UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TESIS PROFESIONAL

APLICACIÓN DIDÁCTICA PARA MÉTODOS DE CIFRADO CLÁSICOS EN LA PLATAFORMA iOS

QUE PARA OBTENER EL TÍTULO DE **INGENIERO EN COMPUTACIÓN**

PRESENTAN:

JUAN JOSÉ ARREOLA SIMÓN

NAYELY VERGARA ORTEGA

DIRECTORA DE TESIS:
M.C MA. JAQUELINA LÓPEZ BARRIENTOS

CIUDAD UNIVERSITARIA 11/09/13

Panorama General

Durante siglos, los seres humanos han tenido la necesidad de comunicarse, utilizando distintos métodos para poder realizarlo. Algunas veces se considera privada dicha información, por lo que resulta relevante ocultarla con el fin de mantenerla a salvo de intrusos que posiblemente pudieran hacer mal uso de ella, o que estén deseosos de conocer lo desconocido. Desde hace mucho tiempo la humanidad ha protegido cuidadosamente sus conocimientos ya que la información es poder y el poder ofrece ventajas sobre los hombres y las sociedades. Sun Tzu en *El arte de la guerra* y Nicolás Maquiavelo en *El Príncipe* mencionaron la importancia de la información y de cómo este conocimiento puede ayudar en la toma de decisiones para estar siempre un paso adelante del adversario.

En la antigüedad existían bibliotecas donde se podía resguardar la información para trasmitirla y evitar que personas no autorizadas la obtuvieran, dando así las primeras muestras de protección de la información. Con el paso del tiempo, se hace cada vez más importante la protección de la información, ya que con el avance de la tecnología, los medios por los cuales se transmite la información están cada vez más al alcance de cualquiera, así pues, el cuidar de la información se ha vuelto una tarea crucial para la sociedad en general. Así es como surge el concepto de Seguridad de la Información, que comprende diversos aspectos entre ellos la disponibilidad, comunicación, identificación de problemas, análisis de riesgos, la integridad, confidencialidad y las medidas preventivas que toma un individuo u organización, con la ayuda de herramientas tecnológicas para la protección de la información.

La reducción o eliminación de los riesgos asociados a cierta información es el objetivo de la Seguridad de la Información y la Seguridad Informática, e involucra la implementación de estrategias que protejan los procesos en donde la información es de gran importancia para el desarrollo de las actividades u objetivos de la organización. Estas estrategias deben tener como punto primordial el establecimiento de políticas, controles de seguridad, tecnologías y procedimientos para detectar amenazas que puedan explotar vulnerabilidades y que pongan en riesgo la confidencialidad de dicha información, es decir, que ayuden a proteger y salvaguardar tanto la información como los sistemas que la almacenan y administran.

Durante muchos años la Seguridad de la Información ha tenido como servicios de seguridad básicos a la confidencialidad, la integridad y la disponibilidad (CIA, por sus siglas en inglés, Confidentiality, Integrity, Availability). La Seguridad de la Información es un tema relevante para los gobiernos, entidades militares, instituciones financieras, hospitales y empresas privadas, pues amasan una gran cantidad de información confi-

dencial sobre sus empleados, clientes, productos, investigación y su situación financiera. La mayoría de esta información es capturada, procesada y almacenada por medio de computadoras y transmitida a través de redes hacia otras computadoras. Actualmente vivimos en un mundo donde el proceso de la transferencia de la información, su almacenamiento y lectura juegan un papel muy importante en la sociedad, el avance de las tecnologías de la información y de las redes de comunicaciones han sido fundamentales para cumplir con el objetivo de proteger a la información.

El incremento de las aplicaciones electrónicas ha hecho que la Seguridad Informática implemente diversas herramientas con el uso de diferentes técnicas criptográficas para tratar de resolver el problema que éstas aplicaciones han ido acumulando con el paso del tiempo, entre los que se encuentran principalmente el robo de información, suplantación de identidad, y ataques de denegación de servicios.

Ya que con cada nuevo avance en el campo de la Criptografía, se realizan esfuerzos similares para tratar de vulnerar los nuevos sistemas y métodos que son integrados a los procesos de seguridad actuales, se vuelve esencial en este proceso de evolución de la Criptografía la enseñanza de la misma a las nuevas generaciones de ingenieros para que adquieran los conocimientos y habilidades necesarias para diseñar, desarrollar y aplicar herramientas criptográficas para que sean capaces de aportar avances en este esfuerzo para asegurar la información.

Actualmente los desarrollos didácticos en el área de la Criptografía en los que el usuario tiene un alto nivel de interacción con los mecanismos de las técnicas de cifrado son escasos. Debido a esta problemática surge la necesidad de crear herramientas que apoyen en el proceso de aprendizaje realizando así un avance para el desarrollo de las tecnologías que nos ayudan a proteger la información.

Las nuevas tecnologías que nos permiten participar de forma cada vez más interactiva con los dispositivos electrónicos resultan una herramienta de gran ayuda para este proceso de aprendizaje; el incremento en la capacidad de procesamiento gráfico y el diseño de interfaces de usuario más amistosas han impulsado la creación de dispositivos y plataformas que resultan ideales para acercar este conocimiento y apoyar en la enseñanza de los nuevos especialistas en seguridad.

Objetivos

Diseñar e implementar una aplicación didáctica que permita interactuar en la mecánica de los métodos de cifrado clásicos en la plataforma iOS, con el propósito de mejorar la enseñanza a futuras generaciones que se encuentren involucradas en la seguridad, también para preparar recursos humanos mejor capacitados en todos los ámbitos relacionados con la Seguridad Informática.

Además es necesario impulsar, fomentar y crear conciencia de la importancia que tiene esta área de conocimiento en prácticamente todas las actividades de la sociedad.

Metodología de Trabajo

El trabajo de tesis fue realizado a través de la investigación y recopilación de información.

- Investigación de algoritmos de cifrado clásicos
- Investigación de aplicaciones relacionadas con la seguridad en iOS

Las fuentes de información se obtuvieron a partir de:

- Investigación bibliográfica
- Buscadores de Internet

Dentro de la actividad de desarrollo de la aplicación, se destaca:

- Análisis de factibilidad
- Elaboración de requerimientos
- Diseño
- Implementación
- Pruebas

Introducción

En el presente trabajo se describen las etapas de elaboración de la aplicación didáctica para métodos de cifrado llamada Cipher.

Comenzamos con las características de las herramientas que ocuparemos para realizar dicha aplicación como lo son: dispositivos electrónicos en los cuales se ejecutará (smartphones y tablets), tomando en cuenta el gran crecimiento que están teniendo estas tecnologías, lenguaje de programación para aplicaciones móviles, el framework Cocoa Touch y su entorno de desarrollo IDE (Integrated Development Environment).

En el capítulo 2 se describen los métodos antiguos de cifrado más importantes, así como una breve reseña histórica, que nos permitirá ubicarlos en la época en la cual se desarrollaron, teniendo en cuenta las condiciones sociales y políticas en las cuales fueron usados, el propósito que tenían durante periodos de guerra o simplemente para ocultar información. Se describirá su funcionamiento tanto en el proceso de cifrado como el de descifrado, implementando algunos ejemplos para poder comprender de una manera más sencilla el comportamiento de cada algoritmo y que así el usuario pueda familiarizarse con la aplicación y que su interacción con ella sea evidente.

El capítulo 3 detalla el análisis de requerimientos , teniendo como beneficio principal la mejor comprensión de cada método. Se declaran las funciones del producto, las restricciones de diseño, atributos del sistema y los requisitos de rendimiento.

El siguiente capítulo llamado Implementación describe el funcionamiento de las clases de Modelo más sobresalientes de la aplicación de acuerdo a la arquitectura Modelo Vista Controlador, se exponen diagramas uml para mejor percepción de la estructura del programa y se presentan partes de código con su correspondiente explicación.

El último capítulo muestra los resultados de las pruebas lógicas realizadas a cada método de cifrado y descifrado para cada algoritmo, se demostró su correcto funcionamiento con las herramientas de Xcode y el framework SenTestingKit.

Índice general

Índice de figuras

Capítulo 1

Marco Teórico

En esta primera parte se presenta el panorama actual del mercado de los teléfonos inteligentes, las principales características del sistema operativo que usaremos para desarrollar la aplicación, así como el conjunto de tecnologías alrededor de éste, como son: Objective-C, el lenguaje de programación que se utiliza para el desarrollo en iOS, OpenGL ES 2.0, una especificación utilizada ampliamente para la creación de programas con gráficos 2D y 3D y el patrón de diseño Model View Controller.

También se hace una descripción general de las herramientas necesarias para el desarrollo en iOS, como son Xcode y el conjunto de utilidades que ayudan a la producción de aplicaciones.

Y por último se muestra una pequeña reseña de Enigmatium, una aplicación en iOS que ejemplifica algunos métodos de cifrado antiguos.

1.1. Teléfonos Inteligentes (Smartphones)

Los smartphones son teléfonos móviles con capacidades más avanzadas que un teléfono convencional, como su capacidad de procesamiento y su conectividad, así como una gran cantidad de funciones que incluyen reproductores de música, cámaras digitales, navegación GPS, pantallas TouchScreen y navegadores de Internet, entre otras.

Una de las principales diferencias entre un Smartphone y un teléfono convencional es la existencia de APIs (Application Programming Interfaces), interfaces de programación de aplicaciones, que permiten el desarrollo de aplicaciones por parte de terceros. Esta característica permite aprovechar de una manera más sencilla las capacidades de hardware del dispositivo y nos proporciona las herramientas necesarias para crear aplicaciones que pueden ser usadas en una gran cantidad de actividades cotidianas como negocios, entretenimiento, medicina, música y educación, entre otras.

Actualmente el mercado de los smartphones sigue creciendo en todo el mundo. Según un informe de la consultora Gartner, las ventas mundiales a usuarios finales llegaron a 207.7 millones de unidades en el año 2012, un 38.3 por ciento más que en el año 2011, mientras que la de los celulares tradicionales bajó un 19.3 por ciento. Samsung cerró 2012 como primer fabricante con una cuota del mercado del 22 por ciento, 2.3 puntos más que el año anterior y 384.6 millones de unidades vendidas, seguido de Nokia con un 19.1 por ciento y Apple con el 7.5 por ciento, 2.5 más que en el 2011 y 130.13 millones de unidades. Por sistemas operativos, Android fue el primero con el 69.7 por ciento del mercado, seguido de iOS con una cuota de 20.9 por ciento. [1]

1.2. Tablets

Una tablet es un dispositivo que tiene prestaciones muy similares a las de una computadora pero que se presenta en una sola pieza, una pantalla táctil utilizada como dispositivo primario de entrada. Cuentan con un diseño plano, ligero y compacto, lo que revoluciona el concepto de movilidad.

Está más orientada a multimedia, lectura de contenidos y a la navegación web que a usos profesionales. Entre sus ventajas destaca la facilidad de uso (ya que no es necesario ocupar un teclado o ratón), peso ligero, mayor portabilidad y aumento en la duración

[1] `http://www.gartner.com/newsroom/id/2335616` (última consulta en febrero de 2013)

de la bateria. Las nuevas tablets cuentan con tecnología 3G que permite la transmisión de datos y voz, como si fuera un smartphone.

Debido a estas características y a su gran aceptación en el mercado seleccionamos a la Tablet como dispositivo para el desarrollo de la aplicación.

Según un informe de IDC [2] (International Data Corporation) que es consultoría especializada en estudios de mercado, durante los últimos tres meses del año 2012, las ventas de tablets superaron los 52 millones de unidades, reflejando un crecimiento del 75.3 por ciento sobre el año anterior. Apple cerró el 2012 como primer lugar con 23 millones de unidades vendidas, su máximo récord hasta la fecha, Samsung ocupa el segundo lugar con 7.9 millones de tablets distribuidas, seguido de Amazon con 6 millones de unidades, pero Asus, fabricante de Nexus, registra el mayor crecimiento de ventas, 402 por ciento que representa el 6 por ciento del mercado. Véase figura 1.1.

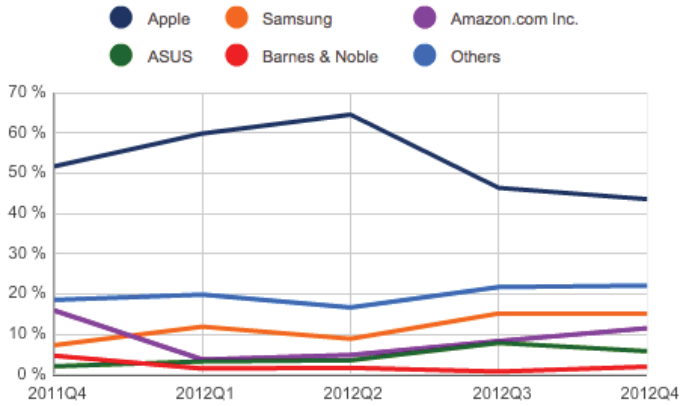


Figura 1.1: Vendedores de Tablets, cuarto trimestre de 2012(2012) En: `http://www.idc.com/getdoc.jsp?containerId=prUS23926713#.UWsUUoJeY1L`

1.3. iOS

En la figura anterior podemos observar que la tablet más vendida es la iPad de Apple, por esta razón seleccionamos el sistema operativo iOS para el desarrollo de la aplicación.

[2] `http://www.idc.com/getdoc.jsp?containerId=prUS23926713#.UWsUUoJeY1L` (última consulta en febrero de 2013)

iOS (anteriormente denominado iPhone OS) es un sistema operativo móvil de Apple. Originalmente desarrollado para el iPhone, siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV.

Cocoa Touch es el framework de programación encargado de controlar la interacción con el usuario en iOS y está orientado a gestos multitáctiles, los cuales son movimientos específicos realizados con los dedos sobre la pantalla, por ejemplo cuando el usuario arrastra sobre la pantalla para desplazar una lista, pellizca para contraer una imagen o toca un botón para activarlo. La interfaz multitáctil le da al usuario una sensación de conexión inmediata con su dispositivo y mejora su sensación de manipulación directa de los objetos en pantalla.

iOS se deriva de Mac OS X, que a su vez está basado en Darwin BSD, y por lo tanto es un sistema operativo Unix.

1.3.1. Características clave en iOS 6.1

La versión actual de iOS es 6.1 y algunas de sus características más importantes son [3]:

iCloud Storage APIs

Estas APIs permiten a la aplicación escribir documentos de usuarios y datos en una locación central y acceder a esos documentos desde todos los dispositivos iOS y computadoras del usuario; esto significa que el usuario puede ver o editar sus archivos desde cualquier dispositivo sin la necesidad de sincronizarlos o transferirlos explícitamente.

Automatic Reference Counting

Automatic Reference Counting (ARC) es una característica a nivel del compilador que simplifica el proceso de administrar el ciclo de vida de los objetos en Objective-C, en lugar de que el programador tenga que recordar cuando retener o liberar los recursos, ARC evalúa los requerimientos del ciclo de vida del objeto y automáticamente inserta los métodos apropiados en tiempo de compilación.

[3] `https://developer.apple.com/library/ios/#releasenotes/General/RN-iOSSDK-6_1/` (última consulta en enero de 2013)

Storyboards

Los Storyboards son una nueva forma de definir la interfaz de usuario de la aplicación. En el pasado se utilizaban los archivos *nib* para definir la interfaz de usuario. Un archivo Storyboard captura la interfaz completa en un solo lugar y permite definir los controladores de vista y las transiciones entre ellos. Como resultado, los Storyboards capturan el flujo de toda la interfaz de usuario adicionalmente a los contenidos que se presenten. Las aplicaciones pueden usar un solo archivo Storyboard para almacenar todos los controladores y vistas. En tiempo de construcción, el Interface Builder toma el contenido del Storyboard y lo divide en piezas más pequeñas que pueden ser usadas individualmente para un mejor desempeño.

GLKit Framework

Desde la versión 5.0 de iOS se añadieron nuevos Frameworks entre ellos el GLKit Framework que contiene un conjunto (set) de clases útiles basadas en Objective-C que simplifican el esfuerzo necesario para crear una aplicación OpenGL ES 2.0. GLKit provee soporte para cuatro áreas clave del desarrollo de aplicaciones:

1. Las clases GLKView y GLKViewController proveen una implementación estándar de una vista habilitada para OpenGL ES y el ciclo de renderizado asociado. La vista administra el objeto framebuffer subyacente en favor de la aplicación; la aplicación solamente dibuja en ella.

2. La clase GLKTextureLoader provee rutinas de carga y conversión de imágenes en la aplicación, permitiéndole cargar imágenes de textura automáticamente al contexto. Puede cargar texturas síncrona y asíncronamente. Cuando se cargan texturas asíncronamente, la aplicación provee un bloque manejador que se llama cuando la textura se carga en el contexto.

3. EL GLKit framework provee implementaciones de vector, matriz, y cuaterniones así como también operaciones de pila de matrices para tener la misma funcionalidad encontrada en OpenGL ES 1.1.

4. Las clases GLKBaseEffect, GLKSkyboxEffect y GLKReflectionMapEffect proveen shaders configurables que implementan operaciones gráficas comunes. En particular, la clase GLKBaseEffect implementa el modelo de iluminación y material encontrado en la especificación OpenGL ES 1.1, simplificando el esfuerzo requerido para migrar una aplicación de OpenGL ES 1.1 a OpenGL ES 2.0.

1.4. Objective-C

Objective-C es un lenguaje de programación orientado a objetos creado como un superset de C pero que implementase un modelo de objetos parecido al de Smalltalk. Originalmente fue creado por Brad Cox y la corporación Stepstone en 1980. En 1988 fue adoptado como lenguaje de programación de NEXTSTEP y en 1992 fue liberado bajo licencia GPL para el compilador GCC (GNU Compiler Collection). Actualmente se usa como lenguaje principal de programación en Mac OS X y GNUstep.

Objective-C fue creado principalmente por Brad Cox y Tom Love a inicios de los 80's en su compañía Stepstone. Ambos fueron iniciados en Smalltalk mientras estaban en el Programming Technology Center de ITT en 1981. Cox se vio interesado en los problemas de reutilización en el desarrollo de software. Se dio cuenta de que un lenguaje como Smalltalk sería imprescindible en la construcción de entornos de desarrollo potentes para los desarrolladores en ITI Corporation. Cox empezó a modificar el compilador de C para agregar algunas de las capacidades de Smalltalk. Pronto tuvo una extensión para añadir la programación orientada a objetos a C, a la cual llamó OOPC (Object-Oriented Programming in C). Para ilustrar que se hizo un progreso real, Cox mostró que para hacer componentes de software verdaderamente intercambiables sólo se necesitaban unos pequeños cambios en las herramientas existentes. Específicamente, estas necesitaban soportar objetos de manera flexible, venir con un conjunto de bibliotecas que fueran utilizables, y permitir que el código (y cualquier recurso necesitado por el código) pudiera ser empaquetado en un formato multiplataforma.

Cox y Love luego fundaron una nueva empresa, Productivity Products International (PPI), para comercializar su producto, el cual era un compilador de Objective-C con un conjunto de bibliotecas potentes. En 1986, Cox publicó la principal descripción de Objective-C en su forma original en el libro Object-Oriented Programming, An Evolutionary Approach.

Objective-C consiste en una capa muy fina situada por encima de C, y además es un estricto superset de C. Debido a esto, es posible compilar cualquier programa escrito en C con un compilador de Objective-C, y también puede incluir libremente código en C dentro de una clase de Objective-C.

La sintaxis de objetos de Objective-C deriva de Smalltalk. Toda la sintaxis para las operaciones no orientadas a objetos (incluyendo variables primitivas, pre-procesamiento, expresiones, declaración de funciones y llamadas a funciones) son idénticas a las de C, mientras que la sintaxis para las características orientadas a objetos es una implementación similar a la mensajería de Smalltalk.

1.5. OpenGL ES 2.0

OpenGL ES es una API libre, multiplataforma para una funcionalidad completa de gráficos 2D y 3D en sistemas embebidos, incluyendo consolas, teléfonos, y vehículos, consiste en un subset bien definido de la versión para desktops de OpenGL, creando una poderosa y flexible interfaz de bajo nivel entre el software y el hardware dedicado a gráficos.

OpenGL ES incluye perfiles para sistemas de punto flotante y punto fijo y la especificación EGL para portabilidad nativa en sistemas basados en ventanas. OpenGL ES 2.X es definido en relación con OpenGL 2.0 y hace énfasis en un pipeline gráfico programable 3D con la habilidad de crear objetos tipo programa y de escribir shaders de vértices y fragmentos en el OpenGL ES Shading Language. [4]

1.6. Git

Git es un software libre que se encarga del control de versiones basado en BitKeeper y en Monotone, diseñado por Linus Torvalds y escrito en C.

Entre las características más importantes de Git se encuentran:

- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos.
- Git está escrito en C, reduciendo la carga de las rutinas asociadas con lenguajes de alto nivel, la rapidez y el rendimiento han sido el objetivo principal de Git desde su creación.
- Es distribuido, lo que significa que en lugar de tener solamente el código fuente actual, se clona todo el repositorio, así, cada usuario tiene la historia completa del proyecto, cada una de estas copias pueden ser usadas o reemplazadas en el servidor si existe una corrupción del repositorio principal.
- Garantía de los datos, el modelo de datos que usa Git asegura la integridad de cada bit del proyecto: de cada archivo y commit se hace un checksum que garantiza que los datos sean los correctos.

[4] `http://www.khronos.org/opengles/` (última consulta en mayo de 2012)

Git es liberado bajo la licencia de código abierto GPLv2. Esto significa que se puede examinar el código fuente en cualquier momento o contribuir al proyecto mismo.

Bajo licencia de software de Git GPLv2, se puede:

- Usar Git en proyectos abiertos o de propiedad de forma gratuita.
- Descargar, inspeccionar y modificar el código fuente de Git.

1.7. Modelo Vista Controlador (MVC)

Modelo Vista Controlador es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

- Modelo: esta es la representación específica de la información con la cual el sistema opera. Por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.
- Vista: este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- Controlador: este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

Muchos de los sistemas informáticos utilizan un Sistema de Gestión de Base de Datos que corresponde a la parte del Modelo en la arquitectura MVC. La unión entre capa de presentación y capa de negocio conocido en el paradigma de la Programación por capas representaría la integración entre Vista y su correspondiente Controlador de eventos y acceso a datos. MVC no pretende discriminar entre capa de negocio y capa de presentación pero si pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el Controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

1.8. Xcode

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con Mac OS X. Xcode trabaja conjuntamente con Interface Builder, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo, pero no limitado a Cocoa, Carbón y Java. En Xcode 4 se incluye el compilador Apple LLVM basado en el proyecto de código libre LLVM.org, el cual esta optimizado para iPhone, iPad y la Mac. Según Apple, el compilador LLVM compila código el doble de rápido que GCC y produce aplicaciones que también funcionan más rápido. Fue construido desde cero como un set de librerías optimizadas, fáciles de extender, fáciles de optimizar y diseñado para arquitecturas de chips modernos.

En Xcode 4 la pila completa del compilador Apple LLVM, desde el parser hasta el optimizador de código, tienen gran soporte para C, Objective-C, y C++. El realce de código, autocompletar, y muchas otras características son manejadas por el parser del LLVM. Si el compilador conoce un símbolo, también Xcode lo conoce. C, C++ y Objective-C son exactamente conocidos en tiempo de edición, exactamente como lo son en tiempo de construcción.

Además de todo esto, Xcode 4 cuenta con varios instrumentos que ayudan en el proceso de desarrollo de aplicaciones que incluyen instrumentos para medir el desempeño de gráficos OpenGL ES; un monitor de manejo de memoria que puede detectar aumentos de memoria no intencionales; o Time Profiler en iOS para recolectar muestras con una sobrecarga mínima y un rastreador de sistema para ver como todos los procesos interactúan.

1.9. Enigmatium

Enigmatium es una aplicación presentada como proyecto final de carrera desarrollado por Judith Medina González y Patricia López Peña de la Universidad Carlos III de Madrid en Agosto de 2011, dicha aplicación es utilizada como recurso docente para

las asignaturas que tienen como contenido temático algoritmos criptográficos, la cual se divide en dos secciones: Mini juegos: esta sección se divide en dos más, las cuales son jugar y crear. En la primera se pueden encontrar juegos relacionados con los algoritmos como César, Playfair, Vigenére, Diffie - Hellman y Scítala. Además el usuario puede crear sus propios juegos e intercambiarlos con otros usuarios de Enigmatium mediante bluetooth.

En el modo aventura, el usuario tendrá que resolver un crimen, recogiendo pistas por la ciudad de Leganés, en Madrid (España) para lo cual tendrá que superar diferentes pruebas basadas en los mini juegos.

Capítulo 2

Técnicas clásicas de cifrado

2.1. Introducción

Las técnicas de cifrado son aquellas que permiten ocultar un mensaje mediante un determinado procedimiento, para que solo las personas autorizadas tengan acceso a dicha información. El procedimiento para cifrar un mensaje es el siguiente:

- El emisor cuenta con el mensaje a cifrar, también llamado mensaje en claro o texto plano.
- El método de cifrado transforma dicho mensaje de manera que sea incomprensible, este procedimiento puede ser auxiliado con una clave. El mensaje resultante recibe el nombre de mensaje cifrado o criptograma.
- Después el receptor al conocer la clave, transforma el criptograma con ayuda del algoritmo de descifrado en el mensaje en claro.

Cuando se utiliza la misma clave para cifrar y descifrar, se le conoce como sistema de cifrado simétrico. Si se utiliza una clave para cifrar y otra distinta para descifrar se le llama sistema de cifrado asimétrico.

Las técnicas clásicas de cifrado fueron usadas desde el inicio de la Criptografía y hasta el año 1948, cuando fue desarrollada la teoría matemática de las comunicaciones

por Claude Shannon, esto implicó que la Criptografía ya no fuera un misterio, sino que se considerara una rama de las matemáticas.

Los sistemas de cifrado clásidos utilizan principalmente las operaciones de sustitución y transposición para modificar el mensaje en claro.

2.1.1. Algoritmos de sustitución

Estos algoritmos se encargan de reemplazar cada carácter del mensaje en claro por otro correspondiente al alfabeto de cifrado. Se pueden clasificar a su vez en dos grupos: monoalfabéticos y polialfabéticos.

Los algoritmos de sustitución monoalfabética utilizan un alfabeto para el cifrado como para el descifrado, además de que el proceso de sustitución se puede realizar de dos maneras distintas:

- Sustitución monográmica: donde el cifrado se efectúa carácter por carácter. Ejemplos de este tipo de sustitución son Polybio y César.

- Sustitución poligrámica: el proceso de cifrado se lleva a cabo por bloques de caracteres. Playfair es un ejemplo de este tipo de sustitución.

Los algoritmos de sustitución polialfabética utilizan varios alfabetos para reemplazar cada carácter del mensaje en claro, este proceso de sustitución puede realizarse de dos maneras distintas:

- Sustitución periódia: es aquella que se repite con frecuencia a intervalos determinados. Ejemplos de este tipo de sustitución son Alberti y Vigenère.

- Sustitución no periódica: se repite a intervalos indeterminados. Enigma es un ejemplo de este tipo de sustitución.

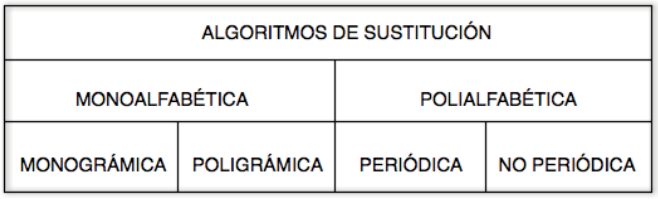


Figura 2.1: Clasificación de los algoritmos de sustitución

2.1.2. Algoritmos de transposición

Estos algoritmos se encargan de cambiar de posición a los elementos que conforman el mensaje en claro siguiendo un esquema bien definido, se basan en el uso de artefactos mecánicos o diseños geométricos y son de cifrado simétrico. Se dividen en inversa, simple, doble, grupos, series, columnas, filas y máscaras rotativas, de entre estos algoritmos sólo nos enfocaremos en máscaras rotativas.

2.2. Scítala

2.2.1. Descripción

La scítala puede considerarse como el primer sistema de criptografía por transposición. En tiempos Romanos, la scítala fue usada para enviar mensajes a través de las líneas enemigas. Aunque la scítala fue documentada por primera vez desde el siglo VII a. de C., no fue hasta 800 años después que Plutarco describió como los generales de Esparta usaban la scítala para enviar mensajes secretos. Cada general o almirante en el ejército de Esparta tenía un bastón y uno idéntico era guardado por el magistrado. Cuando querían enviar un mensaje, envolvían una tira de papiro o piel alrededor del bastón y escribían el mensaje. Cuando la tira era removida del bastón el mensaje se volvía ininteligible y tenía la ventaja de poder ser usada como cinturón o para envolver un paquete y ser transportada fácilmente. Para ser leída correctamente se tenía que envolver de nuevo en un bastón del mismo tamaño.

Figura 2.2: Scítala (2007) En: `http://es.wikipedia.org/wiki/Esc%C3%ADtala`

2.2.2. Funcionamiento

Proceso de cifrado

El mensaje en claro es: "La criptografía es la ciencia encargada del ocultamiento de la información." De acuerdo al largo y ancho del bastón podemos escribir el mensaje, en este caso las filas representan el ancho y las columnas la longitud de la vara.

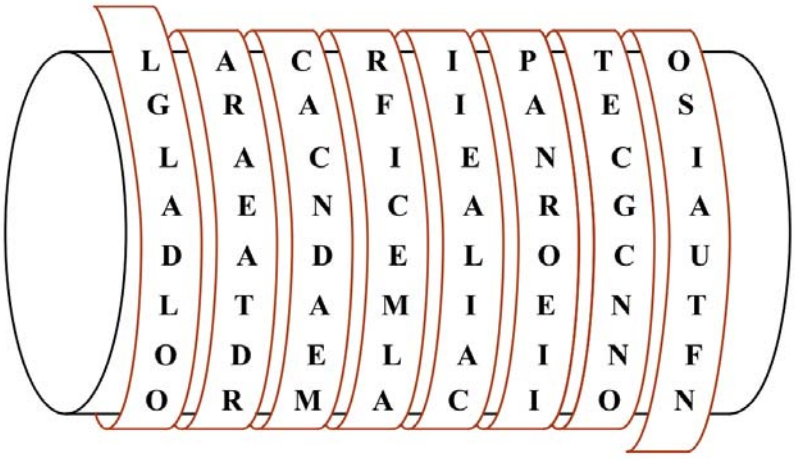


Figura 2.3: Proceso de cifrado de Scítala

Una vez escrito se procede a desenrollar el pedazo de papel, ahora el mensaje cifrado queda de la siguiente forma:

LGLADLOO ARAEATDR CACNDAEM RFICEMLA IIEALIAC PANROEII TECGCNNO OSIAUTFN

Proceso de descifrado

Ahora se tiene el criptograma:

PTOOS REGRI IMRTC MAARI ECFAO RRINN SICSX IPOPX STPOX

Para poder descifrarlo, es necesario conocer el largo y ancho del bastón donde se colocará el papiro. Al hacer varias pruebas con distintos bastones podemos llegar al mensaje en claro.

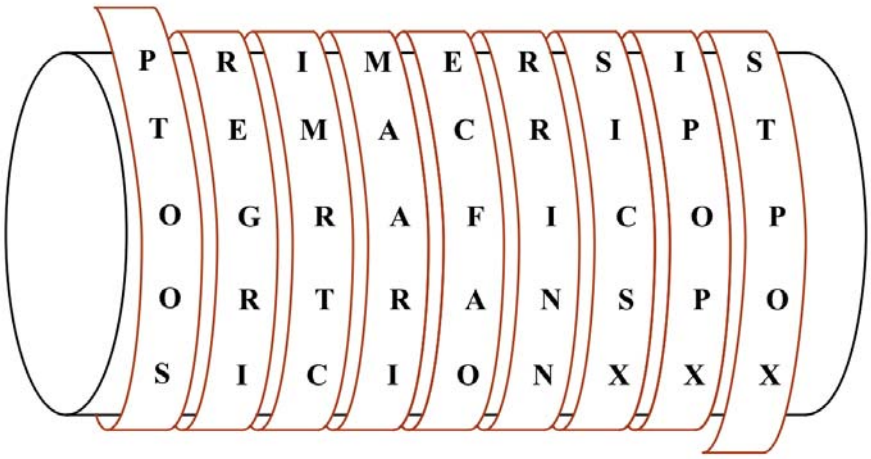


Figura 2.4: Proceso de descifrado de Scítala

El mensaje en claro se constituye por la rotación del bastón, obteniendo las letras de cada fila:

PRIMER SISTEMA CRIPTOGRAFICO POR TRANSPOSICION

2.3. Cuadro cifrador de Polybio

2.3.1. Descripción

Polybio nació en Grecia en el año 200 a. de C. Es considerado uno de los historiadores más prestigiosos de la antigüedad, debido a que es el primero que escribe una historia universal además de vivir en carne propia los acontecimientos políticos y militares de su época. Entre sus principales pensamientos destaca el cuidado de la veracidad, él pensaba que "La verdad, es expuesta por la naturaleza a los hombres como algo supremo en divinidad y poder, tarde o temprano, la verdad prevalecería sobre cualquier oposición" lo que lo llevó a inventar un cuadro de 5x5 en el cual asignaba un cuadro para cada letra del alfabeto, con el paso del tiempo se fue adaptando para poder emplear varios alfabetos como lo son el español y el inglés en dicho cuadro. En la figura 2.5 podemos observar su cifrador adaptado al alfabeto inglés.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Figura 2.5: Cuadro Cifrador de Polybio

2.3.2. Funcionamiento

Se trata de un algoritmo en donde cada letra del alfabeto es reemplazada por las coordenadas de su posición en un cuadrado. Es un caso particular de transposición mono-alfabética.

Proceso de cifrado

Lo primero es establecer el alfabeto que se va a ocupar, en este caso, el alfabeto inglés. Véase figura 2.6.

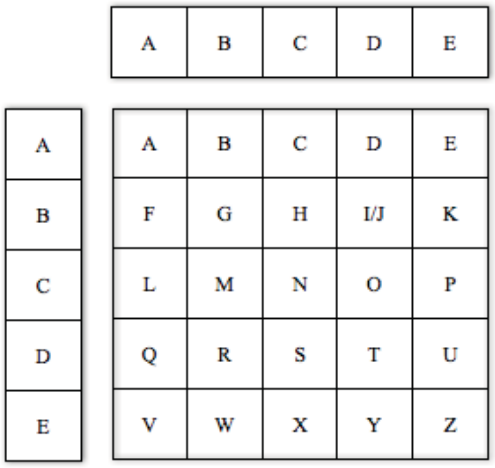


Figura 2.6: Proceso de cifrado de Polybio

El mensaje en claro es: CRIPTOGRAFIA

Para poder cifrarlo se ubican las coordenadas de las letras empezando a cifrar por la fila y después por la columna, quedando entonces:

Mensaje cifrado: AC DB BD CE DD CD BB DB AA BA BD AA

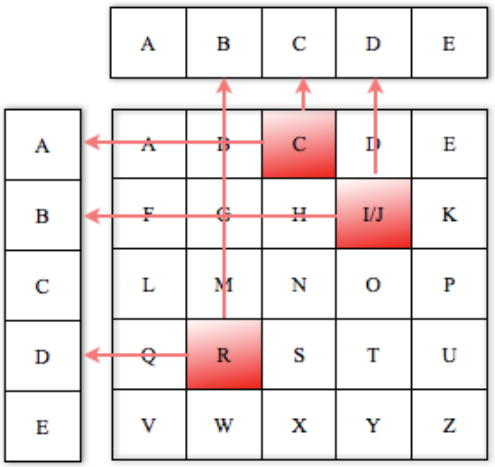


Figura 2.7: Ejemplo de cifrado de Polybio

Proceso de descifrado

Lo primero es establecer el alfabeto que se va a ocupar, en este caso, el alfabeto inglés.

El mensaje cifrado es: AACABBCDDBBDDDCBCD

Para poder descifrarlo se separa el criptograma en pares de letras: AA CA BB CD DB BD DD CB CD

Se toman los dos primeros caracteres, de manera que el primer carácter es ubicado con su similar en la primera fila de la tabla y el segundo con su símil de la primera columna, para poder localizar la intersección de estas letras y encontrar el carácter correspondiente al descifrado.

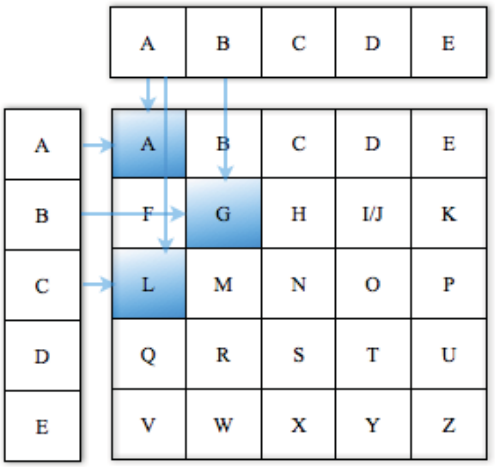


Figura 2.8: Proceso de descifrado de Polybio

Mensaje descifrado: ALGORITMO

2.4. Cifrado del César

2.4.1. Descripción

En el siglo I a.de C. surge el cifrado de César que es una de las técnicas de cifrado más simples y más usadas. Es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. El cifrado César recibe su nombre en honor a Julio César, al cual se le atribuye que utilizó por primera vez este método. No se sabe cuán efectivo resultaba realmente el cifrado César en esa época, pero debió ser razonablemente seguro, ya que pocos enemigos de César habrían sabido leer, y mucho menos podría haber llevado a cabo el criptoanálisis necesario. Asumiendo que el atacante pudiera leer el mensaje, no existen pruebas de la existencia de técnicas para solucionar este tipo de cifrado.

2.4.2. Funcionamiento

Proceso de cifrado

Se utiliza el alfabeto inglés para llevar a cabo este método, en la figura 2.9 se muestra el nuevo alfabeto desplazado 3 posiciones a la izquierda.

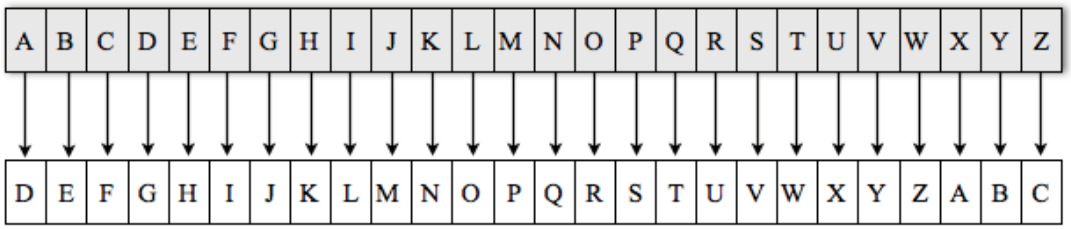


Figura 2.9: Alfabeto desplazado

Mensaje en claro: CIFRADO DE CESAR

Mensaje cifrado: FLIUD GRGHF HVDU

Proceso de descifrado

Para descifrar se realiza el desplazamiento de las letras 3 posiciones a la izquierda.

Mensaje cifrado: FLIUDGR SRU VXVWLWXFLRQ

Mensaje en laro: CIFRADO POR SUSTITUCIÓN

2.5. Disco de Alberti

2.5.1. Descripción

Concibe un sistema polialfabético, esto es, un cifrador que emplea varios alfabetos, saltando de uno a otro cada tres o cuatro palabras. En este sistema tanto el emisor como el destinatario deben ponerse de acuerdo para fijar la posición inicial de dos círculos concéntricos.

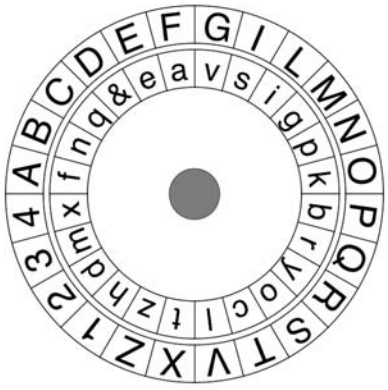


Figura 2.10: Disco de Alberti

En estos círculos o discos, véase figura 2.10, se encuentran 24 celdas en las cuales está grabado un alfabeto latín convencional de 20 caracteres, en el primer disco se encuentran además los números 1, 2, 3 y 4, en el segundo se encuentran el símbolo & y las letras *h, k* y *y*, en este segundo disco las letras pueden estar en cualquier orden. Un disco puede girar con respecto al otro, haciendo un total de 24 alfabetos diferentes para usar en el cifrado, además se puede fijar una rotación de los discos cada cierto tiempo.

2.5.2. Funcionamiento

Proceso de cifrado

Mensaje en claro: CIFRADO DE ALBERTI

Se elije una letra del disco interno que sólo es conocida por el receptor y emisor, en este ejemplo se utiliza la letra g y se hace coincidir con la letra A del disco externo. La posición de los discos es la siguiente:

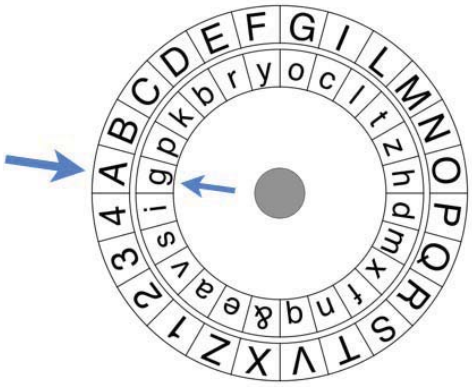


Figura 2.11: Posición inicial

Después se ubica cada uno de los caracteres del mensaje a cifrar en el disco externo de manera que el criptograma correspondiente es el indicado en el disco interno. Lo destacado de esta técnica es que el alfabeto seleccionado para cifrar puede cambiar, de manera que cada determinado número de caracteres se gire el disco interior n posiciones a la izquierda o a la derecha.

La clave es: (Ag, 4, 3der)

Lo que significa que se hace coincidir la letra A con la letra g y cada 4 sustituciones se gira el disco 3 posiciones a la derecha.

Las primeras cuatro letras cifradas son: $kcyx$

En este caso, después de 4 letras se gira el disco tres posiciones a la derecha.

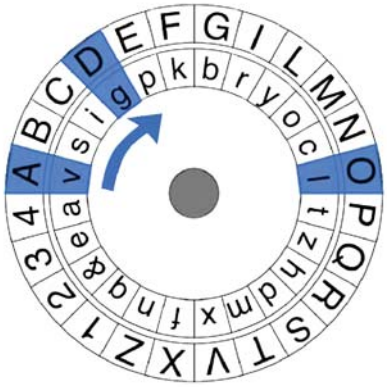


Figura 2.12: Segunda posición del proceso de cifrado

Procedemos al cifrado de las siguientes 4 letras, quedando: *vglg*

Y repetimos el proceso hasta obtener el mensaje cifrado.

Mensaje cifrado: kcyx vglg s&ke eycs

Proceso de descifrado

Se tiene la siguiente clave: (An, 3, 1 izq) con su correspondiente mensaje cifrado: ngsrcgzby

Se hace coincidir la letra conocida por el emisor y receptor n con la letra A del disco externo y se obtienen las primeras letras del mensaje.

Primeras cuatro letras en claro: ALG

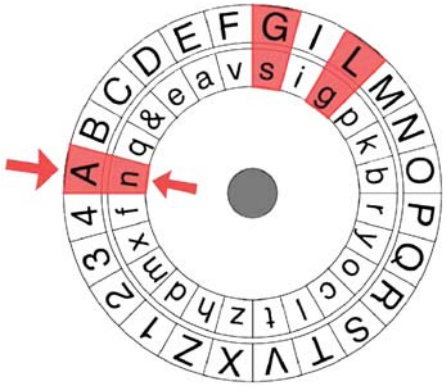


Figura 2.13: Primera posición del proceso de descifrado

Después de estas tres letras se hace girar el disco móvil el número de posiciones acordadas, 1 en este caso, hacia el lado izquierdo y se obtienen las siguientes 3 letras.

Letras en claro: ALG ORI

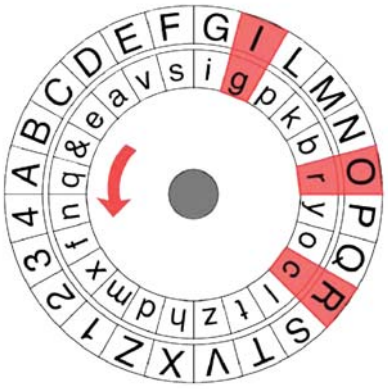


Figura 2.14: Segunda posición del proceso de descifrado

Se repite el mismo proceso hasta obtener el mensaje descifrado.

Mensaje en claro: ALGORITMO

2.6. Cifrado de Vigenère

2.6.1. Descripción

Es un cifrado basado en diferentes series de caracteres o letras formando una matriz cuadrada, llamada tabla de Vigenère, véase figura 2.15. El cifrado de Vigenère es un cifrado de sustitución simple polialfabético. En este algoritmo se ubican los siguientes elementos:

- Columnas: cada elemento correspondiente a la clave.
- Renglones: cada elemento correspondiente al mensaje en claro.
- Celdas: cada elemento que conforma el mensaje cifrado o criptograma.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figura 2.15: Matriz de Vigenère

2.6.2. Funcionamiento

Proceso de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Mensaje en claro (Mcla): CODIGO INDESCIFRABLE

Clave (K): CRIPTO

Lo primero que se tiene que hacer es tomar la primera letra del mensaje en claro y la de la clave, así, se podrá hacer referencia a las columnas y renglones respectivamente. Se hace igual para la segunda letra, y así sucesivamente.

Mcla	C	O	D	I	G	O	I	N	D	E	S	C	I	F	R	A	B	L	E
K	C	R	I	P	T	O	C	R	I	P	T	O	C	R	I	P	T	O	C

Figura 2.16: Ejemplo de proceso de cifrado de Vigenère

Al tener esto, se procede a cifrar tomando las letras como coordenadas en la matriz:

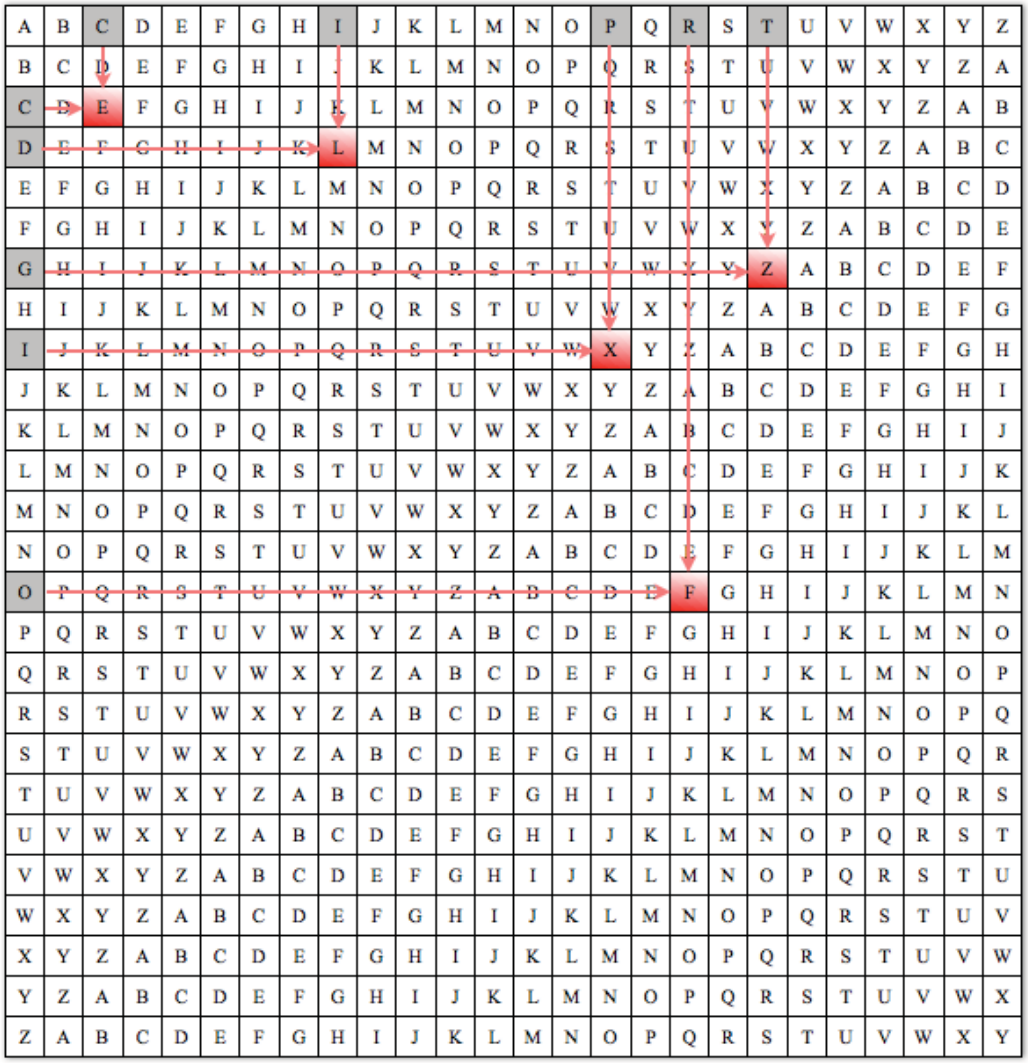


Figura 2.17: Representación de proceso de cifrado de Vigenère

El mensaje cifrado es: EFLXZ CKELT LQKWZ PUZG

Proceso de descifrado

Para llevar a cabo el descifrado, se tiene el mensaje cifrado y la clave con la cual se podrá realizar dicho proceso.

Mensaje cifrado: SRZCX CRAQB VLIFM NMQIU

Clave: ANTIGUO

Al igual que con el proceso de cifrado, se toma la primera letra tanto del mensaje cifrado como de la clave, para poder localizar la primera letra del mensaje en claro, se ubica la letra de la clave *ANTIGUO* en la matriz de Vigenère y en esa misma columna se localiza la letra del mensaje cifrado, con esto su puede ubicar la fila a la cual pertenece y con que letra corresponde. Véase figura 2.18.

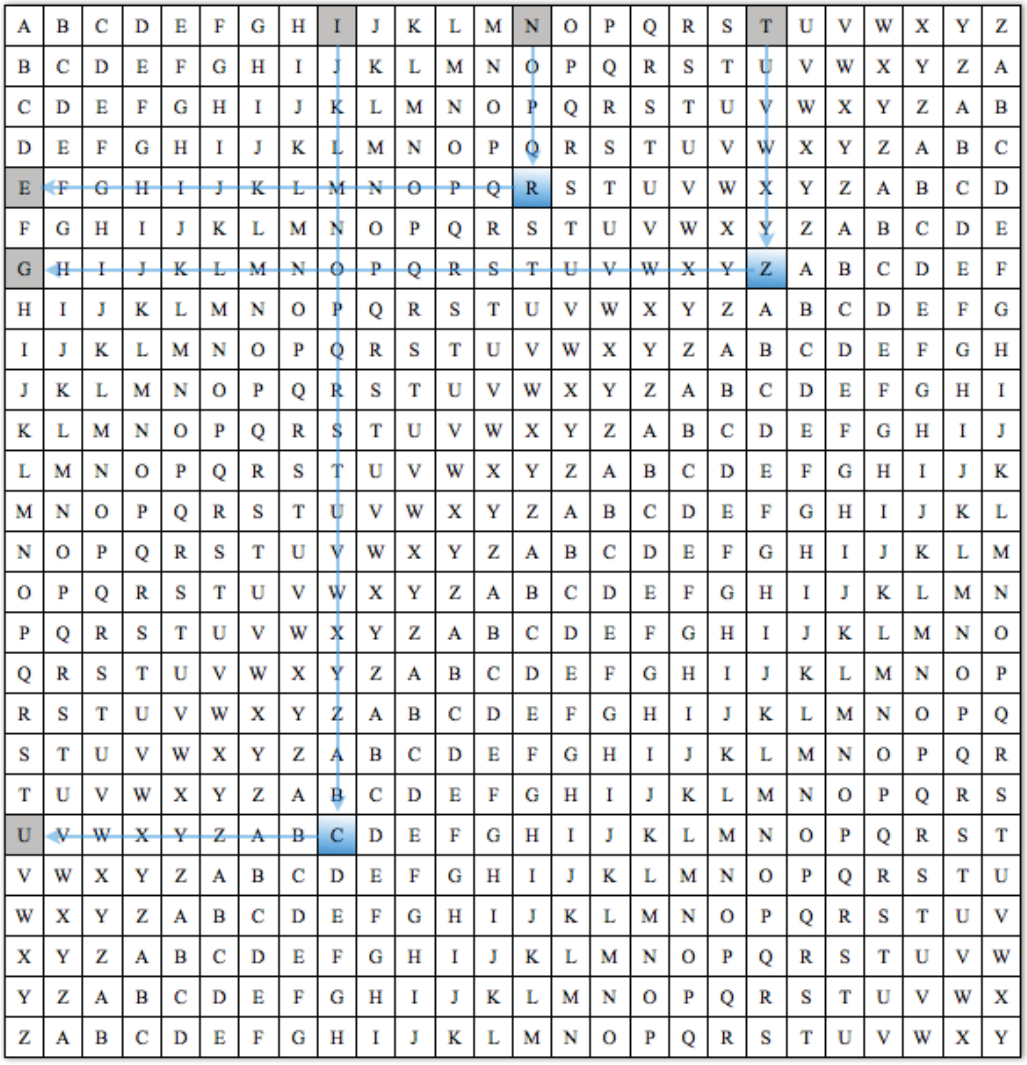


Figura 2.18: Representación de proceso de descifrado de Vigenère

Mensaje en claro: SEGURIDAD INFORMATICA

2.7. Reja de Cardano o Máscaras Rotativas

2.7.1. Descripción

Desarrollado por Girolamo Cardano en 1550, es un sistema basado en una carta o tarjeta perforada, de tal manera que el mensaje en claro se obtenía al colocarlo sobre un determinado texto preconcebido. Dicho sistema en su momento se conoció como la reja de Cardano y que después de un tiempo fue modificándose a máscaras rotativas. Este algoritmo fue utilizado en la primera guerra mundial por los alemanes. El algoritmo consiste en utilizar un par de tarjetas y en cada una de ellas colocar una matriz cuadrada (de igual número de filas que de columnas), una de ellas se utiliza para verter el mensaje junto con otros caracteres que fungen como relleno y cuya función es crear difusión de los carácteres dentro del contenido del mensaje y confusión a un posible criptoanalista; la segunda matriz es la que opera como máscara ya que en ella se marcan las celdas que representan los orificios a través de los cuales se puede ver el mensaje en claro.

2.7.2. Funcionamiento

Proceso de cifrado

En este caso el mensaje a cifrar será: RESGUARDAR Y PROTEGER LA INFORMACIÓN

Se crea la máscara con una matriz de 8x8 de manera que al girarla 4 veces se muestre el mensaje en claro. Tomando como referencia uno de los lados de dicha máscara. Véase figura 2.19.

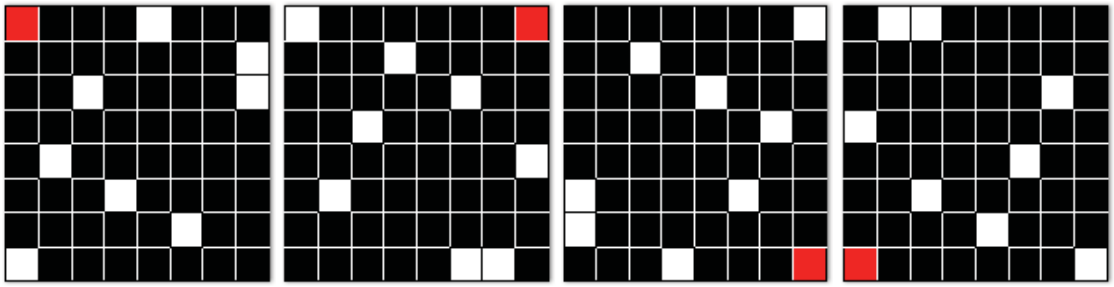


Figura 2.19: Máscaras rotativas

Ahora de acuerdo a la máscara y sus posiciones, en una matriz del mismo tamaño, pondremos nuestro mensaje en claro.

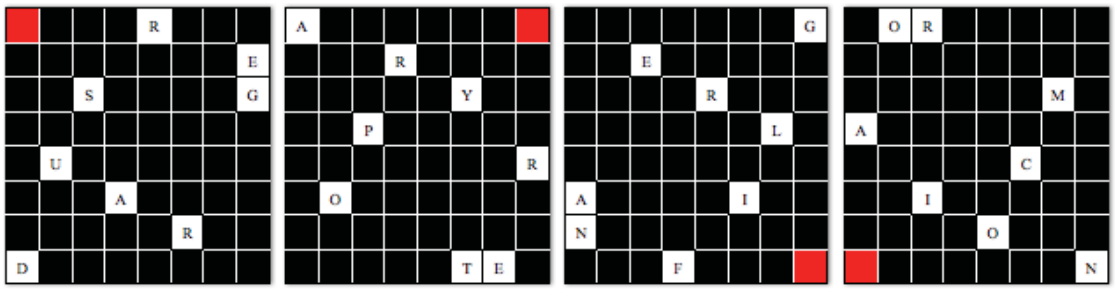


Figura 2.20: Ejemplo de máscaras rotativas

Después se tendrá que rellenar las celdas con letras al azar para que cumpla con su objetivo, esconder el mensaje en claro, según se muestra en la figura 2.21.

A	O	R	A	R	O	S	G
N	C	E	R	R	I	A	E
A	L	S	O	R	Y	M	G
A	T	P	S	H	O	L	A
N	U	N	C	A	C	I	R
A	O	I	A	H	I	Y	U
N	O	M	A	O	R	X	Z
D	I	F	F	I	T	E	N

Figura 2.21: Matriz con mensaje en claro

Proceso de descifrado

Se tiene la matriz con el mensaje cifrado, además de que es necesario contar con la máscara para poder descifrar el mensaje. En este caso, se cuenta con la matriz:

E	J	P	G	O
T	F	G	C	C
R	A	T	A	R
U	A	Z	O	I
N	I	A	X	A

Figura 2.22: Matriz con mensaje cifrado

Se tiene la máscara a utilizar, se superpone sobre la matriz con el mensaje cifrado y dejará ver algunas letras, al realizar un giro de 90 grados, se mostrarán nuevas letras. Véase figura 2.23. Hacer este giro cuatro veces.

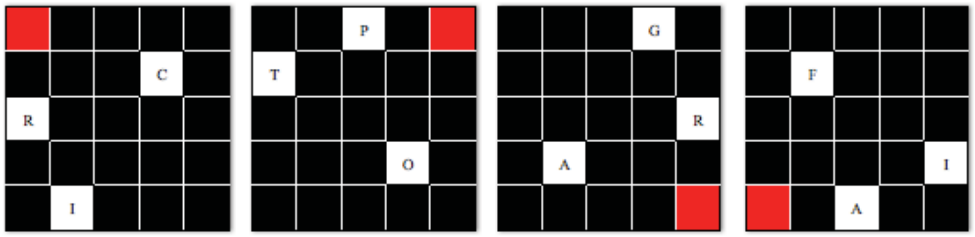


Figura 2.23: Descrifrado con máscara rotativa

Quedando el mensaje en claro: CRIPTOGRAFIA

2.8. Rueda de Jefferson

2.8.1. Descripción

La Rueda de Jefferson fue una de las primeras máquinas criptográficas, elaborada por Thomas Jefferson en 1790. Consistía en 26 ruedas de madera, cada una de ellas tenía una letra del alfabeto en desorden, dichas ruedas estaban colocadas en un eje común formando un cilindro. El orden de las letras en cada rueda era único. Véase figura 2.24.

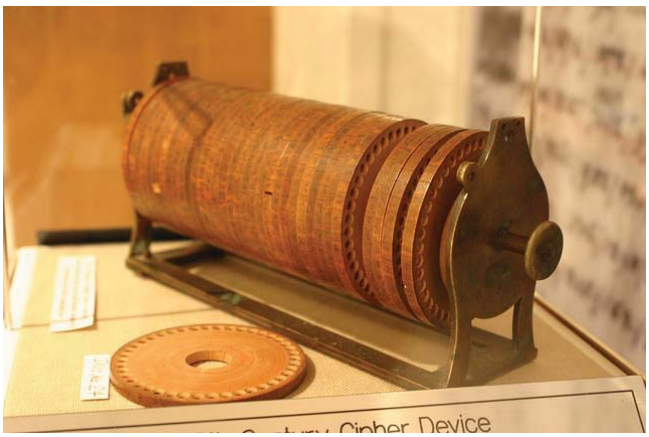


Figura 2.24: Rueda de Jefferson (1980) `http://www.flickr.com/photos/ideonexus/5176168673/`

2.8.2. Funcionamiento

Proceso de cifrado

Para poder cifrar un mensaje es necesario colocar una letra en cada rueda de forma consecutiva, así en una fila se tiene el mensaje en claro y en las otras filas se tiene el mensaje cifrado, de esta manera se puede usar cualquier fila para tener el mensaje cifrado. También es posible intercambiar el lugar de una rueda por otra, en caso de que esto suceda se necesita especificar la posición de los discos. Véase figura 2.25.

N	J	V	H	X	F	D	B	B	X	T	Z	P	V	C	H	Z	O	W	C	V	P	E	F	C	K
O	Q	R	G	Y	Z	C	Q	M	Y	R	F	T	C	G	I	J	R	X	V	Y	U	Z	E	W	D
C	O	N	F	I	D	E	N	C	I	A	L	I	D	A	D	D	E	L	M	E	N	S	A	J	E
Q	Y	S	C	H	R	O	O	Q	H	K	B	D	W	X	G	O	N	A	R	N	O	U	S	G	R
R	H	J	U	F	U	N	Z	W	P	X	R	N	J	T	W	T	P	Y	U	D	A	I	I	R	P

Figura 2.25: Proceso de cifrado de la Rueda de Jefferson

Mensaje en claro: CONFIDENCIALIDAD DEL MENSAJE

En este caso, tomamos la siguiente fila para cifrar.

Mensaje cifrado: QYSCHROOQHKBDWXGONARNOUSGR

Proceso de descifrado

Para realizar el proceso de descifrado, es necesario colocar en cada rueda la letra correspondiente al mensaje cifrado, después el receptor tiene que examinar todas las filas para encontrar el mensaje en claro. Véase figura 2.26.

Mensaje cifrado: TQBYSLHMLKJGTVGLJRKBRVDKNT

N	J	K	Q	T	S	T	P	P	Q	Y	X	P	B	C	R	Z	O	B	H	M	S	T	N	Y	N
T	Q	B	Y	S	L	H	M	L	K	J	G	T	V	G	L	J	R	K	B	R	V	D	K	N	T
C	O	D	I	G	O	S	Y	T	E	C	N	I	C	A	S	D	E	C	I	F	R	A	D	O	X
R	Y	M	Z	J	Q	P	G	D	L	W	P	D	D	X	X	O	R	S	C	Z	D	P	G	H	G
M	H	A	M	V	X	Y	H	S	M	L	O	N	W	T	P	T	P	D	V	K	W	C	O	E	V

Figura 2.26: Proceso de descifrado de la Rueda de Jefferson

En este caso, la siguiente fila muestra el mensaje descifrado.

Mensaje en claro: CODIGOS Y TECNICAS DE CIFRADOX

2.9. Cifrado Playfair

2.9.1. Descripción

Algoritmo desarrollado por Wheatstone en 1850, es llamado así en honor a su amigo Lord Playfair. En este algoritmo se hace uso de poligramas (conjunto de bloques de caracteres, en este caso dos), que a cada par de letras del mensaje en claro hace corresponder otras dos letras en el mensaje cifrado. Se usa con el alfabeto inglés de 26 letras que se disponen sobre un cuadro de cinco filas y cinco columnas. La disposición de las letras en el cuadro es la clave del cifrado, aparte de esto, es necesario contar con una clave, la cual no llevará letras repetidas, por ejemplo la palabra CIFRADO, después se escriben las letras del alfabeto, cuidando que no coincidan con las de la clave, así se obtendrá la matriz de la figura 2.27.

C	I/J	F	R	A
D	O	B	E	G
H	K	L	M	N
P	Q	S	T	U
V	W	X	Y	Z

Figura 2.27: Cifrado Playfair

Para efectuar el cifrado se emplean las siguientes reglas:

- Si el par de letras a cifrar están situadas en filas y columnas diferentes, se forma el rectángulo que tiene como vértices opuestos las dos letras. Las letras de los otros dos vértices forman el mensaje cifrado, ordenadas por filas de la misma forma que en el mensaje en claro. Es decir que se pone antes en el mensaje cifrado la letra que se encuentra en la misma fila que la primera letra del mensaje en claro. Por ejemplo, con la tabla anterior al par LI le corresponde KF y al par ZO le corresponde WG.
- Si ambas letras se encuentran en la misma fila, se sustituyen por las que se encuentran en la misma fila a su derecha. Si alguna de ellas está en la última columna se sustituye por la letra de la misma fila en la primera columna. Por ejemplo IC se cifraría como FI y OE como BG.
- Igualmente, si están en la misma columna, se cifran mediante las letras que se encuentran justamente debajo de ellas. Si alguna está en la quinta fila, por la de la primera. Con la tabla anterior, BS se cifra como LX, y AZ como GA.

2.9.2. Funcionamiento

Proceso de cifrado

Considerando que se tiene:

Mensaje en claro: SEGURIDAD PROVIENE DEL LATIN SECURITAS.

Ahora bien, lo primero a realizar es la formación de digramas:

SE GU RI DA DP RO VI EN ED EL LA TI NS EC UR IT AS

Ocupando como clave la palabra: CIFRADO

Nuestra matriz queda:

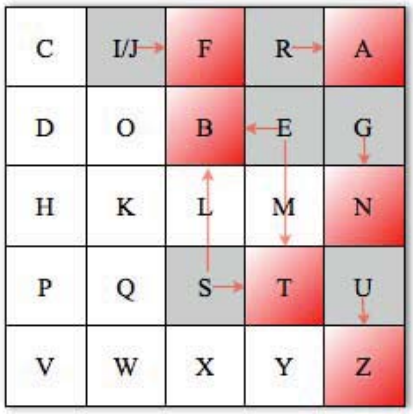


Figura 2.28: Ejemplo de cifrado de Playfair

El mensaje cifrado es:

TB NZ AF GC HV IE WC GM GO BM NF QR LU DR TA RQ FU

Proceso de descifrado

Ahora se considera la palabra clave *SEGURO*, se obtiene la matriz mostrada en la figura 2.29:

S	E	G	U	R
O	A	B	C	D
F	H	I/J	K	L
M	N	P	Q	T
V	W	X	Y	Z

Figura 2.29: Matriz de cifrado de Playfair

Y el mensaje cifrado es: AW MD ET AO GE GU RS LB BO

Ahora aplicando las reglas antes mencionadas, pero de manera inversa, por ejemplo, en la regla 2 en lugar de que haya un desplazamiento a la derecha, esta vez será a la izquierda.

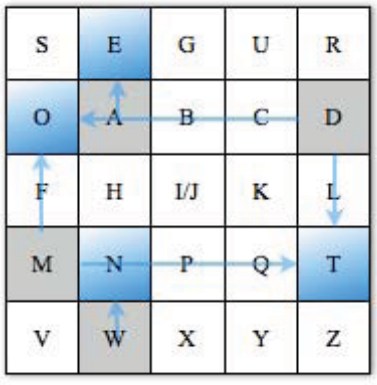


Figura 2.30: Matriz de cifrado de Playfair

Mensaje en claro: ENTORNO DE SEGURIDAD

2.10. Enigma

Desarrollada por el holandés Alexander Koch y el alemán Arthur Scherbius, este último fundó una compañía para llevar a cabo la producción comercial de dicha má quina, así la primera versión comercial fue puesta a la venta en 1923. En 1926 la marina alemana adquirió una máquina Enigma solicitando que se le agregara una cuarta rueda para que la comunacacíon fuera más segura que con las máquinas de tres rotores.

El cifrado operaba de la siguiente manera: se hacía pasar corriente eléctrica a través de un cierto mecanismo, de manera que la letra que se imprimía era ya el resultado de un complejo sistema de cifrado, en el que había una serie de ruedas colocadas tocándose por sus caras y formando un cilindro. Al oprimir una tecla, la corriente llegaba a la primera rueda. Las ruedas tenían contactos eléctricos delante y detrás, y en su interior estaban conectados los de adelante con los de atrás con base en un patrón previamente establecido. Véase figura 2.31.

Figura 2.31: Enigma

2.10.1. Descripción

La operación de la máquina Enigma consiste fundamentalmente en un teclado similar al de las máquinas de escribir, que tiene conectado un rotor compuesto de 26 cables (uno por cada letra del alfabeto inglés). En la primera máquina Enigma, se contaba con tres rotores conectados entre sí y tres ranuras para poder introducirlos, de forma

que el primer rotor estaba conectado con el segundo y el segundo con el tercero. El tercer rotor se conectaba a un reflector y la salida de este se volvía a conectar al tercer rotor, para realizar el mismo proceso pero en sentido contrario.

Un rotor se compone de dos partes principales, en primer lugar el anillo exterior móvil que contiene el alfabeto y que se encuentra visible en la ventanilla de la máquina. En segundo lugar el núcleo con el cableado interno que se utiliza para ajustar manualmente la posición del rotor. El cambio de posición del anillo interior con respecto al anillo exterior se llama ring setting. Véase figura 2.32.

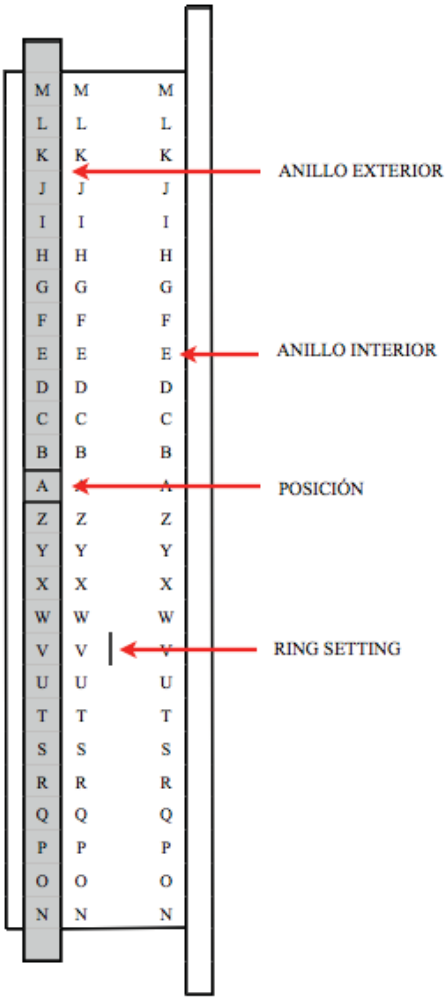


Figura 2.32: Estructura del rotor

Al teclear una letra se iluminaba una lámpara con su correspondiente criptograma que podría ser siempre diferente, ya que eso dependía de la posición del rotor, la cual era la clave para cifrar el mensaje.

La clave tenía que estar compuesta por el orden de los rotores, el ring setting de los rotores y las conexiones de los cables en el plugborad. Era necesario conectar los 3 pares de cables con los pares de letras indicados, de manera que la máquina quedara configurada para cifrar y descifrar, debido a que su operación es simétrica. Así, la clave es típicamente representada de la siguiente forma:

Rotores: I - II - III

Posiciones: A - A - B

Plugborad: G/L, K/J, Y/X, M/T, Z/I, V/O

Ring setting: A01 - A01 - B02

También se toman en cuenta las especificaciones de los rotores, ya que cada uno de ellos tiene una configuración distinta con respecto al notch, que es una muesca que tiene el rotor que hace que el rotor a la izquierda gire en el momento en que un mecanismo dentro de la máquina coincide con esta muesca. Por ejemplo si el rotor III, que se encuentra a la derecha, tiene un notch en la letra Q, el rotor II, colocado en el centro, tendrá que avanzar una posicón cuando el rotor III cambie de Q a R.

2.10.2. Funcionamiento

Proceso de cifrado

En este ejemplo se utilizan los rotores I, II y III en las posiciones A - A - B y un ring setting A01 - A01 - A01 respectivamente, y el reflector B con las siguientes especificaciones. Véanse figuras 2.33 y 2.34.

ENTRADA	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ROTOR I	E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J
ROTOR II	A	J	D	K	S	I	R	U	X	B	L	H	W	T	M	C	Q	G	Z	N	P	Y	F	V	O	E
ROTOR III	B	D	F	H	J	L	C	P	R	T	X	V	Z	N	Y	E	I	W	G	A	K	M	U	S	Q	O

Figura 2.33: Especificaciones de los rotores

ENTRADA	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
REFLECTOR B	Y	R	U	H	Q	S	L	D	P	X	N	G	O	K	M	I	E	B	F	Z	C	W	V	J	A	T

Figura 2.34: Especificaciones del reflector

Se procede a hacer el cifrado de la letra A. En el rotor III se ha establecido en la posición B. Primero, la señal llega a la letra A, pero entra en el contacto B, por la posición establecida, de acuerdo a las especificaciones de los rotores, la letra correspondiente a la letra de entrada B para el rotor III es D, después se resta la posición para obtener la primer letra de salida del primer rotor, la cual es C.

En el segundo rotor, la letra de entrada es C, como la posición establecida es A, no se realiza un desplazamiento, así, para la letra de entrada C corresponde una letra de salida D.

En el tercer rotor, el rotor I recibe la letra D que de acuerdo a las especificaciones, la salida es la letra F.

El reflector toma la letra F y la cifra como una letra S, que es regresada al rotor I.

El rotor I recibe la letra S y de acuerdo a las especificaciones de los rotores, obtiene una letra S, que es la nueva entrada del segundo rotor.

El rotor II codifica la letra S como una E para pasarla al último rotor, el cual realiza un desplazamiento debido a la posición establecida, entonces la letra correspondiente a E es una F, que es codificada como una C, pero es necesario restarle el desplazamiento para poder obtener la letra cifrada correspondiente a la A, que es B.

Mensaje en claro: A

Mensaje cifrado: B

El proceso se ejemplifica en la figura 2.35.

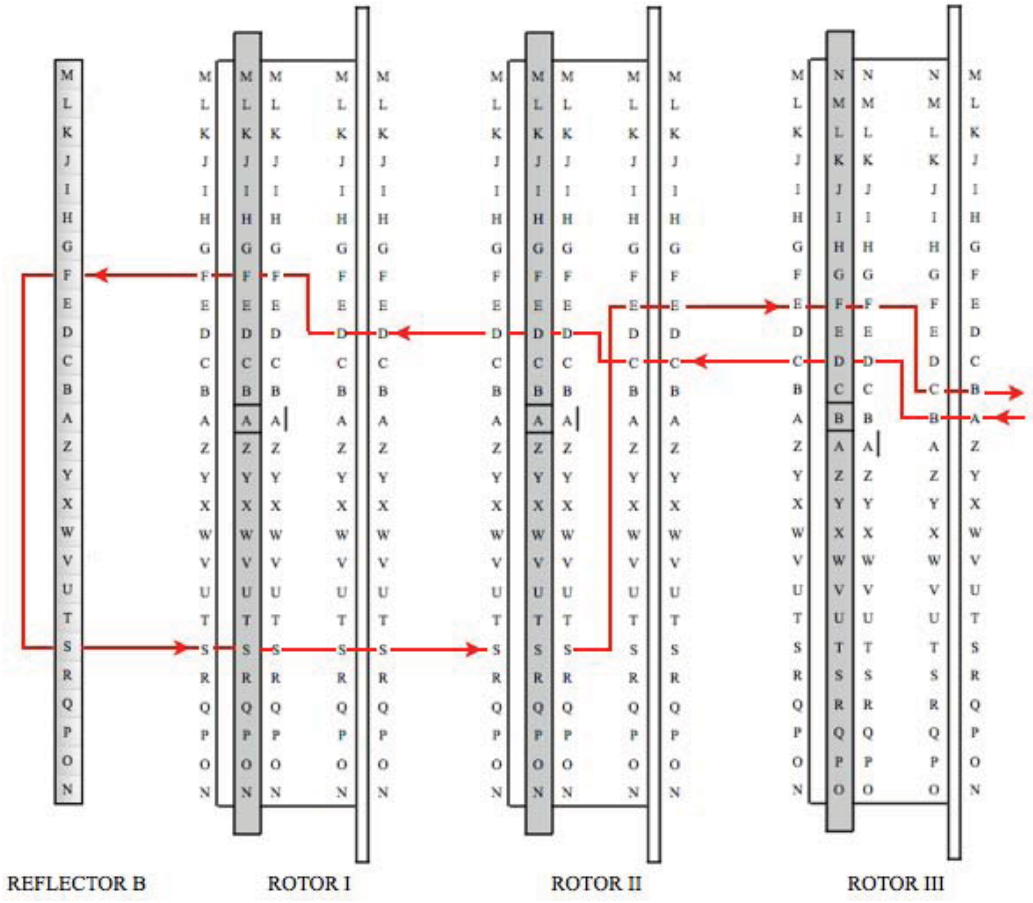


Figura 2.35: Proceso de cifrado de la máquina Enigma

Proceso de descifrado

El proceso de descifrado en la máquina Enigma es simétrico, esto significa que se utiliza el mismo proceso para cifrar o descifrar mensajes. La figura 2.36 muestra el procedimiento de descifrado.

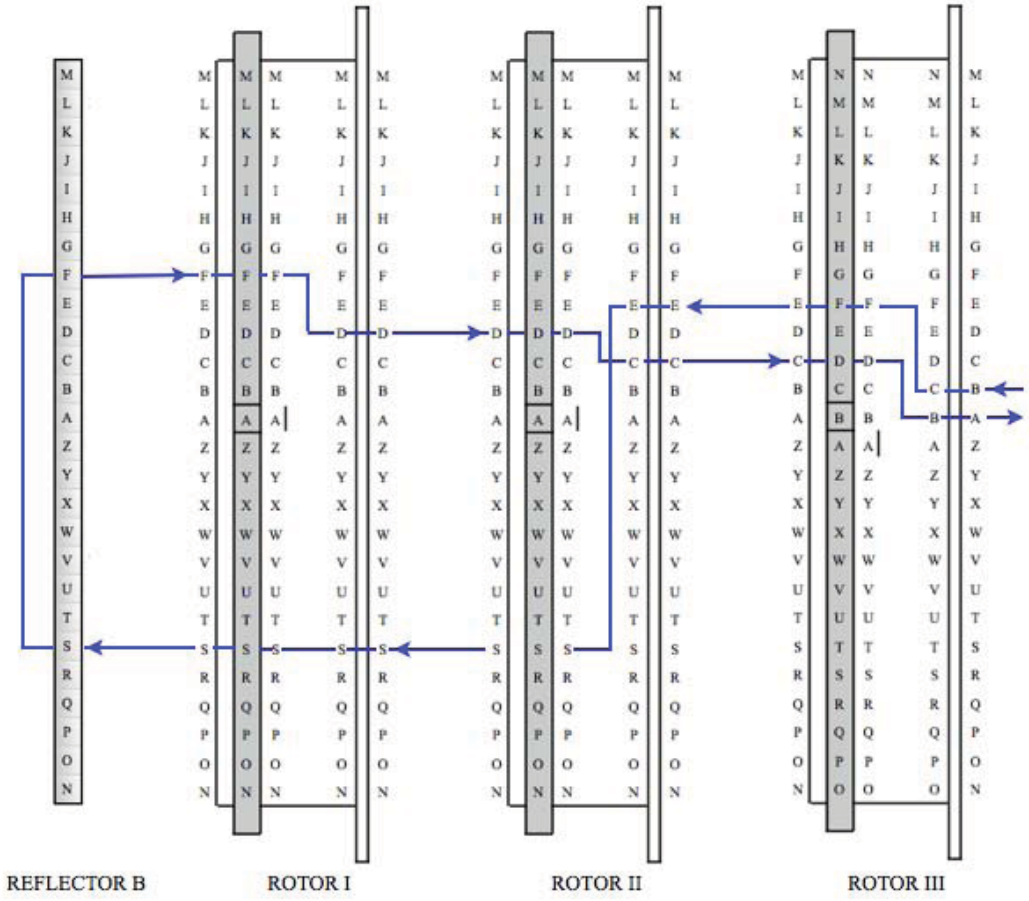


Figura 2.36: Proceso de descifrado de la máquina Enigma

Capítulo 3

Diseño

3.1. Análisis de requerimientos

3.1.1. Introducción

El presente capítulo tiene como objetivo reúnir la información necesaria para ayudar a analizar y entender todos los requerimientos funcionales del sistema que servirán para el desarrollo de la aplicación y sus posteriores pruebas, está basado en el estándar de IEEE 830.

3.1.2. Ámbito del sistema

Cipher

El sistema permitirá al usuario interactuar con diversos métodos criptográficos antiguos con el fin de mejorar sus conocimientos en esta área.

El sistema no esta dedicado a cifrar mensajes ni a descifrarlos para su utilización en aplicaciones reales.

El beneficio principal del sistema es la mejor y más rápida adquisición de conocimiento en el ámbito de la Criptografia antigua así como una mejor comprensión del contexto histórico de cada uno de los métodos.

3.2. Descripción General

3.2.1. Funciones del producto

La aplicación contará con un menú principal donde se podrá navegar entre cada uno de los métodos, opciones de configuración y créditos.

En cada uno de los métodos se mostrarán opciones que guiarán al proceso de cifrado, descifrado y una breve descripción del contexto histórico. En los procesos de cifrado y descifrado se podrá interactuar directamente con una simulación del dispositivo necesario para cada método. En el proceso de cifrado se podrán guardar los mensajes para su posterior descifrado. En el proceso de descifrado se podrá elegir entre una lista de mensajes cifrados guardados anteriormente. En cada método se mostrará un tutorial de cómo interactuar con el dispositivo para cifrar o descifrar.

3.2.2. Características de los Usuarios y Restricciones

Es necesario tener presente que en un desarrollo de esta naturaleza, los usuarios juegan un papel muy importante debido a que ellos son los que van a interactuar con dicha aplicación por lo que es necesario que cuenten con las siguientes características:

- Los usuarios deben tener una noción general de los objetivos de la Criptografía así como un interés en aprender los diferentes métodos de cifrado clásicos.
- Los usuarios pueden reforzar sus conocimientos, si éstos ya cuentan con los conceptos de los diferentes algoritmos que forman parte de la historia de la Criptografía.

En cuanto a las restricciones, estas se refieren al entorno de desarrollo y la plataforma de ejecución de tal forma que:

- El sistema se desarrollará en Objective-C sobre la plataforma iOS 6.1 para los dispositivos iPad, iPod Touch y iPhone, tomando en cuenta que el iPad es la tablet más vendida en la actualidad.

3.3. Requisitos Específicos

3.3.1. Funciones

Scítala

- En la pantalla para cifrar se mostrará una scítala con una tira de papiro a su alrededor.
- Se mostrarán controles para modificar el largo y el diámetro de la misma, al ir cambiando, la tira de papiro se irá ajustando al nuevo tamaño.
- Para introducir el mensaje en claro se tendrá que hacer touch en la primera celda del papiro, de tal manera que se mostrará el teclado para ir introduciendo el mensaje en claro.
- El usuario podrá girar la scítala según sea necesario.
- Cuando el mensaje a cifrar esté completo, el usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.
- En la pantalla de descifrado, el usuario seleccionará un mensaje cifrado de una lista de mensajes guardados previamente, además podrá ver los detalles de los criptogramas, como son la fecha de creación y el autor.
- El usuario será capaz de ir modificando el largo y diámetro del bastón hasta descifrar el mensaje, además se le indicará si el mensaje descifrado es correcto.

Cuadro cifrador de Polybio

- En la pantalla para cifrar se mostrará una matriz con el alfabeto y adicionalmente una fila de entradas en la parte superior y una columna de entradas al lado izquierdo, las cuales formarán la combinación que aparece en el mensaje cifrado.
- Se tendrá la opción de solicitar el cifrado con letras o con números. Esta opción estará en la parte inferior de la matriz.
- Se mostrará el teclado del iPad para que el usuario ingrese el mensaje a cifrar, o si lo desea, puede hacer touch en cada celda de la matriz. En cualquiera de las dos formas, se resaltará la letra y sus intersecciones con las letras o números correspondientes.
- Cuando el usuario escriba el mensaje en claro, el mensaje cifrado irá apareciendo en un campo de texto.
- Una vez terminado el proceso de cifrado, el usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.
- En la pantalla de descifrado, el usuario seleccionará un mensaje cifrado de una lista de mensajes guardados previamente, además de poder ver los detalles de los criptogramas, como son la fecha de creación, el autor y el modo (letras o números).
- El usuario tendrá la opción de escribir el mensaje en claro mediante el teclado del iPad o al hacer touch en la matriz, en el caracter correspondiente a la fila y columna especificado en el criptograma, en cualquier opción, se resaltará la letra y sus intersecciones pertenecientes.
- Debajo del mensaje cifrado, irá apareciendo el mensaje en claro, donde se indicará al usuario si está realizando el descifrado de manera correcta, se realizará la revisión por cada letra.

Cifrado del César

- En la pantalla para cifrar se mostrará el alfabeto inglés repetido dos veces, el primer alfabeto estará fijo y el segundo podrá deslizarlo el usuario para establecer el desplazamiento de las letras.
- Se mostrará el teclado del iPad para que el usuario ingrese el mensaje a cifrar, o si lo desea, puede hacer touch en cada letra del alfabeto fijo.
- El mensaje cifrado irá apareciendo en un cuadro de texto mientras el usuario escriba el mensaje en claro.

- Una vez terminado el proceso de cifrado, el usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.
- En la pantalla de descifrado, el usuario seleccionará un mensaje cifrado de una lista de mensajes guardados previamente, además podrá ver el detalle del criptograma, como la fecha de creación, el autor y el desplazamiento.
- Se tendrá que deslizar el segundo alfabeto para poder descifrar el mensaje, de esta manera se irá actualizando el mensaje cifrado, hasta que se tenga un mensaje coherente.
- El mensaje en claro se mostrará debajo del mensaje cifrado, se le indicará al usuario si ha descifrado el mensaje de manera correcta.

Disco de Alberti

- En la pantalla inicial se mostrará un menú, para indicar si el usuario desea construir por sí mismo un disco, si desea utilizar un disco previamente guardado o si desea utilizar el disco por defecto, el cual contendrá en el disco exterior el alfabeto latín convencional de 20 carácteres y los números 1,2,3 y 4, en el disco interior se encuentran las letras en desorden, además se cuenta con el simbolo & y las letras H,Y y K.
- Si se selecciona la opción de crear nuevo disco, se mostrará el teclado del iPad y el usuario tendrá que introducir los caracteres que desee utilizar y después guardarlo. Una vez guardado, regresa al menú principal y selecciona la opción de utilizar un disco previamente guardado.
- Si se elige usar un disco guardado, el usuario seleccionará de una lista de discos guardados previamente.
- En la pantalla para cifrar se mostrarán dos discos, el externo y el interno, los cuales serán móviles, debajo de estos discos se podrá introducir el mensaje en claro mediante el teclado del iPad, para empezar el proceso de cifrado se eligirá una posición inicial de los discos, además de establecer la frecuencia, desplazamiento y dirección que componen la clave mediante un botón (Settings). El usuario podrá escribir el mensaje en claro.
- Mientras el usuario escriba el mensaje en claro, el mensaje cifrado irá apareciendo en un cuadro de texto, cuando termine de escribir el mensaje, el usuario podrá guardarlo para poder usarlo posteriormente en el proceso de descifrado.

- En la pantalla de descifrado se podrá seleccionar un mensaje cifrado de una lista, cuando se seleccione uno, aparecerán los discos con los cuales fue cifrado, así como su clave.
- El usuario podrá rotar el disco de acuerdo a como fue rotado en el proceso de cifrado, además de establecer el desplazamiento, frecuencia y dirección e ir introduciendo el caracter del mensaje cifrado mediante el teclado del iPad en un cuadro de texto.
- Por cada letra que introduzca, se le indicará al usuario si esta descifrando el mensaje de manera correcta.

Cifrado de Vigenère

- En la pantalla para cifrar se mostrará, en la parte superior, la primera fila de la matriz compuesta por el alfabeto inglés, así como la primera columna. La primera fila es para introducir el mensaje en claro y la columna es donde se introduce la llave para poder realizar el proceso de cifrado. Se tendrá un detalle de una parte de la matriz y una imagen que hace referencia a la matriz completa.
- Lo primero que se tendrá que hacer es introducir la llave, esto mediante el teclado del iPad.
- Después, se tendrá que ingresar el mensaje en claro, se puede hacer desde el teclado del iPad o al hacer touch en el caracter correspondiente al alfabeto de la primera fila, se iluminará la letra conveniente al caracter de la llave y al de la columna especificada. Además, en la imagen, se presentará la posición actual en la matriz.
- El mensaje cifrado irá apareciendo en un cuadro de texto.
- Una vez terminado el proceso de cifrado, el usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.
- En la pantalla de descifrado, el usuario seleccionará un mensaje cifrado de una lista de mensajes guardados previamente, además de poder consultar algunos detalles como la fecha de creación, el autor y la clave del criptograma.
- El usuario tendrá la opción de descifrar el mensaje desde el teclado del iPad o al hacer touch en el caracter correspondiente en la fila superior del alfabeto, de esta forma, se iluminarán las letras de la llave y el criptograma.

- Arriba de la llave irá apareciendo el mensaje en claro, por cada letra que se introduzca, se le indicará al usuario si esta descifrando el mensaje de manera correcta.

Máscaras Rotativas

- En la pantalla para cifrar se mostrará una matriz de 10x10, en la parte inferior el usuario podrá modificar el tamaño. Una vez realizada esta elección, del lado inferior derecho de la matriz se contará con la opción de editar la máscara. El usuario tendrá que seleccionar que casillas de la matriz serán perforadas.
- El usuario podrá introducir el mensaje en claro seleccionando la casilla perforada de la matriz e ir introduciendo los caracteres mediante el teclado del iPad. El mensaje en claro aparecerá en la parte inferior de la pantalla. Se podrá girar la máscara con los botones que aparecen en el lado superior derecho de la matriz.
- Una vez que se realizaron los tres giros, en la parte inferior izquierda de la matriz, se contará con una opción que permitirá al usuario llenar los espacios vacíos con caracteres aleatorios.
- Una vez terminado el proceso de cifrado, el usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.
- En la pantalla de descifrado, el usuario seleccionará una matriz cifrada de una lista de matrices guardadas previamente, también podrá ver los detalles de dicho criptograma, como el autor y la fecha de creación.
- Aparecerá la matriz y el usuario tendrá que seleccionar el botón que se encuentra del lado inferior derecho de la matriz para poder editar la máscara, de esta manera podrá elegir las casillas que crea que fueron perforadas, se le indicará si ha seleccionado las casillas correctas, además de que en la parte inferior de la pantalla se mostrará el mensaje descifrado.

Cifrado de Playfair

- En la pantalla para cifrar se mostrará una matriz de 5x5 en donde se podrá introducir la llave del mensaje al hacer touch en las casillas e introduciendo los caracteres mediante el teclado del iPad.

- Una vez realizado esto, en la parte inferior de la matriz, se tiene una opción para completarla con las letras adecuadas.
- El usuario tendrá que introducir en el campo de texto el mensaje en claro mediante el teclado del iPad. En la matriz se iluminarán los digramas y las letras que conformarán el criptograma, además el mensaje cifrado irá apareciendo en un campo de texto.
- El usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.
- En la pantalla de descifrado, el usuario seleccionará un mensaje cifrado de una lista de mensajes guardados previamente, además de poder consultar algunos detalles como la fecha de creación, el autor y la clave del criptograma.
- El usuario tendrá que poner la clave en la matriz y después completarla con las letras restantes del alfabeto o con el botón Completar que se encuentra en la parte inferior de la matriz.
- Para descifrar el mensaje, el usuario tendrá que hacerlo desde el teclado del iPad, se le indicará si cada digrama introducido es correcto.

Rueda de Jefferson

- En la pantalla para cifrar se mostrará una rueda de Jefferson con 26 cilindros.
- El usuario podrá intercambiar el orden de los discos mediante un botón ubicado debajo de la rueda de Jefferson.
- Cada cilindro se podrá rotar para conseguir el caracter deseado, abajo de la rueda de Jeferson, se introducirá el mensaje en claro y el mensaje cifrado, también se podrá hacer mediante el teclado del iPad.
- Una vez que se introdujo el mensaje, se podrá dar touch al botón Bloquear Mensaje
- Después el usuario tendrá que establecer la fila para poder cifrar el mensaje, esto lo hará girando el disco que se encuentra a la izquierda de la rueda de Jefferson.
- El usuario tendrá la opción de guardar el mensaje cifrado para poder utilizarlo en el proceso de descifrado.

- En la pantalla de descifrado, el usuario seleccionará un mensaje cifrado de una lista de mensajes guardados previamente, además de poder consultar algunos detalles como la fecha de creación, el autor y el orden de los discos.
- El usuario podrá intercambiar el orden de los discos según el detalle del criptograma.
- El usuario podrá ir moviendo cada cilindro hasta obtener el mensaje cifrado en los cilindros, así podrá hacer touch en cada caracter que aparece en los cilindro o introducirlo mediante el teclado del iPad, se le indicará si esta descifrando el mensaje de manera correcta.

Enigma

- En la pantalla para cifrar se mostrará un menú que contendrá las máquinas disponibles (Enigma I, Enigma G, Enigma K, Enigma M3 y Enigma M4) para realizar el proceso de cifrado.
- El usuario podrá configurar la posición inicial de los rotores y del reflector, dando touch en cada uno de ellos.
- El usuario tendrá la opción de configurar en el panel de conexiones las sustituciones de letras, dando touch en cada una de las letras que desee conectar.
- El usuario podrá introducir el mensaje en claro presionando las teclas de la máquina.
- El mensaje cifrado irá apareciendo en la parte superior derecha de la máquina, debajo del mensaje en claro.
- Se podrá guardar el mensaje cifrado para poder utilizarlo posteriormente en el proceso de descifrado.
- En la pantalla de descifrado se podrá seleccionar un mensaje guardado anteriormente.
- Se mostrará la configuración que se usó para cifrar el mensaje.
- El usuario podrá realizar la configuración inicial de la máquina de la misma manera que en el proceso de cifrado.
- Para descifra el mensaje, se tendrá que presionar una tecla de la máquina.

- El mensaje descifrado irá apareciendo en la parte superior derecha de la máquina, arriba del mensaje cifrado.
- Se le indicará al usuario si esta descifrando el mensaje de manera correcta.

3.3.2. Requisitos de rendimiento y de diseño

La aplicación será usada por un sólo usuario en un determinado momento y es independiente para cada dispositivo. La capacidad de almacenamiento para los criptogramas solo esta limitada por la capacidad del dispositivo. Para cada método el tiempo de cifrado o descifrado que tarda un algoritmo con una cadena de texto de 200 caracteres no debe ser mayor a 83.33 milisegundos, debido a que el ojo humano puede procesar de 10 a 12 imagenes por segundo, tomando como referencia las 12 imagenes por segundo, el tiempo para procesar cada imagen es de aproximadamente 83.33 milisegundos.

Debido al tamaño limitado de la pantalla se deberá mostrar un número adecuado de elementos en cada método para facilitar su visualización por parte del usuario.

3.3.3. Atributos del sistema

- Fiabilidad.

  El sistema debe realizar de manera correcta el proceso de cifrado y descifrado de cada método criptográfico.

- Corrección.

  El sistema satisface la mecánica de los métodos de cifrado, cumpliendo con el objetivo de mejorar la enseñanza de la criptografía.

- Eficiencia.

  Para cada método, el tiempo de cifrado o descifrado de una cadena de texto, no mayor a 200 caracteres, no debe ser mayor a 83.33 milisegundos.

- Integridad.

  No se tiene la necesidad de controlar el acceso al software o a los datos por personas no autorizadas, debido a que el propósito del sistema no es guardar información confidencial.

- Usabilidad.

  Debido a que el sistema es una aplicación didáctica, debe ser lo más amigable posible hacia el usuario, además de complementarlo con la ayuda necesaria.

- Mantenibilidad.

  Debido a la alta cohesión, al bajo acoplamiento del sistema y al uso de la programación orientada a objetos, será relativamente sencillo encontrar errores en el código, para posteriormente resolverlos.

- Flexibilidad.

  Utilizando el patrón MVC y un bajo acoplamiento, agregar nuevos métodos de cifrado o nuevas características a los existentes debe resultar sencillo.

- Portabilidad.

  Ya que es necesario usar el framework Cocoa Touch para iOS, el código no sería portable en otra plataforma, por ejemplo Android.

- Reusabilidad.

  Es sencillo usar el Modelo para una aplicación usando Cocoa en MacOSX.

Capítulo 4

Implementación

4.1. Introducción

El presente capítulo describe el funcionamiento de las clases de Modelo de la aplicación en el patrón Modelo Vista Controlador, en la sección 4.2 se describe el sistema de forma simplificada en un diagrama de clases UML y en la sección 4.3 se detallan las partes relevantes de código de cada método dedicadas a cifrar y descifrar.

4.2. Diagramas de clase

De acuerdo a las especificaciones mencionadas en el capítulo anterior, los diagramas de clases simplificados para la aplicación son los siguientes. Véanse figuras 4.1, 4.2, 4.3, 4.4 y 4.5. El prefijo `Cy` de cada clase de la parte de modelo está escrito con una *y* en lugar de una *i* para evitar una confusión con el prefijo `Ci` utilizado por Core Image, un framework de iOS.

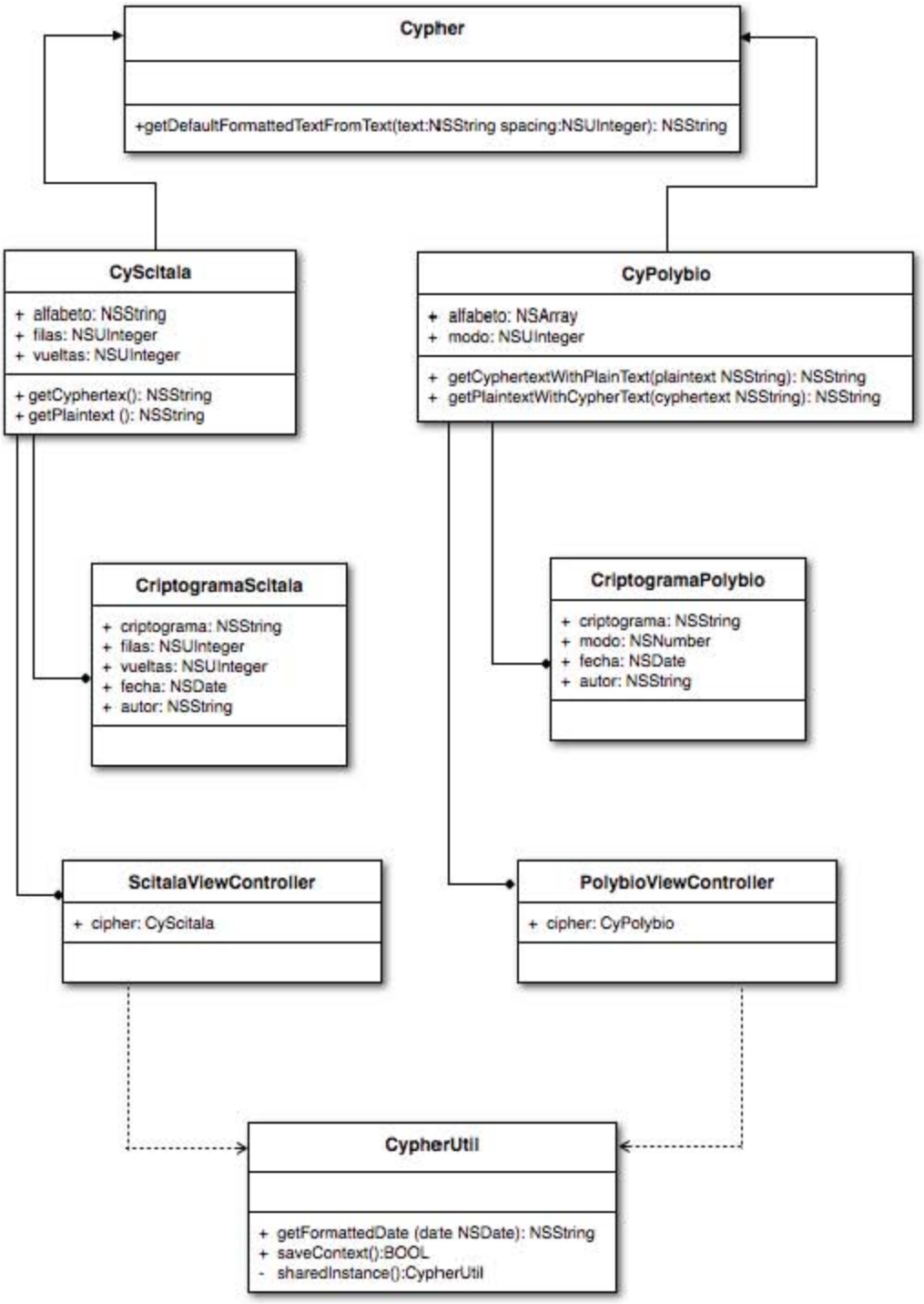


Figura 4.1: Diagrama de clases (Scítala y Polybio)

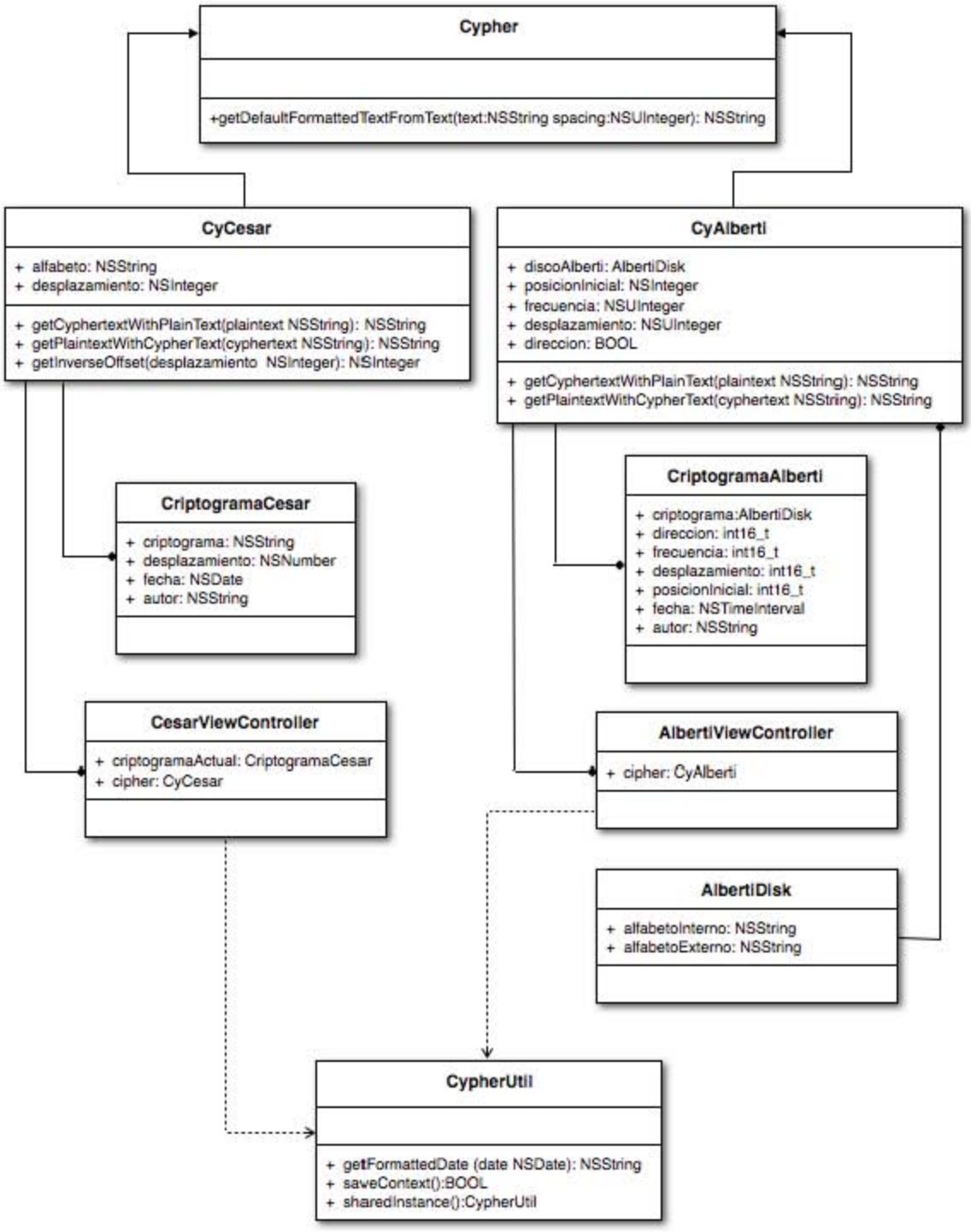


Figura 4.2: Diagrama de clases (César y Alberti)

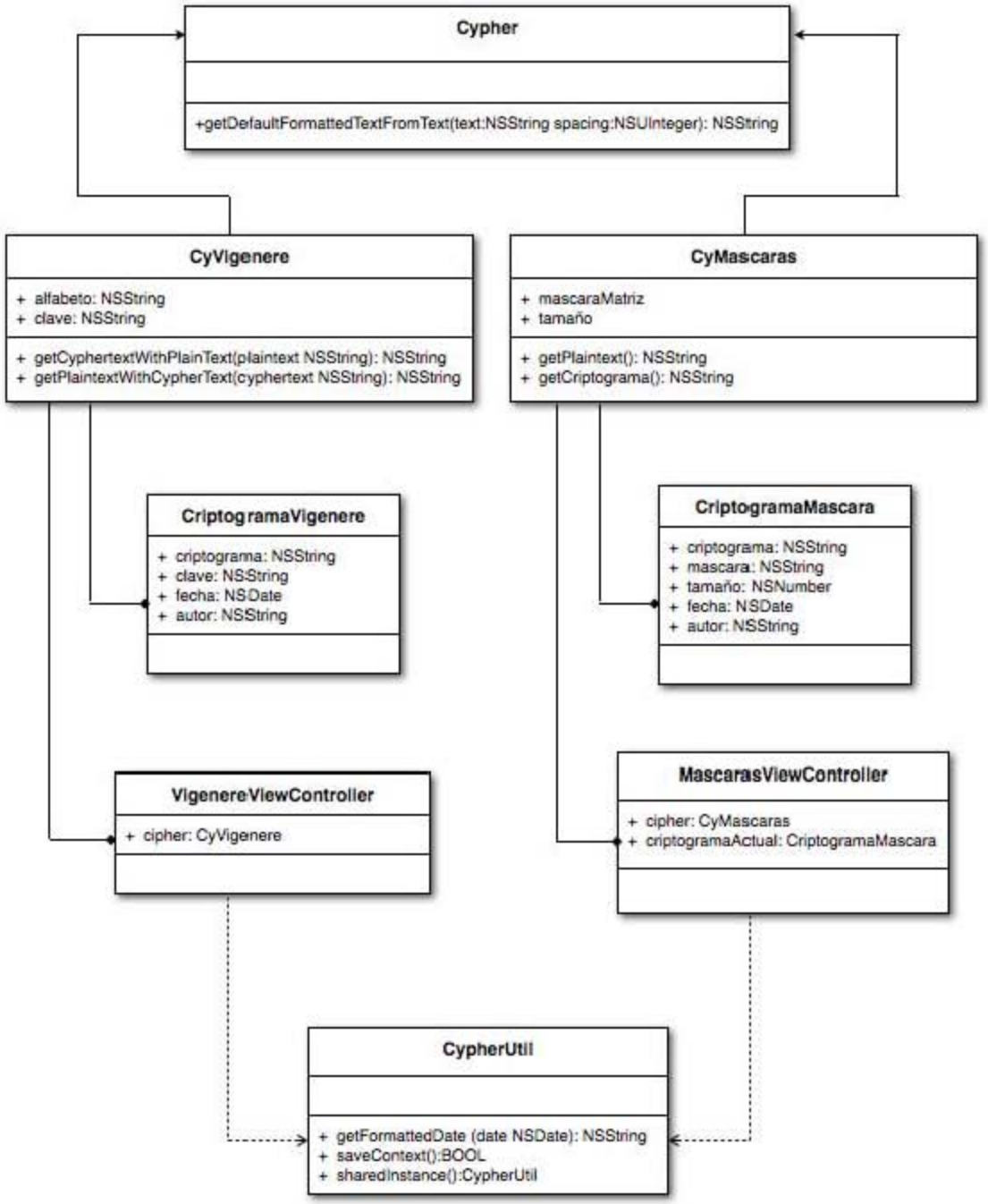


Figura 4.3: Diagrama de clases (Vigenère y Máscaras Rotativas)

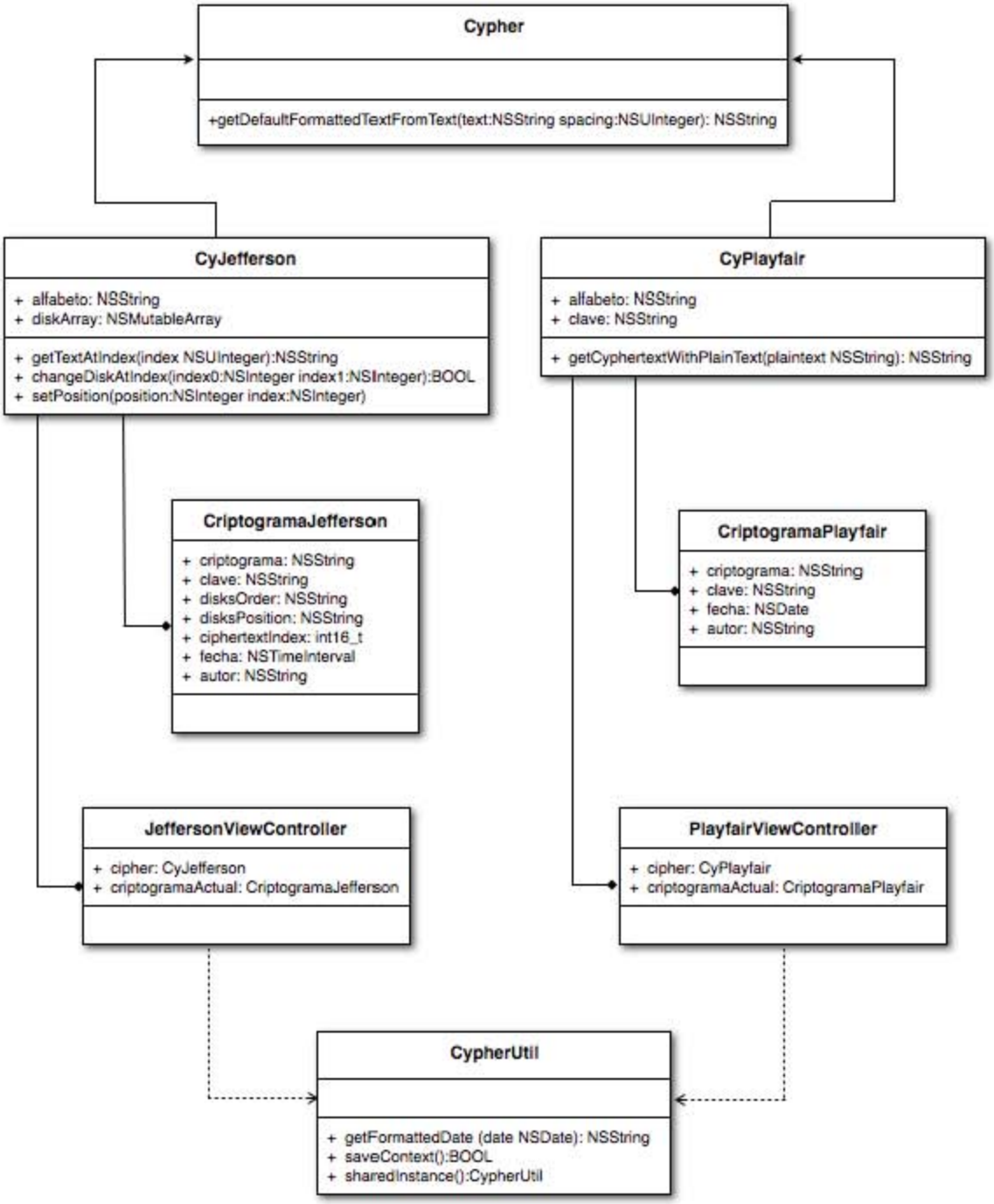


Figura 4.4: Diagrama de clases (Jefferson y Playfair)

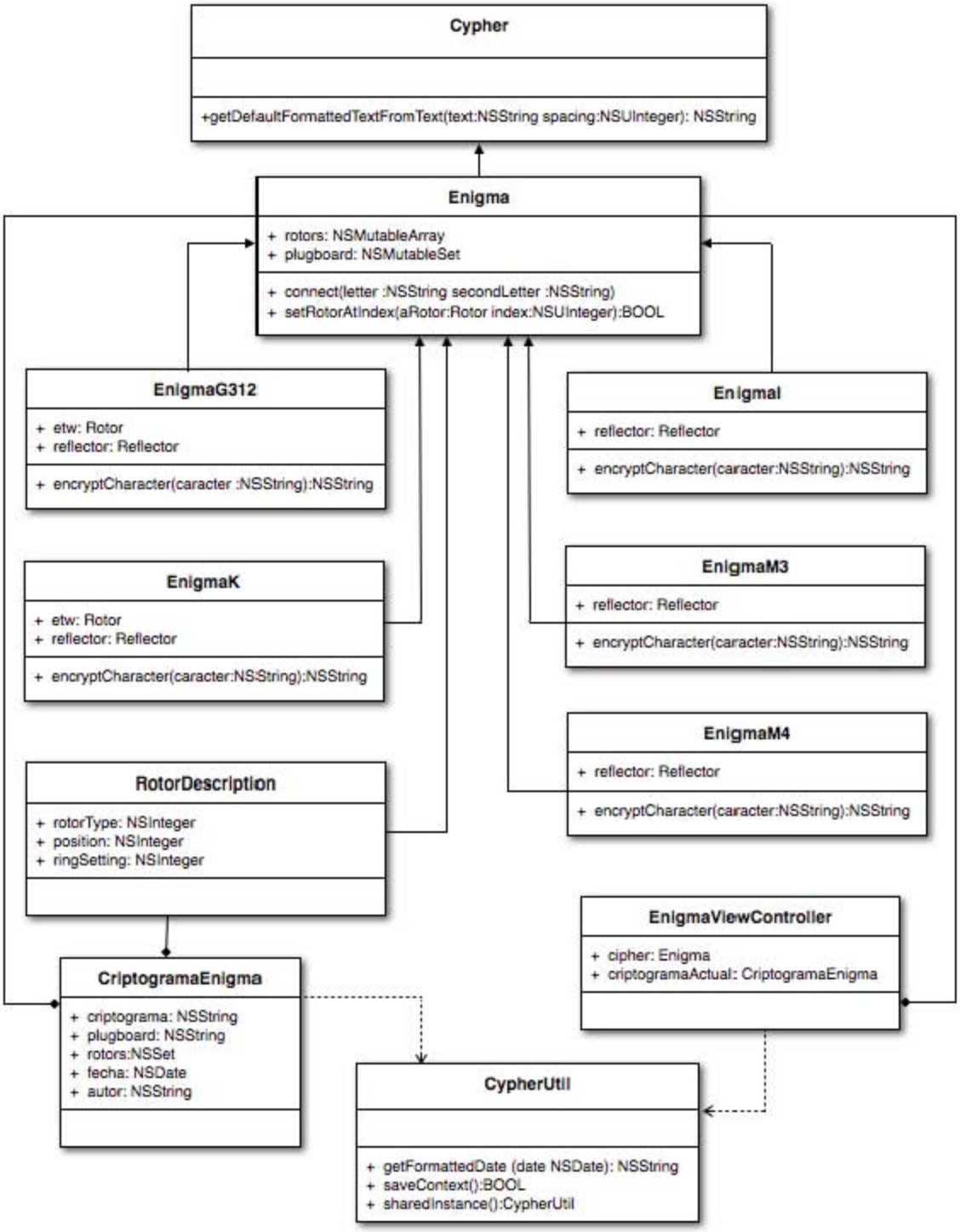


Figura 4.5: Diagrama de clases (Enigma)

4.3. Algoritmos

En esta sección se presentan las partes de código relevantes de los algoritmos encargados de cifrar y descifar una cadena de texto. Estas líneas de código se encuentran en las clases correspondientes al modelo en el patrón MVC y se pueden observar en las figuras de la sección anterior con el prefijo Cy. Cada subsección muestra un método criptográfico y una breve explicación de la implementación.

4.3.1. Scítala

Como podemos observar en la figura 4.6 la clase `CyScitala` implementa los métodos `getCiphertext` y `getPlaintext`, los cuales son los métodos principales que consulta el controlador `ScitalaViewController` para actualizar sus vistas cuando el texto en claro y el criptograma han cambiado, a continuación se muestra la implementación del primero de ellos:

```
1  - (NSString *)getCiphertext
2  {
3      NSMutableString *result = [NSMutableString stringWithCapacity:self
           .rows * self.loops];
4      for (NSUInteger index = 0; index < self.rows * self.loops; index
           ++) {
5          if ([_ciphertext[index] length] == 1) {
6              [result appendString:_ciphertext[index]];
7          } else {
8              [result appendString:@" "];
9          }
10     }
11     return [NSString stringWithString:result];
12 }
```

En la línea 3 se crea la cadena que va a almacenar el criptograma y en las líneas 4 a 10 se itera en el arreglo `_ciphertext` que representa el papiro donde se escribe el texto y agrega cada letra a la cadena `result`, el límite del `for` está dado por el producto de las filas y vueltas actuales.

La implementación del método `getPlaintext` es muy parecida a la de `getCiphertext` pero en este caso la cadena que representa el texto en claro se obtiene concatenando la primera letra dentro del arreglo `_ciphertext` con cada letra que se encuentra a un

espacio igual a los multiplos del número de filas actuales dentro del arreglo, y así sucesivamente para las siguientes letras hasta completar el total de filas.

```
1  - (NSString *)getPlaintext
2  {
3      NSMutableString *result = [NSMutableString stringWithCapacity:self
               .rows * self.loops];
4      for (NSUInteger row = 0; row < self.rows; row++) {
5          for (NSUInteger loop = 0; loop < self.loops; loop++) {
6              if ([_ciphertext[loop * self.rows + row] length] == 1) {
7                  [result appendString:_ciphertext[loop * self.rows +
                           row]];
8              } else {
9                  [result appendString:@" "];
10             }
11         }
12     }
13     return [NSString stringWithString:result];
14 }
```

En la línea 3 se itera entre cada fila de la scítala y en el `for` de la línea 4 se itera entre las vueltas, el índice dentro del arreglo `_ciphertext` se obtiene multiplicando la vuelta actual con el número de filas y sumando la fila actual, en las líneas 7 y 9 se agrega la letra correspondiente a la cadena resultante.

4.3.2. Polybio

El siguiente método recibe una cadena de texto en claro y devuelve el texto cifrado.

```
1  static const NSInteger COLUMNS = 5;
2  static const NSInteger ROWS = 5;
3
4  - (NSString *)getCyphertextWithPlaintext:(NSString *)plaintext
5  {
6      NSMutableString *result = [NSMutableString stringWithCapacity:[
               plaintext length] * 2];
7      NSRange range;
8      NSUInteger row, column;
9      for (NSUInteger index = 0; index < [plaintext length]; index++) {
10         range = [@"ABCDEFGHIKLMNOPQRSTUVWXYZ" rangeOfString:[plaintext
                   substringWithRange:NSMakeRange(index, 1)]];
11         if (range.location != NSNotFound) {
12             row = range.location % COLUMNS;
13             column = range.location / ROWS;
```

```
14
15              [result appendString:[self.cypherColumn substringWithRange
                        :NSMakeRange(column, 1)]];
16              [result appendString:[self.cypherRow substringWithRange:
                        NSMakeRange(row, 1)]];
17
18              if ([[plaintext substringWithRange:NSMakeRange(index, 1)]
                        isEqualToString:@"I"])
19                  index++;
20          }
21      }
22      return result;
23  }
```

El `for` de la línea 9 itera entre cada una de las letras del texto en claro y encuentra su fila y columna en la matriz de Polybio, dependiendo del modo en el que se este cifrando, letras o números, a la cadena `resultado` se le agrega la letra de la columna y fila, ya sea de `@"ABCDE"` o `@"12345"`, que le corresponde.

Para descifrar un criptograma se usa el siguiente método.

```
1   - (NSString *)getPlaintextWithCyphertext:(NSString *)cyphertext
2   {
3       cyphertext = [cyphertext stringByReplacingOccurrencesOfString:@" "
                withString:@""];
4
5       // No es un criptograma valido si no tiene una longitud par
6       if ([cyphertext length] % 2 != 0)
7           return nil;
8       NSMutableString *result = [NSMutableString stringWithCapacity:[
                cyphertext length] / 2];
9       for (NSUInteger i = 0; i < [cyphertext length]; i+=2) {
10          NSRange rangeColumn = [self.cypherColumn rangeOfString:[
                    cyphertext substringWithRange:NSMakeRange(i, 1)]];
11          NSRange rangeRow = [self.cypherRow rangeOfString:[cyphertext
                    substringWithRange:NSMakeRange(i + 1, 1)]];
12          if (rangeColumn.location == NSNotFound || rangeRow.location ==
                    NSNotFound)
13              return nil;
14          [result appendString:[[self.alfabeto objectAtIndex:rangeColumn
                    .location] objectAtIndex:rangeRow.location]];
15      }
16      return result;
17  }
```

En la línea 3 se sustituyen las ocurrencias de los espacios para tener una cadena que solo contenga caracteres válidos, en la línea 6 se comprueba la longitud del criptograma,

debe ser un múltiplo de 2, en el `for` de la línea 9 se itera entre cada par de letras del criptograma y se calcula la posición de su fila y columna dentro de la matriz, en la línea 14 la letra se agrega a la cadena del texto en claro.

4.3.3. César

En la clase CyCesar el valor del `self.alfabeto` es `@"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`. Para cifrar el texto en claro se usa el siguiente método.

```
1  - (NSString *)getCiphertextWithPlaintext:(NSString *)plaintext
2  {
3      NSMutableString *result = [NSMutableString stringWithCapacity:[
           plaintext length]];
4      NSRange range;
5      for (int i = 0; i < [plaintext length]; i++) {
6          range = [self.alfabeto rangeOfString:[plaintext
               substringWithRange:NSMakeRange(i, 1)]];
7          if (range.location != NSNotFound) {
8              [result appendString:[self.alfabeto substringWithRange:
                   NSMakeRange(getNextIndex(range.location, self.
                   desplazamiento, [self.alfabeto length]), 1)]];
9          }
10     }
11     return result;
12 }
```

En el `for` se itera entre cada letra del texto en claro y se encuentra su índice dentro del alfabeto, después la función `getNextIndex` calcula el índice correspondiente a la letra cifrada dependiendo del desplazamiento actual y la agrega a la cadena del texto cifrado.

La implementación de la función `getNextIndex` es la siguiente:

```
1  static inline NSInteger getNextIndex(NSUInteger index, NSInteger
       desplazamiento, NSUInteger length)
2  {
3      NSInteger new = index + desplazamiento;
4      if (new < 0) {
5          new = new + length;
6      }
7      else if (new >= length){
8          new = new - length;
9      }
```

```
10      return new;
11  }
```

El nuevo índice se calcula sumando el índice con el desplazamiento actual, si el nuevo índice es menor a 0 o mayor a la longitud del alfabeto se le suma o resta, respectivamente, la longitud del alfabeto.

La implementación de `getPlaintextWithCypherText` es realmente un mensaje a `getCiphertextWithPlaintext` y aparece solamente para evitar confusiones con el usuario de la clase.

En el momento de descifrar un criptograma se usa el siguiente método:

```
1  - (NSInteger)getInverseOffset: (NSInteger)desplazamiento
2  {
3      return [self.alfabeto length] - desplazamiento;
4  }
```

Simplemente le resta el desplazamiento actual a la longitud del alfabeto para obtener un desplazamiento inverso al que se usó para descifrar.

4.3.4. Disco de Alberti

La clase CyAlberti usa una instancia de la clase AlbertiDisk que contiene la información de los alfabetos externo e interno que es utilizada en el siguiente método para cifrar un texto en claro.

```
1  - (NSString *)getCiphertextWithPlaintext:(NSString *)plaintext
2  {
3      NSMutableString *string = [NSMutableString stringWithCapacity:
           plaintext.length];
4      NSInteger offset;
5      NSInteger innerIndex;
6      for (NSUInteger index = 0; index < plaintext.length; index++) {
7          offset = [self inverseIndexWithIndex:[self.disk.
               alfabetoExterno rangeOfString:[plaintext
               substringWithRange:NSMakeRange(index, 1)]].location];
8          innerIndex = (index / self.turnFrequency) * self.displacement;
9          innerIndex *= self.turnLeft ? -1 : 1;
10         innerIndex += self.initialPosition;
```

```
11          innerIndex += offset;
12          innerIndex = [self normalizedIndex:innerIndex];
13          innerIndex = [self inverseIndexWithIndex:innerIndex];
14          [string appendString:[self.disk.alfabetoInterno
                substringWithRange:NSMakeRange(innerIndex, 1)]];
15      }
16      return [NSString stringWithString:string];
17  }
```

El método itera entre cada letra del texto en claro y en la línea 7 establece el valor a `offset` calculado con el método `inverseIndexWithIndex` el cual devuelve el índice en el sentido contrario, ya sea en el sentido de las manecillas del reloj o en contra de ellas, al sentido del índice de una letra en el alfabeto externo, la razón de usar los dos sentidos es debido a que el giro del disco en el sentido de las manecillas del reloj produce un ángulo positivo y es el sentido natural para llenar las letras al crear el disco lo cual tiene como consecuencia que el índice de las letras en el disco sea inverso al sentido de giro.

La implementación de `inverseIndexWithIndex` simplemente resta el índice de la longitud del alfabeto del disco.

```
18  - (NSInteger)inverseIndexWithIndex:(NSInteger)index
19  {
20      if (!index) return 0;
21      return self.disk.alfabetoExterno.length - index;
22  }
```

En la línea 8 se obtiene el valor del índice interno dividiendo su posición entre la frecuencia de giro, lo cual nos indica el número de veces que ha girado el disco interno, y multiplicándolo por el desplazamiento, en la línea 9 se multiplica por -1 si la dirección de giro es a la izquierda, en la línea 10 se le suma la posición inicial que fue establecida por el usuario previamente y se le suma el valor de `offset` para encontrar el valor final del índice en el disco interno, sin embargo, el valor necesita ser normalizado, debe ser un valor entre 0 y la longitud del alfabeto interno del disco menos 1, y ya que fue calculado como un índice inverso se llama de nuevo a la función `inverseIndexWithIndex` en la línea 13.

Para normalizar un índice se usa el siguiente método.

```
23  - (NSInteger)normalizedIndex:(NSInteger)index
```

```
24 {
25     if (index < 0) {
26         return self.size - (ABS(index)  % self.size);
27     }
28     if (index > self.size) {
29         return index  % self.size;
30     }
31     return index;
32 }
```

Si el índice ya se encuentra en un valor entre 0 y la longitud del alfabeto externo del disco actual, el valor de `self.size` es igual a la longitud del alfabeto, el valor del índice no cambia, debido a que el valor de `index` no esta limitado en la clase se debe usar el operador módulo entre el valor absoluto y la longitud del alfabeto restado de la longitud del alfabeto si el índice es menor a 0, en caso de que el índice sea mayor a la longitud del alfabeto el nuevo índice es simplemente el módulo de `index` entre la longitud del alfabeto.

4.3.5. Vigenère

La implementación del método `getCyphertextWithPlainText` requiere que la longitud de la clave sea mayor a 0, en la siguiente sección de código la línea 5 comprueba la longitud y en caso de ser igual a 0 regresa un `NSString` vacío.

```
1  - (NSString *)getCyphertextWithPlainText:(NSString *)plaintext
2  {
3      NSMutableString *result = [NSMutableString stringWithCapacity:[
           plaintext length]];
4      NSUInteger index = 0;
5      if ([self.key length]) {
6          for (NSUInteger element = 0; element < [plaintext length];
               element++) {
7              NSUInteger row = [self.alfabeto rangeOfString:[self.key
                   substringWithRange:NSMakeRange(index, 1)]].location
                   ;
8              NSUInteger column = [self.alfabeto rangeOfString:[
                   plaintext substringWithRange:NSMakeRange(element, 1
                   )]].location;
9              if (row != NSNotFound && column != NSNotFound) {
10                 NSInteger new = column + row;
11                 if (new < 0) {
12                     new = new + [self.alfabeto length];
13                 }
14                 else if (new >= [self.alfabeto length]){
```

```
15                      new = new - [self.alfabeto length];
16                  }
17                  [result appendString:[self.alfabeto substringWithRange
                        :NSMakeRange(new, 1)]];
18                  index++;
19                  if (index >= [self.key length]) {
20                      index = 0;
21                  }
22              }
23          }
24      }
25      return [NSString stringWithString:result];
26  }
```

Dentro del `for` de la línea 6 se itera entre cada letra del texto en claro y en la línea 7 se calcula la fila de la letra actual de la clave, la cual va cambiando junto con la letra del texto en claro, en la línea 8 se encuentra la columna a la que pertenece la letra del texto en claro, si los índices de la fila y columna son valores válidos, el índice del texto cifrado se calcula sumando la fila con la columna en la línea 10, debido a que los valores de fila y columna son positivos y pueden ir de 0 hasta la longitud del alfabeto menos 1, el nuevo valor del índice calculado tiene que ser normalizado para que no sea mayor a la longitud del alfabeto, esto se realiza en la línea 15, en la línea 17 se agrega la letra del texto cifrado a la cadena `result`, finalmente se incrementa el valor del índice de la clave en la línea 19.

4.3.6. Máscaras Rotativas

En el caso de Máscaras rotativas el controlador no mantiene la estructura de datos que representa las letras dentro de la matriz, en cambio esta estructura es una matriz cuadrada con tamaño igual al máximo número de filas y columnas disponibles para el dispositivo, 10 en el caso del iPad y 8 en el caso del iPhone, llamada `_textMatrix`, debido a esto el controlador tiene que pedir a la clase CyMascaras la cadena que representa el texto en claro, la implementación del método `getCriptograma` es la siguiente:

```
1  - (NSString *)getCriptograma
2  {
3      NSMutableString *criptograma = [NSMutableString stringWithCapacity
           :self.size * self.size];
4      for (NSUInteger row = 0; row < self.size; row++) {
5          for (NSUInteger col = 0; col < self.size; col++) {
6              [criptograma appendString:_textMatrix[row][col]];
```

```
7           }
8       }
9       return [NSString stringWithString:[criptograma
                stringByTrimmingCharactersInSet:[NSCharacterSet
                whitespaceCharacterSet]]];
10  }
```

Simplemente itera entre las columnas y filas de `_textMatrix` y agrega la letra a la cadena `criptograma`.

Para obtener el texto en claro se utiliza el método `getPlaintext`, su implementación es la siguiente:

```
1   - (NSString *)getPlaintext
2   {
3       NSUInteger nextRow, nextColumn;
4       NSMutableString *result = [NSMutableString stringWithCapacity:self
                .size * self.size];
5       for (NSUInteger state = 0; state < 4; state++) {
6           for (NSUInteger row = 0; row < self.size; row++) {
7               for (NSUInteger col = 0; col < self.size; col++) {
8                   switch (state) {
9                       case 0:
10                          nextRow = row;
11                          nextColumn = col;
12                          break;
13
14                      case 1:
15                          nextRow = self.size - 1 - col;
16                          nextColumn = row;
17                          break;
18
19                      case 2:
20                          nextRow = self.size - 1 - row;
21                          nextColumn = self.size -1 - col;
22                          break;
23
24                      case 3:
25                          nextRow = col;
26                          nextColumn = self.size - 1 - row;
27                  }
28                  if (_maskMatrix[nextRow][nextColumn] ==
                        CIPGrilleMaskStateUncovered) {
29                      [result appendString:_textMatrix[row][col]];
30                  }
31              }
32          }
33      }
```

```
34      return [[NSString stringWithString:result]
              stringByTrimmingCharactersInSet:[NSCharacterSet
              whitespaceCharacterSet]];
35  }
```

El método itera entre cada una de las 4 posiciones posibles de la máscara en el `for` de la línea 5 y entre cada fila y columna de la matriz en los `for` de las líneas 6 y 7, para cada entrada de la matriz se calcula la fila y columna para un estado determinado que ocuparán la fila y columna del estado inicial, el estado inicial o estado 0 corresponde con el arreglo bidimensional que representa la matriz de texto, estos nuevos valores representan la fila y columna que ocupará la entrada en los estados de rotación siguientes, finalmente se usan estos valores en la línea 28 para comprobar si un elemento de la máscara, representada también como un arreglo bidimensional en la variable `_maskMatrix`, se encuentra en un estado descubierto, en caso de ser así, se agrega la letra de la matriz de texto a la cadena resultante.

4.3.7. Rueda de Jefferson

La clase CyJefferson utiliza una serie de 26 discos representados cada uno como una clase JeffersonDisk, la cual mantiene una cadena de texto predefinida y constante con un orden particular, además de otras propiedades como un identificador y una posición, que son colocados en orden en un arreglo llamado `diskArray` cuando se crea una instancia de CyJefferson, ya que los usuarios de la clase requieren modificar el orden de los discos, se tiene un método para realizar esta acción llamado `changeDiskAtIndex:with`, su implementación es la siguiente:

```
1  - (BOOL)changeDiskAtIndex:(NSInteger)index0 with:(NSInteger)index1
2  {
3      if (index0 < 0 || index0 >= kSize || index1 < 0 || index1 >= kSize
              ) {
4          return NO;
5      }
6      JeffersonDisk *temp = [self.diskArray objectAtIndex:index0];
7      [self.diskArray replaceObjectAtIndex:index0 withObject:[self.
              diskArray objectAtIndex:index1]];
8      [self.diskArray replaceObjectAtIndex:index1 withObject:temp];
9      return YES;
10 }
```

El método inicia comprobando en la línea 3 si los índices de los discos a intercambiar son válidos, de no ser así, el método regresa un `NO` para indicar que no se ejecutó co-

rrectamente, si los índices son correctos, se realiza el intercambio de estos en las líneas 6 a 8 y se devuelve un `YES` como respuesta.

Para establecer la posición a un disco en particular se requiere hacerlo por medio del método `setPosition:toDiskAtIndex` de la clase CyJefferson, la impementación es la siguiente:

```
1 - (void)setPosition:(NSInteger)position toDiskAtIndex:(NSInteger)index
2 {
3     if (index < 0 || index >= kSize) {
4         return;
5     }
6     ((JeffersonDisk *)[self.diskArray objectAtIndex:index]).position =
              position;
7 }
```

En la línea 3 se comprueba si el índice es válido, de ser así, se establece la nueva posición en la línea 6.

Para obtener el texto en claro y el texto cifrado se le pide a la clase el texto en una determinada fila, esta fila representa un índice dentro del texto en cada objeto de la clase JeffersonDisk, el siguiente método pertenece a la clase CyJefferson.

```
1 - (NSString *)getTextAtIndex:(NSUInteger)index
2 {
3     NSMutableString *ciphertext = [NSMutableString stringWithCapacity:
            kSize];
4     for (JeffersonDisk *disk in self.diskArray) {
5         [ciphertext appendString:[disk encrypt:index]];
6     }
7     return [NSString stringWithString:ciphertext];
8 }
```

En la línea 5 agrega el resultado de la llamada al método `encrypt` de cada objeto de la clase JeffersonDisk dentro del arreglo `diskArray`.

El siguiente bloque de código muestra la implementación del método `encrypt de la clase JeffersonDisk`:

```
1 - (NSString *)encrypt:(NSInteger)index
2 {
3     index = [self normalizeIndex:self.position + index];
```

```
4       return [self.permutation substringWithRange:NSMakeRange(index, 1)
                ];
5   }
```

Primero le asigna a `index` el valor normalizado de la suma de la posición actual del disco con el índice solicitado dentro de la cadena que representa las letras del disco y en la línea 4 regresa la letra correspondiente al nuevo índice calculado.

4.3.8. Playfair

La clase CyPlayfair contiene una matriz cuadrada llamada `_key` que representa la matriz de texto de la clave, en el siguiente método se usa esta matriz para construir el texto cifrado.

```
1   - (NSString *)getCyphertexWithPlaintext:(NSString *)plaintext error:(
          NSError **)error
2   {
3       if (![self isKeyComplete]) {
4           *error = [NSError errorWithDomain:@"Cypher error" code:70
                    userInfo:@{NSLocalizedDescriptionKey:@"Invalid Key"}];
5           return nil;
6       }
7       NSMutableString *result = [NSMutableString stringWithCapacity:[
                plaintext length]];
8       plaintext = [[plaintext stringByReplacingOccurrencesOfString:@" "
                withString:@""] stringByReplacingOccurrencesOfString:@"J"
                withString:@"I"];
9       NSString *text;
10      NSUInteger firstRow, firstColumn, secondRow, secondColumn;
11      for (NSUInteger index = 0; index < [plaintext length] / 2; index
                ++) {
12          text = [plaintext substringWithRange:NSMakeRange(index * 2, 2)
                    ];
13          if (![self getCyphertextIndicesWithText:text nextFirstRow:&
                    firstRow nextFirstColumn:&firstColumn nextSecondRow:&
                    secondRow nextSecondColumn:&secondColumn]) {
14              *error = [NSError errorWithDomain:@"Cypher error" code:72
                        userInfo:@{NSLocalizedDescriptionKey:@"Plaintext
                        error"}];
15              return nil;
16          }
17          [result appendString:_key[firstRow * kSize + firstColumn]];
18          [result appendString:_key[secondRow * kSize + secondColumn]];
19      }
20      return [NSString stringWithString:result];
```

```
21 }
```

El método comienza por comprobar si todos los elementos de la matriz han sido establecidos, en caso de no ser así se crea un error y se regresa `nil`, en la línea 8 se redefine el valor de `plaintext` a partir de su valor original eliminando los espacios en blanco y sustituyendo las ocurrencias de "J" con "I", en el `for` de la línea 11 se itera entre cada par de letras del texto en claro que son usados por el método `getCyphertextIndicesWithText:` para calcular los índices del par de letras cifradas aplicando las reglas del cifrado de Playfair, si el método devuelve un `YES`, el método fue ejecutado correctamente, se agregan a la cadena resultante las letras del texto cifrado en las líneas 17 y 18.

Para encontrar el par de letras que corresponden al texto cifrado se aplican las reglas de Playfair en el siguiente método:

```
1  - (BOOL)getCyphertextIndicesWithText:(NSString *)text nextFirstRow:(
         NSUInteger *)nextFirstRow nextFirstColumn:(NSUInteger *)
         nextFirstColumn nextSecondRow:(NSUInteger *)nextSecondRow
         nextSecondColumn:(NSUInteger *)nextSecondColumn
2  {
3      if ([text length] != 2)
4          return NO;
5      NSUInteger firstIndex = [self indexOfString:[text substringToIndex
           :1]];
6      if (firstIndex == NSNotFound)
7          return NO;
8      NSUInteger firstRow = firstIndex / kSize;
9      NSUInteger firstColumn = firstIndex % kSize;
10     NSUInteger secondIndex = [self indexOfString:[text
           substringFromIndex:1]];
11     if (secondIndex == NSNotFound || secondIndex == firstIndex)
12         return NO;
13     NSUInteger secondRow = secondIndex / kSize;
14     NSUInteger secondColumn = secondIndex % kSize;
15     // Segunda regla
16     if (firstRow == secondRow) {
17         *nextFirstRow = firstRow;
18         *nextSecondRow = firstRow;
19         *nextFirstColumn = firstColumn + 1;
20         *nextSecondColumn = secondColumn + 1;
21         if (*nextFirstColumn >= kSize)
22             *nextFirstColumn = 0;
23         if (*nextSecondColumn >= kSize)
24             *nextSecondColumn = 0;
25     }
26     // Tercera regla
27     else if (firstColumn == secondColumn) {
```

```
28          *nextFirstColumn = firstColumn;
29          *nextSecondColumn = firstColumn;
30          *nextFirstRow = firstRow + 1;
31          *nextSecondRow = secondRow + 1;
32          if (*nextFirstRow >= kSize)
33              *nextFirstRow = 0;
34          if (*nextSecondRow >= kSize)
35              *nextSecondRow = 0;
36      }
37      // Primera regla
38      else {
39          *nextFirstRow = firstRow;
40          *nextFirstColumn = secondColumn;
41          *nextSecondRow = secondRow;
42          *nextSecondColumn = firstColumn;
43      }
44      return YES;
45  }
```

El método recibe una cadena de longitud 2 en la variable `text` y 4 apuntadores a NSUInteger en los cuales se va a establecer las posiciones dentro de la matriz `_key` que corresponden a las letras cifradas, `nextFirstRow` y `nextFirstColumn` para la primera letra y `nextSecondRow` y `nextSecondColumn` para la segunda.

En la línea 3 se comprueba que la longitud de `text` sea igual a 2 y en caso de no ser así se devuelve un `NO` en la línea 4, en la línea 5 se obtiene el índice dentro de la matriz que corresponde a la primera letra, el índice es un número que puede ir desde 0 hasta `(kSize * kSize) - 1` donde `kSize = 5` tomando en cuenta que la matriz es recorrida por filas, en la línea 6 se comprueba si la letra existe dentro de la matriz y de no ser así, se devuelve un `NO`, si la letra se encuentra dentro de la matriz se calcula su fila y columna en las líneas 8 y 9, en las líneas 10 a 14 se sigue un procedimiento similar para la segunda letra. El `if` de la línea 16 comprueba si la fila de la primera letra es igual a la fila de la segunda letra, en ese caso se aplica la primer regla del cifrado y se establecen las nuevas filas y columnas para las letras cifradas, en caso de que cualquiera de los índices de las columnas sea mayor a kSize la columna se hace 0, en la línea 27 se comprueba la tercera regla y se aplican los valores de fila y columna para las letras cifradas en un proceso similar a la segunda regla pero en lugar de columnas se usan filas. Finalmente si se ha llegado al `else` de la línea 38 se aplica la primera regla y se devuelve un `YES` para indicar que el método se ejecutó de manera correcta.

4.3.9. Enigma

Cada clase que hereda de la clase abstracta `Enigma` basa su funcionamiento en la clase `Rotor`, existe en cada implementación de Enigma un arreglo que contiene los rotores que utiliza, al cifrar, cada rotor realiza una sustitución monoalfabética particular para cada tipo de rotor, debido a que en el proceso la corriente viaja desde el rotor rápido (el que se encuentra a la derecha) hacia el reflector y de regreso, cada rotor tiene una referencia al rotor que se encuentra a su izquierda y otra al rotor a su derecha, de este modo el objecto de la clase Enigma llama al método `encryptLeft` de su primer rotor (rotor rápido) el cual llama al mismo método del rotor a su derecha hasta llegar al reflector que devuelve la letra cifrada de regreso al rotor más a la izquierda para realizar la segunda sustitución y devolver la letra a su rotor a la derecha hasta llegar al rotor rápido el cual realiza de nuevo una sustitución y devuelve el resultado al objecto enigma. La siguiente sección de código muestra el proceso.

```
1 - (NSString *)encryptLeft:(NSString *)text
2 {
3     NSInteger index = [alphabet rangeOfString:text].location;
4     NSString *result = [self.wiring substringWithRange:NSMakeRange([
              self normalizeIndex:index + self.position - self.
              ringSetting], 1)];
5     index = [alphabet rangeOfString:result].location;
6     index = (index + self.ringSetting - self.position);
7     index = [self normalizeIndex:index];
8     return [self encryptRight:[self.leftCipher encryptLeft:[alphabet
              substringWithRange:NSMakeRange(index, 1)]]];
9 }
```

En la línea 3 se obtiene el índice dentro del alfabeto normal de la letra a cifrar y se utiliza en la línea 4 para realizar la sustitución definida por el cableado del rotor que se encuentra en la variable `wiring`, antes de obtener la letra se le suma al índice la posición actual del rotor y se le resta su ring setting . En la línea 5 se asigna a la variable `index` la posición dentro del alfabeto del resultado de la sustitución anterior y en la siguiente línea se le suma el ring setting y se le resta su posición para compensar la suma de la línea 4, debido a las sumas realizadas el índice puede ser un valor menor a 0 o mayor a la longitud del alfabeto, por ello se normaliza en la línea 7. En la línea 8 se llama al método `encryptLeft` del rotor a la izquierda el cual realiza los mismos pasos hasta llegar al reflector que realiza una sustitución similar a la de la línea 4 y devuelve la letra al reflector que en la misma línea 8 llama a su método `encryptRight` que se describe a continuación:

```
1 - (NSString *)encryptRight:(NSString *)text
2 {
```

```
3       NSInteger index = [alphabet rangeOfString:text].location;
4       NSString *result = [alphabet substringWithRange:NSMakeRange([self
                normalizeIndex:index + self.position - self.ringSetting], 1
                )];
5       index = [self.wiring rangeOfString:result].location;
6       index = (index + self.ringSetting - self.position);
7       index = [self normalizeIndex:index];
8       return [alphabet substringWithRange:NSMakeRange(index, 1)];
9   }
```

En la línea 3 se obtiene el índice de la letra devuelta por el rotor a la izquierda o el reflector en el caso del rotor más a la izquierda y en la línea 4 se le suma la posición del reflector y se le resta su ring setting para obtener la letra que se utiliza en la línea 5 para realizar la sustitución, en la línea 6 se le suma el ring setting y se le resta su posición para compensar las sumas de la línea 4, en la siguiente línea se normaliza en índice y por último se devuelve la letra del alfabeto que corresponde al índice.

Dependiendo del tipo de máquina enigma que se esté utilizando puede existir un plugborad, el cual está representado por la clase `Plugboard`, que puede realizar una simple sustitución antes y después del cifrado que realizan los rotores, en las máquinas que utilizan el plugboard existe una instancia de la clase `Plugboard` que mantiene un set de conexiones representadas por la clase `PlugboardConnection`, si una letra existe dentro del set de conexiones el plugboard utiliza el siguiente método para cifrarla:

```
1   - (NSString *)encrypt:(NSString *)character
2   {
3       // Si existe la conexion se cifra la letra
4       for (PlugboardConnection *connection in self.connections) {
5           if ([connection.start isEqualToString:character]) {
6               return connection.end;
7           } else if ([connection.end isEqualToString:character]) {
8               return connection.start;
9           }
10      }
11      return character;
12  }
```

En la línea 4 se itera entre cada conexión existente en el set `connections`, si la letra es el comienzo o el fin de la conexión se devuelve la letra correspondiente al otro extremo de la conexión, si no se encuentra la letra en alguna conexión del plugborad simplemente se devuelve la letra sin cambios.

Otro método importante de la clase enigma es el de `notch` que realiza el giro de los rotores dependiendo de unas muescas particulares para cada rotor, estas muescas ha-

cen que los rotores giren cambiando su posición dependiendo de la posición que ocupen dentro del arreglo de rotores de la máquina enigma. El rotor de la derecha gira cada vez que el usuario presiona una tecla, cuando el mecanismo de la máquina encuentra una muesca en este rotor hace que el rotor a su izquierda gire también, a su vez si el segundo rotor se encuentra en una posición con una muesca el rotor a su izquierda gira, debido a este mecanismo el rotor más a la derecha gira con más frecuencia que el rotor a la izquierda. Es importante notar que este giro se realiza antes de cerrar el circuito que enciende la lámpara de la letra cifrada. A continuación se muestra la implementación de este mecanismo:

```
1  - (void)notch
2  {
3      if ([[self.rotors objectAtIndex:0] isInNotchPosition] || [[self.
               rotors objectAtIndex:1] isInNotchPosition]) {
4          if ([[self.rotors objectAtIndex:1] isInNotchPosition]) {
5              [[self.rotors objectAtIndex:2] rotate];
6          }
7          [[self.rotors objectAtIndex:1] rotate];
8      }
9      [[self.rotors objectAtIndex:0] rotate];
10 }
```

En la línea 3 se comprueba si el rotor 0 (derecha) o el rotor a su izquierda (rotor 1) se encuntran en una posición con muesca, si es así, el rotor 1 gira en la línea 7, antes del giro del rotor 1 se comprueba si éste esta en una posición con muesca, de ser así, el rotor a su izquierda (rotor 2) gira en la línea 5, por último en la línea 9, debido a que el rotor de la derecha siempre gira cada vez que se presiona una tecla, el rotor 0 gira.

A continuación se muestran las implementaciones del método `encryptCharacter` de cada tipo de máquina enigma presentes en la aplicación.

Enigma I

```
1  - (NSString *)encryptCharacter:(NSString *)character
2  {
3      [self notch];
4      NSString *plugboardCharacter = [self.plugboard encrypt:character];
5      return [self.plugboard encrypt:[[self.rotors objectAtIndex:2]
               encryptLeft:plugboardCharacter]];
6  }
```

Enigma G

```
1 - (NSString *)encryptCharacter:(NSString *)character
2 {
3     [self notch];
4     return [self.etw encryptLeft:character];
5 }
```

Antes de realizar el cifrado se llama al método `notch` de la máquina para hacer girar los rotores necesarios, en la línea 4 se realiza el cifrado sin un plugboard debido al tipo de enigma. La variable `etw` es de tipo `Rotor` y representa el mecanismo de entrada al primer rotor llamado Entry Wheel que realiza una sustitución, este dispositivo nunca gira.

Enigma K

```
1 - (NSString *)encryptCharacter:(NSString *)character
2 {
3     [self notch];
4     return [self.etw encryptLeft:character];
5 }
```

La implementación de `encryptCharacter` es similar a la de la Enigma G.

Enigma M3

```
1 - (NSString *)encryptCharacter:(NSString *)character
2 {
3     [self notch];
4     NSString *plugboardCharacter = [self.plugboard encrypt:character];
5     return [self.plugboard encrypt:[[self.rotors objectAtIndex:0]
              encryptLeft:plugboardCharacter]];
6 }
```

En la línea 4 se obtiene la letra cifrada por el plugboard y se utiliza en la siguiente línea por el rotor de la derecha, al terminar el cifrado, se devuelve la letra al plugboard, en caso de que exista una conexión se cifra la letra.

Enigma M4

```
1 - (NSString *)encryptCharacter:(NSString *)character
2 {
3     [self notch];
```

```
4       NSString *plugboardCharacter = [self.plugboard encrypt:character];
5       return [self.plugboard encrypt:[[self.rotors objectAtIndex:0]
                encryptLeft:plugboardCharacter]];
6   }
```

La implementación de `encryptCharacter` es similar a la de la Enigma M3.

4.3.10. Controladores

Cada una de las técnicas antes mencionadas están representadas por una clase con el prefijo Cy encargada de cifrar el mensaje en claro y que corresponde a la parte de modelo en el patrón MVC, para cada una de estas técnicas existe un controlador que mantiene un objeto de una de estas clases que se encarga de procesar los eventos y mantener sincronizados el modelo y las vistas.

La forma general en que funcionan estos controladores en la mayoría de los nueve métodos de cifrado consiste en mantener un componente de vista llamado `plaintext` de tipo `UITextView` que se encarga de mostrar al usuario un campo de texto, este componente contiene una cadena de texto llamada `text` que usamos para representar el mensaje en claro, cuando el usuario intenta modificar esta cadena de texto el objeto `plaintext` llama al método `textView:shouldChangeTextInRange: replacementText:` que se encuentra en el controlador el cual consulta al objeto modelo si la letra que intenta cambiar se encuentra dentro de su alfabeto, en caso de ser así, el controlador permite que se cambie el texto en claro y solicita al modelo la nueva cadena que representa el texto cifrado, en cada controlador la implementación de este método se encarga de manejar los casos particulares de la técnica de cifrado que representa, por ejemplo el caso del par de letras IJ en Polybio o el límite del mensaje en Jefferson. El proceso antes descrito se muestra en la figura 4.6.

Cuando el usuario guarda el criptograma el controlador solicita al modelo crear un objeto de tipo Criptograma correspondiente a la técnica de cifrado actual, este objeto es administrado por Core data y se guarda en una base de datos de SQLite dentro de los archivos de la aplicación.

Otra de las cosas importantes de las que se encarga el controlador es actualizar el modelo y las vistas cuando el usuario selecciona un criptograma para descifrar y de validar el texto en claro cada vez que se modifica para indicar al usuario si es correcto.

Además de todo lo anterior existen una gran cantidad de otras tareas que realiza el controlador por ejemplo: mostrar la ayuda al usuario, actualizar el acomodo de los componentes de vista cuando se cambia la orientación del dispositivo, realizar algunas animaciones en casos particulares y suministrar la información necesaria a la vista que muestra la lista de criptogramas, entre otras.

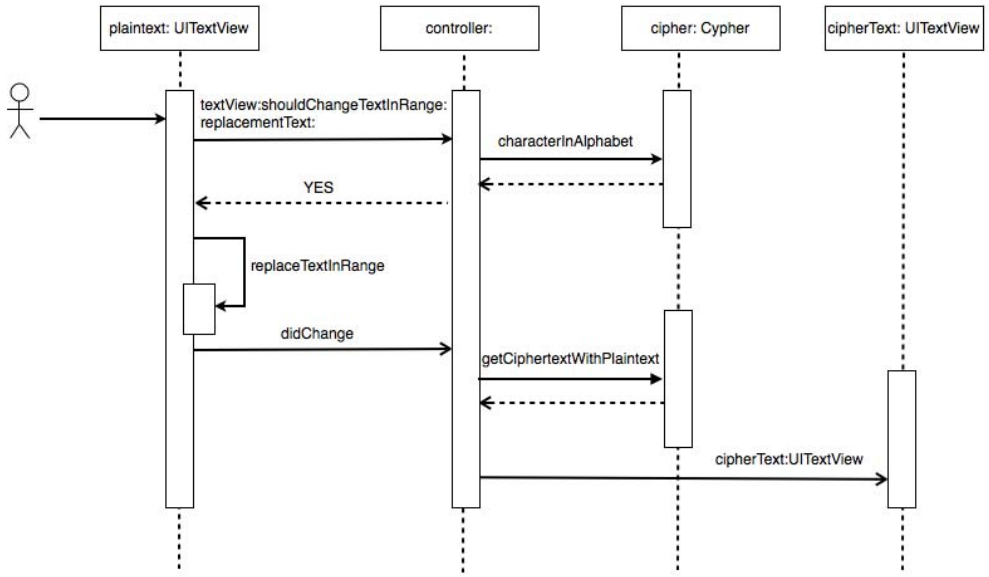


Figura 4.6: Diagrama de secuencia general de los controllers

Capítulo 5

Pruebas

5.1. Introducción

El presente capítulo muestra los resultados de las pruebas realizadas en la aplicación para demostrar el correcto funcionamiento de los algoritmos de cada método. La sección 5.2 presenta las pruebas lógicas realizadas en los algoritmos encargados de cifrar y descifrar cadenas de texto, éstas fueron hechas con el framework SenTestingKit que proporciona Xcode. La sección 5.3 muestra los tiempos aproximados que tardan los algoritmos en realizar su función.

Además de estas pruebas la aplicación fue probada por usuarios a través de la plataforma TestFlight, la cual es usada para distribuir versiones beta de aplicaciones y es gratuita.

En la dirección `http://cipherappunam.blogspot.mx` se muestra una breve descripción de la aplicación y sus características así como el link para descargar la aplicación.

5.2. Resultados de las pruebas unitarias

Las líneas que comienzan con `Test Case` son una salida directa, sin modificaciones, mostrada en la consola al momento de ejecutar las pruebas.

5.2.1. Scítala

Prueba de cifrado de una cadena válida

El resultado de cifrar la cadena:

LACRIPTOGRAFIAESLACIENCIAENCARGADADEL
OCULTAMIENTODELAINFORMACION

Estableciendo 8 filas y 8 vueltas, debe ser igual a:

LGLADLOOARAEATDRCACNDAEMRFICEMLAIIEAL
IACPANROEIITECGCNNOOSIAUTFN

Test Case '-[ScitalaTests testEncryptValidString]' passed (0.002 seconds).

Prueba de descifrado de una cadena válida

El mensaje cifrado es:

PSAOCRSRICGOAIISRRPNCMTIAOSIEEPFRPORMTITON

Estableciendo 7 filas y 6 vueltas, el mensaje descifrado:

PRIMERSISTEMACRIPTOGRAFICOPORTRANSPOSICION

Test Case '-[ScitalaTests testDecryptValidString]' passed (0.001 seconds).

5.2.2. Polybio

Prueba de cifrado de una cadena válida

El resultado de cifrar la cadena:

CRI/JPTOGRAFI/JA

debe ser igual a:

ACDBBDCEDDCDBBDBAABABDAA

Test Case '-[PolybioTests testEncryptValidString]' passed (0.000 seconds).

Para la opción de cifrado con números, se ocupa la cadena anterior teniendo como resultado:

134224354434224211212411

Test Case '-[PolybioTests testEncryptValidStringWithNumbers]' passed (0.000 seconds).

Prueba de cifrado de una cadena inválida

Se introducen caracteres inválidos, los cuales debe ignorar.

La cadena a cifrar es:

CRI/J#PTOÑGRA@FIJA$

y debe ser igual a:

ACDBBDCEDDCDBBDBAABABDAA

Test Case '-[PolybioTests testEncryptInvalidString]' passed (0.000 seconds).

Para la opción de cifrado con números, se ocupa la cadena anterior teniendo como resultado:

1342243544342242112124 11

Test Case '-[PolybioTests testEncryptInvalidStringWithNumbers]' passed (0.000 seconds).

Prueba de descifrado de una cadena válida

El mensaje cifrado es:

AACABBCDDBBDDDCBCD

Mensaje descifrado:

ALGORITMO

Test Case '-[PolybioTests testDecrypt]' passed (0.000 seconds).

Para la opción de cifrado con números.

El mensaje cifrado es:

113122344224443234

Mensaje descifrado:

ALGORITMO

Test Case '-[PolybioTests testDecryptValidString]' passed (0.000 seconds).

Prueba de descifrado de una cadena inválida

Cuando la longitud de la cadena es impar, significa que es inválida y el resultado de descifrarla es nil.

El mensaje cifrado es:

AACABBCDDBBDDDCBC

Test Case '-[PolybioTests testDecryptInvalidString]' passed (0.000 seconds).

Lo mismo pasa con la opción de descifrado con números.

El mensaje cifrado es:

11312234422444323

Test Case '-[PolybioTests testDecryptInvalidStringWithNumbers]' passed (0.000 seconds).

5.2.3. César

Prueba de cifrado de una cadena válida

El resultado de cifrar la cadena:

CIFRADO DE CESAR

con un desplazamiento de 3

debe ser igual a:

FLIUDGRGHFHVDU

Test Case '-[CesarTests testEncryptValidString]' passed (0.000 seconds).

Prueba de cifrado de una cadena inválida

Se introducen caracteres inválidos, los cuales debe ignorar.

CIFRA@DO# DE CES$AR

con un desplazamiento igual a 3 y debe ser igual a:

FLIUDGRGHFHVDU

Test Case '-[CesarTests testEncryptInvalidString]' passed (0.000 seconds).

Prueba de descifrado

Para realizar el proceso de descifrado, se consideran los siguientes datos:

Mensaje cifrado:

FLIUDGR SRU VXVWLWXFLRQ

con un desplazamiento igual a 3.

Mensaje en claro:

CIFRADO POR SUSTITUCIÓN

Test Case '-[CesarTests testDecryptValidString]' passed (0.000 seconds).

5.2.4. Disco de Alberti

Prueba de cifrado de una cadena válida

El resultado de cifrar la cadena:

CIFRADO DE ALBERTI

con el disco que se tiene por default, estableciendo la letra g que coincida con la letra A del disco externo, con una frecuencia de 4, desplazamiento igual a 3 y dirección a la derecha, debe ser igual a:

kcyx vglg s&ke eycs

Test Case '-[CesarTests testDecryptValidString]' passed (0.000 seconds).

Prueba de cifrado de una cadena inválida

Cuando se introducen caracteres inválidos,el resultado es nil y se produce un error.

Mensaje en claro:

DISCO$DEA@#LBERTI

con la clave del ejemplo anterior

Test Case '-[AlbertiTests testEncryptInvalidString]' passed (0.006 seconds).

Prueba de descifrado de una cadena válida

Se tiene el mensaje cifrado:

bcfklgpvkeslcs

y la misma clave para cifrar que es: establecer la letra$g$ para que coincida con la letra A del disco externo, una frecuencia de 4, desplazamiento de 3 y dirección a la derecha.

Mensaje descifrado:

DISCO DE ALBERTI

Test Case '-[CesarTests testDecryptValidString]' passed (0.000 seconds).

5.2.5. Cifrado de Vigenère

Prueba de cifrado de una cadena válida

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Mensaje en claro (Mcla):

CODIGO INDESCIFRABLE

estableciendo la clave (K) CRIPTO

El mensaje cifrado debe ser igual a:

EFLXZ CKELT LQKWZ PUZG

Test Case '-[VigenereTests testEncryptValidString]' passed (0.000 seconds).

Prueba de cifrado de una cadena inválida

Se introducen caracteres inválidos, los cuales debe ignorar.

Mensaje cifrado:

EFLXZ CKELT LQKWZ PUZG

estableciendo la clave (K) CRIPTO

Mensaje en claro (Mcla):

CO#DIGOINDES$CIFRA@BLE

Test Case '-[VigenereTests testEncryptInvalidString]' passed (0.000 seconds).

Prueba de descifrado

Para realizar el proceso de descifrado, se consideran los siguientes datos:

Mensaje cifrado:

SRZCX CRAQB VLIFM NMQIU

Estableciendo la clave ANTIGUO

El mensaje en claro debe ser igual a:

SEGURIDAD INFORMATICA

Test Case '-[VigenereTests testDecryptValidString]' passed (0.000 seconds).

5.2.6. Máscaras Rotativas

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Mensaje a cifrar:

RESGUARDAR Y PROTEGER LA INFORMACIÓN

Estableciendo el tamaño de la máscara en 8 y perforaciones en las coordenadas: (1,5),(2,8),(3,3),(3,8),(5,2),(6,4),(7,6),(8,1) Obteniendo una matriz con los siguientes caracteres:

AORAROSGNCERRIAEALSORYMGATPSHO
LANUNCACIRAOIAHIYUNOMAORXZDIFFITEN

Test Case '-[MascarasTests testEncryptValidString]' passed (0.000 seconds).

Prueba de descifrado

Para realizar el proceso de descifrado, se consideran los siguientes datos:

Matriz con los caracteres:

JXPGRTFRCWRTPQRAANOIGIAVD

Se establecen las perforaciones en las coordenadas: (2,4),(3,1),(5,2) y el tamaño de la máscara: 5

Obteniendo el mensaje en claro:

CRIPTOGRAFIA

Test Case '-[MascarasTests testDecryptValidString]' passed (0.000 seconds).

5.2.7. Rueda de Jefferson

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos: Mensaje en claro:

FABRICADOENSERIEPORETIENNE
Fila: 22

El mensaje cifrado resultante:

ECDPHJFCILVDMGWOMRTYLMZKOR

Test Case '-[JeffersonTests testEncryptValidString]' passed (0.000 seconds).

Prueba de descifrado

Para realizar el proceso de descifrado, se consideran los siguientes datos:

Orden de los discos: 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25
Posicion de los discos: MOSRGQJPNJVVHDGTHISWZJBSTK
Fila: 21

Mensaje en claro resultante:

PRIMERDISPOSITIVOMECANICO

Test Case '-[JeffersonTests testDecryptValidString]' passed (0.018 seconds).

5.2.8. Playfair

Prueba de cifrado de una cadena válida

Mensaje en claro:

SEGURIDAD PROVIENE DEL LATIN SECURITAS

Estableciendo la clave CIFRADO

El resultado del mensaje cifrado:

TBNZAFGCHVIEWCGMGOBMNFQRLUDRTARQFU

Test Case '-[PlayfairTests testEncryptValidString]' passed (0.000 seconds).

Prueba de cifrado de una cadena inválida

Se introducen caracteres inválidos, los cuales debe ignorar. Mensaje en claro:

S$EGURIDA@D PR#OVIENE DEL LATINÑ S$ECURITAS

Estableciendo la clave CIFRADO

El resultado mensaje cifrado:

TBNZAFGCHVIEWCGMGOBMNFQRLUDRTARQFU

Test Case '-[PlayfairTests testEncryptInvalidString]' passed (0.000 seconds).

Prueba de descifrado

Para realizar el proceso de descifrado, se consideran los siguientes datos: Mensaje cifrado:

AW MD ET AO GE GU RS LB BO

Estableciendo la clave SEGURO

El resultado del mensaje en claro es:

ENTORNO DE SEGURIDAD

Test Case '-[PlayfairTests testDecryptValidString]' passed (0.001 seconds).

5.2.9. Enigma

Se realiza sólo el proceso de cifrado, debido a que es un algoritmo simétrico, esto significa que se utiliza el mismo proceso para cifrar o descifrar mensajes.

Enigma I

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Orden de los rotores: I, II, III
Posición de los rotores: A, A, A
Reflector: B
Mensaje en claro: ARCO

El resultado del mensaje cifrado es:

BCEM

Test Case '-[EnigmaLogicTests testEnigmaI]' passed (0.000 seconds).

Enigma G312

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Orden de los rotores: I, II, III
Posición de los rotores: A, A, A

Reflector: B
Mensaje en claro: ARCO

El resultado del mensaje cifrado es:

CDBW

Test Case '-[EnigmaLogicTests testEnigmaG]' passed (0.000 seconds).

Enigma K

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:
Orden de los rotores: I, II, III
Posición de los rotores: A, A, A
Reflector: B
Mensaje en claro: ARCO

El resultado del mensaje cifrado es:

CDBW

Test Case '-[EnigmaLogicTests testEnigmaK]' passed (0.001 seconds).

Enigma M3

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Orden de los rotores: VI VII, VIII
Posición de los rotores: A, A, A
Reflector: B
Mensaje en claro: ARCO

El resultado del mensaje cifrado es:

GVSF

Test Case '-[EnigmaLogicTests testEnigmaM3]' passed (0.000 seconds).

Enigma M4

Prueba de cifrado

Para realizar el proceso de cifrado, se consideran los siguientes datos:

Orden de los rotores: Gamma, IV,V,VI
Posición de los rotores: A, A, A
Reflector: B
Mensaje en claro: ARCO

El resultado del mensaje cifrado es:

HBUA

Test Case '-[EnigmaTests testEnigmaM4]' passed (0.002 seconds).

5.3. Resultados de las pruebas de eficiencia

Todas las pruebas que se describen a continuación fueron realizadas con un iPad 3ª generación con iOS 6.1.3 utilizando Instruments 4.6 con la aplicación en modo Release.

5.3.1. Scítala

La figura 5.1 muestra el resultado del Time Profiler ocultando las llamadas a librerías del sistema en el cual se modifica el texto en claro de la scítala.

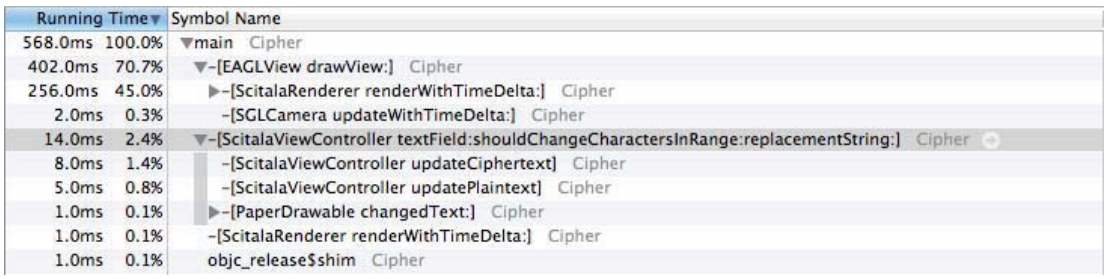


Figura 5.1: Resultado del Time Profiler para Scítala

Se realizaron 4 cambios en el texto con el máximo número de filas y vueltas disponibles que en ambos casos es igual a 10, en la línea resaltada se puede observar que el total de tiempo que se tarda en calcular el nuevo texto cifrado y en claro es de 14.0 [ms], aproximadamente 3.5 [ms] por cada cambio en el texto.

En el caso de la scítala se utiliza una vista de OpenGL ES 2.0 con 30 fps (frames por segundo), la figura 5.2 muestra el desempeño de la aplicación en un momento determinado durante el uso para cifrar de la scítala.

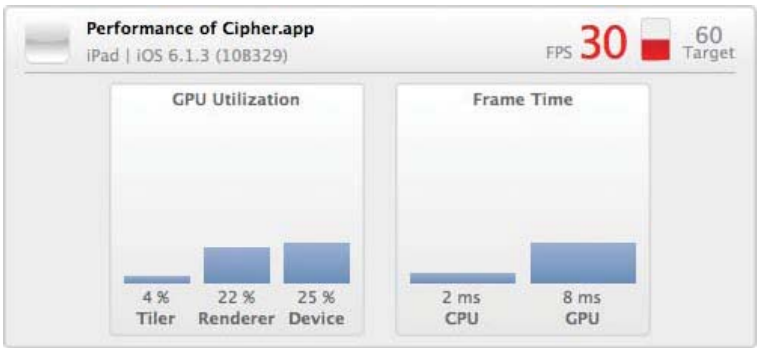


Figura 5.2: Desempeño de OpenGL para Scítala

Como se puede ver en la parte de Frame Time el tiempo que utiliza el GPU es de solo 8 [ms] de un total de 33 [ms] disponibles para el frame actual, esto nos garantiza un uso fluido de esta parte de la aplicación por parte del usuario.

5.3.2. Polybio

Las pruebas del Time Profiler en Polybio se realizaron cifrando 10 veces una cadena a partir de 200 letras.

Running Time		Symbol Name
320.0ms	98.1%	Main Thread 0xad5f0
320.0ms	98.1%	main Cipher
91.0ms	27.9%	-[PolybioMatrixView handleTap:] Cipher
76.0ms	23.3%	-[PolybioViewController matrix:didSelectElementAtRow:andColumn:] Cipher
54.0ms	16.5%	-[PolybioViewController updateCiphertext] Cipher
28.0ms	8.5%	-[CyPolybio setPlaintext:] Cipher
23.0ms	7.0%	-[CyPolybio getCiphertextWithPlaintext:] Cipher
6.0ms	1.8%	-[CyPolybio getFormattedCriptograma] Cipher
6.0ms	1.8%	+[CyPolybio getFormattedCriptogramaWithText:] Cipher
6.0ms	1.8%	+[Cypher getDefaultFormattedTextFromText:spacing:] Cipher
12.0ms	3.6%	-[PolybioMatrixView setColorsWithMainView:atRow:andColumn:active:animated:] Cipher
1.0ms	0.3%	DYLD-STUB$$objc_autoreleaseReturnValue Cipher

Figura 5.3: Resultado del Time Profiler para Polybio

En la figura 5.3 la línea resaltada en gris muestra el tiempo que tardó el método `handleTap` que es de 91 [ms] y todas las llamadas necesarias para obtener el texto cifrado, cada actualización lleva aproximadamente 9.1 [ms]. El algoritmo que realiza la

función principal de cifrar llamado `getCiphertextWithPlaintext` tarda 23 [ms] para las 10 llamadas.

5.3.3. César

La figura 5.4 muestra los resultados de cifrar una cadena de texto en claro con una longitud de 200 letras 10 veces.

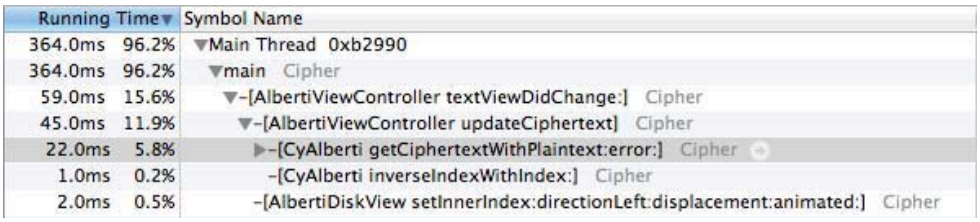
Running Time		Symbol Name
1786.0ms	98.1%	▼Main Thread 0xb019d
1786.0ms	98.1%	▼main Cipher
424.0ms	23.2%	▼-[iCarousel step] Cipher
342.0ms	18.7%	▶-[iCarousel didScroll] Cipher
38.0ms	2.0%	▼-[CesarViewController updateCiphertext] Cipher
14.0ms	0.7%	▼-[CyCesar setPlaintext:] Cipher
12.0ms	0.6%	-[CyCesar getCiphertextWithPlaintext:] Cipher
5.0ms	0.2%	▼-[CyCesar getFormattedCriptograma] Cipher
4.0ms	0.2%	▶+[CyCesar getFormattedCriptogramaWithText:] Cipher
21.0ms	1.1%	▶-[CesarViewController carouselDidEndScrollingAnimation:] Cipher
10.0ms	0.5%	▶-[iCarousel depthSortViews] Cipher
3.0ms	0.1%	-[iCarousel stopAnimation] Cipher
1.0ms	0.0%	-[iCarousel transformItemViews] Cipher

Figura 5.4: Resultado del Time Profiler para César

La línea resaltada en gris muestra el método principal llamado `getCiphertext-WithPlaintext` el cual tarda 12.0 [ms] en total por las 10 veces que cifra el mensaje en claro.

5.3.4. Disco de Alberti

En el método de Alberti se cifró una cadena mayor a 200 letras en 10 ocasiones, los resultados se muestran a continuación.



Running Time		Symbol Name
364.0ms	96.2%	Main Thread 0xb2990
364.0ms	96.2%	main Cipher
59.0ms	15.6%	-[AlbertiViewController textViewDidChange:] Cipher
45.0ms	11.9%	-[AlbertiViewController updateCiphertext] Cipher
22.0ms	5.8%	-[CyAlberti getCiphertextWithPlaintext:error:] Cipher
1.0ms	0.2%	-[CyAlberti inverseIndexWithIndex:] Cipher
2.0ms	0.5%	-[AlbertiDiskView setInnerIndex:directionLeft:displacement:animated:] Cipher

Figura 5.5: Resultado del Time Profiler para Alberti

En la figura 5.5 se observa que el tiempo de cifrado total por las 10 veces es de 22.0 [ms], aproximadamente 2.2 [ms] por ocurrencia.

5.3.5. Cifrado de Vigenère

En Vigenère se cifra de nuevo una cadena mayor a 200 letras 10 veces, en la figura 5.6 se muestran los resultados.

Running Time		Symbol Name
1191.0ms	95.7%	Main Thread 0xb45c4
1191.0ms	95.7%	main Cipher
206.0ms	16.5%	-[VignereViewController textView:shouldChangeTextInRange:replacementText:] Cipher
188.0ms	15.1%	__56-[VignereViewController highlightMatrixView:completion:]_block_invoke330 Cipher
55.0ms	4.4%	-[VignereViewController updateCiphertext] Cipher
28.0ms	2.2%	-[CyVignere setPlaintext:] Cipher
19.0ms	1.5%	-[CyVignere getCiphertextWithPlaintext:] Cipher
1.0ms	0.0%	DYLD-STUB$$objc_release Cipher
1.0ms	0.0%	-[CyVignere alfabeto] Cipher
4.0ms	0.3%	-[CyVignere getFormattedCriptograma] Cipher
2.0ms	0.1%	-[VignereViewController scrollViewDidScroll:] Cipher

Figura 5.6: Resultado del Time Profiler para Vigenère

En gris se resalta las llamadas al método `getCiphertextWithPlaintext`, cada llamada tarda aproximadamente 1.9 [ms] haciendo un total de 19.0 [ms] por las 10 llamadas aproximadamente.

5.3.6. Máscaras Rotativas

En el caso de Máscaras Rotativas se realizó la prueba usando el número máximo de celdas (100 celdas) cambiando el texto en claro 20 veces.

Running Time		Symbol Name
1146.0ms	96.7%	▼Main Thread 0xb84c1
1146.0ms	96.7%	▼main Cipher
187.0ms	15.7%	▼-[MascarasViewController textField:shouldChangeCharactersInRange:replacementString:] Cipher
39.0ms	3.2%	▼-[MascarasViewController updatePlaintext] Cipher
38.0ms	3.2%	-[MascarasViewController didChangePlaintext:] Cipher
1.0ms	0.0%	-[CyMascaras getPlaintext] Cipher

Figura 5.7: Resultado del Time Profiler para Máscaras Rotativas

En la figura 5.7 se muestra el resultado de la prueba, en este caso se cambio el texto en claro un mayor número de veces ya que el tiempo de cifrado solo aumenta con el número de celdas (fijo a 100 en este caso) y el tiempo en obtener el texto en claro era muy pequeño para aparecer en la lista de símbolos realizando solamente 10 cambios. El tiempo total de obtener el texto en claro es de aproximadamente 1.0 [ms] para las 20 modificaciones.

5.3.7. Rueda de Jefferson

En este método, el texto en claro y el criptograma siempre tienen la misma longitud (26), en la figura 5.8 se muestran los resultados de cifrar el texto en claro 10 veces.

Running Time		Symbol Name
2200.0ms	93.6%	▼Main Thread 0xb9346
2200.0ms	93.6%	▼main Cipher
125.0ms	5.3%	▼-[JeffersonViewController textViewDidChange:] Cipher
35.0ms	1.4%	▼-[JeffersonViewController updateCiphertext] Cipher
3.0ms	0.1%	-[CyJefferson getTextAtIndex:] Cipher
2.0ms	0.0%	-[JeffersonDisk indexOfCharacter:] Cipher
1.0ms	0.0%	objc_getProperty$shim Cipher
67.0ms	2.8%	▶-[JeffersonViewController saveCriptograma] Cipher
33.0ms	1.4%	▶-[JeffersonViewController pickerView:didSelectRow:inComponent:] Cipher
10.0ms	0.4%	▶-[JeffersonViewController updateCiphertext] Cipher
8.0ms	0.3%	▶-[JeffersonViewController pickerView:titleForRow:forComponent:] Cipher
6.0ms	0.2%	▶-[JeffersonViewController textView:shouldChangeTextInRange:replacementText:] Cipher
4.0ms	0.1%	▶-[JeffersonViewController encryptRowAction:] Cipher
1.0ms	0.0%	▶-[AbstractCipherController keyboardWillShow:] Cipher
1.0ms	0.0%	DYLD-STUB$$objc_msgSend Cipher

Figura 5.8: Resultado del Time Profiler para la rueda de Jefferson

El tiempo total de cifrar 10 veces el texto en claro es de 3.0 [ms].

5.3.8. Playfair

En el método de Playfair se cifró una cadena mayor a 200 letras en 10 ocasiones, los resultados se muestran a continuación.

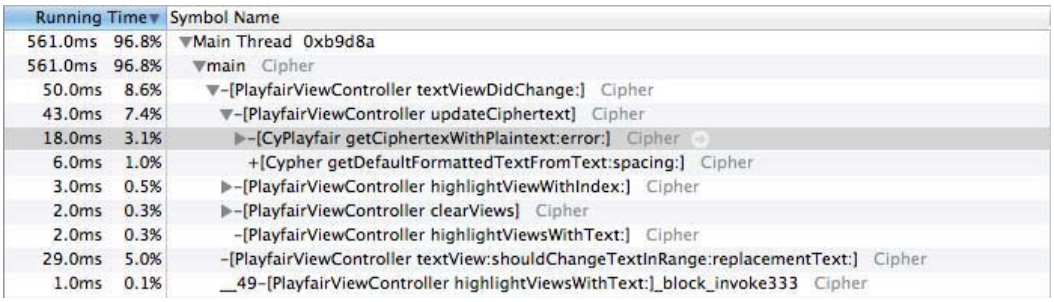


Figura 5.9: Resultado del Time Profiler para Playfair

En la figura 5.9 la línea resaltada en gris muestra el tiempo que tardó el método `getCiphertextWithPlaintext` para obtener el texto cifrado, cada llamada tarda aproximadamente 1.8 [ms]. El tiempo total es de 18 [ms] aproximadamente para las 10 llamadas.

5.3.9. Enigma

En el caso de Enigma se utilizó una instancia de Enigma M4, ya que tiene más rotores que los otros modelos, debido a que el tiempo de cifrado no aumenta con la longitud del texto en claro y de que el tiempo que tarda el método `encryptCharacter` es muy pequeño para aparecer en la lista de simbolos con pocas llamadas, se cifraron 40 letras. Los resultados se muestran a continuación.

Running Time		Symbol Name
3250.0ms	93.4%	▼Main Thread 0xbaa22
3250.0ms	93.4%	▼main Cipher
682.0ms	19.6%	▶-[iCarousel step] Cipher
335.0ms	9.6%	▼-[EnigmaViewController buttonDown:] Cipher
9.0ms	0.2%	▶-[EnigmaViewController updateRotorsAnimated:] Cipher
8.0ms	0.2%	▶-[EnigmaM4 encryptCharacter:] Cipher
97.0ms	2.7%	▶-[EnigmaViewController rotorDidPan:] Cipher
66.0ms	1.8%	▶-[WheelMenuViewController carousel:didSelectItemAtIndex:] Cipher
11.0ms	0.3%	▶-[EnigmaViewController viewDidLoad] Cipher
6.0ms	0.1%	▶-[iCarousel resizeSubviewsWithOldSize:] Cipher
6.0ms	0.1%	▶-[WheelMenuViewController viewDidLoad] Cipher
3.0ms	0.0%	-[iCarousel startAnimation] Cipher
3.0ms	0.0%	-[PlugboardView drawRect:] Cipher
2.0ms	0.0%	-[WheelMenuViewController awakeFromNib] Cipher
2.0ms	0.0%	▶-[iCarousel layoutSubviews] Cipher

Figura 5.10: Resultado del Time Profiler para Enigma M4

La línea en gris muestra que el método que cifra una letra llamado 40 veces tarda 8 [ms] un aproximado de 200 μs. por cada ocurrencia.

Todos los tiempos en los resultados anteriores se encuentran dentro de los 83.33 [ms] establecidos en los requerimientos de rendimiento.

Conclusiones

Uno de los objetivos de este trabajo fue el de crear conciencia de la importancia que ha tenido la Criptografía desde épocas remotas hasta nuestros días, usando para este fin los algoritmos más sobresalientes de la antigüedad. Aprovechando los avances de la tecnología que actualmente nos permite asociarnos con los dispositivos electrónicos de una manera más interactiva, tenemos como resultado una herramienta que el usuario puede utilizar para familiarizarse con dichos métodos.

El trabajo comenzó con un estudio a fondo de las principales técnicas de cifrado clásicas, desde su descripción y la importancia de su uso en el periodo en el que fue desarrollada y utilizada, mostrando un poco de su historia y explicando su funcionamiento, tanto en el proceso de cifrado como en el de descifrado, con el próposito de un mejor entendimiento del algoritmo. Esta investigación nos proporcionó una idea más precisa del proceso utilizado en cada una de las técnicas y de su contexto histórico, lo cual nos llevó a la traducción de estas ideas en un diseño que representará de una manera simple el concepto de la técnica sin perder de vista las caracteristicas físicas del dispositivo empleado, en los casos requeridos, para presentar una interfaz consistente con el proceso y dispositivo.

A partir del diseño fueron creados los requerimientos funcionales que sirvieron de base para la implementación utilizando el Framework Cocoa Touch, el cual nos proporcionó la mayoría de los componentes que se adaptaban a las necesidades descritas anteriormente, sin embargo, la falta de componentes que representarán a la scítala sin desviarse demasiado del concepto nos llevó a considerar otras opciones. La necesidad de crear una representación de la scítala que realmente ayudara a su mejor comprensión resultó en el uso de la API OpenGL, la cual nos proporcionó los medios para representar gráficamente el concepto de una manera fiel al diseño.

Así pues, la aportación principal de este trabajo consiste en la implementación de una aplicación para contribuir al aprendizaje de los métodos de cifrado clásicos que

se encuentre disponible de manera gratuita en la App Store de Apple para cualquier persona con los conocimientos básicos de los objetivos de la Criptografía que cuente con un dispositivo iPad, iPhone o iPod Touch.

Esta aplicación fue desarrollada inicialmente para iPad, pero considerando el gran número de dispositivos móvilles como iPhone y iPod Touch, se decidió adaptar la aplicación para poder usarla en estos dispositivos, además de que el tiempo de desarrollo fue relativamente corto ya que se utiliza el mismo código para las clases de controlador y modelo en el patrón MVC, y solo el código de vista tuvo que ser creado para estos dispositivos. A pesar de esto, resulta un poco más complicada la interacción del usuario con la aplicación en el iPod Touch y en el iPhone que en el iPad, debido a que el espacio para las vistas es menor y algunos componentes no son adecuados para poder utilizarlos, por esta razón el dispositivo Enigma no fue incluido en el iPhone y el iPod Touch.

Como trabajo posterior se puede considerar la implementación de otros métodos de cifrado clásicos para ser añadidos a la aplicación en una siguiente versión, así como el de adaptar todos los métodos para plataformas con sistema operativo Android debido a que es el sistema más utilizado en el mercado de la tecnología móvil y con mayor crecimiento en la actualidad.

Glosario

API (Application Programming Interface), interfaz de programación de aplicaciones. Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación.

Cifrado por transposición es un tipo de cifrado en el que unidades de texto plano se cambian de posición siguiendo un esquema bien definido.

Cipher Nombre de la aplicación desarrollada en este trabajo.

Core data es una colección de herramientas y frameworks para almacenar, accesar y compartir datos en aplicaciones de Cocoa Touch orientadas a objetos.

Criptografía del griego *Kryptós*, criptos "ocultar" y *graphé*, grafos "escribir" la criptografía es: escritura oculta. Es la ciencia encargada de transformar la información de manera que sea encubierta e incomprensible para las personas no autorizadas para acceder a ella.

Criptograma es un mensaje cifrado cuyo significado resulta ininteligible hasta que es descifrado.

Framebuffer Es el destino de las operaciones que generan una imagen en OpenGL, consiste en una colección de arreglos de dos dimensiones utilizadas para almacenar información de color y profundidad principalmente.

Framework es un directorio jerárquico que encapsula los recursos compartidos, como una biblioteca compartida, archivos nib, archivos de imágenes, secuencias localizadas, archivos de cabecera y documentación de referencia en un solo paquete.

IOS sistema operativo móvil de la empresa Apple Inc.

MVC Modelo Vista Controlador es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

Notch es el giro que tiene que dar el siguiente rotor con respecto al actual en la máquina Enigma.

Plugborad Arreglo de conexiones en algunos modelos de la máquina enigma utilizado para intercambiar dos letras antes y después del cifrado realizado por los rotores.

Ring setting cambio de posición del anillo interior con respecto al anillo exterior del rotor en la máquina Enigma.

Shader es un programa de software que incluye instrucciones sobre cómo la GPU debe renderizar ciertas imágenes en la pantalla. Son utilizados para realizar transformaciones y crear efectos especiales, por ejemplo iluminación, fuego o niebla.

Storyboard es una representación visual de la interfaz de usuario de una aplicación iOS, mostrando las pantallas y las conexiones entre ellas.

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en ingles) de Apple Inc.

Bibliografía

[1] López Barrientos, Ma. Jaquelina. *Criptografía*. México: Universidad Nacional Autónoma de México, Facultad de Ingeniería, 2009.

[2] Singh, Simon .*The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. New York: Anchor Books, Inc. 1999.

[3] tripp, Alan. *Code Breakers*.UK, Oxford University Press 1994.

[4] Kahn,David. *The Codebreakers*. New York: Scribner, 1996.

[5] Wolff, David. *OpenGL 4.0 Shading Language Cookbook*. UK: Packt Publishing Ltd, 2011.

[6] F.L Bauer. *Decrypted Secrets*.Berlin: Springer, Fourth Edition, 2007.

[7] Welch, Shawn. *iOS 5 Core Frameworks:Develop and Design*. Berkeley, CA, 2012.

[8] Buck, Erik M. *Learning OpenGL ES for iOS. A Hands-On Guide to Modern 3D Graphics Programming*. Estados Unidos: Pearson, 2013.

`http://developer.apple.com/library/ios/navigation/`

`http://en.wikipedia.org/wiki/Scytale`

`http://www.cryptool-online.org/index.php`

`http://www.cypher.com.au/crypto_history.htm`

`http://ciphermachines.com/`

http://www.cs.trincoll.edu/~crypto/index.html

http://www.cryptomuseum.com/crypto/enigma/i/index.htm

http://serdis.dis.ulpgc.es/~ii-cript/PAGINA%20WEB%20CLASICA/CRIPTOGRAFIA/

https://en.wikipedia.org/wiki/Enigma_machine

http://en.wikipedia.org/wiki/Enigma_rotor_details

http://www.pbs.org/wgbh/nova/military/how-enigma-works.html

http://users.telenet.be/d.rijmenants/en/enigmatech.htm

https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/