Universidad de las Ciencias Informáticas

Facultad 10

Título: Rubik, contenedor cifrado de datos personales

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autores: Ernesto Rodriguez Ortiz
George Sánchez Amaya

Tutores: 1er Tte. Ing. Joaquín Quintas Santiago
Ing. Susel Vázquez Acuña

Ciudad de la Habana, Cuba
Año 52 de la Revolución

DECLARACIÓN DE AUTORÍA

Declaramos ser autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Autores

_____ _____

Ernesto Rodriguez Ortiz George Sánchez Amaya

Tutores

_____ _____

1er Tte Ing. Joaquín Quintas Santiago Ing. Susel Vázquez Acuña

Dedicatoria

A mis padres, que desde pequeño me han mostrado el camino a seguir a través de los estudios, de mi preparación personal y profesional, por todo su sacrificio y esfuerzo.

George

A la memoria de mi padre, por quien me esfuerzo cada día mucho más para que pueda sentirse orgulloso de mí.

Ernesto

Resumen

Desde el comienzo de la era digital la seguridad de la información ha sido un importante asunto a considerar, llegando en ocasiones a constituir un serio problema que necesita ser resuelto. Variadas son las acciones que se llevan a cabo y las medidas que se implementan en aras de aumentar al máximo posible la invulnerabilidad de los datos. El cifrado de la información es una de estas acciones, cuya puesta en práctica aumenta el nivel de seguridad, evitando que intrusos o personas sin la debida autorización tengan acceso a la misma. Actualmente las Fuerzas Armadas Revolucionarias (FAR) se encuentran inmersas en un proceso de migración hacia el sistema operativo GNU/Linux Nova, que no cuenta con una herramienta de cifrado de información destinada a sus usuarios finales. Pueden encontrarse en Internet variadas ofertas de dichas herramientas, que de una u otra forma realizan el cifrado de información, pero en torno a estas existen problemas de licencia, falta de mantenimiento, vulnerabilidades conocidas, entre otros, que impiden que estas realicen un buen trabajo en la protección de la información. Esta situación hace imprescindible la construcción de una herramienta de cifrado de datos personales, propia de las FAR, para la distribución GNU/Linux Nova, que permita proteger la información sensible de los usuarios finales, mediante el cifrado y descifrado automático, haciendo uso de los mejores algoritmos conocidos hoy en día.

Palabras claves: herramientas de cifrado, confidencialidad de la información.

Índice general

Introducción **1**

1. Fundamento teórico **5**
1.1. Control de acceso . . . 5
1.1.1. Módulos de autenticación conectados (PAM) . . . 5
1.2. Criptografía . . . 6
1.2.1. Criptosistemas simétricos o de clave privada . . . 7
1.3. Metodología de desarrollo . . . 8
1.4. Herramientas . . . 11
1.4.1. Herramienta de cifrado a utilizar . . . 11
1.4.1.1. Loop-AES . . . 11
1.4.1.2. DM-Crypt . . . 12
1.4.1.3. DM-Crypt/Cryptsetup . . . 13
1.4.1.4. DM-Crypt/LUKS . . . 14
1.4.1.5. eCryptfs . . . 15
1.4.1.6. Scramdisk4Linux (SD4L) . . . 16
1.4.2. Lenguaje de programación . . . 18
1.4.3. IDE de desarrollo . . . 19
1.4.4. Otras herramientas . . . 19

2. Características del sistema **22**
2.1. Descripción del problema . . . 22
2.2. Propuesta del sistema a desarrollar . . . 23
2.3. Modelo de dominio . . . 23
2.4. Especificación del entorno operacional . . . 24
2.5. Descripción de las funcionalidades . . . 25
2.5.1. Requisitos funcionales . . . 25
2.5.2. Requisitos no funcionales . . . 27
2.6. Definición de los casos de uso y abuso . . . 28
2.6.1. Definición de los actores y actores de abuso. . . . 28
2.6.2. Diagrama de casos de uso y casos de abuso . . . 29

2.6.3. Realización de los casos de uso. Descripción textual detallada. 29
2.6.4. Realización de los casos de abuso. Descripción textual detallada. 38

3. Diseño del sistema **41**
3.1. Arquitectura y patrones de diseño 41
3.1.1. Arquitectura 41
3.1.2. Principios de diseño seguro 42
3.1.3. Patrones de diseño 43
3.2. Diagramas de clases de diseño 43
3.3. Realización de los casos de uso de diseño 46
3.4. Análisis y modelado de amenazas 54
3.4.1. Amenazas mitigadas 57
3.4.2. Amenazas mitigadas parcialmente 58
3.4.3. Amenazas sin mitigar 59
3.5. Ayuda 60

4. Implementación y pruebas **61**
4.1. Estructuración del modelo de implementación 61
4.2. Revisión de código 62
4.3. Pruebas 64
4.3.1. Pruebas unitarias 64
4.3.2. Pruebas de integración 65
4.3.3. Pruebas de penetración 65

Conclusiones **66**

Recomendaciones **67**

Referencias **69**

Glosario de términos **70**

Introducción

Desde mucho antes del surgimiento de las computadoras, el hombre se ha preocupado por mantener en secreto información a la que no quisiera que accedieran todas las personas. Con el advenimiento de la era informática y el surgimiento de un nuevo concepto de información[1], esta tarea de protección y salvaguarda no solo está vigente como antaño, sino que se hace más compleja y difícil para usuarios inexpertos, dándoles ventaja a intrusos y atacantes.

En este sentido, las aplicaciones informáticas libres, por sus características de seguridad, robustez, estabilidad y alta configurabilidad, cada día ganan más espacio entre los usuarios; y en los servidores se han consolidado como un fuerte oponente a las aplicaciones privativas [22]. Por las ventajas que brindan las aplicaciones informáticas libres, muchos países han migrado hacia ellas como lo hace Venezuela con el decreto 3390 ó la municipalidad del Rosario en Argentina [11]. En Cuba la migración al Software Libre responde a sus ventajas y a una estrategia soportada por cuatro puntos fundamentales: político, social, tecnológico y económico [22]. Como parte de este proceso de migración se ha creado una distribución de GNU/Linux cubana en la UCI, llamada Nova.

Actualmente las diferentes distribuciones de GNU/Linux no permiten a sus usuarios mantener su información personal completamente confidencial, debido a que varias situaciones atentan contra ello. La primera, es que existe un usuario administrador global con todos los privilegios en el sistema operativo, llamado (*root*) que puede acceder a la información personal del resto de los usuarios en cualquier momento, que si se ve vulnerado por un atacante al obtener su contraseña o por otro usuario del sistema que escale en permisos, provocará el compromiso de la seguridad de todos los datos de los usuarios. La segunda es que se puede acceder libremente a todo el sistema de archivos del disco duro mediante un LiveCD u otro sistema operativo instalado en la misma computadora, evadiendo los mecanismos que garantizan el cumplimiento de las políticas de seguridad definidas en el sistema GNU/Linux que coexiste con este otro sistema operativo presente en una misma computadora. Por estas razones surge el siguiente problema a resolver: **¿Cómo garantizar una mayor confidencialidad de la información personal de los usuarios en el sistema operativo GNU/Linux Nova?**

Para dar solución al problema planteado, esta investigación tiene como objeto de estudio **los mecanismos de protección de la confidencialidad de la información personal de los usuarios en el sistema operativo GNU/Linux**, delimitando su campo de acción a **las herramientas que permitan el cifrado y descifrado automático de la información personal de los usuarios en las distribuciones de GNU/Linux.**

Se puede incrementar la seguridad de la información mediante la combinación de mecanismos de cifrado y autenticación, haciendo posible que los usuarios accedan a su información y evitando que pueda ser accedida por personal no autorizado. Actualmente existen formas de poner en funcionamiento el cifrado automático mediante el uso de paquetes y configuraciones. Sin embargo estas son extremadamente difíciles de poner en marcha para

[1] Información digital, almacenada no de forma física, sino lógica

cualquier usuario inexperto en el uso del sistema operativo GNU/Linux, enfrentándose al riesgo de cometer errores en el proceso, que pueden acarrear resultados indeseables como pérdida de información o mal funcionamiento del sistema.

En estos momentos en la distribución GNU/Linux Nova no existe una herramienta destinada a que los usuarios finales puedan, de una forma relativamente sencilla, mantener su información personal de manera confidencial, por lo que este trabajo se plantea como objetivo **crear una aplicación para la distribución GNU/Linux Nova que combine mecanismos de cifrado y autenticación para aumentar la confidencialidad de la información sensible de sus usuarios.**

Defendiendo la idea de que **una vez desarrollada y puesta en funcionamiento una aplicación que combine mecanismos de cifrado y autenticación, aumentará la confidencialidad de la información sensible de los usuarios de la distribución GNU/Linux Nova.**

Se podrían ejemplificar algunas situaciones en las cuales esta herramienta sería de gran utilidad:

- Un usuario que utiliza sus claves SSH para la autenticación, pero no las bloquea con contraseñas, ya que las utiliza para una serie de operaciones automáticas (por ejemplo copias de seguridad), quiere un lugar donde se pueda almacenar con seguridad la clave privada SSH sin usar una contraseña SSH, criptográficamente protegida en caso de que le roben su computadora portátil.

- Un usuario utiliza frecuentemente su sistema para compilar código abierto no sensible. Aunque tiene datos en el mismo que quisiera proteger criptográficamente, no quisiera tener que cargar con el costo que para el rendimiento tendría el cifrado de todo el sistema, que afectaría operaciones de entrada/salida altamente intensas. Preferiría cifrar algunos documentos que guarda en su directorio /home, en lugar de todos los del directorio raíz.

- Un gerente de una empresa, en la cual el informático es el encargado de la instalación, reparación y mantenimiento de las máquinas quisiera tener un simple mecanismo mediante el cual tener su propio directorio privado donde almacenar ficheros con información sensible, con posibilidad de lectura al autenticarse, pero cifrados en el disco e imposibles de leer cuando el usuario cierra su sesión para evitar que el informático que es administrador de la máquina tenga acceso a la información sensible.

En cada una de estas situaciones, la funcionalidad propuesta ofrece protección mediante el cifrado, más allá del control de acceso discrecional de GNU/Linux para cada usuario del sistema, protegiendo además la confidencialidad de los datos, en caso de robo físico.

Para lograr el objetivo de este trabajo han sido definidos los siguientes objetivos específicos:

1. Determinar las herramientas de desarrollo y cifrado, libres y de código abierto, que se usarán para desarrollar la aplicación.

2. Combinar los mecanismos de cifrado y autenticación.
3. Determinar los riesgos de seguridad a los que estará expuesto el contenedor cifrado de datos personales[2].
4. Definir las características de seguridad y usabilidad idóneas para la interfaz de usuario.

Desarrollándose para alcanzarlos, las siguientes tareas de investigación:

1. Realizar un estudio del estado del arte de las diferentes herramientas de desarrollo y cifrado libres y de código abierto, en aras de determinar cuáles se usarán para desarrollar la aplicación.
2. Analizar cómo combinar los mecanismos de autenticación con los de cifrado mediante los cuales se implementará la solución.
3. Determinar los riesgos de seguridad que pueden estar presentes en el contenedor cifrado de datos personales y seleccionar los mecanismos de seguridad necesarios para mitigarlos.
4. Identificar las características de seguridad y usabilidad de la interfaz de usuario requeridas para las herramientas de protección de la información.
5. Diseñar e implementar la herramienta.
6. Validar la seguridad que ofrece la herramienta mediante la comprobación experimental.

Entre los Métodos de trabajo científico a utilizar se encuentran:

- El método histórico-lógico y el dialéctico para el estudio crítico de los trabajos anteriores, y para utilizar estos como punto de referencia y comparación con los resultados alcanzados.
- El método analítico-sintético al descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución de la propuesta.
- El método coloquial para la presentación y discusión de los resultados.
- El método experimental para comprobar la utilidad de la herramienta obtenida.

El presente trabajo de diploma está estructurado en cuatro capítulos tal como se describe a continuación.

- En el Capítulo 1 "Fundamento teórico", se hace un estudio de los diferentes mecanismos de Control de Acceso (principalmente los utilizados por los sistemas operativos GNU/Linux) y de las diferentes herramientas de cifrado existentes para las plataformas libres y sus características.

[2]Dispositivo creado por Rubik, en el cual el usuario guardará su información para que permanezca confidencial.

- En el Capítulo 2 “Características del sistema”, se establece qué debe hacer el sistema y cómo lo hará, definiendo su alcance y requisitos.
- En el Capítulo 3 “Diseño de la aplicación”, se muestran tanto el diseño orientado a objetos de las clases que se implementarán para cumplir las funcionalidades requeridas por la herramienta, como las características de seguridad y usabilidad a tener en cuenta para la aplicación y su funcionamiento.
- En el Capítulo 4 “Implementación y pruebas”, se define la organización del código y se diseñan varios tipos de pruebas para verificar que la aplicación construida cumpla con las especificaciones requeridas.

Finalmente se presentan las Conclusiones, Recomendaciones, Referencias, Bibliografía, Anexos y el Glosario de Términos.

Capítulo 1

Fundamento teórico

La creación de un sistema automático de cifrado depende en gran medida de los componentes a utilizar, algoritmos de cifrado y herramientas. Actualmente existen una gran cantidad de estos algoritmos y herramientas con distintas características y para diversos sistemas operativos. Con el fin de realizar la mejor selección para la construcción de la aplicación de cifrado automático en el presente capítulo se hace un análisis detallado de cada uno de los aspectos que intervienen y que deben ser tomados en cuenta para la creación de dicha herramienta, tales como el control de acceso, los principales conceptos y herramientas criptográficas, la metodología de desarrollo y las herramientas a utilizar. Por último se presentan las conclusiones donde los autores definen su postura respecto al análisis teórico realizado.

1.1. Control de acceso

El control de acceso es el proceso de conceder permisos a usuarios o grupos de usuarios para acceder a objetos, tales como ficheros o dispositivos en la red. El mismo está basado en tres conceptos fundamentales: identificación, autenticación y autorización.

Identificación: es la manera en que el usuario le dice al sistema quién es él [15], usualmente se usa un identificador de usuario.

Autenticación: es el proceso que prueba que un sujeto (por ejemplo, un usuario o un sistema) es realmente quien dice ser [15], generalmente por medio de una contraseña.

Autorización: es el procedimiento para conceder permisos a usuarios o procesos [15] para que puedan acceder a los recursos, luego de haberse identificado y autenticado.

En los sistemas GNU/Linux se usa un control de acceso discrecional siguiendo el modelo de Unix tradicional conocido como UGO (*User, Group, Other*) y este se logra mediante el uso de PAM que provee autorización dinámica para aplicaciones y servicios.

1.1.1. Módulos de autenticación conectados (PAM)

PAM por sus siglas en ingles "*Pluggable Authentication Modules*", no es un modelo de autenticación en sí sino un mecanismo que proporciona una interfaz entre las aplicaciones de usuario y diferentes métodos de autenticación, tratando de esta forma de solucionar uno de los problemas clásicos de la autenticación de usuarios: el hecho de que

una vez se ha definido e implantado cierto mecanismo en un entorno, es difícil cambiarlo [10]. A través de PAM se pueden integrar las diferentes utilidades de los sistemas operativos que lo utilizan ya que permite comunicar todas las aplicaciones con los métodos de autenticación de manera transparente.

El proyecto Linux-PAM proporciona la implementación de PAM para los sistemas GNU/Linux y está formado por muchos profesionales que dan soporte a todos los módulos que se han desarrollado y continúan desarrollándose.

La arquitectura PAM es la solución a los diferentes problemas de autenticación de usuarios, especialmente en entornos complejos, como sistemas distribuidos o reinos Kerberos; proporcionando independencia entre los servicios del sistema y los mecanismos de autenticación utilizados en beneficio tanto de los administradores como de los usuarios. Desde 1995, año en que se adoptó la solución propuesta por *Sun Microsystems* hasta la actualidad, cada vez más plataformas integran PAM por defecto, con lo que se ha convertido en el estándar en la autenticación dentro de entornos Unix [10].

Debido a las grandes ventajas de PAM que ha logrado imponerse como un estándar en la autenticación de los sistemas Unix y los sistemas basados en este como es el caso de las diferentes distribuciones de GNU/Linux, su uso es necesario en el desarrollo de la solución que se propone.

1.2. Criptografía

Según el Diccionario de la Real Academia de la Lengua Española, la palabra Criptografía proviene del griego *kriptos*, que significa oculto, y *graphos*, que significa escritura, y se define como el "Arte de escribir con clave secreta o de un modo enigmático" [16]. La criptografía es una de las grande ramas de la criptología (del griego *krypto* y *logos*, estudio de lo oculto, lo escondido), es la ciencia[1] que trata los problemas teóricos relacionados con la seguridad en el intercambio de mensajes en clave entre un emisor y un receptor a través de un canal de comunicaciones (en términos informáticos, ese canal suele ser una red de computadoras) [10].

La otra gran rama de esta ciencia es el criptoanálisis que estudia los métodos para descifrar los mensajes, al contrario de la criptografía, que estudia como cifrarlos. La criptografía se divide en dos, la criptografía de clave privada o simétrica y la criptografía de clave pública o asimétrica.

Aunque inicialmente la criptografía fue concebida para resolver el problema de la confidencialidad de la información, hoy resuelve cuatro problemas fundamentales en la seguridad:

La Confidencialidad: es la ocultación de la información [2] para que solo pueda ser leída por personas autorizadas.

La Integridad: se refiere a la fiabilidad de la información, expresada en términos de prevención del cambio indebido o no autorizado [2].

[1] Es necesario aclarar que la criptología es una ciencia, aunque en muchos lugares se la considera un arte; por ejemplo, en el Diccionario de la Real Academia de la Lengua Española.

La Autenticidad: se puede confirmar que el mensaje recibido fue enviado por quien dice lo envió y no otra persona [10].

No rechazo o no repudio: es la imposibilidad de negar la autoría de un mensaje enviado [10].

Existen dos tipos fundamentales de criptosistemas:

Criptosistemas simétricos o de clave privada: son aquellos que emplean la misma clave k tanto para cifrar como para descifrar [16] e incluye los sistemas clásicos.

Criptosistemas asimétricos o de clave pública: emplean una doble clave (kp, kP), kp se conoce como clave privada y kP se conoce como clave pública. Una de ellas sirve para la transformación de cifrado y la otra para la transformación de descifrado [16].

El presente trabajo se interesa principalmente por la criptografía simétrica debido a que se puede beneficiar de la velocidad y facilidad de uso de esta con respecto a la criptografía asimétrica.

1.2.1. Criptosistemas simétricos o de clave privada

La criptografía simétrica es el sistema de cifrado más antiguo en el cual se incluye la criptografía clásica y consiste en que tanto el emisor como el receptor cifran y descifran con una clave que ambos conocen y que deben mantener en secreto.

Toda la seguridad de este sistema está basada en la clave privada, por lo que es misión fundamental tanto del emisor como del receptor mantener la clave en secreto, ya que si la clave cae en manos de terceros, el sistema deja de ser seguro, por lo que habría que desechar dicha clave y generar una nueva.

Existen dos condiciones que deben cumplir los algoritmos que se usan en este tipo de criptografía para que sean considerados fiables:

1. Una vez conocido el texto cifrado no se pueden obtener de él ni el texto en claro ni la clave.
2. Debe resultar más caro en tiempo o dinero descifrar la clave a partir del texto en claro y el texto cifrado que el valor posible de la información obtenida por terceros.

El mayor inconveniente de la criptografía simétrica es la necesidad de enviar la clave de cifrado del emisor al receptor, para que éste sea capaz de descifrar el mensaje. Por tanto, se incurre en los mismos peligros al enviar la clave, por un sistema que debería ser supuestamente seguro, que al enviar el texto plano [10]. Dicho inconveniente no afecta la herramienta en construcción (Rubik) pues esta solo va a ser usada por un usuario que utiliza el algoritmo criptográfico para cifrar la clave que va a usar en el contenedor.

Entre los principales algoritmos de cifrados simétrico se encuentran DES (*Data Encryption Standard*), 3DES (*Triple DES*), IDEA (*International Data Encryption Algorithm*), RC5 (*International Data Encryption Algorithm*),

AES (*Advanced Encryption Standard*), Blowfish, de los cuales se encuentran disponibles en GNU/Linux a nivel de núcleo, versión 2.6.31 DES, AES, Blowfish, además se pueden encontrar otros como CAST, Serpenter, Twofish, etc.

1.3. Metodología de desarrollo

Existen actualmente numerosas metodologías de desarrollo de software, pero no todas están diseñadas de forma tal que el proceso se enfoque en detallar actividades que se centren en garantizar la seguridad del producto durante todo el ciclo de vida de desarrollo. Solo un grupo de metodologías modifican las actividades tradicionales o incorporan nuevas con el objetivo de reducir el número de debilidades y vulnerabilidades aumentando la fiabilidad del software ante la presencia de amenazas. Estas metodologías son las llamadas "metodologías de desarrollo de software seguro".

Una de las metodologías valoradas para realizar la herramienta fue el "Proceso de desarrollo y gestión" del centro UCID, por una cuestión de homogeneidad, debido a que es la utilizada mayormente por los proyectos desarrollados en el centro. Sin embargo, no fue seleccionada como metodología para el desarrollo de esta aplicación por las siguientes razones:

- Las actividades definidas para la captura de requisitos no se enfocan en los requisitos de seguridad.
- No existen actividades definidas para garantizar que la arquitectura y el diseño de la aplicación sean seguros.
- Los procesos de validación de la implementación están orientados a detectar problemas con la satisfacción de los requisitos funcionales de la aplicación y no incluyen ninguna actividad para detectar vulnerabilidades en el código o en la lógica de la aplicación.
- Al igual que el caso anterior, las pruebas unitarias tampoco traen actividades definidas explícitamente para garantizar que cada uno de los componentes del sistema sea seguro.
- Las pruebas de integración no contemplan ningún elemento de pruebas de penetración u otro tipo de evaluación de seguridad que permita certificar que los componentes, sus interacciones y su funcionamiento como sistema sea seguro.

Por otra parte fueron analizadas metodologías de desarrollo de software seguro como *Microsoft SDL*, CLASP y 7 *Touch Points,* resultando seleccionada CLASP para el desarrollo de la solución propuesta.

Proceso Global y Ligero para Aplicaciones de Seguridad (CLASP)

CLASP, por sus siglas en inglés "*Comprehensive, Lightweight Application Security Process*", ofrece una forma estructurada para hacer frente a los problemas de seguridad durante el desarrollo de software. CLASP está disponible

como una extensión de RUP y contiene las mejores prácticas formalizadas para fomentar la seguridad en existentes o recién comenzados ciclos de desarrollo de software, de forma estructurada, repetible y mensurable [21].

Fue desarrollado por el experto en seguridad John Viega y está diseñado para permitir la fácil integración de sus 24 actividades relacionadas con la seguridad con las actividades definidas en los flujos de trabajo de RUP, manteniendo la flexibilidad de este proceso de desarrollo de software [21]. Las actividades son asignadas como responsabilidades a siete roles dentro del equipo de desarrollo, y para cada rol se brinda una guía de implementación de las mismas, el nivel de riesgo de omitirlas y el costo estimado de implementarlas. También contiene un catálogo amplio de vulnerabilidades que ayuda a evitar o remediar errores en el diseño o codificación que puedan conducir a la explotación de los servicios de seguridad [21].

No ha sido posible integrar CLASP con el "Proceso de desarrollo y gestión" del centro UCID, pues lograr la correcta integración de las actividades de ambos procesos no es objetivo de este trabajo de tesis. Además existen ciertas incompatibilidades entre las actividades que proponen ambos. Por solo poner un ejemplo en el en el "Proceso de desarrollo y gestión" del centro UCID no hay ninguna actividad relacionada con definir, detallar y modelar los casos de uso por lo que se hace muy complejo integrar la actividad de detallar los casos de abuso que propone CLASP y que es propuesta también por otras metodologías de desarrollo de software seguro como 7 *Touch Points*.

El Proceso Unificado de Rational

RUP es una recopilación de prácticas de ingeniería de software que se están mejorando continuamente de forma regular para reflejar los cambios en las prácticas de la industria [4]. Este proporciona un entorno de proceso configurable basado en estándares para satisfacer las necesidades exclusivas de cada proyecto [4]. Se caracteriza por ser iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura.

RUP se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto y consta de cuatro fases: inicio, elaboración, construcción y transición [7]. Cada fase se subdivide en iteraciones en las cuales se desarrollan los flujos de trabajos: Modelado del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, Despliegue, Configuración y Control de Cambios, Gestión del Proyecto y Entorno. Véase la figura 1.1

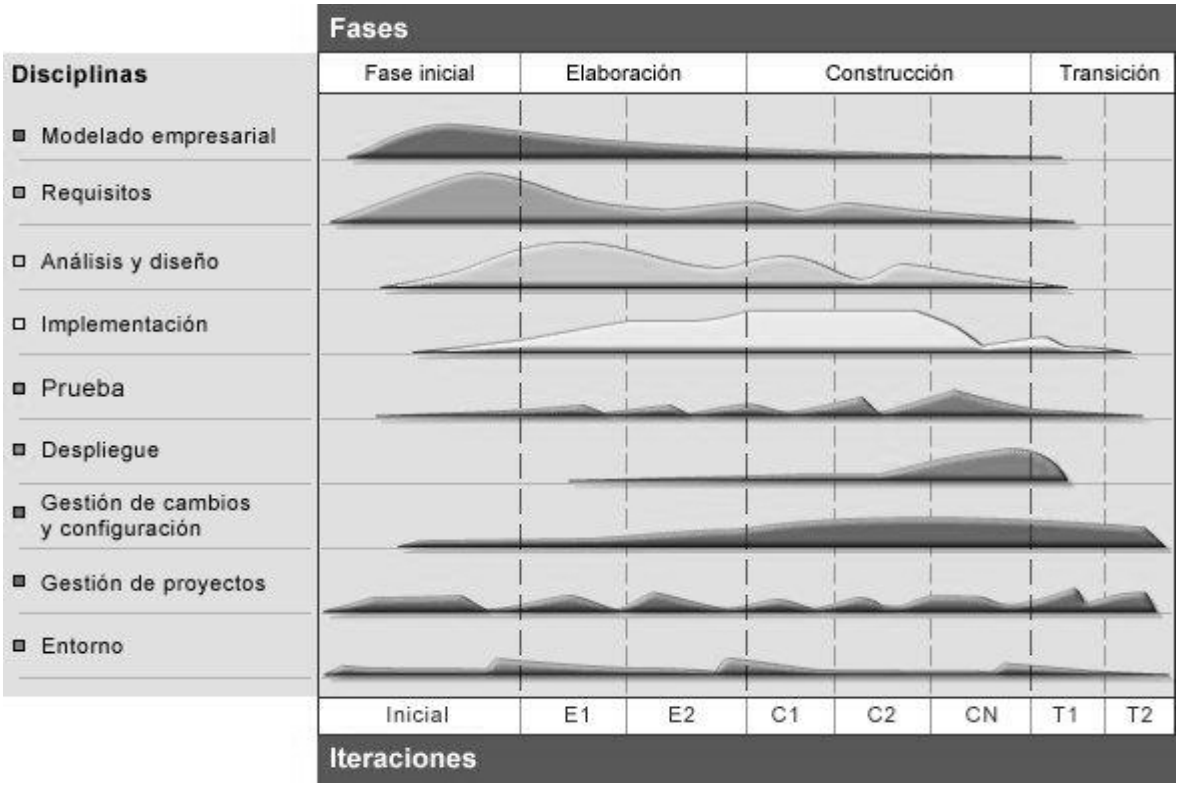

Figura 1.1: RUP

Las razones fundamentales para seleccionar a CLASP como metodología para el desarrollo del proyecto fueron, en primer lugar, que se encuentra disponible como un complemento para RUP que es un proceso de desarrollo robusto y perfectamente configurable a cualquier proyecto; en segundo lugar que como metodología de software seguro CLASP está enfocada en hacer frente a los problemas de seguridad durante todo el ciclo de vida de desarrollo del software, que en este caso es primordial, pues se está hablando de una herramienta para la protección de la confidencialidad de la información; y en tercer lugar los autores de este trabajo poseen mayor experiencia en el uso de RUP como metodología de desarrollo de software que en las restantes, factor que disminuye los riesgos de cometer errores durante el desarrollo e implementación de la solución propuesta y permite utilizar mejor el tiempo, pues minimiza las actividades destinadas a la familiarización con nuevas metodologías y el aprendizaje de sus particularidades.

También serán consultados estudios realizados sobre el análisis de riesgos y requerimientos de seguridad en aplicaciones de cifrado de discos que tienen como objetivo definir los requisitos de seguridad que deben cumplir estas aplicaciones [5].

1.4. Herramientas

Seleccionar correctamente las herramientas con las que se construirá y que conformarán la nueva aplicación informática tiene un papel primordial en la calidad y seguridad final del proyecto, así como en el tiempo de desarrollo dado el nivel de automatización que estas puedan proveer.

1.4.1. Herramienta de cifrado a utilizar

Hoy en día existen un gran número de aplicaciones que hacen posible el cifrado de la información y que son accesibles a todos los usuarios, desde herramientas libres hasta propietarias, con interfaces gráficas, sin ellas y que se pueden utilizar en diferentes sistemas operativos, desde las diferentes distribuciones de GNU/Linux, hasta Windows y Mac.

Después de realizar una amplia búsqueda de las mejores y más usadas herramientas de cifrado, fue seleccionada una muestra formada solamente por las herramientas de cifrado de código abierto, con licencias libres como GPL y que funcionen sobre GNU/Linux, debido a que él país se encuentra en un proceso de migración al software libre y no es factible desarrollar una solución con herramientas propietarias. Además estas herramientas libres se encuentran a disponibilidad de todos y se entregan con el código fuente de las mismas, evitando así que existan puertas traseras o fallos de seguridad que hayan pasado por alto los desarrolladores, ya que todo aquel que tenga conocimientos para hacerlo, puede revisar el código, cambiarlo y redistribuirlo.

Debido a la manera intencional con que se escogió la muestra, siguiendo criterios específicos, pudo haberse pasado por alto alguna herramienta que no cumpliera con los requisitos anteriores y que sin embargo fuera superior en cuanto a funcionalidades.

Al final del análisis realizado sobre las diferentes herramientas de cifrado que se podrían utilizar se escogieron para la muestra DM-Crypt/cryptsetup, DM-Crypt/LUKS, eCryptfs, Loop-AES, Scramdisk4Linux, las cuales son las principales aplicaciones de software libre para el cifrado de información.

A continuación se realiza una descripción de las diferentes herramientas seleccionadas en la muestra con el objetivo de determinar cuál sería más factible usar en el desarrollo de la solución.

1.4.1.1. Loop-AES

Loop-AES es una plataforma de cifrado funcional a partir del núcleo 2.0; utiliza el modelo *Loop Devices* a través de una interfaz del tipo *Loopback*, sin embargo la misma aplicación trae su propio módulo para el núcleo y su conjunto de aplicaciones para el usuario: *mount, losetup* y otras.

Utiliza hasta 65 llaves distintas, las cuales crea añadiendo información aleatoria. Cada llave cifra un sector determinado de unos 512 bytes que se va repitiendo continuamente. Este modelo de llaves múltiples se denomina multi-key-v3 y la forma en que las genera y las usa, hace que el sistema sea inmune a los ataques de diccionario [3].

Loop-AES no permite el uso de sistemas de ficheros con *journaling* en archivos *loop* creados en el sistema (sistemas de ficheros creados dentro de un fichero de tamaño elegido). Sin embargo sí permite el uso de estos sistemas de ficheros sobre discos, particiones y dispositivos externos, pues se respeta el orden de escritura requerido (siempre que esté desactivada la escritura mediante *buffer cache*) [3].

Loop-AES presenta una serie de problemas de seguridad, por ejemplo:

1. Para aumentar la seguridad al máximo se debe portar un fichero con las 65 llaves siempre encima, por ejemplo una memoria USB, lo que realmente hace que se pierda la seguridad ganada, al portar uno mismo las claves que deberían estar almacenadas en un lugar seguro.

2. Cuando se hiberna o se suspende el equipo las claves que están siendo utilizadas en el momento y que se encuentran en la memoria RAM se copian al disco para poder reanudar el sistema después, lo que hace inseguros y poco recomendados los mecanismos de hibernación o suspensión.

1.4.1.2. DM-Crypt

DM-Crypt es un *device-mapper* tal que provee cifrado transparente de dispositivos de bloque utilizando la nueva CryptoAPI de Linux 2.6. El usuario puede especificar, básicamente, uno de los sistemas de cifrado simétrico, una llave (de cualquier tamaño máximo permitido), un modo de generación IV y entonces crear un dispositivo de bloque en /dev. Las escrituras a este dispositivo serán cifradas y las lecturas descifradas. Se podrá montar un sistema de archivos en el mismo de la manera habitual, pero no se podrán acceder a los datos sin la clave [20].

El módulo del núcleo DM-Crypt trabaja al nivel del dispositivo de bloques, permitiendo a los usuarios cifrar particiones completas. El proceso es transparente a la aplicación, permitiendo el acceso al usuario que haya sido validado en el sistema. DM-Crypt cifra el denominado dispositivo de respaldo (el disco físico) y hace uso de un dispositivo de bloque virtual para proporcionar acceso al contenido en claro, bajo */dev/mapper*. Los usuarios pueden acceder a este dispositivo de bloques para configurar y montar el sistema de ficheros [17].

DM-Crypt se instala en una capa flexible conocida como dispositivo mapeador. Los módulos del dispositivo mapeador se configuran a través de las denominadas tablas *Device Mapper* (DM), ficheros de texto simples que especifican cómo debe manejar el dispositivo mapeador los accesos a las zonas del disco virtual. El formato de las tablas DM para DM-Crypt es muy pesado para usarlo a diario. El software espera que la clave sea una cadena hexadecimal de tamaño fijo. El módulo utiliza la clave para cifrar los datos del dispositivo de bloques. Sin embargo, almacenar la clave de forma permanente en los ficheros de las tablas DM es como dejar la llave colgando de la puerta. Por ello, la clave hay que introducirla en el momento de montar el dispositivo. Teclear hasta 32 caracteres hexadecimales de memoria puede que no sea fácil, pero cryptsetup puede ayudar [17].

1.4.1.3. DM-Crypt/Cryptsetup

Cryptsetup es una herramienta que genera una clave criptográfica desde la contraseña suministrada por el usuario, y luego se la pasa al núcleo.

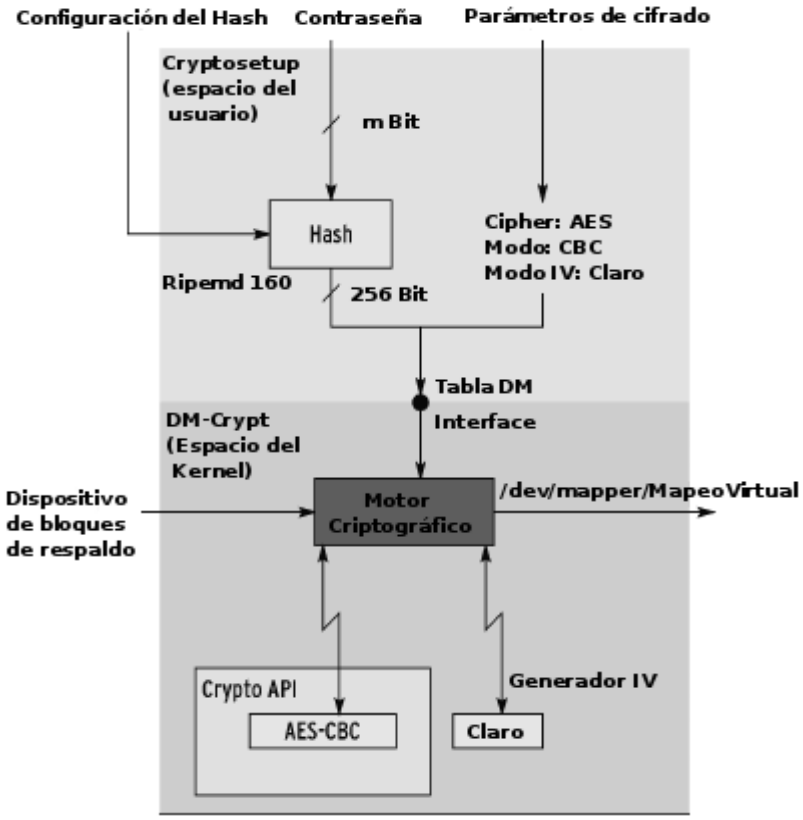

Figura 1.2: Cryptsetup pide al usuario una contraseña y usa la función *hash* Ripemd 160 para crear una clave. DM-Crypt usa la clave para cifrar y descifrar.

Como se muestra en la figura 1.2 es posible configurar dos características de Cryptsetup: la generación de claves mediante la configuración del *hash* y los parámetros de cifrado.

La generación de claves determina cómo producirá Cryptsetup las claves a partir de la contraseña suministrada por el usuario, aunque por defecto usa un algoritmo *hash* que comprime la información para proveer un número fijo de bytes, permitiéndole al usuario escoger una contraseña de cualquier longitud.

Para el proceso de cifrado se seleccionan dos parámetros: el algoritmo y el modo de operación. Cryptsetup pasa estos parámetros y la clave obtenida al núcleo y el módulo DM-Crypt coordina el procedimiento, usando la

CryptoAPI para manejar el cifrado. [17]

Desafortunadamente, Cryptsetup tiene un lado negativo, la mayoría de sus parámetros se encuentran en *scripts* o en ficheros de configuración que, obviamente, no pueden estar en las particiones cifradas. Si se pierden estos ficheros o no puede recordar la configuración de un disco extraíble, se perderá el acceso a los datos cifrados. [17]

1.4.1.4. DM-Crypt/LUKS

LUKS de sus siglas en ingles *"Linux Unified Key Setup"*, es un estándar formal implementado por la herramienta Cryptsetup-LUKS (figura 1.3) que en el cual se eliminan los problemas de la segregación de los datos de DM-Crypt/Cryptsetup [17]. Este último es una versión del Cryptsetup original. Las principales diferencias entre los dos están dadas por la generación de claves y por la manera en que LUKS protege la clave maestra con la que cifra los datos en la partición.

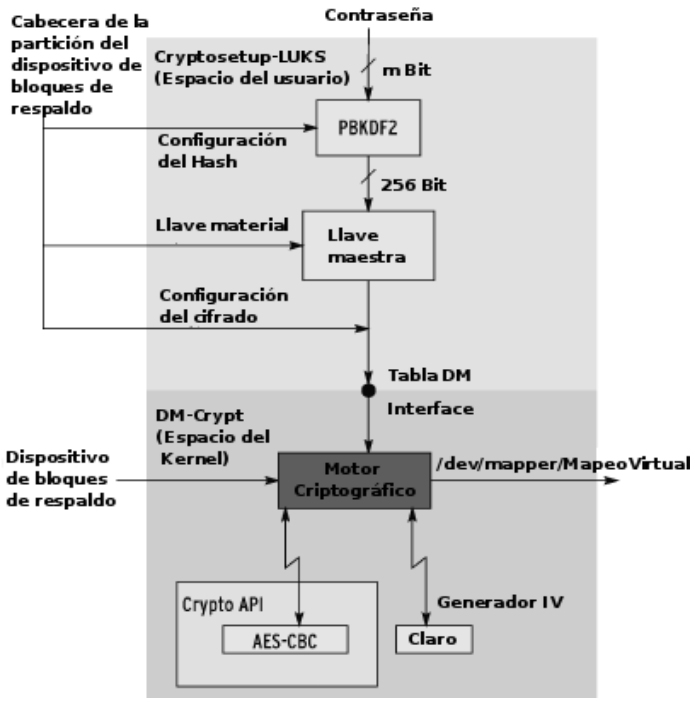

Figura 1.3: Cryptsetup-LUKS almacena los parámetros de la partición cifrada en la cabecera de la partición del dispositivo de bloques de respaldo. La clave obtenida protege la clave maestra que cifra los datos en la partición.

LUKS añade los parámetros que necesita Cryptsetup-LUKS para generar la clave desde una contraseña introducida por el usuario a la cabecera de la partición cifrada. Cada clave contiene una copia cifrada de la clave maestra que DM-Crypt utiliza para proteger los datos (figura 1.4).

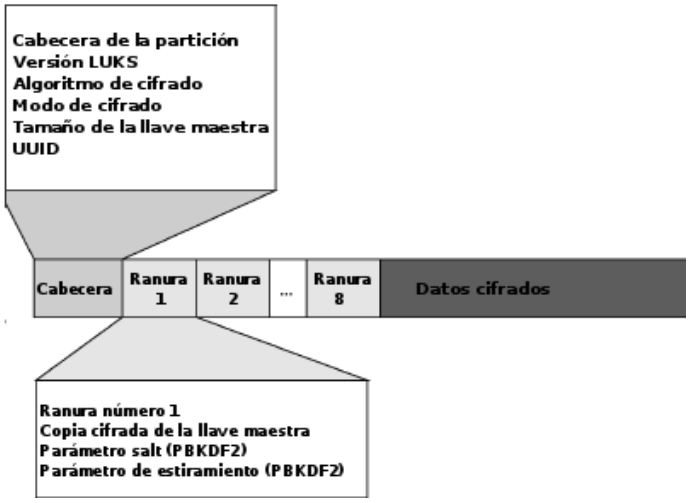

Figura 1.4: LUKS define una cabecera para las particiones DM-Crypt que incluye toda la información para generar una clave de forma segura.

Como es parte de la partición cifrada, la configuración está siempre disponible justo allí donde se necesita. La gestión de claves de LUKS está basada en tres conceptos: jerarquía de claves, PBKDF2 y almacenamiento de información anti forense [17]. Estos conceptos en los que esta basado LUKS hacen posible la seguridad en la gestión de las contraseñas y varias ventajas como son: compatibilidad a través de la estandarización, seguridad contra ataques de baja entropía gracias a PBKDF2, soporte para múltiples contraseñas y varios factores de autenticación usando la jerarquía de contraseñas, revocación efectiva de las contraseñas gracias a AFsplitter [6], que hace posible que se pueda cambiar la contraseña con la que se accede a la información cifrada. Además Dm-Crypt/LUKS tiene el mejor soporte en la distribución Gentoo [6] y Nova la distribución de GNU/Linux cubana está basada en esta distribución de GNU/Linux.

1.4.1.5. eCryptfs

eCryptfs es un sistema de cifrado de archivos para GNU/Linux que aprovecha el servicio de llavero recientemente introducido en el núcleo, la API de cifrado del núcleo (CryptoAPI), PAM, OpenSSL/GPGME, *Trusted Platform Module* (TPM), y el anillo de claves de GnuPG con el fin de hacer el proceso de gestión de la clave y la autenticación, segura a los usuarios finales [9].

eCryptfs es único entre la mayoría de las soluciones de cifrado de sistema de archivos ya que en él almacena un conjunto completo de los metadatos de cifrado, junto con cada archivo de manera individual, al igual que archivos cifrados con formato PGP. Esto permite que los archivos cifrados se puedan transferir a través de dominios de confianza, manteniendo la capacidad de que las personas con las credenciales adecuadas puedan acceder a esos archivos. Debido a que el cifrado y el descifrado se realiza en la capa VFS, el proceso se hace transparente desde la

perspectiva de la aplicación [9].

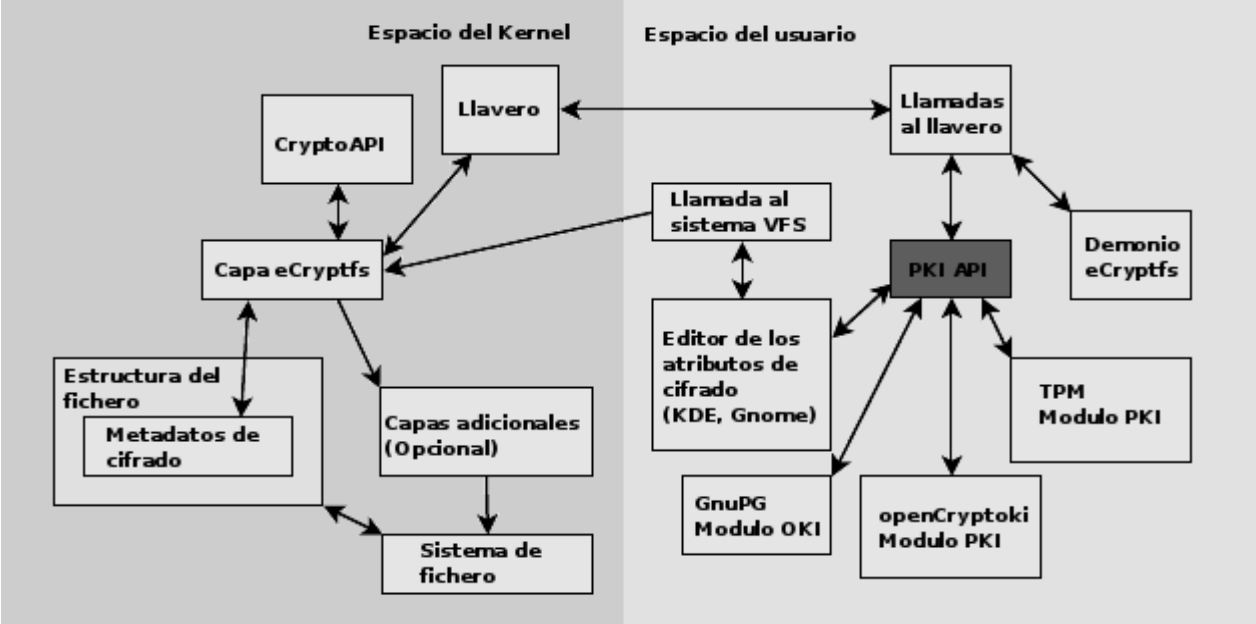

Figura 1.5: Arquitectura de eCryptfs

La principal desventaja de esta herramienta es que al ser un sistema de cifrado de archivos en lugar de un sistema de cifrado por bloques, los metadatos se mantienen visibles: número de ficheros cifrados, permisos de los ficheros, tamaño, fecha de modificación, etc.

Además eCryptfs debe usarse para la protección de la confidencialidad de los datos, en un dispositivo de almacenamiento que está fuera del control de un entorno *host* de confianza y no para cifrar los datos escritos en la memoria swap. Es recomendable que los usuarios empleen DM-Crypt para cifrar la memoria swap en un equipo donde los datos sensibles pueden ser cargados en la memoria en algún momento [9].

1.4.1.6. Scramdisk4Linux (SD4L)

Scramdisk4Linux está escrito en C, las herramientas adicionales están escritas en C++, posee una interfaz visual que utiliza la librería QT. El código criptográfico fue tomado de la librería *Catacomb* escrita por Mark Wooding. El controlador del núcleo es compatible por las versiones 2.4.x y 2.6.x hasta 2.6.32 [12].

Los contenedores ScramDisk4Linux constan de una cabecera y el volumen cifrado que incluye el sistema de archivos y el contenido de todos los archivos. En ambos formatos la cabecera y el volumen son indistinguibles y sin

el conocimiento de la contraseña no se podrían diferenciar de un conjunto de números aleatorios por lo que en estos contenedores no se podrían separar archivos, particiones o medios de almacenamiento [13].

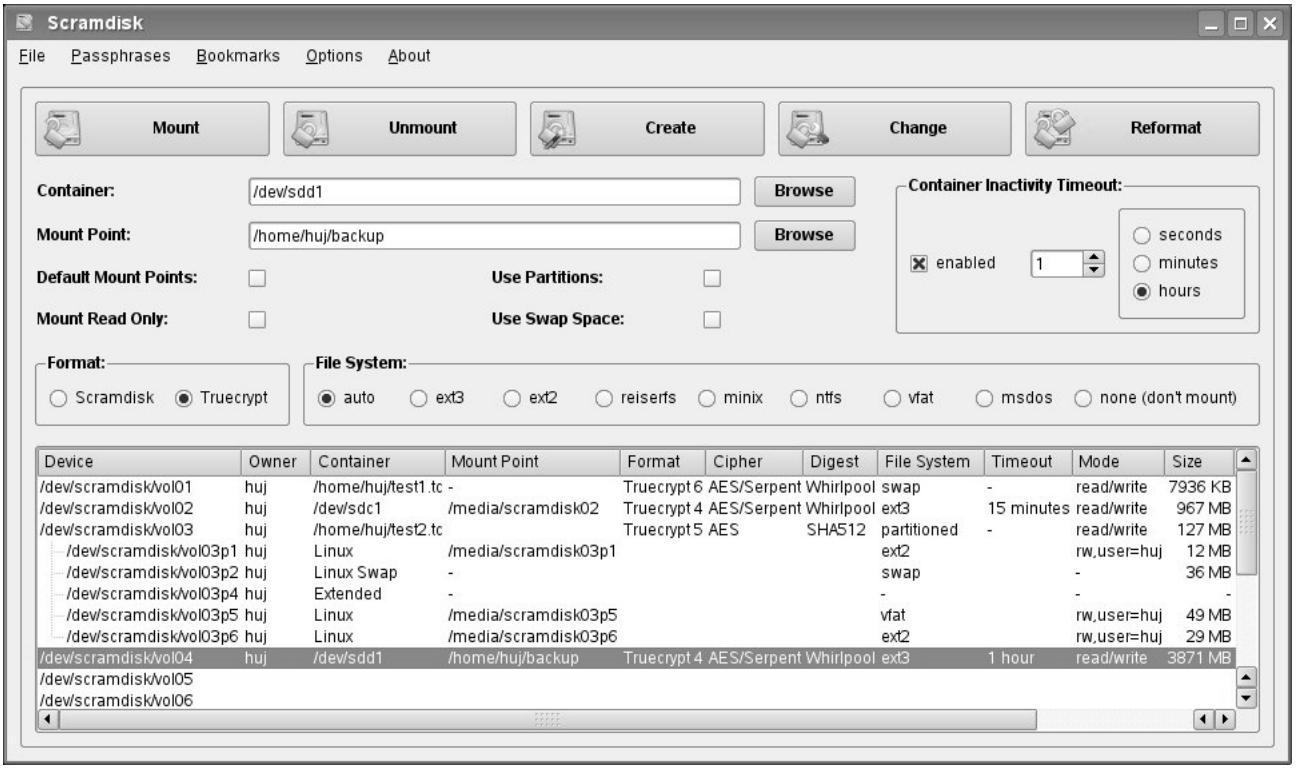

Figura 1.6: Interfaz gráfica de SD4L

Algunas de las funcionalidades que posee la herramienta son:

- Acceso a contenedores creados por Scramdisk en otras plataformas y por otras versiones de SD4L, incluso por la de Windows [12].
- Acceso a contenedores generados por TrueCrypt [2] en otras plataformas, instalaciones o por el software TrueCrypt en sus versiones de la 4.1 a la 6.2 en Windows, MacOS o GNU/Linux [12].
- Creación de contenedores compatibles con TrueCrypt que pueden ser abiertos por versiones de TrueCrypt que sean iguales o superiores a la versión seleccionada [12].

[2] TrueCrypt no ha sido incluida en la selección de las herramientas a comparar para realizar la selección precisamente por ser privativa.

- SD4L incluye una herramienta de reparación para contenedores dañados de ScramDisk y TrueCrypt 6 [12].
- Los contenedores de ScramDisk se pueden formatear a contenedores de TrueCrypt sin la necesidad de crear un nuevo contenedor y copiar los ficheros a este contenedor, los formatos son TrueCrypt 4 ó 5 [12].
- Los contenedores se pueden desmontar automáticamente luego de un tiempo de inactividad [12].
- Un contenedor se puede formatear para ser usado como memoria swap [12].

Luego de analizar cada una de las herramientas, teniendo en cuenta sus características, capas de cifrado y modos de funcionamiento, se decide utilizar DM-Crypt/LUKS por ser la que más ventajas proporciona para el desarrollo de la aplicación y la de mejores resultados en la comparación (Anexos 1, 2, 3 y 4).

1.4.2. Lenguaje de programación

En la actualidad existen innumerables lenguajes de programación, algunos creados con fines bastante específicos. La elección del lenguaje de programación se debe realizar cuidadosamente, revisando si a la larga es el lenguaje que permitirá afrontar todos los retos que surjan a lo largo del proyecto. Una posible causa de fracaso de un proyecto es la mala elección del lenguaje de programación.

Para desarrollar la aplicación que se propone en el presente trabajo se ha seleccionado C++ por un conjunto de condiciones a la hora de implementar la aplicación y por las características propias del lenguaje.

Condiciones a tener en cuenta:

- La aplicación va a interactuar con PAM y DM-Crypt, ambas herramientas escritas en C.
- Siendo una herramienta de seguridad disminuye la posibilidad de potenciales vulnerabilidades al no mezclar varios lenguajes de programación.

Características del lenguaje:

- **Alto nivel**: permite abstraerse de la arquitectura y funcionamiento del hardware.
- **Orientado a objetos**: al orientar la programación a objetos el programador será capaz de diseñar aplicaciones desde un punto de vista más cercano a la vida real, permitiendo la reutilización de código de forma más lógica y productiva.
- **Velocidad**: el código resultante de una compilación en C++ es muy eficiente, gracias a su capacidad de actuar como lenguaje de alto y bajo nivel y a la reducida medida del lenguaje.
- **Programación modular**: un cuerpo de aplicación en C++ puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Además, esta característica permite unir código en C++ con código producido en otros lenguajes de programación como Ensamblador o el propio C.

1.4.3. IDE de desarrollo

La selección del IDE de desarrollo para C++ en GNU/Linux también requiere una investigación pues de él depende el nivel de automatización con el que se construye la aplicación.

Luego de revisar las funcionalidades ofrecidas por los diferentes IDEs disponibles dos de ellos cumplían con las características necesarias, Eclipse con el componente CDT y el QtCreator.

Características:

1. Editor de C/C++ (Funcionalidades básicas, realce de sintaxis, autocompletado de código, etc).
2. Depurador C/C++.
3. Lanzador C/C++.
4. Parser[3].
5. Herramienta de búsqueda.
6. Proveedor de asistencia de contenido.
7. Generador de *Makefile*[4].
8. Potente administrador de proyecto.
9. Altamente configurable.

Añadiéndose a QtCreator su integración con la GUI de QT, razón por la cual finalmente se decidió usarlo.

1.4.4. Otras herramientas

Se mencionan a continuación otras herramientas usadas tanto para la construcción de la aplicación propuesta, como para la confección del documento de tesis.

Doxygen: generador de documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft), PHP, C# y D.

Gimp: programa de edición de imágenes. Ha sido utilizado en la edición de algunas imágenes recogidas en el documento para facilitar la comprensión del contenido.

[3]Programa analizador sintáctico.

[4]Programa analizador sintáctico. Un fichero de configuración usado por el programa make definiendo la ubicación del código fuente, como deberá ser compilado y vinculado para crear un programa ejecutable.

TEX: lenguaje de composición con capacidad para macros escrito por Donald E. Knuth. TEX toma una secuencia de comandos de tipografía, escritos en un guión en un archivo ASCII y los ejecuta. Muchos de los "trucos" del proceso de impresión fueron modelados por Knuth como algoritmos de computación e incorporados a TEX, de ahí su excelente apariencia impresa [1].

LATEX: sistema de preparación de documentos diseñado por Leslie Lamport en 1985, desarrollado a partir de un lenguaje de tipografía llamado TEX [1]. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LATEX es comparable a la de una editorial científica de primera línea.

LYX: sistema de preparación de documentos construido sobre el sistema de proceso de documentos LATEX. Es excelente para crear complejos artículos técnicos y científicos con matemáticas, referencias cruzadas, bibliografías, índices, etc. Es muy bueno para documentos de cualquier longitud en los que se requieren las capacidades de procesamiento usuales: paginación y división automática de secciones, corrección ortográfica, etc. Provee un moderno enfoque a la escritura de documentos con ordenador utilizando un paradigma de lenguaje de diseño, enfoque que rompe con la obsoleta tradición del "como una máquina de escribir". Está diseñado para autores que desean una presentación profesional con rapidez y un mínimo de esfuerzo sin tener que ser especialista en composición gráfica; esta tarea la realiza sobre todo el ordenador, no el autor, que con LYX puede concentrarse en los contenidos [1].

RATS: por sus siglas en inglés *"Rough Auditing Tool for Security" (Herramienta de Auditorías Superficiales para la Seguridad),* es la herramienta recomendada por CLASP para realizar auditorías al código fuente escrito en los lenguajes C, C++, Perl, Python y PHP en busca de errores comunes de programación como desbordamientos de buffer.

Visual Paradigm for UML: herramienta CASE para el modelado de un software durante su ciclo de vida: negocio, requerimientos, análisis y diseño, construcción, pruebas y despliegue. Esta aplicación informática de modelado ayuda a una rápida construcción de la aplicación con alto nivel de calidad. Soporta todos los tipos de diagramas UML, ofrece amplias características de modelado de casos de uso incluyendo la función completa de UML, diagramas de casos de uso, flujo de eventos, la generación de un diagrama de actividad, etc [19].

Conclusiones del capítulo

Luego de realizar un análisis profundo y detallado del estado del arte de las aplicaciones necesarias para desarrollar el contenedor cifrado de datos personales Rubik, han sido definidas las herramientas a utilizar, a modo de resumen: DM-Crypt/LUKS y PAM para dar solución a la situación problemática usando RUP y CLASP como metodología de desarrollo y las herramientas de apoyo QtCreator para desarrollar la interfaz gráfica de la aplicación que se

desarrollará en C++; Visual Paradigm para el modelado UML, Gimp para el tratamiento de imágenes, RATS para la revisión automática de código, Doxygen para generar la documentación de la aplicación y Lyx para la confección del documento. La elección anterior se ha realizado teniendo en cuenta las ventajas del uso de cada una respecto a la solución que se construirá.

Capítulo 2

Características del sistema

El éxito del desarrollo de una aplicación informática depende en alto grado de la comprensión del problema que resuelve y la manera en que debe hacerlo. Es por ello que identificar las características que debe presentar dicha aplicación y sus funciones básicas es de tanta importancia.

En este capítulo se caracteriza la herramienta a desarrollar teniendo en cuenta las particularidades del problema que resuelve. Se parte de describir el dominio del problema y se realiza una propuesta del sistema teniendo en cuenta requisitos funcionales y no funcionales. Se modelan además los casos de uso y abuso, mediante los cuales se representa el sistema desde el punto de vista tanto de sus usuarios como de sus posibles atacantes. Estas actividades sirven como base para el futuro diseño de la aplicación, pues mediante ellas queda definida en qué dirección deberán concentrarse los esfuerzos.

2.1. Descripción del problema

Como se explicó en el capítulo introductorio existen diferentes factores que atentan contra la confidencialidad de la información personal de los usuarios finales, tanto de las diferentes distribuciones de GNU/Linux, como del sistema operativo GNU/Linux Nova, hacia el cual pretende migrar el país. Se hace necesario entonces el desarrollo de una aplicación que incremente la confidencialidad de dicha información ante accesos no autorizados a la misma.

La necesidad de esta aplicación se hace mayor en determinadas áreas o lugares en los que el acceso a la información debe ser más controlado, debido a la manipulación de información sensible o que deba mantenerse en secreto.

Un caso concreto es una unidad militar, donde el jefe, por lo general, no posee grandes conocimientos sobre informática, y donde además se encuentra a cargo de la instalación y mantenimiento de las computadoras un oficial de menor rango. Si la información confidencial que se encuentra en la computadora del jefe de la unidad no está protegida debidamente, el oficial de menor rango podría acceder a ella, violando la privacidad de la misma. En casos peores un usuario con malas intenciones podría alterar o eliminar la información, llegando a causar grandes problemas.

Este tipo de situación puede darse en todo lugar, ya sea una empresa, organización, universidad, etc. Siempre existe información que debe ser protegida del acceso no autorizado, pues de ello depende la estabilidad y correcto funcionamiento de las actividades que se desempeñan en dichos lugares.

2.2. Propuesta del sistema a desarrollar

El contenedor cifrado de datos personales, Rubik, poseerá tener ciertas características para garantizar que los usuarios finales puedan, sin grandes dificultades, proteger ellos mismos la información que deseen mantener confidencial. Para ello Rubik tendrá una interfaz intuitiva y fácil de usar, que posibilitará al root crear un contenedor cifrado para cada uno de los usuarios del sistema operativo en sus respectivos directorios personales y eliminarlo en caso necesario, de manera tal que dichos usuarios puedan guardar, dentro de sus contenedores, la información que deseen mantener confidencial.

Rubik debe permitir a los usuarios propietarios de un contenedor, especificar más de una contraseña para acceder al mismo, cambiarlas o eliminarlas siempre que lo deseen. El usuario puede escoger la forma de descifrar su contenedor, las cuales podrían ser de manera automática al autenticarse.

Estas son las características esenciales que tendrá la propuesta del sistema a desarrollar. A estás se suman otras, que contribuirán a incrementar la confidencialidad de la información personal de los usuarios, las cuales serán detalladas en el transcurso del capítulo.

2.3. Modelo de dominio

Un Modelo de Dominio es una representación visual de conceptos u objetos del mundo real significativos para un problema o área de interés. En él se captura y expresa el entendimiento logrado en una zona del conocimiento bajo investigación, como paso previo a la construcción de un sistema. Su utilidad radica además en que es un medio efectivo y fácil para comprender el entorno en el cual el sistema se insertará y deberá servir. En la figura 2.1 se muestra el diagrama de clases del dominio del problema a resolver en esta investigación.

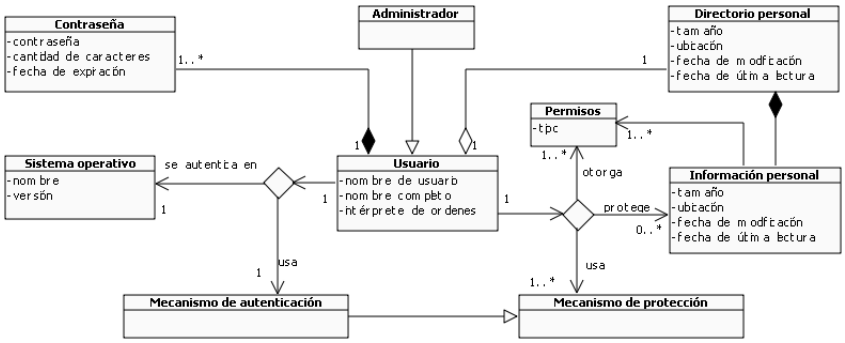

Figura 2.1: Modelo de dominio

A continuación se muestra un glosario con los conceptos del domino del problema donde se inserta esta investigación, para una mejor compresión de su entorno.

Sistema operativo: específicamente sistema operativo GNU/Linux Nova-Baire.

Mecanismo de protección: mecanismos existentes para proteger la información, como el control de acceso o el cifrado.

Usuario: del sistema operativo.

Administrador: usuario *root*, único administrador en el sistema operativo GNU/Linux que posee todos los privilegios.

Mecanismo de autenticación: mecanismos usados por los sistemas operativos para realizar la autenticación de los usuarios.

Contraseña: conjunto de caracteres usado por el usuario para autenticarse en el sistema operativo.

Directorio personal: directorio que cada usuario del sistema operativo GNU/Linux posee dentro del directorio /home. En este se guardan las configuraciones de los programas que el usuario utiliza y sus documentos.

Información personal: información de cada usuario ubicada en sus directorios personales.

Permisos: permisos que poseen los ficheros en el sistema operativo. Pueden ser de lectura, escritura y ejecución.

2.4. Especificación del entorno operacional

La especificación del entorno operacional permite a los miembros del equipo, entender el contexto operativo a tener en cuenta para el diseño de mecanismos de protección o la construcción de guías de seguridad operacional [21]. El entorno operacional en el que deberá funcionar la aplicación de forma óptima deberá contar con las siguientes características:

1. Sistemas operativos GNU/Linux, específicamente las distribuciones de Gentoo y Nova-Baire.
2. Tener instalado los siguientes paquetes:
   - qt-core en su versión 4 o superior.
   - cryptsetup versión 1.0.6-r2 o superior siempre con su respectivo parche cryptsetup-versión-rubik.patch.
   - pam versión 1.1.0 o superior.
   - núcleo versión 2.6.30 o superior con los módulos correspondientes activados (Anexo 5).
3. Las instalaciones deberán proporcionar una protección eficaz contra las escuchas no autorizadas y la transmisión de datos (cortafuegos correctamente configurado, con la base de datos actualizada de antivirus, anti-spyware, anti-rootkit, etc)

4. Las aplicaciones PAM y DM-Crypt/LUKS serán usadas por la aplicación, por tanto se debe velar por su correcta actualización ante problemas de seguridad u otros que puedan comprometer el correcto funcionamiento del contenedor.

5. Para evitar el volcado de memoria de la swap y la lectura de los archivos temporales es recomendable ambos esten cifrados (Anexo 6).

2.5. Descripción de las funcionalidades

Un requisito de software especifica un comportamiento del sistema observable externamente; tales como entradas, salidas, funciones, atributos del mismo o atributos de su entorno [4].

En las siguientes subsecciones se definen los requisitos funcionales y no funcionales que debe tener la aplicación a desarrollar.

2.5.1. Requisitos funcionales

Los requisitos funcionales son condiciones o capacidades que el sistema debe cumplir, definiendo qué debe hacer el producto.

Requisitos Funcionales		
ID.	Nombre	Descripción
RF1	Crear contenedor.	La aplicación debe permitir que un usuario con privilegios de administración pueda crear un nuevo contenedor para un usuario que aún no posea ninguno.
RF2	Eliminar contenedor.	La aplicación debe permitir que un usuario con privilegios de administración pueda eliminar un contenedor existente.
RF3	Configurar contenedor.	La aplicación debe permitir tanto a un usuario con privilegios de administración, como a un usuario propietario de un contenedor, configurar sus parámetros: forma de montaje (al iniciar sesión o a petición del usuario); permitir o no otros inicios de sesión mientras el contenedor está montado y punto de montaje.
RF4	Montar contenedor.	La aplicación debe permitir que un usuario propietario de un contenedor existente pueda montarlo para acceder a la información almacenada en él.

continúa en la próxima página...

RF5	Montar contenedor automáticamente.	La aplicación debe montar el contenedor automáticamente, cuando el propietario inicie sesión si este lo ha definido así en la configuración.
RF6	Desmontar contenedor.	La aplicación debe permitir que un usuario propietario de un contenedor existente pueda desmontarlo para proteger la información almacenada en él.
RF7	Desmontar contenedor automáticamente.	La aplicación debe desmontar automáticamente el contenedor de un usuario que cierre su sesión.
RF8	Adicionar contraseña para montar contenedor.	La aplicación debe permitir que un usuario propietario de un contenedor existente pueda añadir una nueva contraseña mediante la cual montarlo.
RF9	Eliminar contraseña de montar contenedor.	La aplicación debe permitir que un usuario propietario de un contenedor pueda eliminar una contraseña existente mediante la cual se monta el contenedor.
RF10	Impedir eliminación de única contraseña de montar contenedor.	La aplicación debe impedir que un usuario propietario de un contenedor elimine la contraseña mediante la cual se monta el mismo, si es la única.
RF11	Modificar contraseña de montar contenedor.	La aplicación debe permitir que un usuario propietario de un contenedor pueda modificar las contraseñas mediante las cuales se monta el mismo.
Seguridad		
RFS1	Bloquear acceso al sistema operativo con contenedor montado.	La aplicación debe bloquear el acceso de otros usuarios al sistema operativo si hay un contenedor montado.
RFS2	Alertar existencia de otros usuarios autenticados en sistema operativo, al montar contenedor.	La aplicación debe alertar a los usuarios que monten un contenedor cuando existan otros usuarios autenticados en el sistema.

2.5.2. Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que debe poseer el sistema, una vez resuelto qué debe hacer, debe determinarse cómo debe hacerlo. Generalmente se encuentran relacionados con los requisitos funcionales y en muchos casos determinan el éxito de un proyecto.

Requisitos No Funcionales		
Identificador	Nombre	Descripción
Interfaz externa		
RNF_IE1	La interfaz debe ser sencilla.	La interfaz de la aplicación deberá ser lo más simple e intuitiva posible para el usuario, sin limitar sus funciones.
Diseño e implementación		
RNF_DI1	Se debe usar C y C++ como lenguajes de programación.	Serán usados los lenguajes de programación C y C++ para lograr una buena comunicación con el resto de los componentes del sistema operativo que están programados en dichos lenguajes.
RNF_DI2	Se debe usar el estilo de código BSD KNF.	Al desarrollar la aplicación será utilizado el estilo de código BSD KNF por todos los programadores.
Hardware		
RNF_HDW1	Mínimas prestaciones requeridas.	Se requiere de una arquitectura x86 con al menos 256 MB de RAM.
Software		
RNF_SWF1	Sistema operativo requerido.	Se requiere el uso de los sistemas operativos GNU/Linux, específicamente Nova-Baire o Gentoo.
RNF_SWF2	Librería gráfica requerida.	Se requiere disponer de QT 4.4.x, GCC 4.x. o versiones superiores.
RNF_SWF3	Versión del núcleo requerido.	Se requiere disponer de una versión del núcleo 2.6.30 o superior, con soporte para DM-Crypt. (Anexo 5)
RNF_SWF4	Terceras aplicaciones requeridas.	Se requiere disponer de las aplicaciones pam, device-mapper, cryptsetup y multipath-tools.
Seguridad		

continúa en la próxima página...

RNF_SG1	Validar lecturas.	La aplicación debe validar cada uno de los datos leídos ya sea desde el teclado o desde los ficheros de configuración.
RNF_SG2	Validar la integridad de los ficheros de configuración.	La aplicación debe validar que el contenido de los ficheros de configuración esté sintácticamente correcto, según la estructura que se ha definido para ello.

2.6. Definición de los casos de uso y abuso

Los casos de uso representan la interacción entre los usuarios y el sistema. Son "fragmentos" de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa, un caso de uso especifica una secuencia de acciones que el sistema lleva a cabo interactuando con sus actores, incluyendo alternativas dentro de dicha secuencia [7].

Los casos de abuso describen situaciones de uso indebido que el sistema debe ser capaz de detectar e impedir. Pueden ser definidos como acciones llevadas a cabo por atacantes que resultan en pérdida para la organización o para un actor en específico [8]. Al describirlos se contribuye a conformar la documentación necesaria para conocer las acciones malintencionadas que un atacante podría llevar a cabo en los distintos escenarios concebidos [18].

2.6.1. Definición de los actores y actores de abuso.

Los actores representan terceros fuera del sistema, ya sean usuarios o sistemas externos, que juegan un rol determinado cuando interactúan con el sistema [7]. En la tabla 2.3 se muestran los actores que interactúan con el sistema acompañados de una breve descripción.

Cuadro 2.3: Definición de los actores

Actores	Descripción
usuario	Usuario sin privilegios de administración que no puede crear ni eliminar contenedores solamente montarlo, desmontarlo o modificar la configuración de un contenedor que le pertenezca.
root	Usuario administrador con todos los privilegios, encargado de crear los contenedores, eliminarlos y realizar las demás operaciones que realizan los usuarios.

Los actores de abuso son los que intencionalmente o no inician un caso de abuso [8].

Cuadro 2.4: Definición de los actores de abuso

Actores de abuso	Descripción
usuario malicioso	Usuario común pero que usa el contenedor con fines maliciosos.

2.6.2. Diagrama de casos de uso y casos de abuso

Los diagramas de casos de uso no son más que diagramas que ilustran a los actores, casos de uso y las relaciones entre ellos [4]. En los diagramas de casos de uso pueden ser representados de igual manera los casos de abuso, ver figura 2.2.

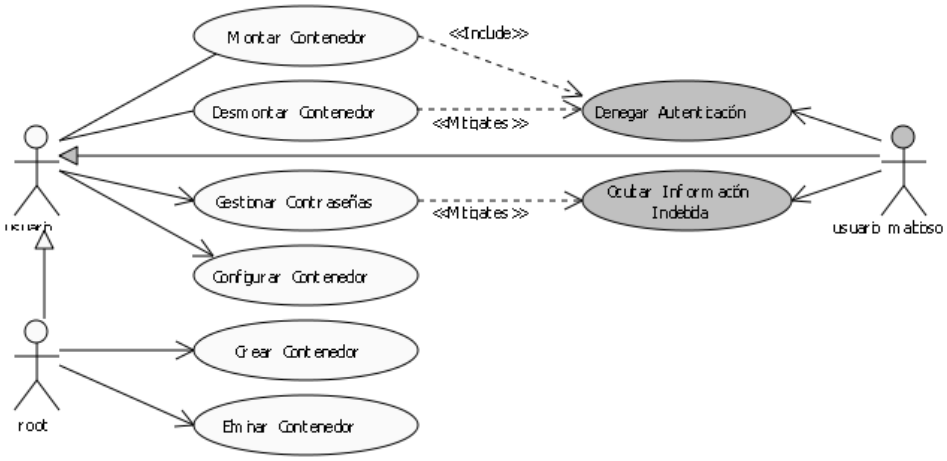

Figura 2.2: Diagrama de casos de uso y casos de abuso.

2.6.3. Realización de los casos de uso. Descripción textual detallada.

RUP se caracteriza por ser iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura, por lo que es muy importante detallar correctamente cada uno de los casos de uso del sistema, para tener buena una comprensión de las funcionalidades que debe brindar la aplicación a los usuarios y basar todo el proceso de desarrollo en lo que el usuario final realmente desea y necesita. A continuación se listarán y detallarán cada uno de los casos de uso del sistema.

Cuadro 2.5: CU Crear Contenedor.

Caso de Uso:	Crear Contenedor.
Actores:	root.
Resumen:	El caso de uso se inicia cuando un usuario con privilegios de administración accede a crear un contenedor.
Prioridad:	Crítico.
Referencia:	RF1, RF3.
Precondiciones:	El usuario debe estar correctamente autenticado en el sistema operativo y debe poseer privilegios de administración.

Flujo Normal de Eventos.	
Acción del Actor	Respuesta del Sistema
1. Accede a **Crear un contenedor**.	2. Muestra los parámetros de configuración para el contenedor que se va a crear: tamaño, algoritmo de cifrado y punto de montaje.
3. Configura las características que tendrá el contenedor. 4. Solicita crear el contenedor con las características especificadas.	5. Valida las características que el usuario seleccionó para la creación del contenedor. 6. Crea el contenedor. 7. Muestra una notificación de creación satisfactoria. Fin del CU.

Flujo alterno # 1 "Errores en parámetros de configuración del contenedor"	
Acción del Actor	Respuesta del Sistema
	(Viene del paso # 5 del Flujo normal) 1. Muestra un mensaje informando al usuario que hay datos incorrectos en las características del contenedor. 2. Retorna al paso # 2 del Flujo normal.

Poscondiciones:	Se ha creado un contenedor.

Cuadro 2.6: CU Eliminar Contenedor

Caso de Uso:	Eliminar Contenedor.
Actores:	root.
Resumen:	El caso de uso se inicia cuando un usuario con privilegios de administración accede a eliminar un contenedor.
Prioridad:	Crítico.
Referencia:	RF2.
Precondiciones:	El usuario debe estar correctamente autenticado en el sistema operativo, debe poseer privilegios de administración y debe existir al menos un contenedor.

Flujo Normal de Eventos.	
Acción del Actor	Respuesta del Sistema
1. Accede a **Eliminar un contenedor**.	2. Muestra un listado con los contenedores existentes.
3. Selecciona el contenedor o los contenedores que desea eliminar. 4. Solicita **Eliminar.**	5. Solicita confirmación para eliminar contenedor.
6. Confirma eliminar el contenedor.	7. El sistema elimina el contenedor. 8. Muestra un mensaje indicando que se eliminó el contenedor satisfactoriamente. Fin del CU.

Flujo alterno #1 "Cancela la eliminación".	
Acción del Actor	Respuesta del Sistema
(Viene del paso # 6 del Flujo normal) 1. Cancela eliminar contenedor**.**	2. Retorna al paso # 2 del Flujo Normal de Eventos.

Poscondiciones:	Se eliminó un contenedor.

Cuadro 2.7: CU Configurar Contenedor.

Caso de Uso:	Configurar Contenedor.
Actores:	usuario.
Resumen:	El caso de uso inicia cuando un usuario accede a cambiar la configuración de su contenedor.
Prioridad:	Crítico.
Referencia:	RF3.
Precondiciones:	El usuario debe estar correctamente autenticado en el sistema operativo, tener un contenedor y tenerlo montado.

Flujo Normal de Eventos.	
Acción del Actor	Respuesta del Sistema
1. Accede a **Cambiar Configuración.**	2. Muestra la configuración actual del contenedor: forma de montaje, acceso a otros usuarios y punto de montaje.
3. Cambia los parámetros de configuración. 4. Solicita guardar la nueva configuración.	5. Valida que no existan errores en el nuevo valor de los parámetros de configuración. 6. Guarda la nueva configuración. Fin del CU.
Flujo alterno #1 "Errores en la configuración"	
	(Viene del paso #5 del Flujo normal de Eventos) 1. Advierte que existen valores incorrectos en los parámetros cambiados por el usuario. 2. Vuelve al paso #2 del Flujo Normal de Eventos.

Poscondiciones:	Se cambia la configuración del contenedor.

Cuadro 2.8: CU Montar Contenedor.

Caso de Uso:	Montar Contenedor.
Actores:	usuario.
Resumen:	El caso de uso se inicia cuando un usuario accede a montar su contenedor.
Prioridad:	Crítico.
Referencia:	RF4, RF5, RFS1, RFS2.
Precondiciones:	El usuario debe estar correctamente autenticado en el sistema operativo y debe ser propietario de un contenedor.
Flujo Normal de Eventos.	
Sección "Montar contenedor a petición del usuario".	
Acción del Actor	Respuesta del Sistema
1. Accede a **Montar contenedor**.	*(El flujo de acciones continúa igual al de la sección "Montar contenedor al iniciar sesión" a partir del paso # 2.)*
Sección "Montar contenedor al iniciar sesión".	
Precondiciones:	El usuario debe tener el contenedor configurado para montarse automáticamente cuando él inicie su sesión.
Acción del Actor	Respuesta del Sistema
1. Inicia sesión en el sistema operativo.	2. Verifica que no existan otros usuarios que hayan iniciado sesión en el sistema operativo. 3. Solicita la contraseña para descifrar el contenedor.
4. Introduce la contraseña.	5. Verifica la contraseña. 6. Verifica la configuración del contenedor 7. Monta el contenedor y le permite el acceso al usuario. Fin del CU.
Flujo alterno # 1 "Otros usuarios en el sistema operativo"	
Acción del Actor	Respuesta del Sistema

continúa en la próxima página...

	(Viene del paso # 2 de la sección "Montar contenedor al iniciar sesión") 1. Detecta otros usuarios autenticados. 2. Advierte los riesgos de montar el contenedor cuando hay otros usuarios trabajando en el sistema operativo y pide confirmación para montar el contenedor.
3. Confirma montar el contenedor.	4. Retorna al paso # 3 de la sección "Montar contenedor al iniciar sesión".
Flujo alterno # 1.1 "Cancelar el montaje"	
(Viene del paso # 3 del Flujo alterno # 1 "Otros usuarios en el sistema operativo".) 1. Cancela montar el contenedor.	2. Cancela la operación de montar el contenedor. Fin del CU.
Flujo alterno # 2 "Contraseña incorrecta"	
	(Viene del paso # 5 de la sección "Montar contenedor al iniciar sesión".) 1. Muestra un mensaje indicando que no se pudo efectuar el montaje del contenedor. Fin del CU.
Flujo alterno # 3 "Bloquear acceso al sistema operativo"	
	(Viene del paso # 6 de la sección "Montar contenedor al iniciar sesión".) 1. Detecta que esta activada la opción de bloquear el acceso de otros usuarios al sistema operativo. 2. Bloquea el acceso de otros usuarios al sistema operativo. 3. Vuelve al paso # 7 de la sección "Montar contenedor al iniciar sesión".
Poscondiciones:	Se monta un contenedor.

Cuadro 2.9: CU Desmontar Contenedor.

<table>
<tr><td>Caso de Uso:</td><td colspan="2">Desmontar Contenedor.</td></tr>
<tr><td>Actores:</td><td colspan="2">usuario.</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de uso se inicia cuando un usuario accede a desmontar su contenedor.</td></tr>
<tr><td>Prioridad:</td><td colspan="2">Crítico.</td></tr>
<tr><td>Referencia:</td><td colspan="2">RF6, RF7.</td></tr>
<tr><td>Precondiciones:</td><td colspan="2">El usuario debe estar correctamente autenticado en el sistema operativo, debe tener un contenedor y este tiene que estar montado.</td></tr>
<tr><td colspan="3">Flujo Normal de Eventos.</td></tr>
<tr><td colspan="3">Sección "Desmontar contenedor a petición del usuario".</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td>1. Accede a Desmontar contenedor.</td><td colspan="2">(El flujo de acciones continúa igual que al de la sección "Desmontar contenedor al cerrar sesión" a partir del paso # 2.)</td></tr>
<tr><td colspan="3">Sección "Desmontar contenedor al cerrar sesión".</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td>1. Cierra sesión en el sistema operativo.</td><td colspan="2">2. Verifica la configuración del contenedor.
3. Desmonta el contenedor. Fin del CU.</td></tr>
<tr><td colspan="3">Flujo alterno # 3 "Desbloquear acceso al sistema operativo"</td></tr>
<tr><td></td><td colspan="2">(Viene del paso # 2 de la sección "Montar contenedor al iniciar sesión".)
1. Detecta que esta activada la opción de bloquear el acceso de otros usuarios al sistema operativo.
2. Desbloquea el acceso de otros usuarios al sistema operativo.
3. Vuelve al paso # 3 de la sección "Desmontar contenedor al cerrar sesión".</td></tr>
<tr><td>Poscondiciones:</td><td colspan="2">Se desmonta el contenedor.</td></tr>
</table>

Cuadro 2.10: CU Gestionar contraseña.

Caso de Uso:	Gestionar contraseña.
Actores:	usuario.
Resumen:	El caso de uso se inicia cuando un usuario accede a gestionar las contraseñas para acceder a su contenedor.
Prioridad:	Crítico.
Referencia:	RF8, RF9, RF10, RF11.
Precondiciones:	El usuario debe estar correctamente autenticado en el sistema operativo, debe tener un contenedor y este tiene que estar montado.

Flujo Normal de Eventos.	
Acción del Actor	Respuesta del Sistema
1. Accede a **Gestionar contraseñas**.	2. Muestra las opciones: ▪ Adicionar contraseña al contenedor (Sección "Adicionar Contraseña"). ▪ Eliminar contraseña del contenedor (Sección "Eliminar Contraseña"). ▪ Cambiar contraseña del contenedor (Sección "Cambiar Contraseña").
3. Accede a una de las opciones mostradas.	*(Continúa en la sección correspondiente a la opcion elegida)*
Sección "Adicionar Contraseña".	
Acción del Actor	Respuesta del Sistema
	1. Solicita la nueva contraseña y su confirmación.
2. Ingresa nueva contraseña y la confirma.	3. Verifica que la confirmación sea válida. 4. Adiciona la nueva contraseña. Fin del CU.
Flujo alterno #1 "Confirmación Incorrecta"	

continúa en la próxima página...

	(Viene del paso # 3 de la sección "Adicionar contraseña") 1. Detecta que la contraseña no se ha confirmado correctamente. 2. Informa imposibilidad de ingresar contraseña debido a confirmación incorrecta. 3. Retorna al paso # 1 del Flujo normal de Eventos.
Sección "Eliminar Contraseña"	
Acción del Actor	Respuesta del Sistema
	1. Solicita contraseña a eliminar.
2. Introduce la contraseña que desea eliminar.	3. Verifica que existe esa contraseña. 4. Verifica que no sea la última. 5. Elimina la contraseña. Fin del CU.
Flujo alterno # 1 "No existe esa contraseña."	
	(Viene del paso # 3 de la Sección "Eliminar Contraseña") 1. Informa que no existe la contraseña que desea eliminar. 2. Retorna al paso # 1 de la sección "Eliminar Contraseña".
Flujo alterno # 2 "No se puede eliminar la última contraseña."	
	(Viene del paso # 4 de la Sección "Eliminar Contraseña".) 1. Informa que no se puede eliminar la única contraseña que existe para montar el contenedor. 2. Fin del CU.
Sección "Cambiar Contraseña"	
Acción del Actor	Respuesta del Sistema
	1. Solicita contraseña a cambiar, nueva contraseña y su confirmación.

continúa en la próxima página...

2. Ingresa contraseña a cambiar, la nueva contraseña y su confirmación.	3. Verifica que existe la contraseña que se desea cambiar. 4. Verifica que la confirmación de la nueva contraseña es correcta. 5. Se cambia la contraseña. Fin del CU.
Flujo alterno #1 "No existe esa contraseña."	
	(Viene del paso #3 de la Sección "Cambiar Contraseña".) 1. Informa que no existe la contraseña que desea cambiar. 2. Retorna al paso #1 de la sección "Cambiar Contraseña".
Flujo alterno #2 "Confirmación incorrecta"	
	(Viene del paso #4 de la Sección "Cambiar Contraseña".) 1. Informa que no se ha confirmado correctamente la contraseña. 2. Retorna al paso #1 del Flujo normal de Eventos.
Poscondiciones:	Se adiciona, elimina o actualiza una contraseña de un contenedor.

2.6.4. Realización de los casos de abuso. Descripción textual detallada.

De manera similar a los casos de uso, los casos de abuso deben ser detallados. De esta forma se podrá detectar un mayor número de posibles puntos débiles a tener en cuenta a la hora de desarrollar la aplicación, logrando con ello aumentar su seguridad. Las acciones indebidas, en forma de casos de abuso, deberán ser mitigadas o eliminadas de ser posible, y en el peor de los casos, al menos, bien documentadas.

Cuadro 2.11: CU Ocultar información indebida.

CU indebido:	Ocultar información indebida.
Actores:	usuario malicioso.

continúa en la próxima página...

Resumen:	Un usuario malicioso oculta información indebida en el contenedor pudiendo resultar perjudicial para otros usuarios, para la organización a la que pertenece o ir en contra de las políticas establecidas para regular el tipo de información que se guarda.
Precondiciones:	El usuario malicioso debe tener un contenedor.
Flujo básico:	**fb1:** Un usuario malicioso monta su contenedor y oculta en este información indebida.
Puntos de captura:	**pc1:** Existe otra contraseña para auditar los contenedores en poder de las personas autorizadas con el objetivo de revisar la información que se almacena en los mismos.
Desencadenantes:	**dc1:** Siempre puede suceder.
Poscondiciones:	**Peor de los casos:** El usuario malicioso guarda la información indebida en el contenedor y no es detectado. **Mejor de los casos:** El usuario malicioso nunca guarda información indebida en el contenedor debido al temor de ser sorprendido por una auditoría o es sorprendido con información indebida dentro de su contenedor.
Amenaza:	Las políticas establecidas para regular el tipo de información que se debe guardar en los contenedores.

Cuadro 2.12: Caso de uso indebido Denegar autenticación.

CU indebido:	Denegar autenticación.
Actores:	usuario malicioso.
Resumen:	Un usuario malicioso monta su contenedor para evitar que otros usuarios puedan iniciar sesión en el sistema operativo.
Precondiciones:	El usuario malicioso debe tener un contenedor.

continúa en la próxima página...

Flujo básico:	**fb1:** Un usuario malicioso monta su contenedor y aunque no esté trabajando con la información del contenedor lo mantiene montado para impedir que otros usuarios inicien sesión. **fb3:** Cuando termina de trabajar no cierra su sesión ni desmonta el contenedor, solo bloquea la sesión, impidiendo que usuarios se conecten remoto o por protocolos de red como ssh.
Puntos de captura:	**pc1:** Establecer un tiempo máximo de inactividad que hará que el contenedor se desmonte automáticamente.
Desencadenantes:	**dc1:** Siempre puede suceder.
Poscondiciones:	**Peor de los casos:** El usuario malicioso evita que otros usuarios inicien sesión en el sistema por tiempo ilimitado. **Mejor de los casos:** El usuario malicioso no puede provocar una denegación de inicio de sesión por un tiempo ilimitado debido a que el sistema desmonta los contenedores de forma automática ante cierto tiempo de inactividad.
Amenaza:	Disponibilidad de uso de la computadora por otros usuarios.

Conclusiones del capítulo

En este capítulo se han definido las características que deberá poseer la aplicación para su correcto funcionamiento, descritas a través de los requisitos funcionales, representados como casos de uso, y los requisitos no funcionales, haciendo énfasis en los de seguridad.

Con el objetivo de garantizar al máximo la seguridad de la aplicación, se han desarrollado actividades específicas definidas por CLASP como la descripción del entorno operacional y de los casos de abuso; siendo estos de gran importancia para detectar vulnerabilidades que ayuden a obtener requisitos de seguridad adicionales.

Lo anteriormente descrito ha servido para definir qué debe hacer el sistema y cuán seguro debe ser, lo cual es la base para diseñar cómo debe hacerlo en el próximo capítulo, para su posterior implementación.

Capítulo 3

Diseño del sistema

Para desarrollar software hay que hacer algo más que conducirse a ciegas a través de los flujos de trabajo guiados exclusivamente por los casos de uso, ya que solamente estos no son suficientes. También es necesaria una buena arquitectura, que de una clara perspectiva del sistema completo, con el objetivo de controlar el desarrollo, describiendo los elementos del modelo más importantes desde el punto de vista arquitectónico. Estos elementos significativos incluyen algunos de los subsistemas, dependencias, interfaces, colaboraciones, nodos y clases que describen los cimientos del sistema, necesarios para comprenderlo y desarrollarlo [7].

A desarrollar una arquitectura sólida contribuye el diseño, quien es el centro de atención hacia el final de la fase de elaboración y principios de la de construcción. Este también permite crear un plano para la implementación, que es el centro de la posterior fase de construcción, una vez que la arquitectura es estable y los requisitos están bien entendidos [7].

3.1. Arquitectura y patrones de diseño

3.1.1. Arquitectura

La arquitectura de un software puede entenderse como aquella estructura del programa que cohesiona las funcionalidades más críticas y relevantes, necesarias para el sistema, y que sirve de soporte al resto de funcionalidades finales, necesarias para el usuario [7].

Para realizar el desarrollo de la aplicación se ha determinado usar una arquitectura en 3 capas, cuyo objetivo primordial es la separación de la presentación, la lógica del negocio y el acceso a los datos. Esta separación, reduce el riesgo y el impacto de los cambios, aumenta la escalabilidad y la tolerancia a fallos y es un escenario propicio para implementar una buena política de pruebas. Además permite acelerar la creación de la aplicación, al poder distribuir entre los desarrolladores el trabajo por niveles, de este modo, cada desarrollador está totalmente abstraído del resto de los niveles que al final son integrados.

Abstracción, encapsulamiento, capas de funcionalidad bien definidas y reusabilidad son algunas de las características que distinguen este tipo de arquitectura que puede ser adaptada a diversos escenarios.

Para el desarrollo de Rubik la arquitectura quedaría como se muestra en la figura 3.1.

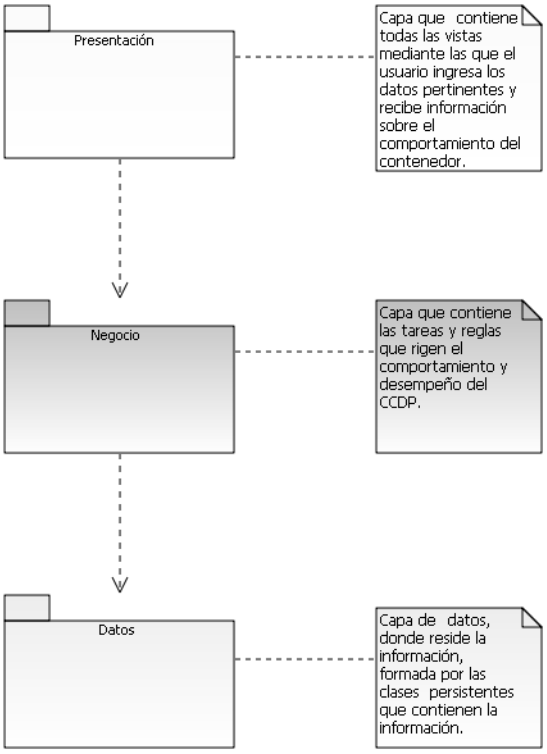

Figura 3.1: Arquitectura

3.1.2. Principios de diseño seguro

Los principios de diseño seguro son mecanismos para garantizar la seguridad del software desde su mismo diseño, haciéndolo más resistente ante los ataques que pudiera sufrir. En el diseño de esta aplicación se han tenido en cuenta los siguientes principios:

Menor privilegio: implica dar a cada componente del software (usuario, proceso, etc.) el mínimo de privilegios necesarios para realizar las tareas autorizadas que le correspondan.

Mediación completa: se debe determinar si se posee la autorización adecuada ante cualquier intento de acceder a un recurso o ejecutar algún proceso, denegándose la posibilidad si no se posee la autorización adecuada.

Fallar de forma segura por defecto: la situación por defecto debe ser la de denegar cualquier solicitud que no haya explícitamente cumplido con las condiciones necesarias para que sea permitida.

Diseño abierto: la seguridad no debe depender de un diseño oscuro sino asegurar la fiabilidad y la posesión por parte de los usuarios de las credenciales adecuadas.

Defensa en profundidad: consiste en crear múltiples barreras protectoras a través de las diferentes capas y dimensiones del sistema.

Economía de mecanismos: mantener el diseño tan simple y pequeño como sea posible en cada aspecto del sistema, minimizando la posibilidad de ocurrencia de errores y facilitando la posibilidad de su análisis.

Reducir la superficie de ataque: consiste en exponer el mínimo indispensable de funcionalidades del software, disminuyendo por tanto las posibilidades de entrada de un atacante.

Asegurar el punto más débil: considerando que los atacantes siempre buscarán el lugar de ataque que menos resistencia ofrezca, se debe analizar la aplicación y determinar sus puntos más vulnerables y tomar medidas que ayuden a aumentar la seguridad.

3.1.3. Patrones de diseño

Un patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas [14].

Patrones de seguridad utilizados

Un patrón de seguridad, es un patrón de diseño, que pretende mostrar a los desarrolladores de software la manera correcta de diseñar y aplicar soluciones, a los problemas de seguridad comunes. Estas soluciones suelen representar las características de seguridad del software.

En el desarrollo de Rubik fueron usados los patrones de seguridad: *Trusted Proxy* (Proxy de confianza) e *Input Validator* (Validador de entradas).

- *Trusted Proxy* (Proxy de confianza): en Rubik se implementa este patrón para interceptar y filtrar todas las comunicaciones entre los usuarios y los componentes a los cuales los usuarios no tienen suficientes privilegios para acceder. Al impedir el acceso directo, puede compensarse las debilidades en los componente que no están debidamente protegidos y asegurarse de que las políticas adecuadas se apliquen de manera uniforme.
- *Input Validator* (Validador de entradas): para garantizar la validez de los datos introducidos por los usuarios o leídos desde los ficheros de configuración, estos son procesados tanto en la capa de presentación, como en la de negocio.

3.2. Diagramas de clases de diseño

Un diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces (las de C++ por ejemplo) en una aplicación. Normalmente contiene clases, asociaciones y atributos;

interfaces, con sus operaciones y constantes; métodos; información sobre los tipos de atributos; navegabilidad y dependencias.

En las figuras 3.2 y 3.3 se representan, con diferente nivel de detalle, las clases de diseño de la aplicación.

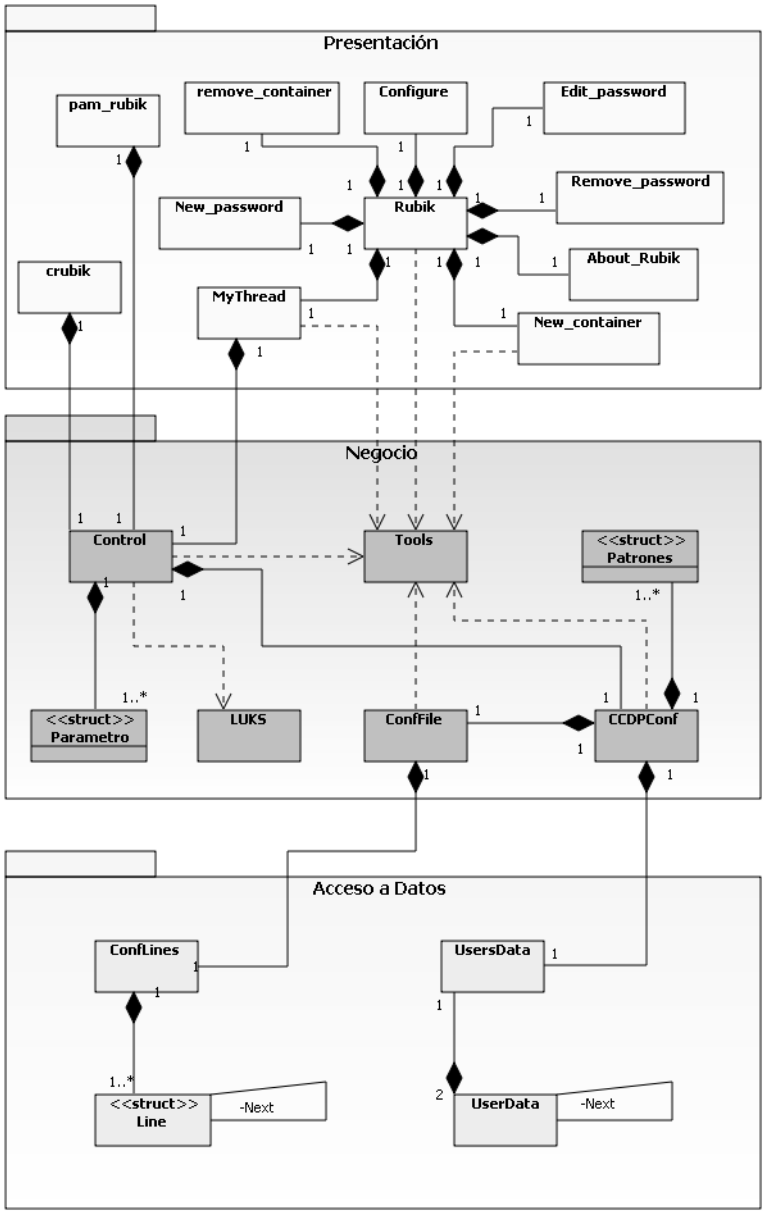

Figura 3.2: Diagrama de clases del diseño organizado por capas.

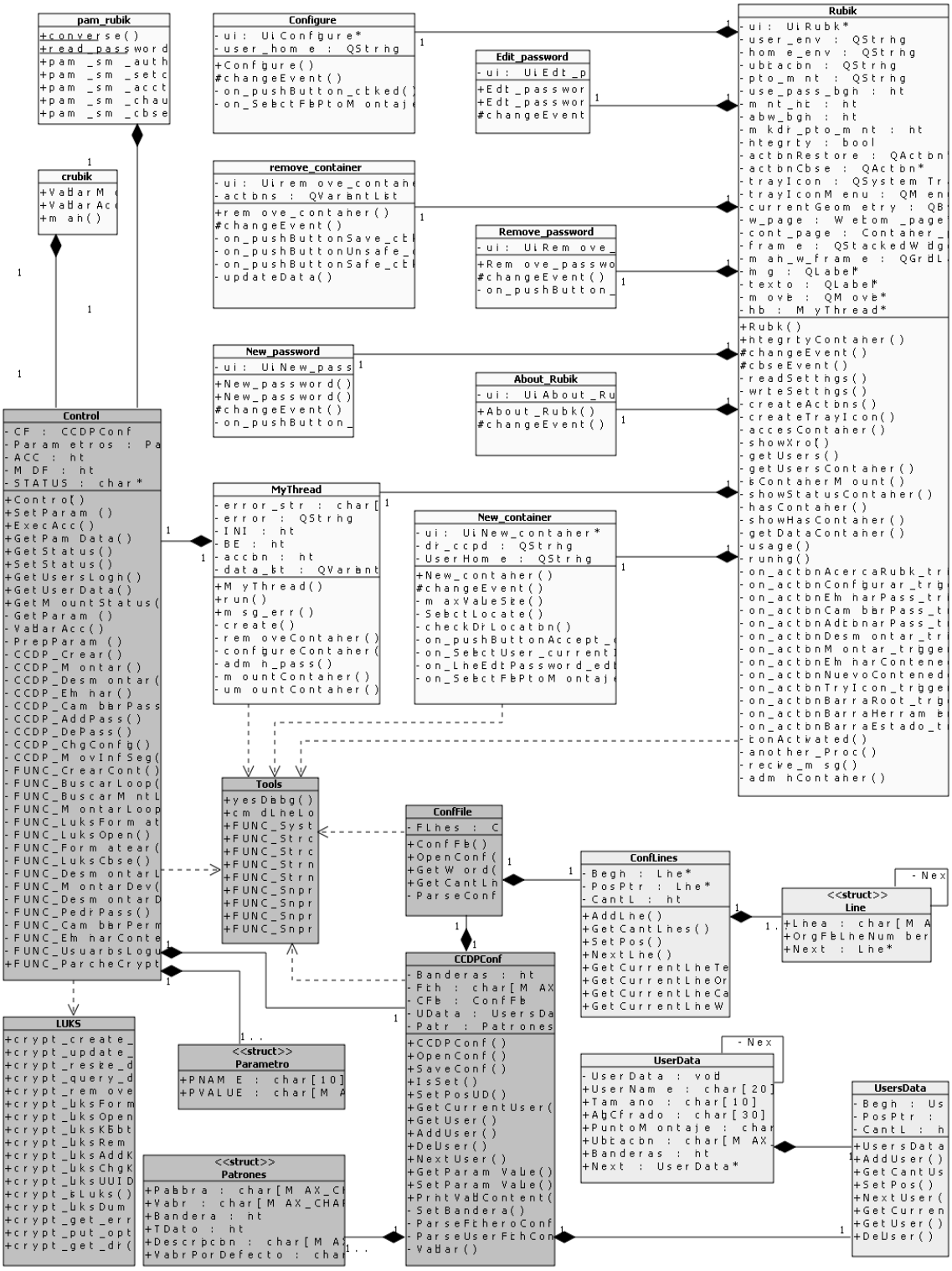

Figura 3.3: Diagrama de clases del diseño.

3.3. Realización de los casos de uso de diseño

Una realización de casos de uso de diseño describe cómo se realiza un caso de uso específico, y cómo se ejecuta, en términos de clases de diseño y sus objetos. Cuenta con una descripción textual del flujo de eventos, diagramas de clases, que muestran sus clases de diseño participantes, y diagramas de interacción (secuencia o colaboración) que muestran la realización de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño [7].

A continuación se muestra la realización de cada caso de uso de la aplicación a través de su respectivo diagrama de clases y diagrama de secuencia. Vale aclarar que los dos primeros diagramas de secuencia no corresponden a casos de uso, sino que representan una secuencia de mensajes común a todos. Esta se ha resumido en dos diagramas aparte figuras 3.4 y 3.5, que se referencian en el resto de las descripciones, en aras de hacerlas más comprensibles.

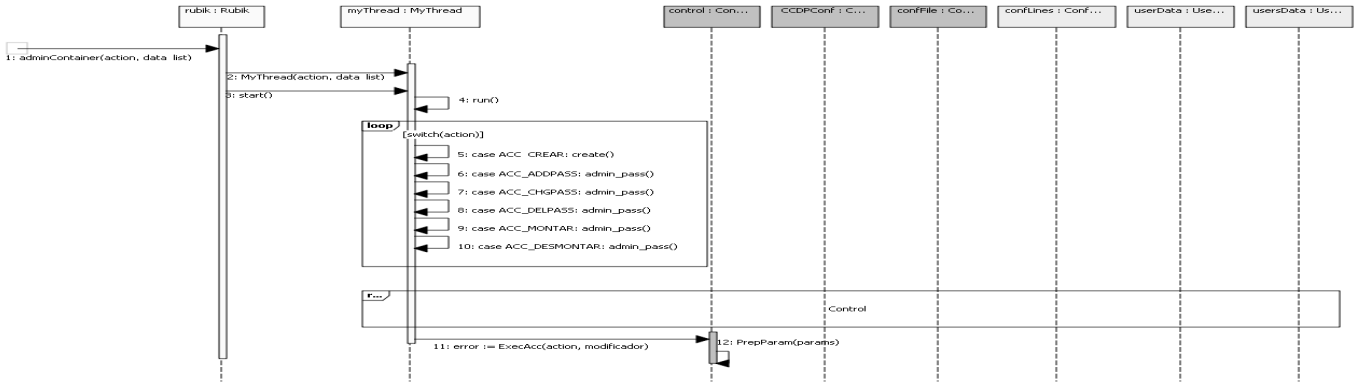

Figura 3.4: MyThread

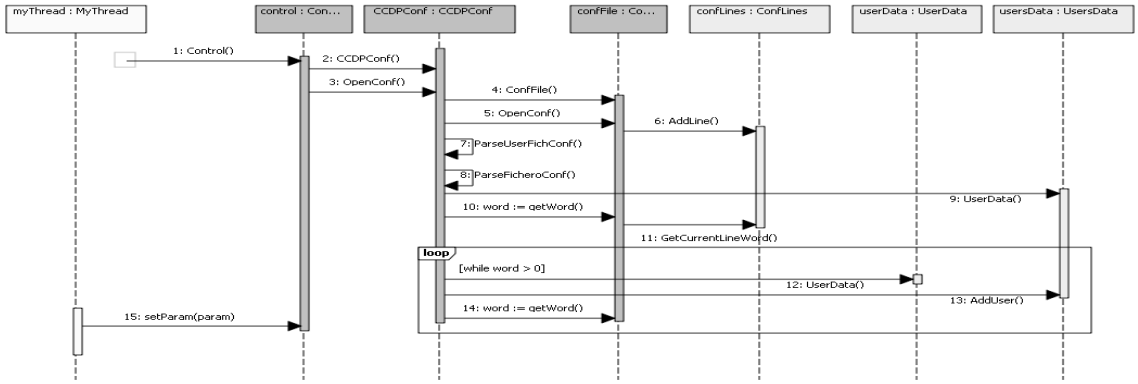

Figura 3.5: Control

Realización del caso de uso "Crear Contenedor".

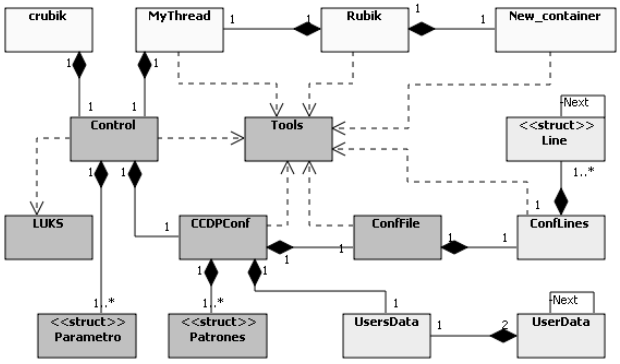

Figura 3.6: Diagrama de Clases del CU "Crear Contenedor"

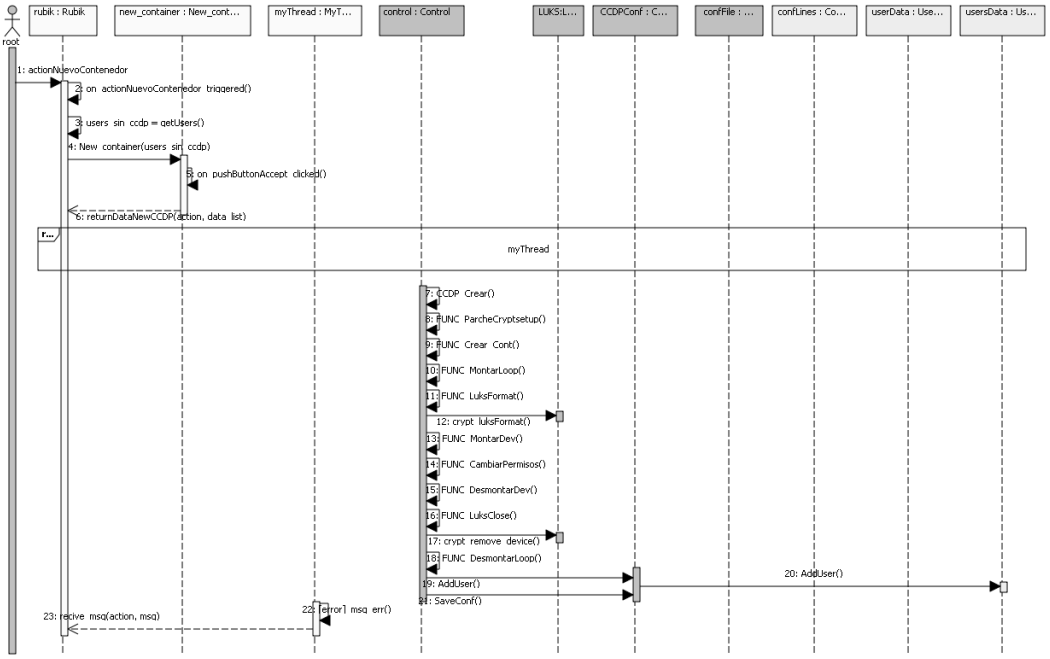

Figura 3.7: Diagrama de Secuencia del CU "Crear Contenedor"

Realización del caso de uso "Eliminar Contenedor".

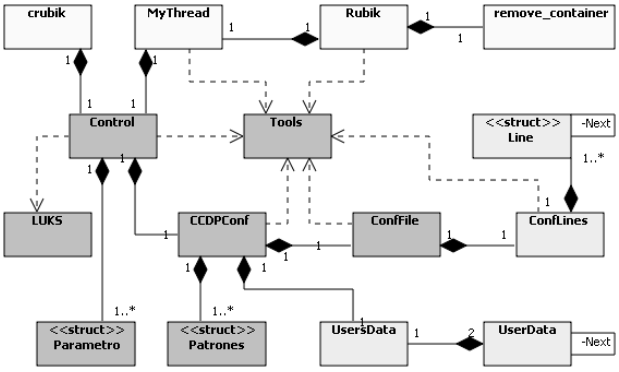

Figura 3.8: Diagrama de Clases del CU "Eliminar Contenedor"

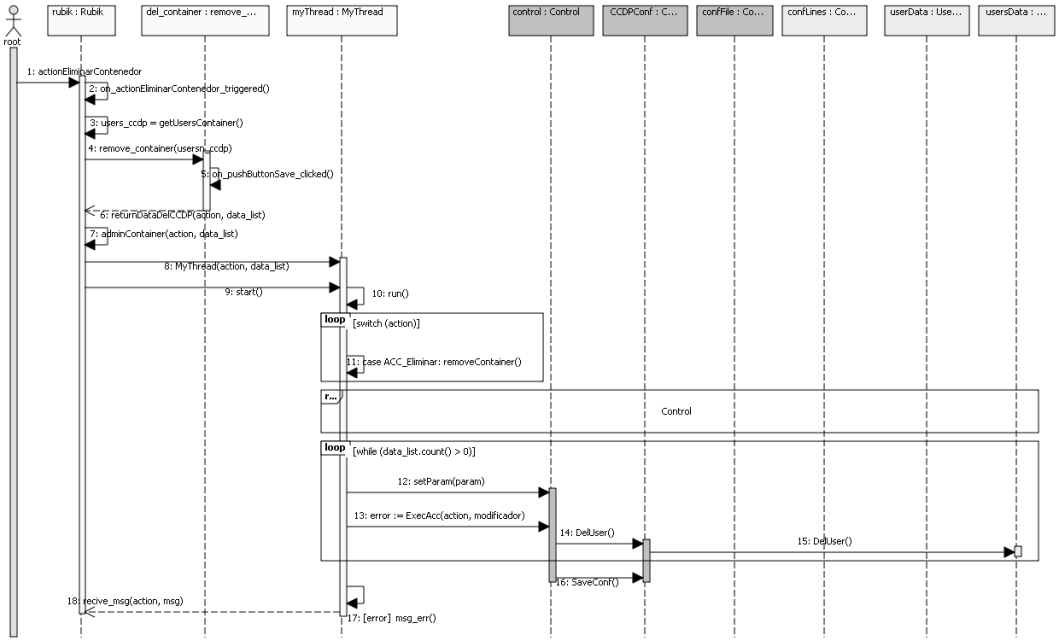

Figura 3.9: Diagrama de Secuencia del CU "Eliminar Contenedor"

Realización del caso de uso "Gestionar Contenedor".

Sección "Adicionar Contraseña".

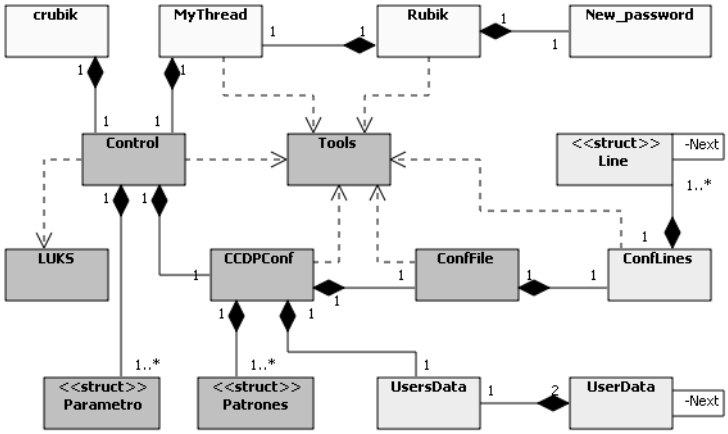

Figura 3.10: Diagrama de Clase del CU "Gestionar Contraseña" sección "Adicionar Contraseña"

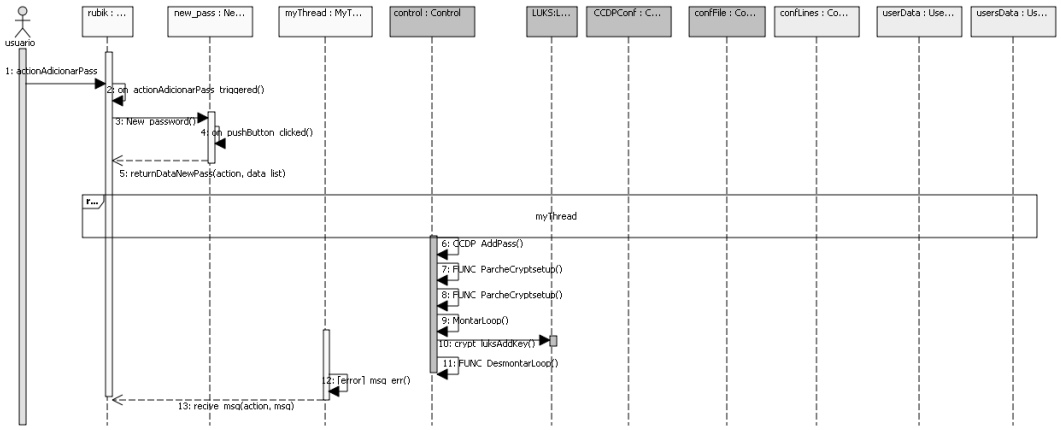

Figura 3.11: Diagrama de Secuencia del CU "Gestionar Contraseña" sección "Adicionar Contraseña"

Sección "Cambiar Contraseña".

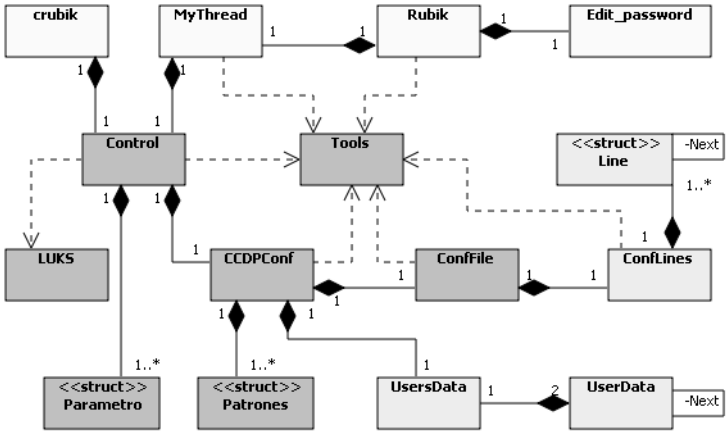

Figura 3.12: Diagrama de Clase del CU "Gestionar Contraseña" sección "Cambiar Contraseña"

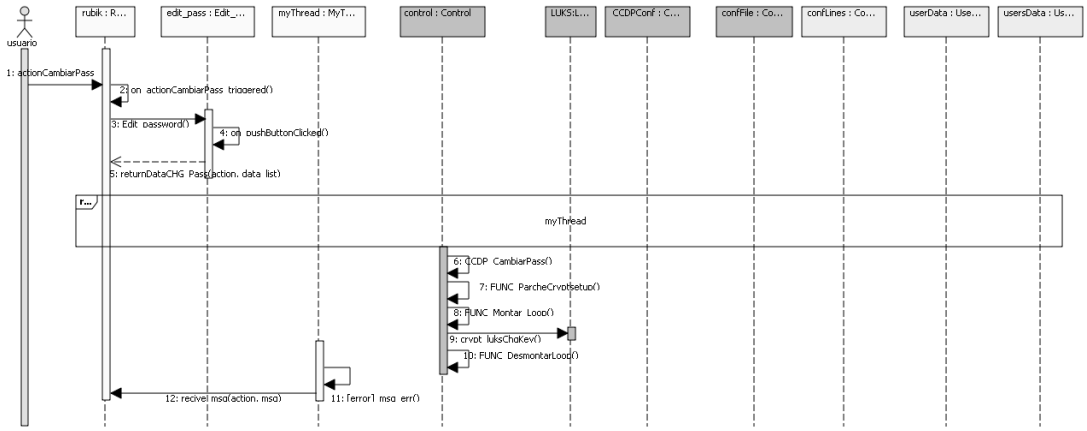

Figura 3.13: Diagrama de Secuencia del CU "Gestionar Contraseña" sección "Cambiar Contraseña"

Sección "Eliminar Contraseña".

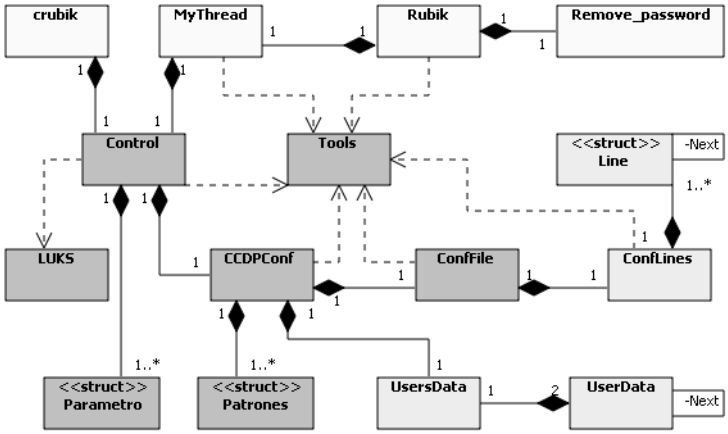

Figura 3.14: Diagrama de Clase del CU "Gestionar Contraseña" sección "Eliminar Contraseña"

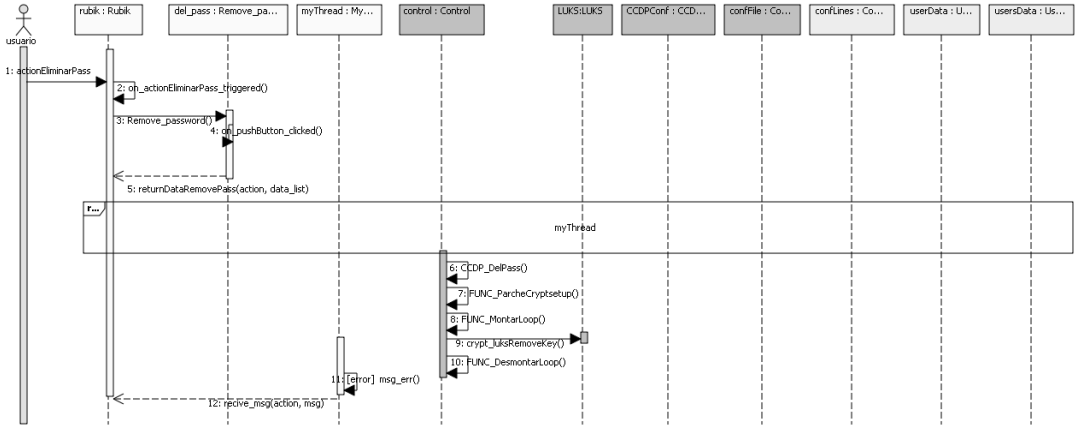

Figura 3.15: Diagrama de Secuencia del CU "Gestionar Contraseña" sección "Eliminar Contraseña"

Realización del caso de uso "Montar Contenedor".

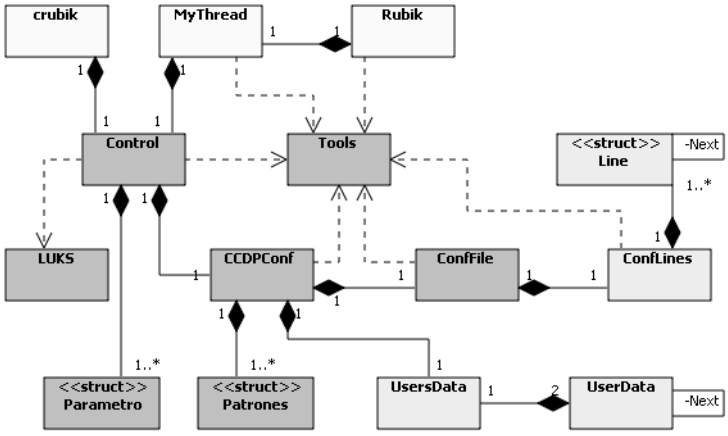

Figura 3.16: Diagrama de Clase del CU "Montar Contenedor"

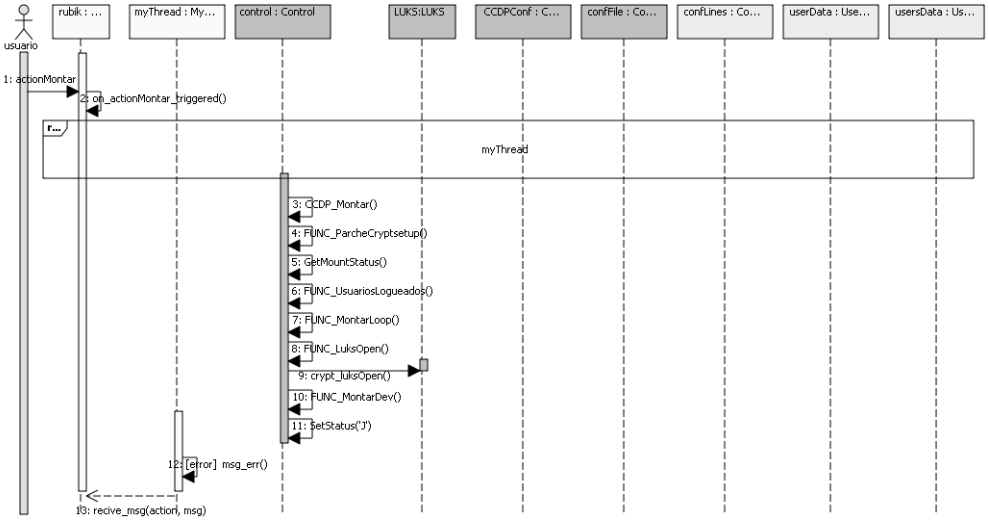

Figura 3.17: Diagrama de Secuencia del CU "Montar Contenedor"

Realización del caso de uso "Desmontar Contenedor".

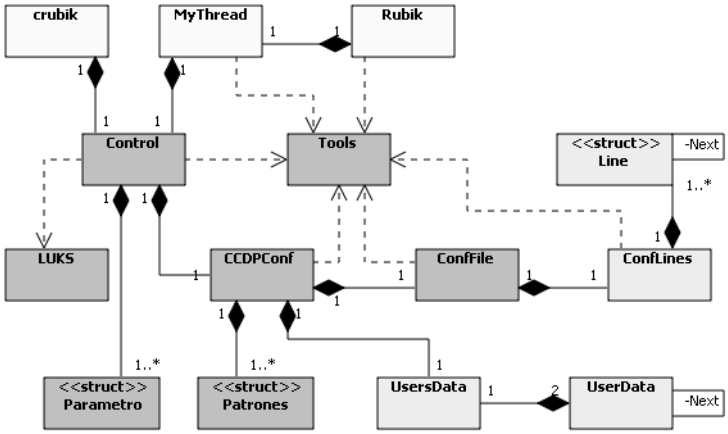

Figura 3.18: Diagrama de Clase del CU "Desmontar Contenedor"

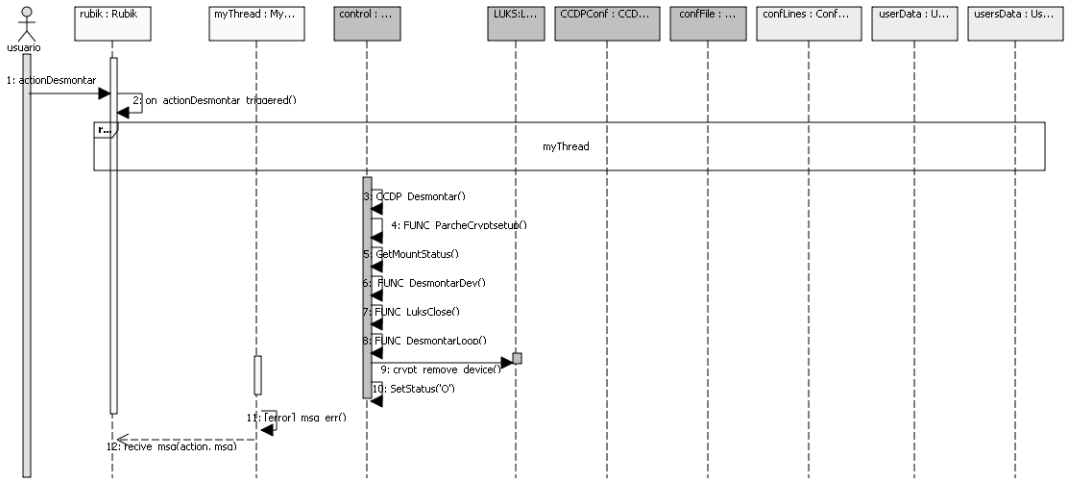

Figura 3.19: Diagrama de Secuencia del CU "Desmontar Contenedor"

3.4. Análisis y modelado de amenazas

El análisis y modelado de amenazas es una técnica de ingeniería de software, que tiene como objetivo ayudar a identificar y planificar correctamente la mejor manera de mitigar las amenazas de una aplicación informática.

Esta tarea de seguridad definida por CLASP tiene como propósito:

1. Evaluar posibles riesgos del sistema de manera oportuna y eficaz en función de los costos, mediante el análisis de los requisitos y el diseño.
2. Identificar amenazas de alto nivel del sistema que no han sido documentadas.
3. Identificar los requisitos de seguridad insuficientes o inadecuados.
4. Evaluar el impacto de la seguridad en los requisitos. [21]

Una vez definidos los requisitos de la aplicación, los casos de uso y abuso, el ambiente operacional y la arquitectura base, es posible comenzar a identificar un grupo de amenazas que deben ser cuidadosamente analizadas y clasificadas, en busca de las vulnerabilidades o puntos débiles a los que se ve expuesta la aplicación.

Determinar el nivel de riesgo

Para determinar el nivel de riesgo se procede a realizar una clasificación y puntuación de las amenazas, de modo que puedan ser priorizadas.

Utilizar DREAD (Damage potential, Reproducibility**,** Exploitability**,** Affected users**,** Discoverability), facilita el uso de un criterio común, para determinar el nivel de riesgo de cada una de las amenazas respondiendo a las siguientes preguntas:

- ***Damage potential*** **(Daño potencial)**: ¿Cuál es el daño que puede originar la vulnerabilidad si llega a ser explotada?
- ***Reproducibility*** **(Reproducibilidad)**: ¿Es fácil reproducir las condiciones que propician el ataque?
- ***Exploitability*** **(Explotabilidad)**: ¿Es sencillo llevar a cabo el ataque?
- ***Affected users*** **(Usuarios afectados)**: ¿Cuántos usuarios se verían afectados?
- ***Discoverability*** **(Descubrimiento)**: ¿Es fácil encontrar la vulnerabilidad?

A continuación se muestra la tabla de puntuación para priorizar las amenazas:

	Puntuación	Alto (3)	Medio (2)	Bajo (1)
D	*Damage potential* (Daño potencial)	El atacante podría utilizar la aplicación para ganar privilegios.	Divulgación de información sensible.	Divulgación de información trivial.
R	*Reproducibility* (Reproducibilidad)	El ataque es fácilmente reproducible.	El ataque se podría reproducir, pero solo en condiciones muy concretas.	Ataque difícil de reproducir, incluso conociendo la naturaleza del fallo.
E	*Exploitability* (Explotabilidad)	Un programador novel podría implementar el ataque en poco tiempo.	Un programador experimentado podría implementar el ataque.	Se requieren ciertas habilidades y conocimientos para explotar la vulnerabilidad.
A	*Affected users* (Usuarios afectados)	Todos los usuarios, configuración por defecto.	Algunos usuarios, no es la configuración por defecto.	Pocos usuarios afectados.
D	*Discoverability* (Descubrimiento)	Existe información pública que explica el ataque. Vulnerabilidad presente en una parte de la aplicación muy utilizada.	La vulnerabilidad afecta solo a una parte de la aplicación. No es muy probable que sea descubierta.	El fallo es trivial, no es muy probable que los usuarios puedan utilizarlo para causar un daño potencial.

Teniendo en cuenta la lista de amenazas identificadas, se le aplica la tabla de puntuación anterior para determinar el riesgo.

Id.	Amenaza	D	R	E	A	D	Total	Puntuación	Mitigada
A1	Elevación de privilegios mediante la aplicación.	3	1	1	1	1	7	Media	Sí
A2	Elevación de privilegios mediante modificaciones en los ficheros de configuración.	3	2	1	1	2	9	Media	Sí
A3	Denegación del servicio mediante modificaciones en los ficheros de configuración.	3	2	1	1	2	9	Media	Sí

continúa en la próxima página...

A4	Denegación de servicio de autenticación mediante la aplicación.	-	3	3	3	3	12	Alta	No
A5	Denegación de servicio de autenticación explotando la memoria compartida.	-	3	2	3	1	9	Media	Sí
A6	Ocultación de información.	-	3	3	3	3	12	Alta	Sí
A7	Divulgación de información.	2	2	2	3	3	12	Alta	Parcialmente
A8	Pérdida de la información.	-	3	-	3	3	9	Media	No
A9	Suplantación de identidad.	2	2	2	3	3	12	Alta	Parcialmente

Una vez conocido el riesgo que representan cada una de las amenazas identificadas y que a continuación se describen, se les dará solución teniendo en cuenta la puntuación del riesgo y el alcance de la investigación a las siguientes:

- A1 - Elevación de privilegios mediante la aplicación.
- A2 - Elevación de privilegios mediante modificaciones en los ficheros de configuración.
- A3 - Denegación del servicio mediante modificaciones en los ficheros de configuración.
- A5 - Denegación de servicio de autenticación explotando la memoria compartida.
- A6 - Ocultación de información.

Serán solucionadas de forma parcial:

- A7 - Divulgación de información.
- A9 - Suplantación de identidad.

Quedando las restantes documentadas para ser resueltas en futuras versiones de la aplicación:

- A4 - Denegación de servicio de autenticación mediante la aplicación.
- A8 - Pérdida de la información.

3.4.1. Amenazas mitigadas

Amenaza A1

Descripción	Elevación de privilegios mediante la aplicación.
Objetivo de la amenaza	Entradas de datos desde la aplicación para crear un desbordamiento de buffer.
Nivel de riesgo	Medio.
Técnicas de ataque	Introducir cadenas extremadamente grandes, al gestionar la contraseña.
Contramedidas	Validar la longitud de las cadenas introducidas por el usuario y todas las funciones conocidas que presentan esta amenaza (strcpy, snprintf...).

Amenaza A2

Descripción	Elevación de privilegios mediante modificaciones en ficheros de configuración.
Objetivo de la amenaza	Ficheros de configuración (/etc/rubik/rubik.conf y /home/usuario/.rubik/config) para crear un desbordamiento de buffer.
Nivel de riesgo	Medio.
Técnicas de ataque	Modificación de los archivos de configuración usados por la aplicación.
Contramedidas	Validación de todos los parámetros de los archivos de configuración y de su longitud al iniciar la aplicación, informando de fallos al usuario y brindándole la oportunidad de corregirlos guiándose por un archivo de configuración de muestra.

Amenaza A3

Descripción	Denegación del servicio mediante modificaciones en los ficheros de configuración.
Objetivo de la amenaza	Ficheros de configuración (/etc/rubik/rubik.conf y /home/usuario/.rubik/config).
Nivel de riesgo	Medio.
Técnicas de ataque	Modificación de los archivos de configuración usados por la aplicación.
Contramedidas	Validación de todos los parámetros de los archivos de configuración al iniciar la aplicación, informando de fallos al usuario y brindándole la oportunidad de corregirlos guiándose por un archivo de configuración de muestra.

Amenaza A5

Descripción	Denegación de servicio de autenticación explotando la memoria compartida.
Objetivo de la amenaza	Aprovechar el uso del módulo de PAM para el contenedor cifrado de datos personales.
Nivel de riesgo	Medio.
Técnicas de ataque	Montar el contenedor mediante la aplicación y desmontarlo por la consola.
Contramedidas	Para determinar si se encuentra montado el contenedor no solo se verifica la memoria compartida, sino también los dispositivos que se encuentran montados en ese momento.

Amenaza A6

Descripción	Ocultación de información.
Objetivo de la amenaza	Guardar información indebida.
Nivel de riesgo	Alto.
Técnicas de ataque	Añadir información indebida al contenedor.
Contramedidas	Generar contraseña de auditoría, para posibilitar revisiones.

3.4.2. Amenazas mitigadas parcialmente

Amenaza A7

Descripción	Divulgación de información.
Objetivo de la amenaza	Acceder al texto en claro que se encuentra en el contenedor.
Nivel de riesgo	Alto.
Técnicas de ataque	Volcado de la memoria swap, y lectura de /tmp.
Contramedidas	Cifrar la memoria swap y el directorio /tmp. Módulo para PAM que no permita autenticarse a otros usuarios mientras exista un contenedor abierto.

La amenaza A7 ha quedado mitigada parcialmente, debido a que mediante Rubik no se podrán llevar a cabo los cifrados de la memoria swap y el directorio /tmp, por lo que ha sido especificado en el ambiente operacional que ambos deben estar cifrados y cómo llevar a cabo esta operación.

Amenaza A9

Descripción	Suplantación de identidad.
Objetivo de la amenaza	Robar identidad al usuario (usuario y contraseña) para abrir el contenedor.
Nivel de riesgo	Alto.
Técnicas de ataque	Ingeniería social, obtención de credenciales por fuerza bruta.
Contramedidas	Brindar al usuario la posibilidad de usar tanto la misma contraseña para iniciar su sesión como una contraseña distinta.

La amenaza A9 ha quedado mitigada parcialmente, pues la aplicación brinda la posibilidad de tener contraseñas diferentes para iniciar la sesión y para acceder a su información confidencial. Quedan expuestos a esta amenaza los usuarios que decidan configurar su contenedor para usar la misma contraseña en ambos procesos, accediendo al sistema operativo y a su información confidencial. Si son obtenidas las credenciales con las que el usuario se autentica en el sistema operativo se podrá acceder también a su información confidencial guardada en el contenedor.

3.4.3. Amenazas sin mitigar

Amenaza A4

Descripción	Denegación de servicio de autenticación mediante la aplicación.
Objetivo de la amenaza	Aprovechar el uso del módulo de PAM para la aplicación.
Nivel de riesgo	Alto.
Técnicas de ataque	Dejar abierto el contenedor cifrado de datos personales.
Contramedidas	Desmontar automáticamente el contenedor transcurrido cierto tiempo de inactividad.

La amenaza A4 aunque tiene un nivel de riesgo alto, no ha sido posible de mitigar debido al grado de dificultad que presenta su solución. Una contramedida para mitigar esta amenaza podría ser desmontar automáticamente el contenedor transcurrido cierto tiempo de inactividad. Sin embargo es posible evadir esta contramedida debido a que un usuario podría montar su contenedor y abrir algún documento guardado en él, de forma tal que el sistema detectaría actividad y no desmontaría el contenedor. Incluso si se implementara un mecanismo para detectar cuándo el usuario no está realmente usando los documentos que tiene dentro de su contenedor, para forzar el desmontaje del mismo, se correría el riesgo de que la información que debe permanecer confidencial quedara desprotegida luego de forzarse el desmontaje del contenedor. Teniendo en cuenta las situaciones antes expuestas, se decidió continuar con la solución de otras amenazas pues esta resulta compleja de resolver y su manifestación no afecta la confidencialidad de la información que es el objetivo central de esta investigación.

Amenaza A8

Descripción	Perdida de la información.
Objetivo de la amenaza	Eliminar el contenedor utilizando las herramientas que brinda el sistema operativo como "rm" o "shred".
Nivel de riesgo	Medio.
Técnicas de ataque	Eliminar el contenedor usando las herramientas que brinda el sistema operativo, ejemplo: rm, shred, etc.
Contramedidas	–

La amenaza A8 no ha podido ser resuelta debido a que no es posible impedir que un contenedor sea eliminado por un usuario con privilegios para hacerlo, usando las herramientas para eliminar información que brinda el sistema operativo. Además esta investigación tiene como objetivo central la confidencialidad de la información y no la disponibilidad de la misma.

3.5. Ayuda

Rubik contará con una ayuda, la cual estará dividida en dos partes. La primera, simple y fácil de entender, estará a disposición del usuario en la propia aplicación, mostrándole cómo usar Rubik correctamente mediante información sobre sus componentes, y un manual de usuario. La segunda, dirigida a los usuarios expertos que deseen saber cómo funciona realmente la aplicación, será detallada y abundante, realizándose mediante el formato simple de documentación del código fuente a través de Doxygen, que puede usarse mezclándose con el código de C y C++ para crear documentación en diversos formatos como RTF, PDF, HTML y LATEX.

Conclusiones del capítulo

Concluido este capítulo se ha definido la arquitectura que sustentará el desarrollo de la aplicación, basada en los patrones de diseño más adecuados según las especificaciones de los requisitos. Se han diseñado los diagramas de clases y se han realizado los diagramas de interacción que permiten comprender de qué forma los objetos se comunicarán entre sí para dar cumplimiento a las funcionalidades definidas. La culminación de todos estos elementos, junto al análisis y modelado de amenazas, permiten dar paso a las actividades de implementación que se definirán y detallarán en el próximo capítulo.

Capítulo 4

Implementación y pruebas

Para llevar a cabo la implementación se comienza con el resultado del diseño y se implementa el sistema en término de componentes: ficheros de código fuente, *scripts*, ficheros de código binario, ejecutables y similares. La mayor parte de la arquitectura del sistema es capturada durante el diseño, siendo el objetivo general de la implementación desarrollar la arquitectura y el sistema como un todo. El modelo de implementación describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizados, y cómo dependen los componentes unos de otros [7].

Durante las pruebas se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas, como intermedias, así como las versiones finales del sistema a ser entregadas a terceros [7].

En este capítulo se estructura y muestra en detalle el modelo de implementación; se muestran los resultados de la revisión hecha al código de la aplicación en busca de vulnerabilidades y de las pruebas: unitarias, de integración y de penetración, que garantizan la calidad y seguridad de la aplicación.

4.1. Estructuración del modelo de implementación

Con el objetivo de estructurar el modelo de implementación en términos de subsistemas y sus relaciones, y para mostrar las dependencias entre los elementos de implementación, se construyen los diagramas de componentes. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo, especificando los subsistemas de implementación y sus dependencias a la hora de importar código y la organización de los mismos.

Se muestra a continuación la estructura de la aplicación en términos de subsistemas de implementación y sus relaciones. Seguidamente los componentes contenidos en el subsistema Rubik y sus dependencias, con un mayor nivel de detalle, debido a que en este subsistema se encuentra la mayor cantidad de componentes que sustentan el grueso de las funcionalidades de la aplicación.

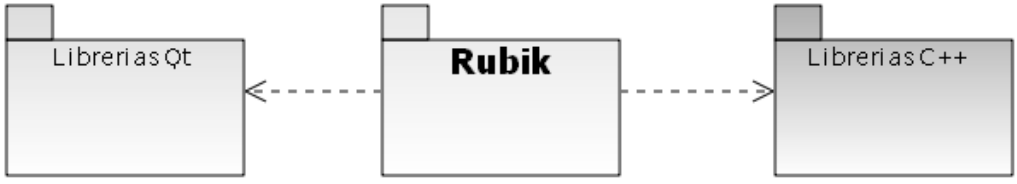

Figura 4.1: Estructuración del modelo de implementación de la aplicación

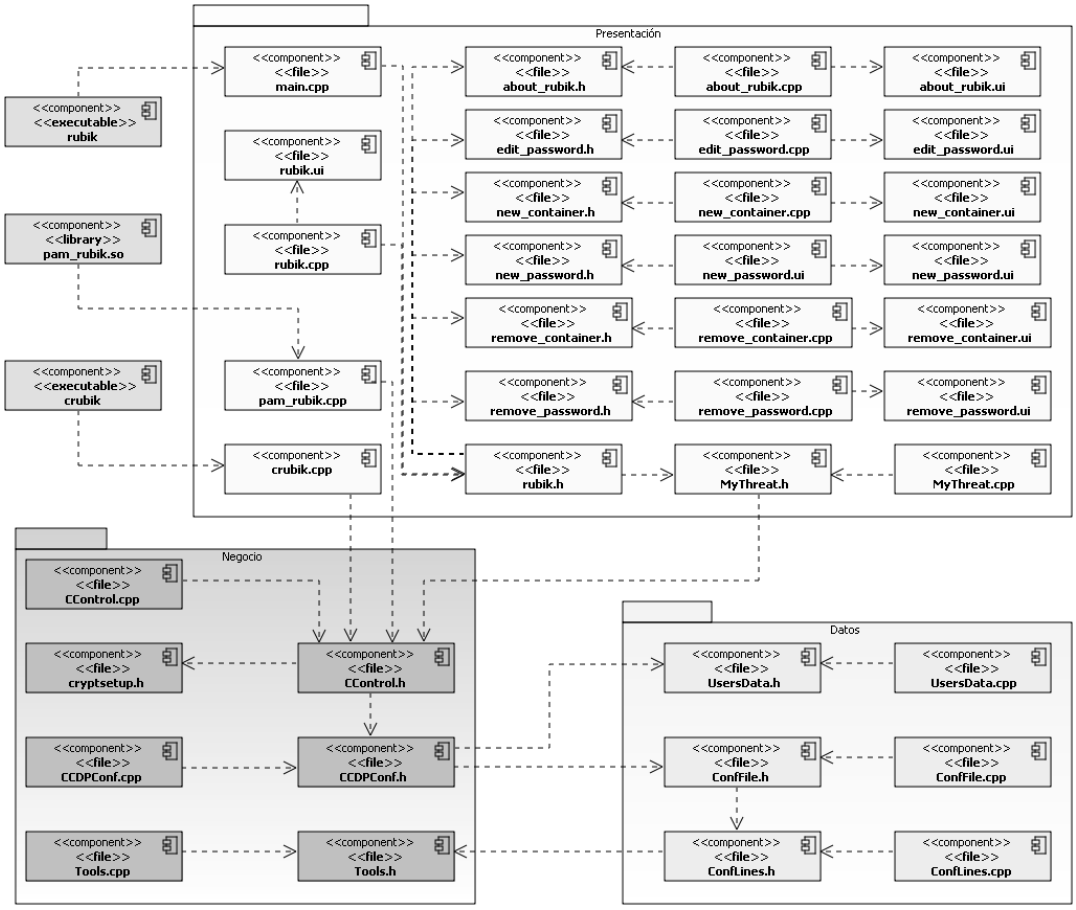

Figura 4.2: Componentes contenidos en el subsistema Rubik

4.2. Revisión de código

La revisión de código se realiza con el objetivo de encontrar y eliminar vulnerabilidades comunes en todo proceso de construcción como desbordamientos de *buffer* y condiciones de carrera. De esta forma se mejora la seguridad del software desarrollado, desde su misma implementación.

Para efectuar esta actividad pueden usarse herramientas como RATS, que automaticen el proceso de revisión de código. Estas herramientas realizan en cuestión de segundos lo que una persona haría en varios días o semanas, en dependencia del tamaño del proyecto a revisar. No obstante las mismas solo realizan una revisión superficial del código, siendo probable que no sean encontradas todas las vulnerabilidades o que se generen falsos positivos. Por esa razón, a pesar de disponer de una herramienta de apoyo de este tipo, se debe realizar además una revisión

manual del código fuente.

En la primera revisión realizada con RATS, se analizaron en total 5441 líneas de código de las cuales se detectaron 204 líneas con vulnerabilidades de gravedad alta, 6 con gravedad media y 38 con gravedad baja.

Teniendo en cuenta estos resultados se implementó el módulo *tools*, el cual contiene un conjunto de funciones creadas para aumentar el nivel de seguridad del software, de manera tal que se asegurara el ambiente en el cual se llama a funciones nativas del lenguaje, que puedan presentar algún tipo de riesgo para la aplicación. En la siguiente tabla se muestran tanto las funciones riesgosas, como las creadas para disminuir las vulnerabilidades.

Función nativa del sistema	Vulnerabilidad	Función creada para controlar la llamada a la función nativa del sistema
strcpy(des, org) des: destino org: origen	No garantiza que el tamaño de la información en el parámetro origen sea menor que el tamaño del parámetro destino, siendo posible un desbordamiento de *buffer*.	FUNC_Strcpy En esta función se llevan a cabo las validaciones necesarias para evitar un desbordamiento de *buffer* antes de usar strcpy.
strncpy(des, org, nc) nc: número de caracteres	No garantiza que el espacio libre del parámetro destino sea mayor que el número de caracteres a copiar del parámetro origen, siendo posible un desbordamiento de *buffer*, al concatenarlos en destino. No verifica que las cadenas terminen en NULL, siendo posible un desbordamiento de *buffer*.	FUNC_Strncpy En esta función se llevan a cabo las validaciones necesarias para evitar un desbordamiento de *buffer* antes de usar strncpy.
strcat(des, org)	No garantiza que el espacio libre del parámetro destino sea mayor que el espacio ocupado por la información del parámetro origen, siendo posible un desbordamiento de *buffer*, al concatenarlos en destino.	FUNC_Strcat En esta función se llevan a cabo las validaciones necesarias para evitar un desbordamiento de *buffer* antes de usar strcat.
strncat(des, org, nc)	No garantiza que el espacio libre del parámetro destino sea mayor que el número de caracteres a copiar del parámetro origen, siendo posible un desbordamiento de *buffer*, al concatenarlos en destino.	FUNC_Strncat En esta función se llevan a cabo las validaciones necesarias para evitar un desbordamiento de *buffer* antes de usar strncat.

continúa en la próxima página...

snprintf(des, cad, for) cad: cadena for: formato	No garantiza que el espacio del parámetro destino sea suficiente para almacenar la información del parámetro cadena, siendo posible un desbordamiento de *buffer*. No verifica que las cadenas terminen en NULL lo que puede provocar un desbordamiento de *buffer*.	FUNC_Snprintf En esta función se llevan a cabo las validaciones necesarias para evitar un desbordamiento de *buffer* antes de usar snprintf.

En la segunda revisión realizada con RATS, luego de la implementación del módulo *tools*, se analizaron en total 6140 líneas de código, de las cuales presentaron vulnerabilidades con una gravedad alta 138, con una gravedad media 6 y con gravedad baja 70.

Hay que señalar, que las vulnerabilidades que no han sido eliminadas han disminuido su peligrosidad, desde vulnerabilidad altas a bajas y que a todas se les ha controlado el ambiente en el que se ejecutan para evitar comportamientos indeseados por parte del software.

Además de usar RATS, se realizó una inspección manual del código, utilizando una lista de comprobación como guía (Anexo 7), corrigiéndose todas aquellas deficiencias y ambigüedades que existían en el mismo.

4.3. Pruebas

A continuación se abordan los diferentes tipos de pruebas realizados a la aplicación, para garantizar su correcto funcionamiento y el máximo nivel de seguridad.

4.3.1. Pruebas unitarias

Las pruebas unitarias, enfocadas en los elementos testeables más pequeños, tales como componentes del modelo de implementación, son una forma de probar el correcto funcionamiento de un módulo de código, para verificar que los flujos de control y de datos estén cubiertos, y funcionen como se espera. Esto sirve para asegurar que cada uno de estos elementos funcione correctamente por separado. Luego, con las pruebas de integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema como un todo.

Se diseñaron y ejecutaron pruebas unitarias a los módulos *tools* y pam_rubik. Al primero por el papel que desempeña en la seguridad de Rubik y al segundo por la importancia que su correcto funcionamiento tiene para el sistema.

El diseño de los casos de pruebas unitarias y su resultado se encuentran recogidos en el Anexo 8.

4.3.2. Pruebas de integración

Las pruebas de integración se utilizan para verificar que los componentes interaccionen entre sí de la forma apropiada después de haber sido integrados en una construcción [7].

En Rubik la capa de presentación se encuentra separada de las otras capas, por tal razón se realizaron pruebas de integración para cada uno de los CU, encontrando algunas incompatibilidades entre la función *waitpid()* de C++ y el objeto QProcess de Qt que entraban en conflicto al competir por un mismo recurso. Finalmente se decidió no usar el objeto QProcess y reemplazarlo por las opciones nativas de C++ disponibles.

El diseño de los casos de pruebas de integración y su resultado se encuentran recogidos en el Anexo 9.

4.3.3. Pruebas de penetración

Las pruebas de penetración constituyen un método para evaluar la seguridad de una aplicación simulando el ataque de un usuario malicioso.

Fueron diseñadas y ejecutadas pruebas que simulan ataques desde los ficheros de configuración o mediante entradas por teclado, a través de la inserción de cadenas largas, caracteres especiales, cadenas codificadas, fragmentos de *scripts* y secuencias de escape.

Además se llevaron a cabo un conjunto de ataques, modificando el entorno operacional, con el objetivo de verificar la estabilidad y seguridad de la aplicación ante dichos cambios.

El diseño de los casos de pruebas de penetración y su resultado se encuentran recogidos en el Anexo 10.

Conclusiones del capítulo

La implementación y la realización de las pruebas sobre la aplicación están estrechamente relacionadas, mayormente durante la fase de construcción del software. En el presente capítulo ha quedado plasmada la estructura de la aplicación en términos de subsistemas de implementación y sus componentes, además de las diferentes pruebas realizadas para garantizar su correcto funcionamiento.

Como resultado del proceso de pruebas que se realizó se corrigieron errores surgidos durante la implementación hasta arrojar resultados satisfactorios. Mediante estas pruebas fueron eliminadas, mitigadas o al menos documentadas las vulnerabilidades que pudiesen atentar contra la seguridad de la aplicación, garantizándose la capacidad de Rubik para pasar a la etapa de despliegue.

Conclusiones

Al culminar el desarrollo de la presente investigación se arribó a las siguientes conclusiones:

- Combinar mecanismos de cifrado y de autenticación, en lugar de emplearlos por separado, es una mejor solución a la hora de desarrollar aplicaciones cuyo objetivo principal sea garantizar la seguridad de la información.
- Usar CLASP como metodología de desarrollo de software seguro para guiar la construcción de la aplicación propuesta, permite elevar la calidad del producto obtenido debido a que propone un conjunto de pasos enfocados a incrementar la seguridad.
- Aplicar principios y patrones de diseño seguro permite construir una arquitectura más robusta y menos vulnerable.
- Detectar y documentar las amenazas que atentan contra la seguridad de la aplicación construida constituye un medio eficaz para hacer frente a los posibles ataques contra dicha aplicación y para garantizar su futuro mantenimiento y actualización.
- Realizar un conjunto variado de pruebas, como parte de las actividades propuestas por CLASP, minimiza las vulnerabilidades de la aplicación, debido a que dichas pruebas son concebidas desde las etapas de diseño e implementación.

Se ratifica la idea a defender debido a que la aplicación desarrollada permite elevar la confidencialidad de la información de los usuarios de la distribución GNU/Linux Nova y constituye un aporte en el campo de las herramientas para la protección de la información, al dotar a dicha distribución de una vía para salvaguardar la información de sus usuarios finales de manera amigable, segura y sencilla.

Recomendaciones

Para mejorar la seguridad de la aplicación y que esta pueda garantizar de una manera más eficaz la confidencialidad de la información de los usuarios finales, se recomienda:

- Solucionar las amenazas que no han sido resueltas en esta investigación y continuar el estudio de posibles nuevas amenazas que puedan comprometer el correcto funcionamiento de Rubik.
- Implementar la comunicación entre Rubik y cryptsetup sin necesidad de utilizar un parche para este último.
- Estudiar e implementar los cambios necesarios para el correcto funcionamiento de la aplicación sobre otras distribuciones de GNU/Linux, comenzando por Ubuntu debido al reciente cambio de plataforma que ha decidido hacer el proyecto Nova, desde Gentoo hacia Ubuntu.

Referencias bibliográficas

[1] Conjunto de autores. *Introducción a LyX*, agosto 2009.

[2] Matt Bishop. *Introduction to Computer Security*. Prentice Hall, Octubre 2004. 784 pp.

[3] David Gascón Cabrejas. Ficheros *www.Linux-magazine.es*, 25:10, 2006.

[4] IBM Corporation. *Rational Unified Process*. IBM, 7.2.0 edition, 2007.

[5] Dirección central de la seguridad de sistemas de información de Francia. *Protection Profile - On-the-fly Mass Storage Encryption Application*. Common Criteria, PP-CDISK-CCv3.1, version 1.4 edition, Agosto 2008.

[6] Clemens Fruhwirth. New methods in hard disk encryption. Documento que investiga el estado del arte en los métodos de cifrados de discos duros y es complementado con la presentación de una variante de TKS1 llamada LUKS., Julio 2005.

[7] James Rumbaugh Grady Booch, Ivar Jacobson. *El Proceso Unificado de Desarrollo de Software*. Addison Wesley, 2000. 458 pp.

[8] Andreas L. Opdahl Guttorm Sindre. Templates for misuse case description. In *Proc. of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality*. Citeseer, 2001.

[9] Michael Austin Halcrow. ecryptfs: An enterprise-class cryptographic filesystem for linux. Technical report, International Business Machines, Inc., Mayo 2005.

[10] Antonio Villalón Huerta. Seguridad en unix y redes, Julio 2002.

[11] Pablo Javkin. Ordenanza para la migración a software libre municipalidad rosario., Marzo 2004. URL `http://lugro.org.ar/biblioteca/textos-gral/ordenanza.html`.

[12] Ulrich Jüttner. Sd4l - scramdisk for linux, Diciembre 2009. URL `sd4l.sourceforge.net`.

[13] Ulrich Jüttner. *ScramDisk for Linux, User Guide and Technical Documentation*, Abril 2009.

[14] Craig Larman. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Prentice Hall, 1999. 507 pp.

[15] R. Lehtinen, D. Russell, and GT Gangemi. *Computer security basics*. O'Reilly Media, Inc., 2da edition, 2006. 296 pp.

[16] Manuel José Lucena López. *Criptografía y Seguridad en Computadores*. Universidad de Jaen, segunda edition, Septiembre 1999. 167 pp.

[17] Clemen Fruhwirth Markus Schuster. Mensajes secretos. *www.Linux-magazine.es*, 14:7, 2005.

[18] Nancy R. Mead. How to compare the security quality requirements engineering (square) method with other methods. Report DAAD19-02-1-0389, Carnegie Mellon University and Software Engineering Institute, 2007.

[19] Visual Paradimg. Visual paradigm for uml - uml tool for software application development, Enero 2010. URL `http://www.visual-paradigm.com/product/vpuml/`.

[20] Christopher Saout. dm-crypt: a device-mapper crypto target. URL `http://www.saout.de/misc/dm-crypt/`.

[21] John Viega. Comprehensive lightweight application security process. Technical report, Secure Software, Inc., 2006.

[22] Yoandys Pérez Villazón. Metodología para la migración a software libre de las universidades del ministerio de educación superior (mes). Tesis, Mayo 2008.

Glosario de términos

/dev Directorio que contiene archivos de dispositivos especiales para los dispositivos de hardware de nuestra computadora, son esenciales para el correcto funcionamiento del sistema.

/dev/mapper Cuando se realiza el mapeo (lectura) de un dispositivo cifrado o protegido para que otros programas puedan utilizar los datos como si estuviesen en un dispositivo de bloques normal el mapeo aparece bajo el directorio /dev/mapper/nombre, donde nombre es la etiqueta asociada a dicho mapeo.

AFsplitter Método para borrar información de un disco duro de manera que no se pueda recuperar, su nombre proviene de anti-forensic splitter que en español sería división anti forense.

anti-rootkit Programa de detección de rootkit.

anti-spyware Programa de detección spyware.

BSD KNF Es un estilo de código cuyas características principales son: abre llaves al finalizar una sentencia de control y utiliza dos tabulaciones de 4 para identación adicional en caso de que una línea continúe en varias líneas y de 8 para identar bloques de código.

CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) Aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

CLASP Comprehensive, Lightweight Application Security Process (Proceso Global y Ligero para Aplicaciones de Seguridad).

CU Caso de uso

confidencial Que se hace o se dice en confianza o con seguridad recíproca entre dos o más personas.

CryptoAPI Colección de algoritmos de cifrado utilizados por diversas partes del núcleo que se ocupan de la criptografía, por ejemplo DM-Crypt. Introducidos a partir del núcleo 2.4.12 se ha actualizado hasta incluir los cifrados de bloques y hash más populares.

desbordamiento de buffer La condición del desbordamiento de buffer se relaciona con la copia de datos a la pila, a los segmentos de datos o al montón, sin tener en cuenta su tamaño, lo que puede provocar una escritura que exceda el intervalo definido y situaciones peligrosas que son el

resultado de posibles sobrescrituras de estructuras que controlan el flujo del programa u otros datos confidenciales.

device-mapper Mapeo de dispositivos. Es una nueva infraestructura en el núcleo Linux 2.6, la cual provee una manera genérica de crear capas virtuales de dispositivos de bloques que pueden realizar diferentes tareas sobre dispositivos de bloques reales, como agotamiento, concatenación, espejado, etc. El device-mapper es utilizado por las herramientas LVM2 y EVMS 2.x.

GNU Sistema operativo cuyo nombre es un acrónimo recursivo que significa "GNU No es UNIX".

GPGME GnuPG Made Easy. Librería diseñada para hacer que el uso de PGP más sencillo para las aplicaciones que lo usen.

GPL Licencia Pública General creada a finales de 1980 para servir de sostén legal a las aplicaciones surgidas bajo el término copyleft.

GUI Interfaz gráfica de usuario, es el artefacto tecnológico de un sistema inactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

GnuPG También conocido como GPG, permite cifrar y firmar los datos y la comunicación, cuenta con un versátil sistema de gestión de claves.

hash Llave generada a través de un algoritmo, representa con un grado de certeza elevado a un documento, registro, archivo, etc.; resumir o identificar un dato a través de la probabilidad.

IDE De sus siglas en inglés Integrated Development Environment, que en español sería Entorno de Desarrollo Integrado

IV Vector de inicialización es el bloque de bits requerido para permitir el cifrado pues éste es el valor usado para modificar el primer texto en claro.

journaling También conocido como registro por diario, es un mecanismo mediante el cual un sistema informático puede implementar transacciones. Consiste en llevar un registro diario almacenando la información necesaria para recuperar la información afectada por la transacción en caso de que esta falle.

Kerberos Protocolo de autenticación de redes mediante el cual es posible demostrar la identidad entre computadoras de manera segura en una red insegura.

Linux Es el núcleo (kernel) del sistema operativo libre GNU

LiveCD Sistema operativo almacenado en un CD o un DVD que puede ejecutarse desde éste sin necesidad de instalarlo en el disco duro de una computadora.

Loop Devices Es un pseudo-dispositivo que permite acceder a un fichero como a si fuese un dispositivo de bloques.

OpenSSL Conjunto de librerías y herramientas de administración relacionadas con la criptografía, que suministran funciones criptográficas a otras aplicaciones como OpenSSH y navegadores web (para acceso seguro a sitios HTTPS).

PAM Pluggable Authentication Modules (Módulos de Autenticación Conectados).

PBKDF2 Password Based Key Derive Function Version 2, es un componente PKCS 5 (Public Key Cryptography Standard 5). PKCS 5 fue especificado en RFC 2898. Entre otras cosas, PBKDF2 hace uso del estiramiento y la dispersión para impedir los ataques basados en diccionarios [6].

PGP Pretty Good Privacy. Software Libre criptográfico de llave pública desarrollado por Paul Zimmerman, se ha convertido en un estándar para la cifrado de correo electrónico y datos.

QT Biblioteca multiplataforma para el desarrollo de interfaces gráficas.

RAM Memoria de acceso aleatoria de la computadora desde donde el procesador recibe las instrucciones y guarda los resultados.

RUP Rational Unified Process (Proceso Unificado de Rational).

Ripemd 160 RACE Integrity Primitives Evaluation Message Digest. Función de hash criptográfica de 128, 160, 256 ó 320 bits desarrollada por Hans Dobbertin, Antoon Bosselaers y Bart Preneel.

root Nombre del usuario administrador, que posee todos los privilegios en el sistema operativo GNU/Linux.

rootkit Herramienta o grupo de ellas que tiene como finalidad esconderse a sí misma y esconder otros programas, procesos, archivos, directorios, claves de registro, y puertos que permiten al intruso mantener el acceso a un sistema para remotamente comandar acciones o extraer información sensible.

SDL Security Development Lifecycle (Ciclo de vida de Desarrollo Seguro).

scripts Programa simple casi siempre en texto plano y estructurado, pueden realizar diversas tareas, desde combinar componentes hasta interactuar con el sistema operativo y el usuario.

secreto Conocimiento que exclusivamente alguien posee y cuidadosamente se tiene reservado y oculto.

spyware Spyware o programa espía, es un programa dentro de la categoría de software malicioso o malintencionado, que se instala furtivamente en una computadora para recopilar información sobre las actividades realizadas en ella.

swap Memoria de intercambio, comúnmente una partición del disco usada para almacenar las imágenes de los procesos que no pueden mantenerse en memoria física.

TPM Trusted Platform Module (TPM) es el nombre de una especificación publicada que detalla un procesador de cifrado en el que se pueden almacenar claves.

TrueCrypt Aplicación privativa para cifrar y ocultar datos utilizando diferentes algoritmos de cifrado.

UCI Universidad de las Ciencias Informáticas.

UCID Unidad de Compatibilización e Integración para la Defensa.

UML Unified Modeling Language (Lenguaje Unificado de Modelado).

VFS Conmutador de sistema de archivos virtual. Permite que las aplicaciones cliente tengan acceso a diferentes tipos de ficheros de una manera uniforme mediante una interfaz entre el núcleo (kernel) y un sistema de archivos en concreto.