
$$E : \mathcal{W} \times \mathcal{K} \to \mathcal{M},$$

tal que

$$\forall c \in \mathcal{C}, k \in \mathcal{K}, m \in \mathcal{M} : E(I(c,k,m),k) = m \tag{5.1}$$

El conjunto de objetos anfitriones $\mathcal{C}$, esta conformado por todos los documentos que cumplan la definición escrita en 5.1. El de los mensajes incrustables está definido por

$$\mathcal{M} = \{m | m = \{0,1\}^l; 0 < l \leq \lceil \log_2[n!] \rceil\},$$

donde n es el número de elementos del objeto anfitrión seleccionado c, y la función logaritmo denota la capacidad de inserción de la función I, su unidad son bits. El conjunto de las claves esteganográficas se denota como

$$\mathcal{K} = \{k | k \in \mathbb{N}; 0 < k < n!\},$$

donde $n!$ es la cardinalidad del conjunto S_n.

Una vez explicados todos los elementos del esquema esteganográfico procedemos a realizar la definición del estegosistema.

Definición 5.3.1. Un estegosistema para documentos de texto es una colección de tres algoritmos de tiempo polinomial (*G*,*I*,*E*) con las siguientes propiedades.

- El algoritmo para generación de claves *G*, toma como entrada el valor n y genera la clave compartida k
- El algoritmo de incrustación *I*, toma como entrada un objeto anfitrión c, un mensaje m, la clave k y produce el objeto marcado w
- El algoritmo de extracción *E*, toma como entrada el objeto tentativamente marcado w, la clave k y regresa el mensaje oculto m o un símbolo especial $\perp$ en caso de error, por ejemplo, cuando no hay un mensaje oculto.

5.4 ALGORITMO PARA GENERACIÓN DE CLAVES

El estegosistema propuesto como mecanismo para enviar y recibir mensajes, tiene el mismo reto que todo sistema criptográfico cuando se trabaja con claves privadas, la compartición del secreto. Aprovechando esta característica se ha decidido utilizar el protocolo de establecimiento de claves Diffie-Hellman. Como sabemos la seguridad del algoritmo Diffie-Hellman depende de la dificultad de resolver el logaritmo discreto [26].

Sin embargo, a diferencia de un sistema criptográfico en donde se define con anterioridad el primo a utilizar, en nuestra propuesta es necesario encontrar un primo que cumpla $p < n!$

Algoritmo 5.5 Generar un grupo cíclico y un elemento generador

ENTRADA: $n \in \mathbb{N}$
SALIDA: $p, g \in Z_p^*$

```
repeat
    repeat
        q ←$ {0,1}^⌈log2[(n−1)!]⌉
    until q es primo
    p = 2p + 1
until p es primo
repeat
    g ←$ Z_p^*
until (g^2 mód p) ≠ 1 y (g^q mód p) ≡ 1
return p, g
```

dado que para cada objeto anfitrión que se utilice el número de elementos es diferente, o al menos para el formato PDF como se describió en la sección 4.2. Por lo tanto necesitamos generar un grupo cíclico de orden primo y uno de sus generadores.

Z_p^* es conocido por ser un grupo cíclico de orden $p-1$ si p es un número primo. Cuando $p = 2q + 1$, donde q es también un número primo, mejor conocido como un primo Sophie Germain, se dice que es un primo seguro, q es muy interesante porque $p - 1 = 2q$ significa que cualquier subgrupo de Z_p^* sólo puede tener orden 2 o q, más aún todos son grupos cíclicos. Llamaremos al subgrupo de orden q como G_q, el cual es de mucho interés, ya que cualquier elemento distinto de la identidad es un generador. El proceso para generar el grupo cíclico y seleccionar p y g se describe en el Algoritmo 5.5.

Una vez seleccionados p y g, las partes a comunicarse, Alicia y Beto debe realizar:

A. Alicia escoge $a \in Z_{p-1}$ al azar, calcula $A = g^a \mod p$ y lo envía a Beto

B. Beto escoge $b \in Z_{p-1}$ al azar, calcula $B = g^b \mod p$ y lo envía a Alicia

C. Alicia calcula $k = B^a \mod p$

D. Beto calcula $k = A^b \mod p$

De esta forma Alicia y Beto comparten la clave k que será utilizada para incrustar y extraer el mensaje.

5.5 ALGORITMO DE INCRUSTACIÓN

La función de *Incrustación* recibe como entrada el objeto anfitrión c, el mensaje a ocultar m y la clave esteganográfica k. A partir del objeto anfitrión se obtienen los n elementos a permutar, almacenados en un arreglo $f = (f_0, f_1, \cdots, f_{n-1})$ ordenados de forma creciente en un orden lexicográfico. Después se calcula

$$x \equiv (k + m) \mod n! \tag{5.2}$$

como se puede apreciar x se encuentra dentro del rango $[0..n! - 1]$ y dada nuestra definición de permutación mostrada anteriormente, x es el índice de una permutación $\pi_x \in S_n$, la cual debe ser computada mediante una función

$$f : \mathbb{N} \to S_n \tag{5.3}$$

como se explicó en la sección 5.2.

En base a π_x y f se crea un nuevo arreglo de elementos f' para mapear el índice de los elementos de f con el valor correspondiente en π_x. Finalmente el arreglo f' se sustituye por f en el archivo original c para dar lugar al archivo alterado w con el mensaje incrustado. Este proceso se muestra en el Algoritmo 5.6.

Algoritmo 5.6 Algoritmo de incrustación

ENTRADA: $c \in \mathcal{C}, m \in \mathcal{M}, k \in \mathcal{K}$
SALIDA: $w \in \mathcal{W}$
1: $f, n \leftarrow$ Leer(c)
2: Ordenar(f)
3: $x \equiv (k + m) \mod n!$
4: $\pi_x \leftarrow$ Permutacion(x) // *Algoritmo 5.2*
5: $f' \leftarrow$ Ordenar(f, π_x)
6: $w \leftarrow$ Escribir(c, f')
7: **return** w

5.6 ALGORITMO DE EXTRACCIÓN

La función de *Extracción* recibe como entrada un objeto tentativamente marcado w y la clave esteganográfica k. Mediante un método de lectura se localizan los n elementos permutados y se almacenan en un arreglo $f' = (f_0, f_1, \cdots, f_{n-1})$, a partir de este arreglo se hace una copia $f = f'$, este nuevo arreglo f se ordena de forma lexicográfica. A partir de ambos arreglos f y f' se construye la permutación π identificando el orden de cada elemento de f' en f.

A partir de esta posible permutación válida π se calcula su índice x, mediante una función

$$f^{-1} : S_n \to \mathbb{N} \tag{5.4}$$

como se describe en el Algoritmo 5.3.

Con la clave esteganográfica k y el índice x se calcula

$$m \equiv (x - k) \mod n! \tag{5.5}$$

y con esto se regresa el mensaje m, si el archivo w, la clave k o ambos son incorrectos el mensaje será ilegible, al cual se le denomina como $\perp$. El proceso se resume en el Algoritmo 5.7

5.7 SEGURIDAD DEL ESTEGOSISTEMA

En un sistema de marcas de agua digitales una propiedad muy importante es la *robustez* del sistema, que en pocas palabras se puede explicar como la capacidad de poder recuperar la

Algoritmo 5.7 Algoritmo de extracción

ENTRADA: $w \in \mathcal{W}, k \in \mathcal{K}$
SALIDA: $m \in \mathcal{M}$
1: $f', n \leftarrow$ Leer(w)
2: $f = f'$
3: Ordenar(f)
4: $\pi_x \leftarrow$ construir una permutación a partir de f y f'
5: $x \leftarrow$ Índice(π_x) // *Algoritmo 5.3*
6: $m \equiv (x - k) \mod n!$
7: **return** m

marca después de que el medio digital marcado fue sometido a procesos de distorsión. En el escenario de marcas de agua la atacante Isabel puede tener conocimiento de la existencia de la marca y su objetivo es separarla de su objeto anfitrión.

Sin embargo en el escenario de las herramientas esteganográficas, como lo describen Cox et al. [9], nuestro adversario no sabe de la existencia de un mensaje oculto y sus metas son 1) saber si un objeto en particular tiene un secreto y 2) recuperar el mensaje. Teniendo estos objetivos del adversario presentes damos algunas características del estegosistema definido para indicar su robustez o mejor dicho su *seguridad*.

La definición de seguridad de un sistema esteganográfico presentada por Cachin [7] asume que el adversario permite a Alicia enviar cualquier objeto a Beto que este dentro del grupo de elementos permitidos. Para nuestro esquema son los archivos ODF y PDF, cualquier otro objeto enviado será destruido. Pero aunque permite la comunicación de objetos válidos, Isabel revisa cada uno de los objetos enviados para tratar de identificar información oculta.

En la definición de Cachin c, s son objetos válidos, sin información y con información oculta respectivamente, y P_C, P_S son las distribuciones de probabilidad que cualitativamente indican el conocimiento de Isabel sobre la legitimidad de la comunicación entre Alicia y Beto. Isabel como observadora (atacante pasivo) tiene que decidir el tipo de objeto enviado, asignando un valor a la distribución de probabilidad correspondiente, la diferencia de entropía entre P_C y P_S indica la seguridad del estegosistema.

En nuestro esquema esteganográfico el atacante pasivo debe poder diferenciar un objeto anfitrión de un objeto marcado, es decir, tomar la decisión de si un documento ODF o un archivo PDF contienen información oculta. Tomando como referencia la especificación técnica del formato estándar, omitiendo detalles de implementación y sin realizar análisis estadístico, sería muy difícil para cualquier atacante poder diferenciar los archivos ya que no se agrega información adicional al contenido del archivo, el estándar tampoco indica una forma en que deban estar ordenados los elementos que permutamos cualquier combinación es igualmente probable, así por simple definición cualquier objeto marcado puede pasar por un objeto simple si no se tiene conocimiento de la existencia del secreto.

6

LA IMPLEMENTACIÓN DEL ESQUEMA

Diseñamos y desarrollamos una aplicación de software multiplataforma que permite a los usuarios enviar y recibir mensajes secretos de forma segura, implementando un esquema de ocultamiento de información. Los aspectos teóricos más importantes del esquema se analizaron y probaron a través del desarrollo. El prototipo cubre las funcionalidades escenciales para el correcto ocultamiento de información en documentos de texto y muestra la posibilidad de realizar implementaciones reales a mayor escala. Como se mencionó anteriormente, el esquema propone el uso de archivos de texto estándares, específicamente los de tipo PDF y ODF por las características presentadas en el capítulo 4.

En este capítulo describimos los detalles de nuestro prototipo de software a través de las diferentes funcionalidades que presenta, las principales fueron creadas en un paquete esteganográfico independiente, con la intención de proporcionar un Interfaz de Programación de Aplicaciones (siglas del inglés *Application Programming Interface*) (API) de desarrollo para facilitar su reutilización, los algoritmos implementados para lograr este propósito son los presentados en el capítulo 5. Tanto el paquete esteganográfico como las funciones de negocio que le permiten la interacción con la interfaz del usuario, fueron escritas completamente en go [18], mientras que la interfaz de usuario utiliza una combinación de tecnologías web para sacar la potencialidad de los nuevos equipos de cómputo. A diferencia de cualquier aplicación web, los usuarios solo deben tener el archivo binario para poder hacer uso de la aplicación, o bien el compilador del lenguaje go para ejecutarlo desde el código fuente, no se requiere de un servidor web, o de un servidor de aplicaciones o algún tipo de contenedor.

Finalmente se describen las pruebas realizadas al paquete esteganográfico para verificar el correcto ocultamiento de los datos en los archivos, y la correcta interpretación de éstos en los visores como es el caso de *Acrobat Reader*, y aquellas realizadas a la aplicación web para verificar la compatibilidad entre los diferentes tipos de dispositivos.

6.1 UNA BREVE DESCRIPCIÓN DEL SISTEMA

Con el propósito de brindar una herramienta para hacer uso del esquema de ocultamiento de información, se desarrolló un paquete que puede ser instalado y vinculado a cualquier proyecto para el lenguaje go , todos los algoritmos fueron escritos desde cero para asegurar el correcto cómputo de las permutaciones y la correcta generación de claves esteganográficas seguras.

Las funciones públicas son,

- *Generador* de claves esteganográficas.
- El método de *Incrustación* de información.
- El método de *Extracción* de información

mientras que las funciones privadas se encargan de realizar las siguientes operaciones,

- Manipulación de archivos, para la lectura y escritura de los documentos PDF y ODF
- Cómputo de permutaciones

los detalles de cada función se describen en la sección 6.2.

Por otro lado, el desarrollo de la aplicación multiplataforma se diseñó con la idea de permitir al usuario conocer todas las etapas del esquema de ocultamiento, como un mecanismo de aprendizaje y mostrar la capacidad del sistema, es decir, la cantidad de información que se puede ocultar en un determinado archivo, además de la longitud de la clave secreta.

A continuación se presenta en forma de lista, el procedimiento que debe realizar el usuario para poder ocultar información en un documento de texto y la forma de recuperarlo, suponemos que la aplicación está en ejecución en algun equipo con acceso a la red.

A. El usuario utiliza su equipo de cómputo, cualquiera que cuente con un navegador de Internet reciente, para ingresar la dirección de red del servidor.

B. Dependiendo del equipo de cómputo utilizado, el sistema muestra una interfaz dedicada para mejorar el desempeño y hacer uso de las ventajas del equipo cliente, por ejemplo si es un dispositivo móvil la interfaz se adapta al comportamiento de las aplicaciones nativas y a las pantallas táctiles.

C. El sistema le guía los pasos necesarios para su instalación, (específicamente para equipos con plataforma iOS) y muestra la pantalla principal.

D. El usuario debe ingresar la dirección de red del archivo que desea usar para ocultar la información, en caso de ingresar desde una computadora de escritorio puede seleccionar un archivo local.

E. El sistema analiza el archivo para verificar que cumpla los requerimientos estándares del formato, en caso de no cumplirlo muestra un mensaje.

F. El usuario ingresa el mensaje a ocultar, cuya longitud dependerá del archivo seleccionado, el sistema indica la capacidad de inserción.

G. El sistema genera una clave esteganográfica y realiza la función de incrustación para generar el documento alterado.

H. El sistema muestra la clave esteganográfica, que será utilizada para recuperar el mensaje.

I. En el proceso de recuperación el usuario solamente debe proporcionar el nombre del archivo y la clave esteganográfica.

Ahora se describen los detalles de cada componente de software creado.

6.2 EL PAQUETE ESTEGANOGRÁFICO

Como se mencionó anteriormente el primer grupo de funciones escritas en el lenguaje de programación go, son el resultado de implementar los algoritmos y procedimientos presentados en el capítulo 5, los que componen el esquema esteganográfico o de ocultamiento de información. Pero antes se describen las funciones privadas.

6.2.1 *Manipulación de archivos*

Como todo sistema esteganográfico se requiere de un objeto anfitrión, el cual será utilizado para ocultar la información, como ya se ha mencionado en varias ocaciones, dos son los tipos de archivos aceptados PDF y ODF, de tal forma es necesario el desarrollo de algunas funciones básicas para modificar los archivos.

La primera de estas funciones tiene como propósito identificar el tipo de archivo seleccionado, lo cual se logra leyendo el encabezado del archivo y localizando su tipo, en la Figura 6.1 se encuentra la cadena para cada uno. En go esta operación se puede realizar usando las funciones básicas del lenguaje, para la lectura de archivos y otras que permiten la comparación entre cadenas de caracteres.

```
a) %PDF
b) mimetypeapplication/vnd.oasis.opendocument
```

Figura 6.1: Cadenas para identificar el tipo de archivo en el encabezado
a) Define un archivo PDF, y b) uno del tipo ODF

Una vez identificado el tipo de archivo anfitrión es necesario leer el contenido del mismo, en formato binario para buscar los elementos que pueden ser permutados.

6.2.1.1 *Lectura de archivos ODF*

Como está especificado en el formato estándar 4.1.1, cualquier documento ODF es un archivo JAR, es decir, un comprimido en ZIP que contiene un archivo extra que indica su tipo, el archivo `mimetype`. El primer paso necesario es descomprimir el archivo para tener acceso al contenido. Afortunadamente go tiene dentro de sus funciones algunas para trabajar con comprimidos de tipo ZIP y otras para escanear apropiadamente documentos en lenguaje XML, lo que facilitó la implmementación.

Se descomprime el archivo ODF para buscar entre los archivos XML los elementos a permutar, como la especificación del lenguaje XML establece, la primer línea de cada archivo indica la versión del formato utilizada, esta se descarta, las siguientes se analizan para buscar las etiquetas y los atributos a utilizar, una lista completa de los atributos puede ser localizada en la especificación del formato estándar [33]. Una etiqueta en el formato ODF se compone como sigue:

```
<nom_etiqueta:id
  (nom_atributo:id_atributo="valor_atributo")* >
```

```
</nombre_etiqueta:id>
```

y en donde

```
< : = " > /
```

son símbolos en el lenguaje.

Al momento de localizar la o las etiquetas correspondientes los atributos

```
nombre_atributo:identificador_atributo="valor_atributo"
```

son almecenados en un arreglo, el cual posteriormente será ordenado lexicográficamente.

6.2.1.2 *Lectura de archivos PDF*

El tipo de archivo PDF es más sencillo de leer, todo el contenido se guarda en formato binario, go permite leer archivos y realizar filtros por palabra, línea o mediante alguna función personalizada.

Se utilza un escaner para filtrar el contenido en búsqueda del texto `xref` que indica el inicio de la tabla de referencias cruzadas, la siguiente línea se descompone para conocer el tamaño de la tabla y crear un arreglo de esta capacidad, en el cual se guardarán una a una cada entrada de la tabla. Posteriormente este arreglo se ordena, como se describe en el Algoritmo 5.6.

En ambas funciones de lectura de archivos, el arreglo resultante de elementos seleccionados es utiliza como una permutación, como se ilustra en la Figura 6.2 la permutación inicial del algoritmo de incrustación.

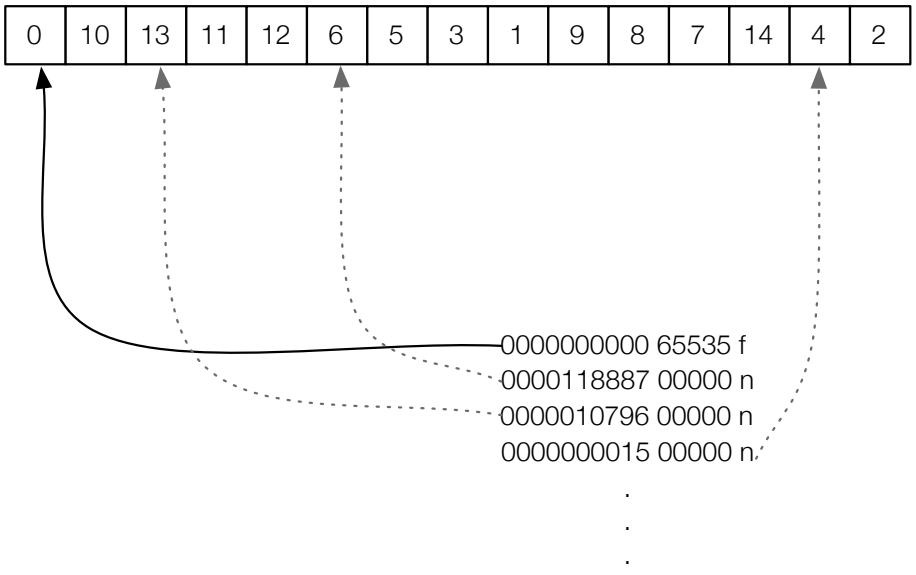

Figura 6.2: Permutación inicial del objeto anfitrión

LENGTH	METHOD	SIZE	CMPR	NAME
39	Stored	39	0 %	mimetype
45	Defl:N	37	18 %	layout-cache
807121	Defl:N	76945	1 %	Pictures/18.png
10403	Defl:N	2178	79 %	content.xml
11784	Defl:N	2165	82 %	styles.xml
9187	Defl:N	1424	85 %	settings.xml
1053	Stored	1053	0 %	meta.xml
899	Defl:N	261	71 %	manifest.rdf

Tabla 6.1: Contenido de un empaquetado ODF

6.2.1.3 *Escritura de archivos ODF*

Como es lógico una función que permita modificar o escribir documentos ODF es necesaria, al término de la función de incrustación se tiene un nuevo órden de los elementos que originalmente fueron seleccionados en el archivo. Por las mismas causas explicadas en la lectura de este tipo de archivos, la escritura también es más complicada que en los archivos PDF. Primero se descomprime el archivo ODF para conocer la estructura y los archivos que contiene, aquellos que no se modifican simplemente se guardan en memoria mientras se termina el proceso.

Los archivos utilizados para recuperar los elementos permutables, se leen de forma independiente para encontrar las etiquetas y atributos utilizados, en donde se realiza el reemplazo de los elementos, en base a la posición inicial de los atributos, es posible escribirlos en el nuevo orden, sin alterar el resto del documento, de esta forma el contenido de los archivos, la forma en que se muestra la información tanto en pantalla como en papel al momento de imprimirse es idéntico al original, los cambios solo afectan el orden de lectura por parte de los visores de documentos.

Al terminar de crear los archivos del documento se procede a crear el empaquetado, en este punto se debe tener cuidado ya que el orden de los archivos dentro del ZIP es importante, incluso es necesario que el primero sea el de `mimetype` y éste no debe tener compresión, como se muestra en la Tabla 6.1.

6.2.1.4 *Escritura de archivos PDF*

La función que permite la escritura de archivos PDF lee y escanea el contenido del archivo línea a línea, hasta encontrar la palabra reservada para la tabla de referencias cruzadas, y mientras no se encuentra, se almacena en memoria el contenido del documento original, que posteriormente se utilizará para escribir el nuevo archivo. Al encontrar el inicio de la tabla de referencias cruzadas y el número de elementos permutables, se sustituye por el nuevo órden de registros de la tabla, generado por la función de incrustación. Y se termina al crear el nuevo documento PDF.

6.2.2 *Cómputo de permutaciones*

Las funciones encargadas de generar los índices de una permutación y la permutación misma se basan en los Algoritmos 5.2 y 5.3, descritos en el capítulo anterior.

A. *El manejo de números muy grandes*, para un valor de n el número de permutaciones posibles es $n!$ y el rango de índices a elegir está entre $[0..n! - 1]$, en la mayoría de los equipos de cómputo actuales el límite en las variables estándares es de 64-bits, es decir la posibilidad de guardar en memoria un valor entre 0 y $2^{64} - 1$, que en relación con el crecimiento de las posibles permutaciones de un entero solo se podría calcular para $n \leq 20$, por tal motivo se hizo uso de la biblioteca de precisión múltiple incorporada en el lenguaje, identificada como el paquete `math.Big` [19].

B. *El uso dinámico de arreglos*, como se describe en el algoritmo de cómputo de permutaciones, es necesario utilizar arreglos, a diferencia de otros lenguajes de programación en donde los arreglos típicamente tienen una longitud definida o es necesario asignar la cantidad de memoria en el momento de la definición, en go se tiene un tipo de dato que permite incrementar el tamaño del arreglo denominado `slice`, utilizado para facilitar el cómputo de permutaciones e índices.

6.2.3 *Generador de claves esteganográficas*

Una de las tres funciones públicas del paquete esteganográfico, implementa el Algoritmo 5.5 descrito en la sección 5.4, de tal forma que recibe como entrada el número de elementos a permutar en el documento, número que puede ser fácilmente localizado como se describió anteriormente. Las operaciones aritméticas se realizan con el paquete `math.Big` [19] y como es esperado regresa un número primo y un generador del campo $p, g \in Z_p^*$ en donde $p < n!$, valores que se utilizan para compartir la clave esteganográfica, como sabemos:

A. Alicia escoge $a \in Z_{p-1}$ al azar, calcula $A = g^a \mod p$ y lo envía a Beto

B. Beto escoge $b \in Z_{p-1}$ al azar, calcula $B = g^b \mod p$ y lo envía a Alicia

C. Alicia calcula $k = B^a \mod p$

D. Beto calcula $k = A^b \mod p$

De esta forma Alicia y Beto comparten la clave k que será utilizada para incrustar y extraer el mensaje.

Además de la implementación del esquema de intercambio de claves, se agregaron dos métodos que no se definieron previamente en el diseño del esquema,

A. La posibilidad de ingresar una contraseña como clave,

B. La generación de números aleatorios para una Clave de un solo uso (en inglés *One-time Pad*) (OTP)

En el primero, un usuario puede ingresar una contraseña personalizada para ocultar la información, de la misma forma en que se procesa el mensaje se procesa la contraseña para utilizarla como un número en el rango $[0..n! - 1]$, la segunda opción es generar un número pseudo-aleatorio con los paquetes predeterminados del lenguaje go de la misma longitud al mensaje, al final del proceso se sugiere al usuario compartir la clave ya sea la ingresada o la generada por medio de un mensaje SMS para los usuarios con dispositivo móvil.

6.2.4 *Incrustación del mensaje*

La función principal es la encargada de ocultar la información en el archivo especificado, de tal forma recibe como entrada el nombre del archivo (compuesto de la localización, el nombre y la extensión) y el mensaje (secuencia de caractéres de la tabla ASCII), en `go` ambas variables son de tipo `string`.

El nombre del archivo es procesado para leer los elementos permutables, como se mencionó anteriormente, mientras que el mensaje ingresado debe ser codificado de tal forma que se interprete como un número entero. Para hacer esta codificación cada caracter en la secuencia se interpreta por su valor decimal dentro de la tabla ASCII

6.2.5 *Extracción del mensaje*

La función encargada de recuperar el mensaje a partir de un archivo tentativamente marcado requiere del nombre del archivo y la clave esteganográfica, el primero compuesto de la localización, del nombre y extensión del archivo, mientras que el segundo es o bien la clave generada por el protocolo Diffie-Helman, la contraseña ingresada en la incrustación o el número aleatorio.

6.3 APLICACIÓN MULTIPLATAFORMA

Se desarrolló una aplicación multiplataforma que hace uso del paquete esteganográfico descrito en la sección anterior, con el propósito de brindar una herramienta que permita enviar mensajes secretos de forma segura, al ocultar la información en archivos comúnes. El sistema fue escrito en el lenguaje de programación `go`, y la interfaz del usuario utilizando `html5, css3, javascript`.

6.3.1 *Creación de la interfaz de usuario*

El usuario sabe que está utilizando una determinada tecnología de cómputo por la interfaz que opera, usualmente cada propietario de una plataforma establece un diseño de los elementos visuales como íconos, botones, incluso el esquema de colores, y todavía más importante crea un patrón en el comportamiento de los elementos. Si una aplicación se adapta en cierta medida a estos elementos, colores, iconos o el mismo comportamiento permite dar la ilusión de ser una aplicación nativa. Varias son las ventajas de crear un sistema que permita tener este comportamiento, entre las más importantes la usabilidad y la utilización de los recursos del dispositivo cliente.

Usando la última versión del lenguaje `html` y un conjunto de herramientas (`css3`, `javascript`) se realizó una interfaz de usuario basada en web que dependiendo del dispositivo cliente muestra un diseño independiente, con el propósito de utilizar el comportamiento y algunas características de cada dispositivo, por ejemplo, si el sitio es visitado desde un equipo con sistema operativo `iOS`, se adaptan los componentes a un tamaño apropiado de acuerdo a la pantalla, además de mostrar los pasos necesarios para incorporar un vínculo del sitio al menú

principal como si de una aplicación se tratara y con esto, hacer uso del caché del dispositivo para evitar descargar las imágenes cada vez que se abra la aplicación.

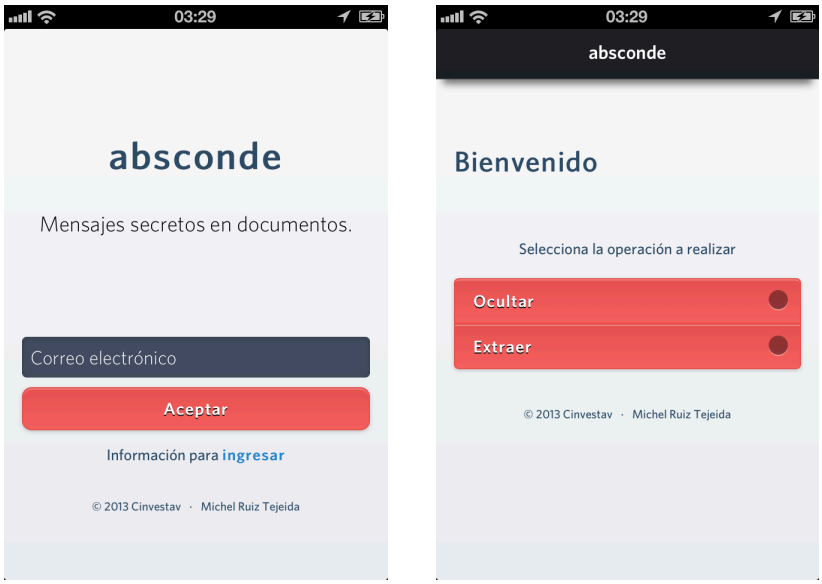

Figura 6.3: Pantalla principal del sistema, en un dispositivo iPhone

En la Figura 6.3, se muestra la pantalla principal del sistema al ser visitado desde un equipo iPhone.

6.3.2 *Capa de negocio*

Dentro de las funciones desarrolladas para la aplicación web se puede mencionar un administrador de sesión para llevar un control sobre los usuarios que hacen uso de la aplicación, así como algunos métodos de autenticación que pueden ser fácilmente incorporados, entre éstos la posibilidad de iniciar sesión a través de una cuenta en un servidor `ldap` o haciendo uso de una cuenta de Google, sin embargo de forma predeterminada no requiere de un nombre usuario y contraseña.

Algunas otras funciones se programaron para facilitar al usuario su uso, entre éstas el determinar la capacidad de un documento antes de iniciar el proceso de incrustación, así cuando se indica el tipo de documento y la ruta en la que se puede localizar el archivo se muestra la cantidad de letras posibles a ocultar.

6.4 PRUEBAS

La gestión de pruebas en nuestra implementación se basa en tres categorías, que se presentan a continuación.

6.4.1 *Cálculos matemáticos*

Aunque se cuidó el correcto diseño de los algoritmos encargados de las permutaciones, presentados en la sección 5.2, se decidió realizar pruebas exhaustivas utilizando el lenguaje de programación `Mathematica` [37], para verificar el correcto cálculo, antes de ser escritos en `go`, con ayuda de las funciones integradas en esta herramienta de álgebra computacional.

Para la generación de permutaciones se utilizó la función incluida `Permutations`, que permite generar todas las posibles permutaciones de una lista de n elementos [38], la cual nos ayudó a verificar

A. la correcta generación de todas las $n!$ permutaciones posibles, y

B. el orden lexicográfico en la generación de la siguiente permutación

el segundo punto es primoridial, porque así podemos estar seguros que una permutación tendrá siempre el mismo índice en el rango de todas las permutaciones para un valor de n en particular.

6.4.2 *Validación de archivos*

Para la definición del esquema esteganográfico nos basamos en el estándar técnico de cada archivo, sin embargo era necesario verificar que los visores de documentos como *Acrobat Reader* y *OpenOffice*, interpretaran adecuadamente los archivos alterados.

Para este propósito se seleccionaron diferentes documentos de texto de ambos tipos con contenido muy diferente para cubrir la mayor cantidad de escenarios posibles. Algunos resultados fueron interesantes, por ejemplo no todas las herramientas de generación de archivos de tipo PDF utilizan la versión estándar, en su lugar utilizan alguna versión anterior o más reciente, entre estas se encuentra `pdflatex`. El resultado de no utilizar la versión estándar es la posible omisión de la tabla de referencias cruzadas, lo que resulta en un archivo no válido.

Otro dato interesante a partir de estas pruebas tiene que ver con los mismos visores, se mostró que algunos visores como `evince, poppler` (visor predeterminado en sistemas operativos `Linux` basados en la distribución `debian`) no soportan el ajustar la tabla de referencias cruzadas [6].

6.4.3 *Funcionales*

La aplicación se ejecutó en un equipo portátil con conexión inalámbrica a una red del Departamento de Computación del CINVESTAV, a través del cual se realizó la petición al sistema utilizando diferentes equipos de cómputo, dispositivos móviles, tabletas y otras computadoras. La interfaz se comportó adecuadamente de acuerdo al dispositivo, una colección de archivos elegidos al azar se utilizaron para ocultar mensajes de diferente longitud, al igual que diferentes formas de generar la clave a utilizar.

Para archivos con una gran capacidad de inserción y seleccionando el protocolo Diffie-Hellman, se registró un impacto en el rendimiento del sistema, provocado por la función de generación de grupos cíclicos para la selección del número primo y su generador a utilizar.

Motivo por el cual se decidió agregar nuevas formas para utilizar las claves secretas como se describió anteriormente. Esta función se implementó en el lenguaje de programación C para identificar si se podía generar una mejor respuesta, sin embargo los resultados no fueron muy diferentes.

7

APLICACIONES

En este capítulo presentamos dos ideas sobre aplicaciones específicas que se pueden realizar tomando como base el diseño del esquema esteganográfico definido en el capítulo 5, y la implementación del paquete de desarrollo descrito en el capítulo 6, como una retroalimentación en nuestro trabajo y aprovechando para mostrar el potencial que tiene un esquema de ocultamiento de información.

7.1 AUTENTICACIÓN DE DOCUMENTOS

En criptografía, un Código de Autenticación de Mensaje (siglas del inglés *Message Authentication Code*) (MAC), es un pequeño fragmento de información que se utiliza para autenticar y proporcionar integridad a un mensaje. Un algoritmo de MAC es comúnmente denominado como una función de picadillo *(hash function)* por clave criptográfica, acepta como entrada una clave y un mensaje de longitud arbitraria, la salida se le conoce como una MAC. El valor MAC protege la integridad de los datos del mensaje, así como su autenticidad, al permitir que el verificador (el poseedor de la clave secreta) pueda determinar si hubo algún cambio en el contenido del mensaje.

En esta sección mostramos un nuevo método para autenticación de documentos, a partir del estegosistema presentado, que brinda los mismos servicios de un MAC criptográfico. Usualmente para usar una MAC, Alicia debe enviar el mensaje a Beto junto con la MAC, como sólo él tiene la clave compartida puede autenticar el mensaje, sin embargo, eso no le impide a Isabel conocer la MAC, si está al pendiente del canal, e intentar hacer algo con ésta para evitar que Beto la verifique. En el procedimiento que se muestra a continuación se impide a Isabel conocer el código de autenticación.

En la Figura 7.1 se muestra el procedimiento que tendría que seguir Alicia para autenticar un mensaje antes de enviárselo a Beto, sea

$$h : \mathcal{C} \to \mathcal{Y} \tag{7.1}$$

una función de picadillo sin clave y $\mathcal{Y}$ el conjunto de todos los posibles picadillos o etiquetas de autenticación, se calcula $y = h(c)$ aplicando la función h en un documento c, se utiliza a y como el mensaje a ocultar en el esquema esteganográfico.

La ventaja de utilizar una determinada función h es que la salida tiene una longitud en bits fija de la forma $h : \{0,1\}^* \to \{0,1\}^n$. En un documento anfitrión puede ajustarse el número de elementos para ocultar la cantidad exacta de bits o bien incluir el código de autenticación concatenando un mensaje adicional. Con este procedimiento realizado, Alicia solamente tiene que enviar el documento y Beto puede autenticarlo al recupear el mensaje, el archivo original y compararlo con la misma función de picadillo utilizada, como se muestra en la Figura 7.2.

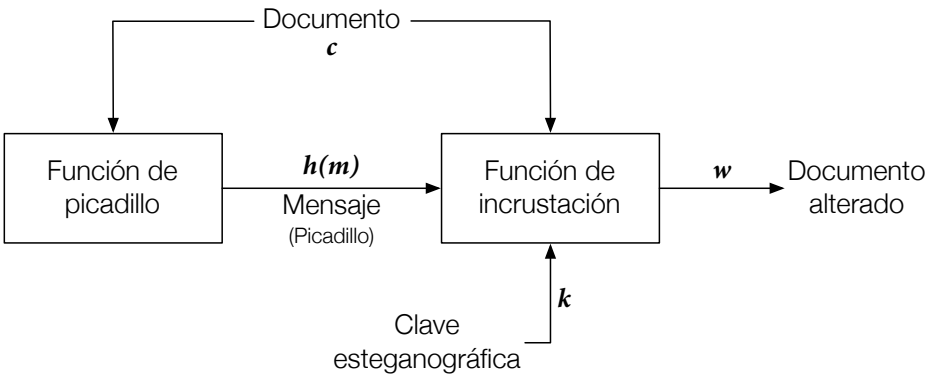

Figura 7.1: Esquema de incrustación para autenticación de documento

7.2 SEGUIMIENTO DE DOCUMENTOS

Describimos el seguimiento de documentos como la capacidad de poder identificar los cambios que ha sufrido un documento desde su creación, un procedimiento que podría ser muy útil en organizaciones en donde se tiene un control estricto de los documentos. Con el esquema esteganográfico propuesto es posible brindar este servicio de seguimiento, si fuese posible modificar la forma en que las aplicaciones de *ofimática* crean los archivos ODF y PDF.

Las aplicaciones que permiten crear archivos ODF y PDF, usualmente no siguen el estándar definido por la ISO al pie de la letra, por ejemplo, una aplicación como `OpenOffice` tiene la función de guardar un documento en el formato PDF, pero si el contenido del documento dentro de la misma aplicación se modifica y se guarda nuevamente en este formato, la aplicación elimina el archivo generado previamente y crea uno nuevo, cuando por especificación del formato estándar es posible actualizar el archivo PDF sin tener que crearlo nuevamente, esta diferencia se ve afectada en la tabla de referencias cruzadas, utilizada por el esquema para ocultar mensajes como se describió en el capítulo 5.

En un archivo PDF se tiene una tabla de referencias cruzadas que indica los elementos que contiene, pero además puede indicar cuales están en uso y cuales no, de acuerdo al formato si un archivo se actualiza solamente se tienen que desactivar los objetos que ya no están en uso y registrar los nuevos, creando una nueva sección en la tabla con sus respectivas entradas. De esta forma por cada cambio en el documento habría una sección en la tabla de referencias cruzadas y el esquema puede implementarse de manera que a cada sección en la tabla se puede incrustar un mensaje diferente.

En el caso de los documentos ODF, `OpenOffice` solamente debe mantener los atributos de las etiquetas de forma intacta, en la actualidad reemplaza los atributos y en su lugar deja las predeterminadas por la aplicación. Con los atributos fijos siempre y cuando el documento sea válido conforme al formato, es posible ocultar $n!-1$ número de mensajes diferentes de longitud $1 < m \leq \lceil \log_2 \lceil n! \rceil \rceil$ bits. En donde la permutación generada por la función de incrustación en el último proceso de ocultamiento realizado se usaría como la permutación inicial en un proceso posterior.

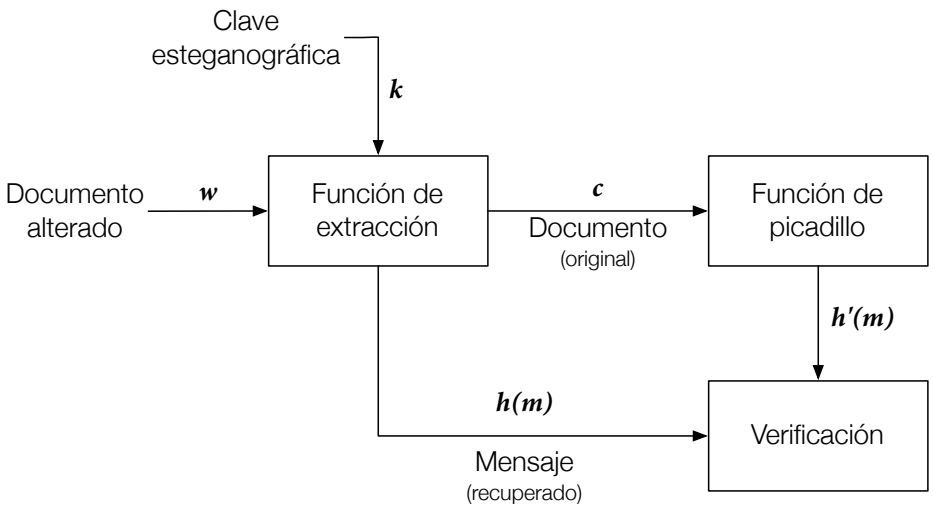

Figura 7.2: Esquema de extracción para verificación de integridad de documento

Para recuperar los mensajes solamente se tiene que recuperar uno a uno con la clave adecuada empezando por la última generada. Y con esto un sistema de seguimiento de documentos que no incremente el tamaño del documento y en caso de alguna modificación indebida se podrá detectar cuando no se pueda recuperar el mensaje, sólo algunos problemas se tendrían que resolver como la administración de las claves.

8

CONCLUSIONES Y PERSPECTIVAS

En este capítulo señalamos los resultados obtenidos en nuestro trabajo y las conclusiones sobre el análisis de nuestra aplicación. Por otra parte proponemos algunas perspectivas para el trabajo futuro.

8.1 CONCLUSIONES

Este trabajo presenta la definición de un nuevo *esquema esteganográfico* diseñado para proveer seguridad en las comunicaciones, cuando la criptografía no es una opción factible [31]. Nuestro interés era utilizar medios digitales de fácil creación, acceso y distribución, que no habían sido completamente explotados para este propósito.

La importancia del proyecto recae en la posibilidad de *ocultar información* en tipos de archivo que no tienen trabajos relacionados, para el caso del formato ODF; y la *seguridad* que brinda el esquema esteganográfico, relacionada con la dificultad computacional de generar todas las permutaciones de un conjunto.

La primer contribución asociada a la definición del esquema es la especificación de algoritmos computacionales, que permiten el cómputo eficiente de una determinada permutación dentro del conjunto de todas las permutaciones y el cálculo de su índice conforme a un orden lexicográfico.

Otra contribución es la creación de un paquete de desarrollo, en donde están implementados los algoritmos definidos en el esquema, el cual se espera sea utilizado en sistemas reales, esto a partir de los resultados obtenidos con la codificación de una aplicación multiplataforma que hace uso del paquete de desarrollo.

Algunos trabajos relacionados con el formato PDF pueden encontrarse en la literatura [15, 40], sin embargo a diferencia de éstos, nuesto esquema no modifica el contenido ni el tamaño del archivo.

Además de permitir una comunicación segura al ocultar la información que se comparte, el esquema presentado en este trabajo puede ser fácilmente modificado para brindar otros servicios de seguridad como la *autenticación de documentos*.

8.2 PERSPECTIVAS

Se describen algunas observaciones para futuro trabajo en el ámbito de la tesis:

A. Un análisis de seguridad profundo sobre el esquema esteganográfico, para encontrar posibles vulnerabilidades y presentar una solución adecuada. El análisis puede estar enfocado a los algoritmos con los que se realiza el cómputo de permutaciones, la generación e intercambio de claves esteganográficas y la probabilidad de poder identificar un archivo alterado de un original.

B. Buscar otros formatos que permitan utilizar el esquema definido, e identificar si es necesario realizar alguna modificación en la especificación.

C. Realizar un análisis para determinar si es posible redefinir el esquema a un estegosistema de clave pública para evitar el intercambio de claves.

NOTA IMPORTANTE: El software descrito en esta tesis estará disponible en http://computacion.cs.cinvestav.mx/~mruiz/software.

A

EL PAQUETE DE DESARROLLO

En este apéndice mostramos parte del código fuente escrito en el lenguaje de programación Go, que implementa el esquema esteganográfico.

```
/*
  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

package estego

import (
  "crypto/rand"
  "estego/mat"
  "estego/archivos"
  "math"
  "math/big"
      "os"
  "sort"
  "strconv"
  "strings"
)

func Generador(n int) (*big.Int, *big.Int) {
  fac := mat.Factorial(big.NewInt(int64(n)))
  for {
        q, err := rand.Prime(rand.Reader, fac.BitLen()-1)
        if err != nil {
            panic(err.Error())
        }
        n := new(big.Int).Mul(q, big.NewInt(2))
        p := new(big.Int).Add(n, big.NewInt(1))
        // p = 2q + 1, order(p)=n=2q
        if p.ProbablyPrime(4) {
            for {
                g, err := rand.Int(rand.Reader, p)
                if err != nil {
                    panic(err.Error())
                }
                if b := new(big.Int).Exp(g, big.NewInt(2), p); b.Cmp(big.
    NewInt(1)) == 0 {
                    continue
                }
                if b := new(big.Int).Exp(g, q, p); b.Cmp(big.NewInt(1)) == 0 {
                    return p, g
                }
```

```
            }
        }
    }
    return nil, nil
}

func Aleatorio(p, g *big.Int) (*big.Int, *big.Int) {
  a, err := rand.Int(rand.Reader, p)
  if err != nil {
    panic(err.Error())
  }
  A := new(big.Int).Exp(g,a,p)
  return a, A
}

func Clave(a, A, p *big.Int) *big.Int {
  k := new(big.Int).Exp(A,a,p)
  return k
}

func aleatorio(n *big.Int) *big.Int {
    a, err := rand.Int(rand.Reader, n)
  if err != nil {
    panic(err.Error())
  }
  return a
}

func Capacidad(nombre string) (uint64, error) {
    var bits uint64
    n, _, err := archivos.Leer(nombre)
    if err != nil {
        return 0, err
    }
    factorial := mat.Factorial(big.NewInt(int64(n)))
    if n <= 170 {
        temp := new(big.Rat)
        temp.SetInt(factorial)
        nfac, _ := temp.Float64()
        log := math.Ceil(math.Log2(nfac))
        bits = uint64(log)
    } else {
        bits = mat.Logaritmo(factorial)
    }

    return bits, nil
}

func Incrustacion(m, c string) (*big.Int, error){
  n, f0, err := archivos.Leer(c)
  if err != nil {
    return nil, err
  }
  mensaje := func(m string) *big.Int {
      var hex string
      bytes := []byte(m)
      for i := range bytes {
          hex_i := strconv.FormatInt(int64(bytes[i]),16)
          if len(hex_i) != 2 { hex_i = "0" + hex_i }
          hex += hex_i
      }
      a := new(big.Int)
      a.SetString(hex,16)
```

```
        return a
    }

    factorial := mat.Factorial(big.NewInt(int64(n)))
    k := aleatorio(factorial)
    x := new(big.Int).Add(k, mensaje(m))
    x.Mod(x,factorial)
    pi := mat.Permutacion(x, n)
    sort.Sort(sort.StringSlice(f0))
    f1 := make([]string,len(f0))
    for i := range f0 {
        f1[i] = f0[pi[i]]
    }
    err = archivos.Escribir(c, f1)
    if err != nil {
        return nil, err
    }

    return k, nil
}

func Extraccion(w string, k *big.Int) (string, error) {
    n, f1, err := archivos.Leer(w)
    if err != nil {
      return "", err
    }

    pi := func(s []string) []int64 {
        perm := make([]int64, len(s))
        t := make([]string, len(s))
        copy(t,s)
        sort.Sort(sort.StringSlice(t))
        for i := range s {
            for j := range t {
                if strings.EqualFold(s[i], t[j]) {
                    perm[i] = int64(j)
                }
            }
        }
        return perm
    }

    mensaje := func(m *big.Int) string {
        bytes := m.Bytes()
        return string(bytes)
    }

    factorial := mat.Factorial(big.NewInt(int64(n)))
    pi_x := pi(f1)
    x := mat.Indice(pi_x)
    m := new(big.Int).Sub(x,k)
    m.Mod(m,factorial)

    os.Remove(w)

    return mensaje(m), nil
}
```

Listing A.1: Funciones públicas

```
package mat

import (
```

```
  "math/big"
  "estego/util"
)

func Logaritmo(x *big.Int) (y uint64) {
    y = 0
  r := new(big.Int).Mod(x,big.NewInt(2))
  if r.Cmp(big.NewInt(1)) == 0 {
      x.Add(x,big.NewInt(1))
  }
  for {
      x.Div(x,big.NewInt(2))
      if x.Cmp(big.NewInt(0)) == 0 {
          y++
          return
      } else {
          y++
      }
  }

  return
}

func Factorial(n *big.Int) (resultado *big.Int) {
  resultado = big.NewInt(1)
  var uno big.Int
  uno.SetInt64(1)
  for n.Cmp(&big.Int{}) == 1 {
    resultado.Mul(resultado, n)
    n.Sub(n, &uno)
  }

  return
}


func Factoradico(entero *big.Int, n int) (factoradico []int64) {
  factoradico   = make([]int64, n)
  residuo, base := new(big.Int), big.NewInt(1)
  i := n - 1
  for entero.Cmp(big.NewInt(0)) > 0 {
    entero, residuo = entero.DivMod(entero,base,residuo)
    factoradico[i] = residuo.Int64()
    base.Add(base,big.NewInt(1))
    i--
  }

  return
}

func Decimal(factoradico []int64) (decimal *big.Int) {
  decimal = new(big.Int)
  j := len(factoradico) - 1
  for i := range factoradico {
    temp := big.NewInt(factoradico[i])
    temp.Mul(temp,Factorial(big.NewInt(int64(j))))
    decimal.Add(decimal,temp)
    j--
  }

  return
}
```

```
func Permutacion(indice *big.Int, n int) (perm []int64) {
  perm = make([]int64, n)
  base := make([]int64, n)
  factoradico := Factoradico(indice,n)
  for i := range base {
    base[i] = int64(i)
  }
  for i := range perm {
    pos := factoradico[0]
    perm[i] = base[pos]
    temp := make([]int64,len(factoradico)-1)
    copy(temp, factoradico[1:])
    factoradico = temp
    base = append(base[:int(pos)],base[int(pos)+1:]...)
  }

  return
}

func Indice(perm []int64) (indice *big.Int) {
  base, factoradico := make([]int64, len(perm)), make([]int64, len(perm))
  for i := range base {
    base[i] = int64(i)
  }
  for i := range factoradico {
    a := perm[i]
    pos := util.Posicion(base, a)
    factoradico[i] = int64(pos)
    base = append(base[:int(pos)],base[int(pos)+1:]...)
  }
  indice = Decimal(factoradico)

  return
}
```

Listing A.2: Funciones matemáticas

BIBLIOGRAFÍA

[1] Sergey Anfinogenov, Valery I. Korzhik, and Guillermo Morales-Luna. Robust digital watermarking system for still images. pages 685–689, 2011.

[2] D. Artz. Digital steganography: hiding data within data. *Internet Computing, IEEE*, 5(3): 75 –80, 5 2001. ISSN 1089-7801. doi: 10.1109/4236.935180.

[3] G. Boccaccio and P.N. Villarosa. *Amorosa visione*. Dalla tipografia di G. Assenzio, 1818. URL http://books.google.com.mx/books?id=4hpJAAAAMAAJ.

[4] J.T. Brassil, S. Low, and N.F. Maxemchuk. Copyright protection for the electronic distribution of text documents. *Proceedings of the IEEE*, 87(7):1181 –1196, 7 1999. ISSN 0018-9219. doi: 10.1109/5.771071.

[5] M. Brookes. The matrix reference manual, 2011. URL http://www.ee.imperial.ac.uk/hp/staff/dmb/matrix/intro.html.

[6] Bugzilla.org. Bug 44488, 1 2013. URL https://bugs.freedesktop.org/show_bug.cgi?id=44488.

[7] Christian Cachin. An information-theoretic model for steganography. In *Information Hiding*, pages 306–318. Springer, 1998.

[8] R.G. Cárdenas. Esteganografía lingüstica. 2004.

[9] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital Watermarking and Steganography*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2008. ISBN 0123725852, 9780080555805, 9780123725851.

[10] A. De Selincourt. *Herodotus. The histories*. Penguin Books, 1996.

[11] J David Eisenberg. *OASIS OpenDocument Essentials*. LuLu. com, 2006.

[12] P. Forczmański and M. Węgrzyn. Open virtual steganographic laboratory. *Elektronika: konstrukcje, technologie, zastosowania*, 50(11):60–65, 2009.

[13] T. Furon. A constructive and unifying framework for zero-bit watermarking. *Information Forensics and Security, IEEE Transactions on*, 2(2):149–163, 2007.

[14] L. Holt, BG Maufe, and A. Wiener. Encoded marking of a recording signal. *UK Patent GB A*, 2196167:1988, 1988.

[15] Manuel García Horta. Autenticación de documentos digitales usando la técnica de marca de agua, 2012.

[16] O. Hosam. Stereo image watermarking using random least significant bit and arnold transform. *International Journal of Research and Reviews in Computer Science (IJRRCS)*, 3(4), 2012.

[17] D. Hunter. *Handmade Paper and Its Water-marks: A Bibliography*. Technical Association of the Pulp and Paper Industry, Committee on Bibliography, 1967.

[18] Google Inc. The Go programming language, 4 2013. URL http://golang.org.

[19] Google Inc. Package big, 4 2013. URL http://golang.org/pkg/math/big.

[20] Adobe Systems Incorporated. *PDF Reference, Sixth edition, version 1.23*, 2006. URL http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

[21] Donald E Kauth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Reading MA, Addison-Wesley, 722p, 1973.

[22] Donald E. Knuth. Computer Programming as an Art. *Communications of the ACM*, 17 (12):667–673, December 1974.

[23] Donald E Knuth. *The Art of Computer Programming, Volume 2: seminumerical algorithms*. Reading, Mass.: Addison-Wesley, 1981.

[24] Sin-Joo Lee and Sung-Hwan Jung. A survey of watermarking techniques applied to multimedia. 1:272 –277 vol.1, 2001. doi: 10.1109/ISIE.2001.931796.

[25] Z. Liu. New trends and challenges in digital watermarking technology: Application for printed materials. *Multimedia Watermarking Techniques and Applications*, pages 289–305, 2006.

[26] Ueli M Maurer. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms. In *Advances in Cryptology-CRYPTO'94*, pages 271–281. Springer, 1994.

[27] Guillermo Morales-Luna. Repaso de permutaciones, 2009. URL http://cs.cinvestav.mx/~gmorales/Biberstein/fvd/node19.html.

[28] Uche Ogbuji. Introducing OpenDocument, 7 2008. URL http://www.ibm.com/developerworks/xml/tutorials/x-odfintro/index.html.

[29] J.J.K. O'Ruanaidh and T. Pun. Rotation, scale and translation invariant digital image watermarking. 1:536–539, 1997.

[30] Robert Sedgewick. Permutation generation methods. *ACM Computing Surveys (CSUR)*, 9(2):137–164, 1977.

[31] Gustavus J Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer, 1984.

[32] Steven Skiena. Implementing discrete mathematics: combinatorics and graph theory with mathematica. 1991.

[33] OASIS Standard. *Open Document Format for Office Applications (OpenDocument) Version 1.2*, 9 2011. URL http://docs.oasis-open.org/office/v1.2/os/OpenDocument-v1.2-os.html.

[34] W. Szepanski. A signal theoretic method for creating forgery-proof documents for automatic verification. In *Carnahan Conf. on Crime Countermeasures, Lexington, KY*, pages 101–109, 1979.

[35] Andreas Westfeld. F5—A steganographic algorithm. 2137:289–302, 2001. URL http://dx.doi.org/10.1007/3-540-45496-9_21.

[36] B Witzschel and M Cantor. *Zeitschrift für Mathematik und Physik*, volume 4. BG Teubner, 1859.

[37] Wolfram. Wolfram mathematica, 1 2013. URL http://www.wolfram.com/mathematica.

[38] Wolfram. Permutations, 3 2013. URL http://reference.wolfram.com/mathematica/ref/Permutations.html.

[39] Y. Xin, S. Liao, and M. Pawlak. Circularly orthogonal moments for geometrically robust image watermarking. *Pattern Recognition*, 40(12):3740–3752, 2007.

[40] B. Zhu, Jiankang Wu, and M.S Kankanhalli. Render sequence encoding for document protection. *Multimedia, IEEE Transactions on*, 9(1):16–24, 2007. ISSN 1520-9210. doi: 10.1109/TMM.2006.886334.