I

HERRAMIENTA SOFTWARE PARA LA PROTECCION CONTRA INSTALACION Y USO NO LICENCIADO DE APLICACIONES SOFTWARE (TRIONIX 1.0)

JUAN GABRIEL QUINTERO PEÑA
WILLIAM VILLAMIZAR VERA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS FISICO-MECANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
2004

HERRAMIENTA SOFTWARE PARA LA PROTECCION CONTRA INSTALACION Y USO NO LICENCIADO DE APLICACIONES SOFTWARE (TRIONIX 1.0)

JUAN GABRIEL QUINTERO PEÑA
WILLIAM VILLAMIZAR VERA

Proyecto de grado para optar el titulo de Ingenieros de Sistemas

Director

LUIS IGNACIO GONZALEZ RAMÍREZ
Magíster en Informática, UIS

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS FISICO-MECANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
2004

CONTENIDO

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

TITULO: **HERRAMIENTA SOFTWARE PARA LA PROTECCIÓN CONTRA INSTALACIÓN Y USO NO LICENCIADO DE APLICACIONES SOFTWARE** (TRIONIX 1.0)[1]

AUTORES: VILLAMIZAR VERA William**
QUINTERO PEÑA Juan Gabriel**

PALABRAS CLAVES: Protección de ejecutables, Proceso unificado de desarrollo, Criptografía, Procesos, Hilos, Manejo de memoria.

La protección de archivos ejecutables para aplicaciones Win32 es una labor que debe ser realizada partiendo de conceptos de manejo de memoria, procesos, manejo de archivos y criptografía. El proyecto busca cubrir las necesidades de protección dentro de la escuela de Ingeniería de sistemas e informática así como de la comunidad universitaria.

Con este proyecto se busca implementar algoritmos de clave pública y de clave secreta que se requieran para soportar el mecanismo de protección, así como un soporte administrativo para los proyectos que se creen. Para llevar a cabo este proyecto se debe conocer aparte de los temas ya mencionados, bibliotecas de vinculo dinámico, algoritmo RSA y DES. La metodología usada fue el proceso Unificado de desarrollo versión 98, soportada con UML 1.0, las fase que se desarrollaron fueron Inicio, elaboración y construcción con sus respectivos flujos de trabajo.

La solución implementada consiste primero en capturar información relacionada con la aplicación a proteger y las condiciones de protección, luego se crea un nuevo archivo ejecutable el cual constara de el archivo original encriptado, mas una serie de procesos y programas encargados de la protección en la maquina del cliente, la protección puede ser renovada por parte del cliente comunicándose con el administrador de la herramienta quien llevara un historial del comportamiento de la aplicación protegida. La mejor protección es la que combine los mecanismos usados por el programador con la criptografía y el manejo del sistema operativo

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Físico-mecánicas.
Ingeniería de Sistemas e Informática.
Director. Mg Luis Ignacio González Ramírez

TITLE: **SOFTWARE TOOLS FOR THE PROTECTION AGAINST THE INSTALLATION AND USE OF NO LICENSED SOFTWARE APPLICATIONS (TRIONIX 1.0)*[2]**

AUTHORS: VILLAMIZAR VERA William**[3]
QUINTERO PEÑA Juan Gabriel**

KEY WORDS: Protection of executable unified process of development
Cryptography, Process, Thread, Memory management.

DESCRIPTION: The executable file protection for the applications Win32 it is a job, which could be realized starting from concepts of memory management, process, files management and Cryptography. This project looks for supply the protection necessities of System Engineering and some software as well as the UIS university community.

With the realization of this project, we find to implement algorithms of public key and secret key that are necessary to support the protection mechanism, as well as an administrative support for the projects that will be create. To develop the project it would be know apart from the topics we have already mentioned dynamics linking libraries, RSA and DES algorithms. The methodology used was the develop unificate process of 98 version, support with UML 1.0, the steps that were developed were beginning, elaboration and construction with their respective flux of work.

The implemented solution consists first, in capture the information related with the application to protect and the protection conditions; then, a new executable file is created which will be have that the original encrypted file. In addition, a process series and programs encharged of the protection in the client's machine; the protection could be renovate for the client, when this is communicated with the administrator, with the tool administrator who writes a record of the protected application conduct. The best protection is that which combines mechanisms used for the programmer with the cryptography and the operative system management.

---

* Graduation Work.

** Physics and Mechanics Sciences Faculty
Systems Engineer School.
Director. Mg Luis Ignacio Gonzalez Ramirez

INTRODUCCIÓN

En la actualidad, el principal problema que se presenta a la hora de desarrollar y distribuir una aplicación software es la inseguridad que le genera a el desarrollador software entregar su aplicación a una persona que puede empezar a distribuir sin ningún permiso el producto que tanto esfuerzo le ha costado construir, sin recibir beneficio alguno.

Este es un temor general al cual se enfrenta cualquier persona o empresa dedicada a desarrollar software; buscando superar este temor, surge Trionix[4] que brinda a el distribuidor de aplicaciones la posibilidad de limitar el uso de su producto en cuanto a acceso y tiempo de uso, con lo cual se puede llegar a implantar una nueva estrategia de venta como seria el alquiler de aplicaciones software restringida; lo cual le permitiría al usuario de la aplicación contar con un producto actualizado y garantizar el cumplimiento del ciclo del software.

Para llevar a cabo este proyecto se utiliza como metodología de desarrollo el proceso unificado, buscando que la documentación presentada de una explicación de las actividades realizadas durante el desarrollo del proyecto.

El presente documento se clasifica por partes y capitulo. Cada una de las parte agrupa un conjunto de capitulo relacionados. La **parte I** refleja los aspectos técnicos básicos mas importantes que comprenden la teoría fundamental usada

[4] Herramienta software contra instalación y uso no licenciado de aplicaciones.

para el desarrollo de este trabajo; en el capitulo 1 plantea los objetivos, antecedentes, justificación, impacto que ofrece este proyecto. El capitulo 2 presenta la información teórica. En el capitulo 3 se presenta una breve reseña sobre el proceso unificado de desarrollo, el cual es la metodología a utilizar.

La parte II describe cada una de las fases del proceso unificado de desarrollo aplicado a este proyecto. Comienza en el capitulo 4 con la fase de inicio; la cual busca justificar la ejecución del proyecto mediante el desarrollo del análisis del negocio. Continua con el capitulo 5 con la fase de elaboración, en la cual se especifica el mayor porcentaje de casos de uso del producto. Por ultimo se presenta la fase de construcción en el capitulo 6, que muestra las iteraciones necesarias para obtener el sistema propuesto.

La **parte III** presenta otra faceta del proyecto fuera del análisis, diseño e implementación. En esta parte se tratan aspectos relevantes de la herramienta Trionix; en el capitulo 7, en el cual se muestra de manera detallada la forma en que se implemento el esquema de seguridad. En el capitulo 8 se exponen las pruebas de auditoria realizada al sistema.

Para finalizar, se presentan las conclusiones y recomendaciones realizadas con base al trabajo realizado, así como los anexos; en los cuales cuentan con información que amplia los conceptos tratados a lo largo del documento.

PARTE I

FUNDAMENTOS

En esta Parte I se reflejan los aspectos técnicos básicos más importantes que comprenden la teoría fundamental usada para el desarrollo de este trabajo; No se pretende mostrar en su totalidad el volumen de información usada, solo se hará énfasis en los temas requeridos para comprender el trabajo realizado.

Los objetivos específicos de esta parte son:

- Presentar los objetivos planteados en el proyecto.
- Justificar la elaboración del proyecto.
- Describir el producto a realizar en cuanto a funcionalidad.
- Formular el marco teórico.
- Definir el marco metodológico, en donde se muestra la metodología utilizada para el desarrollo del trabajo realizado.

CAPITULO I

ÁMBITO DEL PROYECTO

Introducción

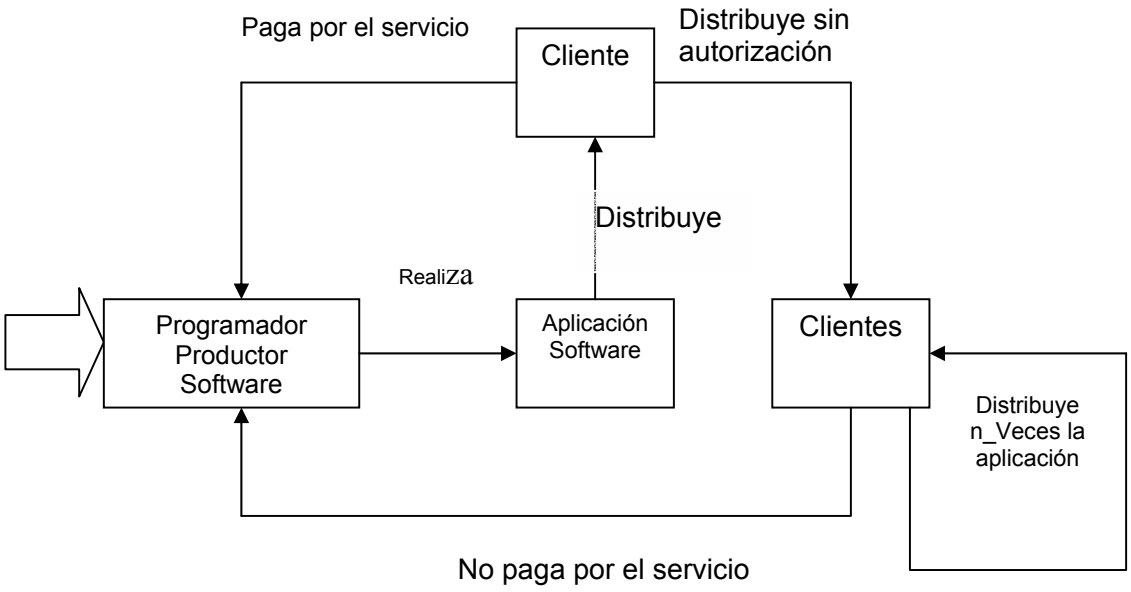


Figura 1.1 *Descripción del problema*

En la actualidad existen dos formas de proteger aplicaciones software:

- Mediante Hardware. Se utiliza un dispositivo llamado KeyLock que se conecta al puerto paralelo del computador para que al momento de ejecutar la aplicación

software se compruebe la existencia de la llave (KeyLock). Este es un buen mecanismo pero es muy costoso y poco practico ya que al tener muchas aplicaciones protegidas se deben tener muchas llaves lo que ocasiona problemas de espacio y de recurso del PC.

- Mediante Software. Existen en el mercado algunas herramientas que pueden ser utilizadas para la protección de Software a costos bastante elevados. Estas herramientas garantizan el uso de una aplicación software por acceso o tiempo de uso, pero no permiten llevar un control acerca del estado de la aplicación protegida.

En el mundo de la informática existen dos áreas de desarrollo: el hardware y el software. El productor de Hardware siempre tendrá reconocimiento y remuneración por su trabajo, mientras que el de software por lo general no cuenta con ninguna de las dos anteriores. A esta conclusión se llego luego de observar la evolución de la tecnología, donde se aprecia claramente como al desarrollador de software se le desconoce su trabajo por causa de la piratería de aplicaciones.

Debido a que la Escuela de Ingeniería de Sistemas no cuenta con medios que le garanticen el control de licencias de las aplicaciones desarrolladas al interior de esta, se propone crear una herramienta software que permita mantener el control sobre las aplicaciones desarrolladas.

1.5 *DESCRIPCIÓN DEL PROYECTO*

1.5.1 *General*

Implementar una herramienta software para regular la instalación y uso no licenciado de aplicaciones, utilizando técnicas de criptografía.

1.5.2 Específicos

- Implementar los algoritmos de clave publica y clave privada (secreta) requeridos para los procesos de encriptamiento de la herramienta software.

- Diseñar e implementar una base de datos que soporte el almacenamiento y organización de la información relacionada con el estado de la aplicación software protegida.

- Crear certificados[5] que permitan establecer un control de la aplicación en cuanto a accesos y tiempos de usos.

- Diseñar e implementar un administrador de perfiles que regule el alcance de las opciones brindadas al usuario por la herramienta software.

[5] Se toma certificado como tipos de protecciones.

1.5.3 Antecedentes

1.5.3.1 Mundial

La piratería mundial de software ha aumentado en los últimos años debido a la falta de leyes y una mayor disponibilidad de programas piratas en Internet. La Alianza de Programas Comerciales (Business Software Alliance) informó en su octavo estudio anual que perdió cerca de 11.000 millones de dólares en ventas por la piratería de software en el 2001.

El dato más revelador, fue que el 40 por ciento de todo el software nuevo instalado por las empresas el año pasado se consiguió en el mercado negro, frente al 37 por ciento en el 2000.

1.5.3.2 Nacional

Actualmente el índice de piratería de software en Colombia es de 53 %, es decir, del total de los programas instalados en el país más de la mitad son ilegales. Esta situación origina pérdidas para la industria del software que llegan a 40 millones de dólares por año, mientras que el estado ha perdido cerca de 217 millones de dólares por concepto de impuestos dejados de percibir. A consecuencia de la piratería, 18 mil colombianos se han quedado sin empleo cada año.

Una de las consecuencias más delicadas que traen estos altos índices de piratería para el país, es el hecho de formar parte de la “Lista especial 301”, la cual fue aprobada en 1988 por el congreso de la Estados Unidos y cuyo objetivo es identificar a todos aquellos países que no cuentan con una protección eficaz ni adecuada de los derechos de propiedad intelectual y derechos de autor.

1.5.4 Justificación

Luego de observar estadísticas publicadas por la BSA[6] que refleja el índice de piratería a nivel mundial y de analizar situaciones presentadas en la EISI[7] con proyectos que generan aplicaciones software, surge la idea del proyecto TRIONIX ya que la oferta existente en el mercado no satisface las necesidades de protección de las aplicaciones desarrolladas en el interior de la Universidad Industrial de Santander ni en el área local. En la UIS se llevan a cabo diferentes tipos de proyecto que dan como resultado aplicaciones software que son utilizadas por dependencias de la universidad o empresas solicitantes del proyecto, las cuales se benefician de la aplicación.

Un caso particular sucede en la Escuela de Ingeniería de Sistemas a la cual recurren diferentes dependencias a la hora de necesitar desarrollar aplicaciones software. Esto da como resultado un proyecto de grado que es utilizado por la dependencia o empresa y aprovechando la coyuntura de la graduación de los creadores del proyecto se pierde completamente el contacto entre la Escuela y el cliente de la aplicación, quedando esta a la deriva sin posibilidad de realizar mantenimiento alguno interrumpiendo el ciclo de vida del software.

Buscando solucionar lo expuesto anteriormente nace TRIONIX que ofrece la protección de aplicaciones mediante mecanismos software utilizando técnicas criptográficas, además cuenta con una base de datos que permite almacenar y administrar la información de las aplicaciones protegidas.

---

[6] Bussines Aliance Software
[7] Escuela de Ingeniería de Sistemas e Informática UIS

1.6 IMPACTO

El trabajo se fundamenta en el mejoramiento de la imagen y el reconocimiento de la Escuela de Ingeniería de Sistemas como productora de Software, buscando consolidarse día a día como una institución capaz de afrontar compromisos de desarrollo frente a su comunidad y a sí misma. También busca el bienestar de los desarrolladores de software al intentar proteger el fruto de su trabajo sin llegar a violar en ningún momento los derechos de los usuarios.

En la Escuela de Ingeniería de Sistemas e Informática de la UIS no existe precedente alguno sobre trabajos encaminados hacia una solución software al problema planteado anteriormente, lo cual crea la posibilidad de iniciar el estudio de esta área que puede significar un gran reconocimiento en la escuela en un futuro no lejano.

1.7 VIABILIDAD

El trabajo no cuenta con un escenario académico disponible al interior de la EISI-UIS, debido a que no existen profesores con conocimiento específicos acerca del tema. Sin embargo en un escenario internacional se cuenta con información y trabajos relacionados con el tema, lo cual nos brinda suficientes bases para la generación de propuestas informáticas acordes con el problema en cuestión.

Disponemos de recursos técnicos para el soporte del desarrollo de la solución software dada como son: los lenguajes de programación, manejadores de base de datos, computadores, internet, etc.

Se garantiza el uso de la herramienta por parte de los productores de aplicaciones software debido a que el mercado cada día es más amplio al igual que las necesidades de protección.

1.8 DESCRIPCIÓN DEL PRODUCTO

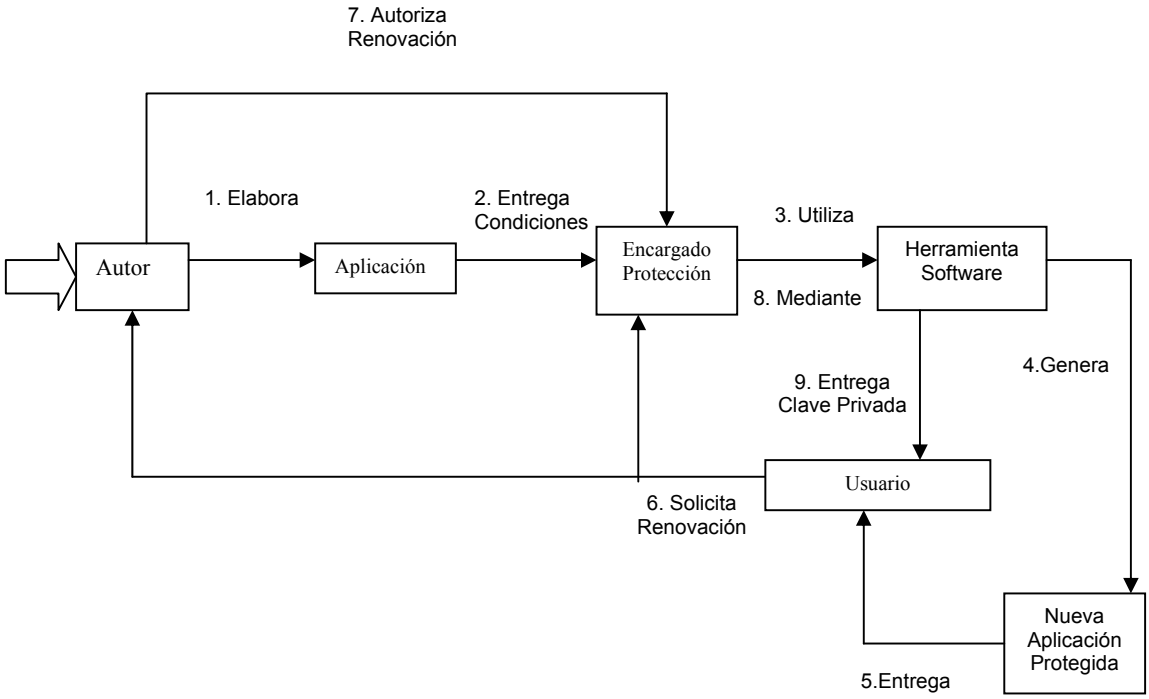


Figura 1.2 *Descripción del Producto***.**

La herramienta software para la protección de aplicaciones brindará al usuario la posibilidad de proteger su aplicación de la siguiente manera:

- El autor luego de obtener el ejecutable de su aplicación procederá a invocar dicho ejecutable desde una interfaz de la herramienta que permite seleccionar el tipo de protección que se le dará a su aplicación (tiempos o usos) definido por el usuario, esto dará origen al proyecto que contiene la aplicación a proteger y las opciones dadas.

- Se procederá al registro del proyecto en la base de datos, en el registro se tendrá información relacionada con: autor de la aplicación, usuario de la aplicación, tipo de protección dada, clave publica asignada y demás información relacionada con la aplicación protegida para poder llevar un pequeño inventario. Una vez registrado el proyecto se obtendrá la aplicación protegida

- Al momento de la instalación de la aplicación por parte del usuario se generará mediante un algoritmo criptográfico la clave pública particular a la aplicación. El usuario debe comunicarse con el responsable de la protección de la aplicación para que este le suministre la clave privada que combinada con la clave pública dada por el usuario cumpla con la condición establecida por el algoritmo de clave pública. Si el usuario escribe la clave publica y privada suministrada, se producirá una validación interna que dará como resultado la habilitación de la aplicación para un uso por un tiempo o por acceso determinado por el encargado de la protección de la aplicación.

- Transcurrido el tiempo de uso de la aplicación esta se bloqueará impidiendo el uso por parte del usuario. La aplicación bloqueada mostrará un mensaje donde se proporciona una nueva clave publica y se solicita una clave privada que al compararlas cumplan la condición establecida por el algoritmo de clave publica; Nuevamente el usuario debe comunicarse con el encargado de la protección de la aplicación, suministrarle la clave publica mostrada por la aplicación bloqueada y esperar a que se le suministre la clave privada que cumpla con la condición establecida. Luego de llevar a cabo este procedimiento se obtendrá una renovación por un tiempo o numero de acceso. La clave publica suministrada por el usuario al encargado de la protección de la aplicación y los datos relacionados con la aplicación protegida serán nuevamente almacenados en la base de datos para poder llevar un control de las operaciones realizadas sobre el proyecto existente.

De esta manera se garantiza que la aplicación será protegida según las condiciones establecidas en la negociación entre el autor y el usuario de la aplicación. También

se obliga al usuario a no perder el contacto con el autor y a no tomar atribuciones que no le pertenecen como lo es la distribución y uso ilegal de la aplicación.

CAPITULO 2

MARCO TEÓRICO

En este capitulo se presentan los conceptos básicos más importantes para la comprensión del trabajo realizado. Se trata de dar una breve descripción de cada uno de los temas relevantes, pero en caso de requerir profundizar sobre alguno de ellos, se debe recurrir a la referencia indicada en la bibliografía.

2.1 ARCHIVO PORTABLE EJECUTABLE (PE)

Es necesario conocer como funciona internamente un archivo ejecutable para poder modificar su contenido, lo cual es de suma importancia en este proyecto, ya que para proteger una aplicación se debe introducir cierto código que permita realizar validaciones que garanticen el cumplimiento de la protección otorgada.

Para conocer un poco mejor el tema se dará una breve introducción sobre la estructura del archivo, origen y evolución.

2.1.1 Antecedentes del formato portable ejecutable (PE)

Existe un formato de archivo ejecutable, Portable Ejecutable (PE) que fue introducido inicialmente por Windows NT en su versión 3.1; este formato PE se inspiró en el de los sistemas operativos UNIX, denominado COFF (Common Object File Format) o Formato de Archivo de Objeto Común, muy eficiente, pero se necesito ir extendiendo para satisfacer todas las necesidades de un sistema operativo moderno. Para mantener su compatibilidad con las versiones del MS-DOS y los sistemas operativos Windows, el formato PE mantuvo la antigua cabecera MZ del MS-DOS.

El nombre de Portable Ejecutable se debe a que este nuevo formato de archivo, es completamente portátil y compatible con cualquier versión de Windows 95/98/NT/Me/2000/XP. Igualmente es usado en micro procesadores diferentes a los Intel x86, tales como MIPS, Alpha y Power PC, entre otros; ciertamente existen algunas diferencias en las instrucciones binarias de la cpu, pero lo más importante es que las herramientas cargadas en los diferentes sistemas operativos no tienen que volverse a montar cada vez que se trabaja con una nueva CPU. Los archivos DLL[8] de 32 bits y los devices drivers (manipuladores de dispositivos o periféricos) también emplean el formato PE.

Conocer acerca del formato PE ayuda a comprender muchos conceptos de los sistemas operativos, como por ejemplo, la forma en que opera el cargador del sistema operativo para ejecutar las funciones dinámicas de los archivos ejecutables DLL cargados en memoria.

También es importante conocer las estructuras de los datos de apoyo usadas por Windows para determinar donde se localiza el código y los datos en memoria. El archivo de PE tiene una colección de campos que define el contenido del resto del archivo. Contiene la información como: tamaños del código y área de los datos, el tamaño de la pila inicial y otras partes vitales de información.

[8] Para mas información ver Apartado 2.2 Bibliotecas de Vínculos Dinámicos

Otro aspecto importante sobre los archivos PE son las secciones; estas son bloques de memoria inmediata sin imposiciones de tamaño; algunas secciones contienen código o datos que se declaran en los programas y se usan directamente mientras se crean otras secciones de los datos. Las secciones también son llamadas objetos.

En términos generales cualquier modificación desarrollada en el formato PE tiene mayor capacidad de ocasionar cambios a los sistemas o archivos en donde tiene influencia, ya que este formato esta altamente difundido por su compatibilidad en diferentes escenarios; es el caso de los sistemas virales que pueden afectar archivos o áreas del sistema preferiblemente usando archivos ejecutables como .EXE y .COM que por ser los mas usados durante cualquier sesión de trabajo se convierten en lugares preferidos para que los programadores puedan activar este tipo de programas. Estos sistemas afectan los registros de los sistemas operativos y generan llaves en el mismo, pueden borrar archivos de extensiones selectivas, formatear el disco duro o dar cualquier instrucción caprichosa de sus desarrolladores, de la misma forma se puede usar para divulgar mensajes que pueden ser de beneficiosos; todo depende de quien desarrolle este tipo de aplicación.

2.2 BIBLIOTECA DE VÍNCULOS DINÁMICOS

Las bibliotecas de vínculos dinámicos, más conocidas como archivos DLL (Dynamic Link Library), son la solución que presenta el sistema operativo Windows ante las desventajas del vínculo estático. Con el vinculo estático el enlace se realiza cuando se genera el archivo .EXE, es aquí cuando el ensamblador vincula el código objeto del programa compilado con el código objeto de la biblioteca temporal y los escribe como un único archivo ejecutable. Si se quiere saber cual es el verdadero código del programa y de la biblioteca por separado seria muy difícil.

La desventaja más grande que presenta el vínculo estático es que no se pueden alterar las funciones llamadas de la biblioteca temporal. Por lo cual si se detecta un

error o si se pretende mejorar la eficiencia de la función habrá que generar un nuevo archivo EXE y distribuirlo entre los usuarios. Además si la biblioteca es utilizada por varios programas que se encuentran en ejecución, las funciones utilizadas de la biblioteca serán cargadas varias veces en memoria; estas situaciones presentadas van en contra de los principios planteados por un sistema multitarea, el cual exige un mínimo de desperdicio de espacio de memoria RAM.

El vinculo dinámico funciona de la siguiente manera: el código del programa no se ensambla con el código del programa de la biblioteca temporal.

La biblioteca temporal aparece como un archivo separado, el cual es vinculado durante la ejecución por el programa ejecutable, de forma que se puede llamar a las funciones contenidas. Esta acción permite actualizar las funciones sin necesidad de compilar de nuevo el programa completo, debido a que sustituyendo el archivo DLL y reiniciado el programa entrarían en ejecución las nuevas funciones.

De esta manera trabaja Windows por medio de las API's; cuenta con mas de mil funciones en diferentes archivos DLL, entre ellas:

- KERNEL32.DLL. Contiene funciones del sistema para el administrador de memoria, sirve para trabajar con procesos y subprocesos, etc.

- USER32. DLL. Rutinas para la interacción con el usuario, como lo son menús, cuadros de diálogos, etc.

- GDI32. DLL. Funciones necesarias para dibujar e imprimir.

2.2.1 Fundamentos de las funciones DLL

En Windows se reconoce un solo tipo de funciones DLL y dos maneras que un programa acceda a ellas: durante el proceso de carga con el shell o durante la

ejecución llamando directamente a las correspondientes funciones del API. El primer caso se conoce como vinculo dinámico en tiempo de carga (Load Time Dynamic Linking), el segundo es un vínculo dinámico en tiempo de ejecución (Run Time Dynamic Linking).

A continuación se hará una breve comparación entre las dos maneras que puede utilizar un programa para acceder una función DLL.

Vinculo dinámico en tiempo de carga.	Vinculo dinámico en tiempo de ejecución
En el código fuente del programa se indica cual función DLL se cargara. Las funciones son tratadas como externas de otros módulos	No se conoce el nombre de la biblioteca y de las funciones que se van a utilizar, sino hasta que el programa entra en ejecución.

Tabla 2.1 *Comparación de tipos de vínculos de una DLL*

2.2.2 Localización de una función DLL

Al momento de utilizar una función almacenada en una librería de vinculo dinámico se introduce el nombre del archivo DLL. Si el nombre del archivo no lleva extensión, se le asignara una .DLL. Si le falta la indicación explicita de un paquete de búsqueda se buscara el archivo en diferentes directorios, teniendo en cuenta el siguiente orden:

1. El directorio del que se cargo el actual proceso.
2. El directorio seleccionado de la unidad vigente.
3. El directorio del sistema de Windows. En Windows XP este directorio es conocido como SYSTEM32, en versiones anteriores a este, el directorio es SYSTEM.
4. El directorio de Windows (Windows).
5. Los directorios incluidos dentro de la variable de entorno PATH.

De no encontrarse el archivo DLL aparecerá un mensaje de error ya que se retorna el llamado de la función NULL. Si se localiza la función, lo primero que se hará es verificar si ya se cargo al proceso algún archivo DLL del correspondiente directorio con el mismo nombre. Si la respuesta es afirmativa, no es necesario cargar de nuevo el archivo DLL indicado; lo único que se haría seria incrementar el contador interno de usuarios. De lo contrario se cargara el archivo DLL y se cargara en la zona de direcciones del proceso.

2.2.*3 Construcción de un archivo DLL*

Los archivos DLL tienen su origen en módulos en los que se empaqueta el código de programas de las funciones DLL a exportar. Las funciones DLL deben hacerse accesibles a otros programas, por lo cual se debe tener cuidado en el tipo de parámetros que recibe y envía la función, ya que si no son compatibles la información nunca llegará. La diferencia de un programa normal y una DLL es que esta última carece de impulso propio, es decir que tiene que ser invocada por otra aplicación alguna de las funciones que contiene para entrar en actividad.

Un archivo DLL no necesita exportar todas las funciones con que cuenta. Solo necesita exportar las que realicen funciones generales requeridas. Para esto se deben caracterizar dichas funciones, lo cual se puede hacer mediante dos métodos:

- ☑ ***Tradicional***

  Se compila un archivo denominado de definición, identificado por su extensión .DEF el cual no es más que un simple archivo ASCII.

- ☑ ***Moderno***

  No sigue la norma ANSI.

El archivo DEF se debe compilar utilizando una sintaxis simple. Este archivo es vinculado a los módulos compilados (archivos OBJ) con la información necesaria para la información de las funciones. Su estructura es de la siguiente manera:

```
LIBRARY     Nombre del archivo DLL
DESCRIPTION Para que se utilizara el archivo DLL
EXPORTS
Función_1_a exportar                @1
Función_2 _a exportar               @2
```

Por cada función a exportar se necesita asignar un identificador para que la función pueda ser accedida desde las aplicaciones, esto se realiza mediante el carácter @.

2.3 CRIPTOGRAFÍA

La criptografía (del griego kryptos, "escondido", y graphein, "escribir"), el arte de enmascarar los mensajes con signos convencionales, que sólo cobran sentido a la luz de una clave secreta.

2.3.1 Introducción histórica

Desde la antigüedad los mensajes cifrados han jugado un papel importante. Son una gran arma de defensa para los militares, diplomáticos y espías; como también para los datos que viajan por Internet.

Sus orígenes se encuentran en las tablas cuneiformes, y los papiros demuestran que los primeros egipcios, hebreos, babilonios y asirios conocieron y aplicaron sus inescrutables técnicas, que alcanzan hoy su máxima expresión gracias al desarrollo de los sistemas informáticos y de las redes mundiales de comunicación.

2.3.2 Conceptos básicos

En los procesos de almacenamiento y transmisión de la información normalmente aparece el problema de la seguridad. En el almacenamiento, el peligro lo representa el robo del soporte del mensaje o simplemente el acceso no autorizado a esa información, mientras que en las transmisiones lo es la intervención del canal.

La protección de la información se lleva a cabo variando su forma. Se llama cifrado (o transformación criptográfica) a una transformación del texto original (llamado también texto inicial o texto claro) que lo convierte en el llamado texto cifrado o criptograma. Análogamente, se llama descifrado a la transformación que permite recuperar el texto original a partir del texto cifrado.

Cada una de estas transformaciones está determinada por un parámetro llamado clave. El conjunto de sus posibles valores se denomina espacio de claves k. La familia de transformaciones criptográficas se llama sistema criptográfico $T = \left\{ \frac{T_k}{k} \in k \right\}$

Para cada transformación criptográfica T_k se definen imágenes de cada una de las palabras de n letras. Es decir, el sistema criptográfico se puede describir como $T = \left\{ T_k{}^n : 1 \le n < \infty \right\}$, siendo $T_k{}^n(x) = y$, donde y es la palabra cifrada que corresponde a la palabra original x.

Para evitar ambigüedades, se hacen las siguientes suposiciones:

- $T^k{}_n$ es biyectiva.
- Se usa el mismo alfabeto para ambos textos, original y cifrado.
- Se define el cifrado de todas las posibles palabras.
- Cada n-palabras se cifra en n-palabras, obteniendo así que el cifrado no cambia la longitud del texto original.

En general no es necesario imponer simultáneamente todas estas condiciones, aunque en algunos casos, como el procesamiento de información digital, si es recomendable porque:

- Se trabaja únicamente con alfabeto binario.
- Debe existir el cifrado de todas las palabras posibles.
- Si el cifrado cambiara la longitud del texto, sería necesario usar un nuevo formato para el texto cifrado.

2.3.3 *Métodos criptográficos*

Existen diferentes métodos criptográficos como son : método clásico, clave publica y clave secreta. Cada uno de estos métodos ofrecen ventajas y desventajas, por lo que se debe analizar el contexto para poder aplicar el algoritmo mas indicado para resolver el problema planteado.

2.3.3.1 Métodos clásicos

Los espartanos utilizaron, en el 400 a.c., la Scitala que puede considerarse el primer sistema de criptografía por transposición, es decir, que se caracteriza por enmascarar el significado real de un texto alterando el orden de los signos que lo conforman. Los militares de la ciudad griega escribían sus mensajes sobre una tela que envolvían en una vara, el mensaje sólo podía leerse cuando se enrollaba sobre un bastón del mismo grosor, que poseía el destinatario lícito.

El método de la scitala era extremadamente sencillo, como también lo era el que instituyó Julio César, basado en la sustitución de cada letra por la que ocupa tres puestos más allá en el alfabeto.

La criptografía resurgió en la Europa de la Edad Media, impulsada por las intrigas del papado.

En 1466, León Battista Alberti, músico, pintor, escritor y arquitecto, concibió el sistema polialfabético que emplea varios abecedarios, saltando de uno a otro cada tres o cuatro palabras. El emisor y el destinatario han de ponerse de acuerdo para fijar la posición relativa de dos círculos concéntricos, que determinará la correspondencia de los signos.

Un siglo después, Giovan Battista Belaso de Brescia instituyó una nueva técnica. La clave, formada por una palabra o una frase, debe transcribirse letra a letra sobre el texto original. Cada letra del texto se cambia por la correspondiente en el alfabeto que comienza en la letra clave.

Pero los métodos clásicos distan mucho de ser infalibles. En algunos casos, basta hacer un simple cálculo para desentrañar los mensajes ocultos. Si se confronta la frecuencia habitual de las letras en el lenguaje común con la de los signos del criptograma, puede resultar relativamente sencillo descifrarlo. Factores como la longitud del texto, el uso de más de una clave juegan un papel muy importante, así como la intuición, un arma esencial para todo criptoanalista[9].

2.3.3.2 Criptográfica de clave pública

En los cifrados asimétricos o de clave pública, la clave de descifrado no se puede calcular a partir de la de cifrado.

En 1975, dos ingenieros electrónicos de la Universidad de Stanford, Whitfield Diffie y Martin Hellman, sugieren usar problemas computacionalmente irresolubles para el diseño de criptosistemas seguros. La idea consistía en encontrar un sistema de cifrado computacionalmente fácil (o al menos no difícil), de tal manera que el descifrado sea, por el contrario, computacionalmente irresoluble a menos que se conozca la clave. Hay que usar una transformación criptográfica T_k de fácil

[9] Persona dedicada a descifrar mensajes secretos mediante un conjunto de técnicas

aplicación, pero de tal forma que sea muy difícil hallar la transformación inversa $T_k^{'}$ sin la clave de descifrado. La función T_k es desde el punto de vista computacional, no inversible sin cierta información adicional (clave de descifrado) y se llama función de una vía o función trampa.

En estos esquemas se utiliza una clave de cifrado (clave pública) K que determina la función trampa T_k, y una clave de descifrado (clave secreta o privada) que permite el cálculo de la inversa $T_k^{'}$.

Cualquier usuario puede cifrar usando la clave pública, pero sólo aquellos que conozcan la clave secreta pueden descifrar correctamente. La seguridad del sistema se basa únicamente en la clave de descifrado.

Por otro lado, hay que resaltar la ventaja que representa en los sistemas asimétricos la posibilidad de iniciar comunicaciones secretas sin haber tenido ningún contacto previo.

A continuación, se describen algunos de los sistemas de clave pública que han tenido más trascendencia:

- **Sistema RSA**[10]

  Se basa en el hecho que no existe una forma eficiente de factorizar números que sean productos de dos grandes primos.

- **Sistema de Rabin**

  Se basa en la factorización de números primos.

- **Sistema de El Gamal**

  Se basa en el problema del logaritmo discreto.

[10] Este algoritmo fue implementado en el trabajo final. Para mayor información ver Apartado 2.3.3.2.1

- **Sistema de Merkle-Hellman**
  Esta basado en el problema de la mochila.

- **Sistema de McEliece**
  Se basa en la teoría de la codificación algebraica, utilizando el hecho de que la decodificación de un código lineal general es un problema NP-completo.

- **Sistemas basados en curvas elípticas**
  En 1985, la teoría de las curvas elípticas encontró de la mano de Miller aplicación en la criptografía. La razón fundamental que lo motivó fue que las curvas elípticas definidas sobre cuerpos finitos proporcionan grupos finitos abelianos, donde los cálculos se efectúan con la eficiencia que requiere un criptosistema y donde el cálculo de logaritmos es aún más difícil que en los cuerpos finitos. Además, existe mayor facilidad para escoger una curva elíptica que para encontrar un cuerpo finito, lo que da una ventaja más frente al sistema de el Gamal.

2.3.3.2.2 Algoritmo RSA

La seguridad del RSA se basa en el hecho de que no existe una forma eficiente de factorizar números que sean productos de dos grandes primos.

Fue desarrollado en 1977 por Ronald Rivest, Adi Shamir y Leonard Adleman, de ahí el nombre de RSA, que corresponde a las iniciales de los apellidos de sus autores.
Para implementar el algoritmo de una manera eficiente se deben seguir los siguientes pasos:

1. Buscar dos grandes números primos, p y q (secretos), y calcular el número n (público) mediante su producto, $n = p * q$. Para la búsqueda de estos primos se

debe utilizar un algoritmo que permita determinar con alta probabilidad que el número sea primo.

2. Encontrar la clave de descifrado constituida por un gran número entero impar, d (secreto), que es primo con el número Φ(n) (secreto). Este parámetro es obtenido de la siguiente manera:

$\phi(n) = (p-1)*(q-1)$. Siendo $\phi(n)$ la función de Euler.

3. Calcular el entero e (publico) tal que $1 \le e \le \phi(\mathrm{n})$, mediante la formula: $e * d = 1(\mathrm{mod}\,\phi(n))$.

4. Hacer pública la clave de cifrado (e, n).

5. Es necesario previamente codificar el texto en un sistema numérico, bien sea decimal o binario, y dividir en bloques M_i de tamaño j o $j-1$ de forma que según sea el alfabeto usado, el decimal o el binario cumpla en cada caso: $10^{j-1} < n < 10^j$ o $2^{j-1} < n < 2^j$.

6. Cifrar cada bloque M_i para transformarlo en un nuevo bloque de números C_i.

$C_i \equiv M_i^{\,e} (\mathrm{mod}\, n)$.

7. Si se quiere descifrar el bloque C_i, se usa la clave privada d según la expresión:

$M_i \equiv C_i^{\,d} (\mathrm{mod}\, n)$.

Las dos principales dificultades en la implementación del RSA son:

- Potencias modulares.
- Búsqueda de números primos.

Para aumentar las dificultades, la seguridad del RSA estriba en que los primos p y q elegidos han de ser muy grandes. Para encontrar los grandes primos p y q se pueden utilizar varios algoritmos, como el de Solovay-Strassen, y el de Lehman y Peralta, que sirven para comprobar la primalidad.

El calculo del numero entero d, primo con $\phi(n) = (p-1)*(q-1)$, se puede calcular tomando simplemente un número primo mayor que $\max\{p,q\}$.

Por ultimo, las operaciones de cifrado y descifrado requieren el calculo de potencias modulares, lo que se puede llevar a cabo en tiempo polinomial. Exactamente son necesarias $2(\log_2(n))$ multiplicaciones modulares, de manera que, por ejemplo, para el calculo de $2^{18} = \left(\left((2^2)^2\right)^2\right)^2 2^2$ son necesarias 5 multiplicaciones modulares.

Ejemplo

Para poder comprender mejor el funcionamiento del RSA, se ilustra el siguiente ejemplo:

1. $p = 3, q = 5$. Se eligen dos números primos.
2. $n = 15$. Se obtiene luego del cálculo del producto ($n = p * q$).
3. $\phi(n) = (3-1)*(5-1) = 8$
4. Tomando $e = 3$, se obtiene $d = 3$ por medio de la formula
   $(e * d) \bmod 8 = (3*3) \bmod 8 = 9 \bmod 8 = 1$
5. Clave publica es $(n, e) = (15,3)$
6. Tomando el mensaje $m = 2$ su cifrado es
   $c = m^e \bmod n$
   $c = 2^3 \bmod 15 = 8$

El mensaje que se enviaría seria el numero 8. Para poder interpretar de manera correcta, el mensaje se debe someter a un proceso de descifrado, como lo es:

$$m = c^d \bmod n = 8^3 \bmod 15$$
$$m = 512 \bmod 15 = 2$$

Con lo cual se obtiene el mensaje correcto. Se debe anotar que si la clave empleada no es la correcta el mensaje que se recibe no será el adecuado.

2.3.3.2.2.1 Tipos de ataques al algoritmo RSA

Para analizar la seguridad del sistema, se asume que el criptoanalista tiene una cantidad ilimitada de pares (M, C) de mensajes originales y sus correspondientes criptogramas. Las posibles maneras en que se puede atacar el sistema son:

2.3.3.2.1.1.1 Factorizar n

De esta forma obtiene el número $\phi(n) = (p-1)*(q-1)$, y con él la clave privada d, puesto que e es publica y se cumple:

$$e * d = 1(\bmod \phi(n))$$

Al ser n el producto de solo dos números primos, un algoritmo de factorización requiere como máximo ‘n pasos, pues uno de los dos factores es necesariamente un primo menor que ‘n. Sin embargo, si n fuera el producto de $N > 2$ primos, un algoritmo de factorización necesitaría como máximo $n^{1/N}$ pasos, que es una cota menor que ‘n, por lo que se concluye que es adecuada la obtención de n como producto de solo dos números primos.

2.3.3.2.1.1.2 Calcular Φ(n)

Si se tiene $\phi(n)$, dado que $p+q=n-\phi(n)+1$ y a partir de la suma se puede calcular $(p-q)^2$ por coincidir con $(p-q)^2-4*n$, luego se consigue la factorización $q=\frac{1}{2}[(p+q)-(p-q)]$ y $p=\frac{1}{2}[(p+q)+(p-q)]$.

2.3.3.2.1.1.3 Ataque por iteración

Si un enemigo conoce (n,e,C), entonces puede generar la secuencia: $C_1-C^e(\mathrm{mod}\,n),...,C_i-[C(i-1)]^e(\mathrm{mod}\,n)$, con lo que sí existe algún C_j tal que $C=C_j$ se deduce que el mensaje buscado es $M=C(j-1)$ pues $[C(j-1)]^e=C_j=C$. En cuanto a la igualdad $C_j=C$ se cumple solo para un valor de j demasiado grande, este ataque se vuelve impracticable. Rivest demostró que si los enteros $p-1$ y $q-1$ contienen factores primos grandes, la probabilidad de éxito mediante este procedimiento es casi nula para grandes valores de n.

2.3.3.2.1.1.4 Ataque de Blakley y Borosh

Blakley y Borosh demostraron que el RSA no siempre esconde el mensaje. Un ejemplo de esta situación es: tomando $e=17,n=35$ y los mensajes a cifrar son $M_1=6$ y $M_2=7$, entonces se obtiene que $6^{17}=6(\mathrm{mod}\,35)$ y $7^{17}=7(\mathrm{mod}\,35)$. Una situación más peligrosa para el sistema aparece, con los valores $p=97$ $q=109$ y $e=865$, ya que el criptosistema resultante no esconde ningún mensaje, pues $M^{865}=M(\mathrm{mod}(97*109))$.

De todo lo anterior se concluye que para que la seguridad del sistema RSA quede perfectamente salvaguardada es necesario escoger cuidadosamente las claves a utilizar.

La principal desventaja del RSA radica en que es mucho más lento que el DES y que los cifrados en flujo. Para tener una idea de la velocidad del funcionamiento del algoritmo se puede decir que el DES trabaja a unos 20 megabits por segundo, mientras que el RSA funciona a una velocidad 1000 veces menor, y aunque se esta investigando al respecto y se ha logrado aumentar la velocidad, la diferencia de tiempo requerido para la ejecución de estos dos algoritmo se mantendrá, ya que también se investiga en los algoritmos de clave secreta para mejorar su velocidad de ejecución.

2.3.3.3 *Criptografía de clave secreta*

En los cifrados de clave secreta, la seguridad depende de un secreto compartido exclusivamente por emisor y receptor.

La principal amenaza criptoanalítica proviene de la alta redundancia de la fuente. El propósito de la difusión consiste en anular la influencia de la redundancia de la fuente sobre el texto cifrado. Hay dos formas de conseguirlo, la primera conocida como transposición, evita los criptoanálisis basados en las frecuencias de las n-palabras. La otra manera consiste en hacer que cada letra del texto cifrado dependa de un gran número de letras del texto original.

El objetivo de la confusión consiste en hacer que la relación entre la clave y el texto cifrado sea lo más compleja posible, haciendo así que las estadísticas del texto cifrado no estén influenciadas por las del texto original.

Algunos tipos de cifrado secreto son:

- **El cifrado por transposición**: consiste en la alteración del orden de las unidades del texto original según una clave.

- **El cifrado por sustitución**: consiste en el reemplazo de las unidades del texto original según una clave.

- **Cifrado producto**: la aplicación iterativa de cifrados sobre textos ya cifrados, es decir, a la composición de varios cifrados. Los cifrados simétricos son cifrados producto de las dos operaciones mencionadas, sustitución y transposición.

- **Cifrado en bloque, DES[11]:** opera sobre textos formados por n-palabras, convirtiendo cada una de ellas en una nueva n-palabras.

- **Cifrado en flujo:** se aplica sobre textos formados por letras combinándolas con un flujo de bits secretos (secuencia cifrante) mediante un operador *****. El descifrado se realiza de una forma análoga, combinando mediante el operador ***^-1** las letras del texto cifrado con la secuencia cifrante, produciendo así las letras del texto original. La secuencia cifrante se produce mediante un generador de bits.

2.3.3.3.1 Algoritmo DES

Es un sistema criptográfico que toma como entrada un bloque de 64 bits del mensaje (lo somete a 16 interacciones), una clave de 56 bits. Se suele utilizar un bloque para la clave de 64 bits, ya que a cada trama de 7 bits se le agrega un bit que puede ser usado como bit de paridad.

El DES tiene diferentes formas de ser implementado dependiendo de la aplicación; las formas de implementar el DES son:

- ☑ **ECB** (Electronic Codebook Mode) Se utiliza para mensajes cortos. (Menos de 64 bits).

[11] Este algoritmo fue implementado en el proyecto final. Para más información ver apartado 2.3.3.3.1. El estándar del algoritmo se encuentra en el Anexo 1

- **CBC** (Cipher Block Chaining Mode) Se utiliza para mensajes largos.
- **CFB** (Cipher Block Feedback) Se utiliza para cifrar bit por bit ó byte por byte.
- **OFB** (Output Feedback Mode) Tiene el mismo uso que el anterior, pero evita la propagación de errores.

No se ha podido romper el sistema **DES** mediante la deducción de la clave simétrica a partir de la información interceptada, sin embargo por medio del método de fuerza bruta (probando alrededor de 2^{56} posibles claves), se pudo romper **DES** en enero de 1999.

Se concluye que es posible obtener la clave del sistema **DES** en un tiempo relativamente corto, por lo que lo hace inseguro para propósitos de alta seguridad. La opción que se ha planteado para poder suplantar al **DES** es usar lo que se conoce como cifrado múltiple, es decir aplicar varias veces el mismo algoritmo para fortalecer la longitud de la clave. Este nuevo sistema de cifrado se conoce como **triple-DES** o **TDES**.

2.3.4 Tipos de ataques

En los sistemas criptográficos se pueden presentar los siguientes ataques:

2.3.4.1 Ataque sólo con texto cifrado

Es la peor situación para el criptoanalista, debido a que se presenta solo cuando se conoce el criptograma[12].

2.3.4.2 Ataque con texto original conocido

El criptoanalista tiene acceso a una relación existente entre el texto inicial y el cifrado. Este caso se puede dar cuando se conoce el tema del cual trata el mensaje,

[12] Se define como el texto cifrado.

ya que proporciona una correspondencia entre las palabras más probables y las palabras cifradas mas repetidas.

2.3.4.3 Ataque con texto original escogido

Además de contar con el criptograma, el criptoanalista tiene acceso al cifrado de cualquier texto que elija. No es que él sepa cifrarlo, sino que lo obtiene ya cifrado.

2.3.4.4 Ataque con texto cifrado escogido

El criptoanalista puede obtener el texto original correspondiente a determinados textos cifrados de su elección.

CAPITULO 3

MARCO METODOLÓGICO

La ingeniería del software establece estándares que facilitan el control del proceso de desarrollo de software de calidad. En este capitulo se describe proceso unificado de desarrollo, metodología utilizada para la realización de este trabajo.

3.*1 METODOLOGÍA*

En el desarrollo de un proyecto software es de gran importancia seleccionar una metodología adecuada para lograr los objetivos trazados. Una de las metodologías mas utilizadas que cuenta con múltiples herramientas que proporcionan una guía para obtener y distribuir correctamente las actividades del equipo de desarrollo y especifica correctamente los productos dentro del proyecto, además de permitir definir claramente los roles dentro del equipo, es el Proceso Unificado de Desarrollo de Software.

Este proceso está basado en un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software, proporcionando normas para el desarrollo eficiente de software de calidad dentro de los plazos y presupuestos planeados.

El proceso unificado de desarrollo software utiliza el lenguaje unificado de modelado (UML), para preparar todos los esquemas de un sistema software. Además permite llevar a cabo el proyecto de una manera eficiente en términos de costo, calidad y tiempo, cumpliendo con el cronograma y presupuesto pactados.

El proceso unificado esta basado en componentes, dirigido por casos de uso, centrado en la arquitectura y es iterativo e incremental.

- ***Basado en componentes*:** el software a desarrollar esta formado por componentes software que son partes físicas del sistema y que pueden ser reemplazadas.

- ***Dirigido por casos de uso*:** un caso de uso es un conjunto de acciones que lleva a cabo el sistema con el fin de proporcionar al usuario un resultado importante y representa un requisito funcional. Los casos de uso guían el proceso a través de todos los flujos de trabajo y permiten desarrollar el sistema en función de las necesidades del usuario; además guían la arquitectura del sistema quien a su vez influye en la selección de los casos de uso.

- ***Centrado en la arquitectura*:** la arquitectura incluye los aspectos más significativos del sistema: su estructura, comportamiento, restricciones, plataforma en la que debe funcionar el software, sistemas heredados, reutilización de componentes y requisitos no funcionales; debe permitir el desarrollo de todos los casos de uso requeridos y estos deben encajar en la arquitectura cuando se llevan a cabo. Los Casos de Uso presentan relación con la Arquitectura, puesto que cada producto tiene tanto una función (Caso de Uso) como una forma (Arquitectura), ambas deben equilibrarse si se desea lograr un producto de éxito; con base en esto, los casos de uso y la arquitectura deben evolucionar en paralelo.

El desarrollo software complejo implica un aumento en el esfuerzo, por esta razón es práctico dividir el desarrollo en pequeñas partes. En el Proceso Unificado de Desarrollo cada entrega corresponde a una iteración, la cual finaliza en un incremento (crecimiento del producto). Cada iteración se realiza con un ciclo en cascada, es decir, se manejan cuatro fases: análisis, diseño, implementación y prueba; al finalizar se realiza una evaluación de los objetivos, si estos se cumplen se

continúa con la siguiente iteración, de lo contrario es conveniente replantear el enfoque con el que se está trabajando.

Es de vital importancia seleccionar primero los casos de uso más relevantes con el fin de construir una arquitectura robusta en las primeras iteraciones que soporte el sistema de una manera eficiente. Se deben seleccionar y programar solo las iteraciones necesarias para lograr los objetivos del proyecto y desarrollar un proceso iterativo controlado.

Cada vez que se hace un incremento en el sistema, se debe validar la funcionalidad del incremento y de los demás componentes desarrollados en ese momento. El proceso unificado reduce el riesgo de retrasos en el calendario previsto, acelera el ritmo de desarrollo e involucra al usuario en todas las fases refinando los objetivos que dirigen el trabajo en cada iteración.

El enfoque iterativo e incremental, guiado por los casos de uso y centrado en la arquitectura, permite una comprensión creciente de los requerimientos, al mismo tiempo que el sistema crece, por esta razón, presenta sustanciales beneficios, debido a que reduce el costo de los riesgos, el costo de un solo incremento, a su vez, logra tener un sistema ejecutable mucho más pronto.

El proceso unificado de desarrollo de software se repite durante la vida del sistema a manera de ciclos que constituyen cada uno una versión del producto. Cada ciclo se subdivide en cuatro fases (inicio, elaboración, construcción, transición) que a su vez se subdividen en iteraciones que se desarrollan a lo largo de 5 flujos de trabajo fundamentales: requisitos, análisis, diseño, implementación y prueba. Como se observa en la figura 3.1.

Otro aspecto relevante de esta metodología es la correcta organización del tiempo que permite cumplir con los plazos planeados, es decir la división del proyecto en iteraciones o mini-proyectos los cuales traen consigo muchos beneficios dentro de los que están, una reducción de costos por desarrollos no óptimos, pues efectuar entregas en las fechas estipuladas evita que la mayoría de los problemas no se

presenten en la fase final como suele ocurrir cuando se siguen otras metodologías, sino que los riesgos se logran identificar en etapas muy tempranas.

Otra característica es la realización de tareas simultáneas, es decir se puede ir recolectando información documentada del tema de investigación, mientras se captura el muestreo de datos.

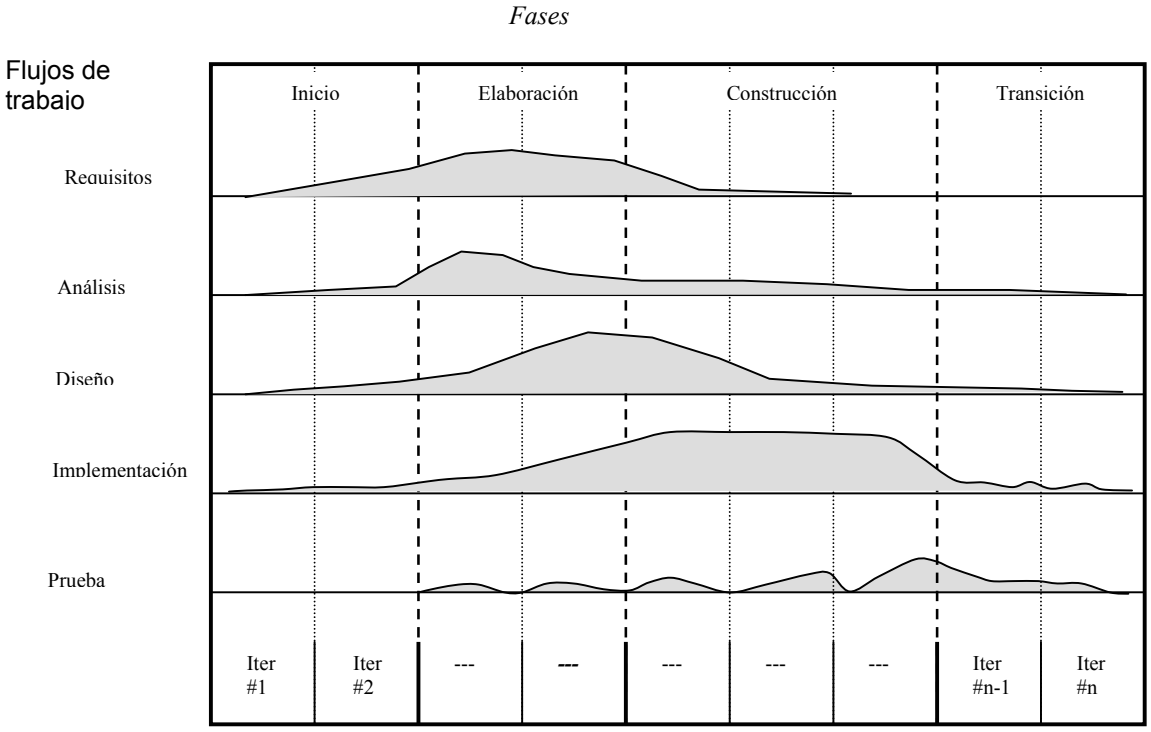


Figura 3.1 *Flujos de trabajo*

3.2 PLAN DE TRABAJO

Para la elaboración del proyecto se seguirá la metodología planteada por el Proceso Unificado de Desarrollo Software. Comprende las siguientes fases: inicio, elaboración, construcción. Cada una de estas fases estará subdivida en iteraciones,

las cuales constituyen pequeños ciclos de vida dentro de los cuales se recorrerán los flujos de trabajo: requisitos, análisis, diseño, implementación y prueba.

A continuación se detalla de manera general cada una de las fases de desarrollo con sus respectivas actividades, mencionando los aspectos más relevantes para la construcción de la herramienta software y las labores que requieren más atención en cada flujo de trabajo. Las actividades mencionadas pueden llegar a ampliarse o reducirse según las dificultades encontradas a lo largo del proyecto.

3.2.1 Fase de inicio

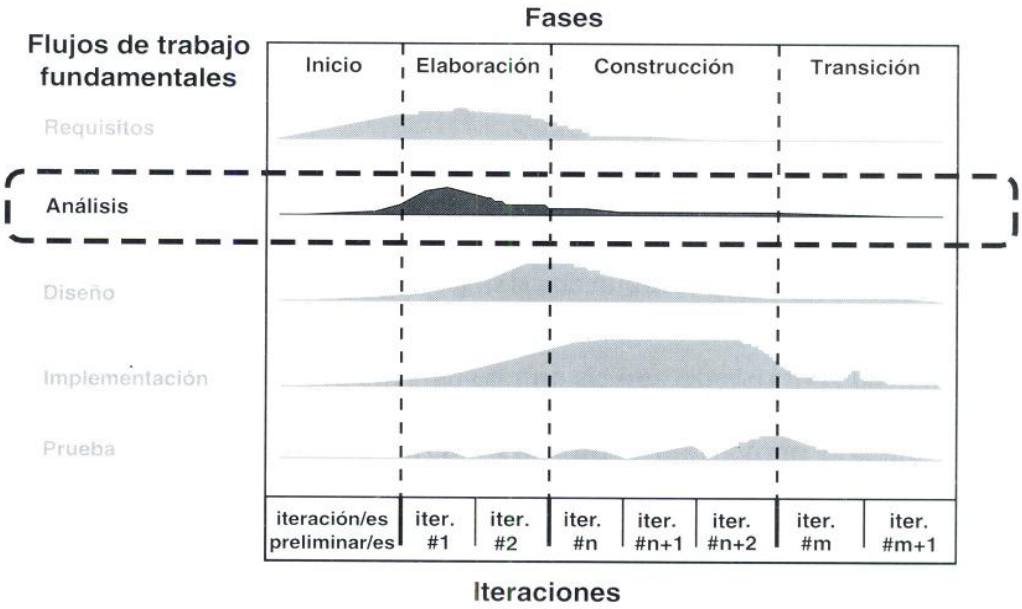


Figura 3.2 *Flujos de trabajo de análisis*

La fase de inicio tiene como objetivo desarrollar una descripción del producto final a partir del objetivo principal del proyecto y hacer un análisis de la problemática que permita determinar cuales son las principales funciones que debe realizar el sistema.

Las actividades que se llevaran a cabo en esta fase son:

- ☑ Realizar una búsqueda de información en el área de seguridad informática para poder conocer las herramientas existentes en el mercado que buscan solucionar la problemática presentada. De esta manera se espera poder ubicarnos de una manera mas precisa en el contexto en el cual se desarrollan las soluciones.

- ☑ Estudiar el lenguaje UML, utilizado como soporte al Proceso Unificado de Desarrollo de Software.

- ☑ Establecer los requisitos fundamentales, empezando por los más representativos.

- ☑ Establecer el modelo de casos de usos iniciales.

- ☑ Definir el ámbito y los límites del sistema.

- ☑ Desarrollar un prototipo que muestre el funcionamiento del sistema.

- ☑ Descripción de la arquitectura candidata.

3.2.2 Fase de elaboración

En esta fase la labor principal es seleccionar una arquitectura estable con el fin de construir el sistema dentro de un marco de trabajo económico y planificar adecuadamente las labores de construcción.

Las actividades a realizar en esta fase son:

- ☑ Construir el modelo de análisis a partir del modelo de Casos de Uso.
- ☑ Crear una base para la arquitectura que cubra la funcionalidad del sistema.
- ☑ Indagar sobre los lenguajes de programación existentes para poder definir cual se adapta más a la idea a desarrollar.
- ☑ Recopilar la mayor parte de los requisitos pendientes de la fase de Inicio.
- ☑ Definir el diseño de la base de datos, que contendrá la información acerca de la aplicación protegida.

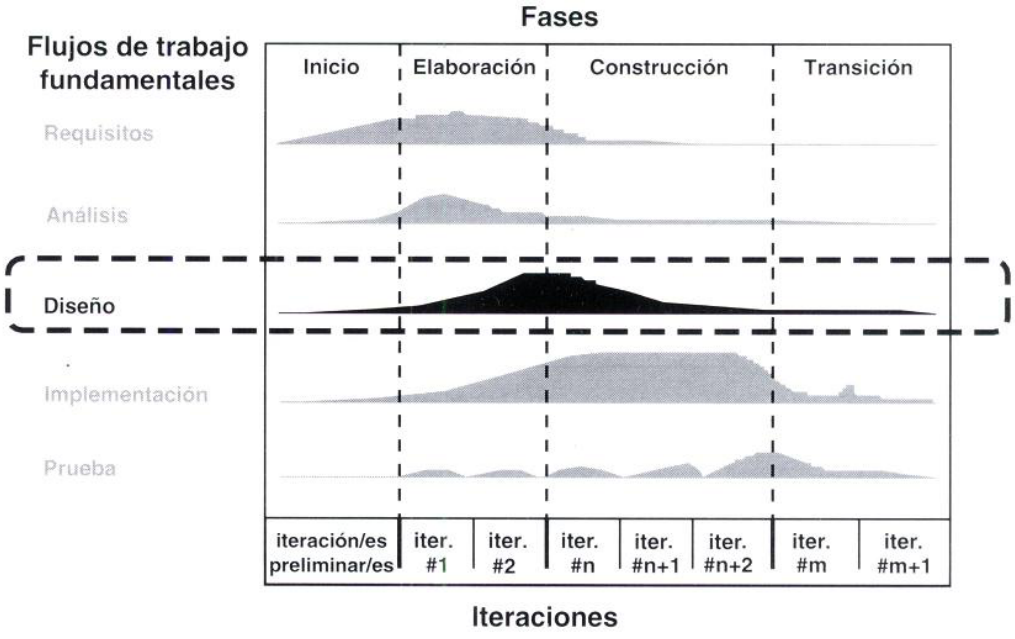


Figura 3.3 *Flujos de trabajo de diseño*

3.2.3 Fase de construcción

El propósito de esta fase consiste en desarrollar el producto software que implemente la funcionalidad del sistema que fue establecido en las fases anteriores y plasmada en la arquitectura. El proceso de desarrollo software se realiza de manera iterativa e incremental, de tal forma que a través del tiempo se generan

prototipos que garantizan el cumplimiento e implementación de las alternativas que satisfacen los requisitos establecidos.

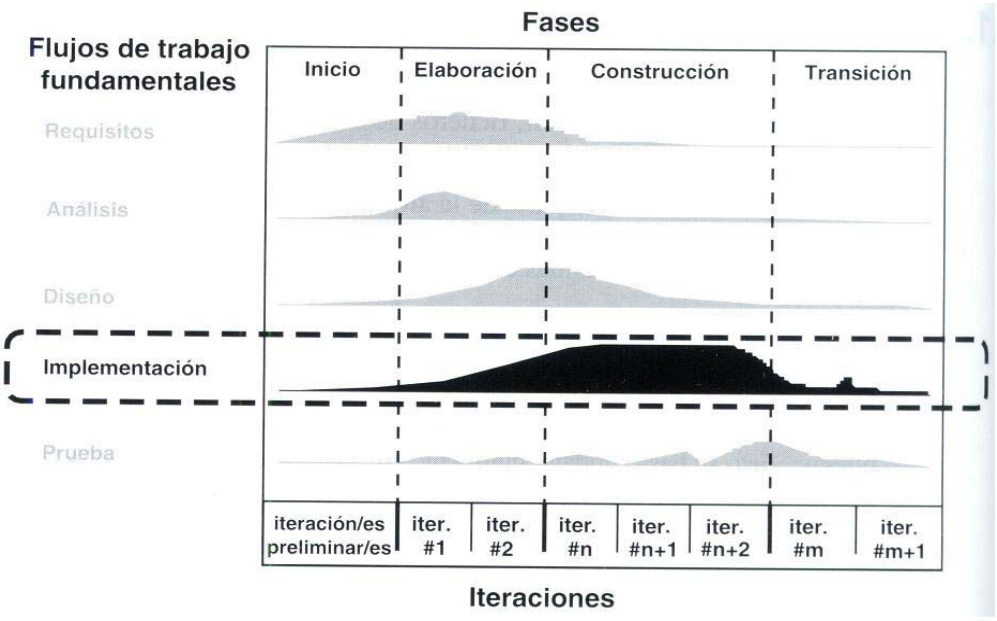


Figura 3.4 *Flujos de trabajo de implementación*

Esta fase cubre las siguientes actividades:

- ☑ Desarrollar los algoritmos necesarios que implementen la funcionalidad del sistema.

- ☑ Implementar la base de datos que contendrá la información relacionada con las aplicaciones protegidas.

- ☑ Construir una interfaz que le permita al usuario interactuar con la herramienta software.

- ☑ Realizar pruebas que garanticen el correcto funcionamiento de la herramienta software.

- ☑ Integrar el sistema completo.

PARTE II

PROCESO UNIFICADO DE DESARROLLO

El proceso unificado de desarrollo software recopila y detalla el conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software; dicho sistema debe realizar con éxito lo que sus futuros usuarios necesitan. Por lo tanto se debe llevar un proceso planificado, ordenado y evolutivo que permita finalmente lograr los objetivos.

Este proceso se centra en la arquitectura donde se incluyen los aspectos estáticos y dinámicos significativos del sistema que surgen de las necesidades del negocio, los cuales se reflejan en los casos de uso. Sin embargo, estos aspectos están acompañados de otros factores como son: plataforma software, arquitectura hardware, sistema operativo, sistema de gestión de base de datos y protocolos de comunicación en red, los cuales con una serie de ciclos constituyen la vida de un sistema.

Cada ciclo concluye en una versión del producto para los usuarios. Cada uno de estos, se desarrolla a lo largo del tiempo y se dividen en fases, con una secuencia de modelos que visualizan lo que esta sucediendo en las fases (Inicio, Elaboración y Construcción), cada una con objetivos a cumplir.

Los objetivos específicos de esta parte son:

Fase Inicio.

- Desarrollar una descripción del producto final a partir de la identificación y análisis de requisitos.

- Desarrollar el análisis del negocio es busca de justificar la ejecución del proyecto.

Fase Elaboración

- Especificar el mayor porcentaje de casos de uso del producto.

Fase Construcción

- Completar la línea base hasta transformarla en una versión operativa del sistema.

CAPITULO 4

FASE DE INICIO

4.1 PLANIFICACIÓN

El objetivo principal de esta fase es identificar las características y funciones que tendrá la herramienta software. Para llegar a esta fase no se partió de la nada, sino de una continua maduración de la idea del proyecto, que condujo al inicio de esta fase con un modelo claro del negocio.

Luego de desarrollar los flujos de trabajo correspondientes a esta fase se espera tener una parte de los requisitos del proyecto definidos, así como los riegos del negocio con su plan de contingencia. El proyecto deberá ser viable y entendible por parte del equipo de trabajo.

4.2 FLUJOS DE TRABAJO

Los flujos de trabajo de esta fase giran entorno a dos grupos de actividades como son la determinación del ámbito del sistema y la comprensión de la arquitectura candidata. Los participantes del proyecto deben adquirir varios roles en esta fase.

4.2.1 Flujo de trabajo de los requisitos

- **Actividad: enumerar los requisitos candidatos**

A continuación se presenta una primera aproximación al listado de requisitos de la herramienta software:

Lista de Requisitos

- ☑ El sistema debe proporcionar protección a una aplicación por fecha y numero de accesos.

- ☑ El sistema debe realizar todas las operaciones sin tener acceso al código fuente de la aplicación a proteger.

- ☑ El sistema debe permitir renovar la protección por un tiempo definido por el usuario.

- ☑ El sistema debe permitir renovar la protección por una cantidad de usos definida por el usuario.

- ☑ Avisar al usuario de la aplicación protegida el vencimiento de la protección con un tiempo de anticipación.

- ☑ Permitir al administrador del sistema llevar información relacionada con sus aplicaciones protegidas mediante una base de datos.

- ☑ Establecer unos permisos de usuarios del sistema que limiten las opciones de renovación y asignación de protección en la herramienta.

- ☑ El sistema debe proporcionar a la aplicación protegida los mecanismos necesarios para desactivarse en caso de vencimiento o corrupción.

Actividad: identificar riesgos críticos

Descripción	Nivel de riesgo	Impacto	Monitor	Responsabilidad	Contingencia
Incumplir con los tiempos y objetivos propuestos en plan de proyecto	Critico	Global	Autores	Autores y director del proyecto	Adaptar los mecanismos de control de la gestión
Cambio en las estructuras de los PE	Critico	Global	Autores	Autores	Limitar la aplicación del sistema
Problemas con el injerto de código	Significativo	Global	Autores	Autores	Estudiar a fondo la forma de realizarlo
Aparición de nuevas formas de hacer vulnerable nuestra protección	Significativo	Global	Autores		Adaptar nuevas versiones del producto que estén de acuerdo con la situación

Tabla 4.1 *Riesgos Críticos*

Actividad: comprender el contexto del sistema

☑ **Descripción del contexto**

Debido a la problemática que existe en el medio con el uso de software sin control, surgió la necesidad de realizar una herramienta propia que permitiera proteger las aplicaciones hechas en la escuela, mediante fechas y usos; así como a su vez llevar un control de todo el comportamiento de una

aplicación y sus clientes, almacenando información referente a esta en una base de datos.

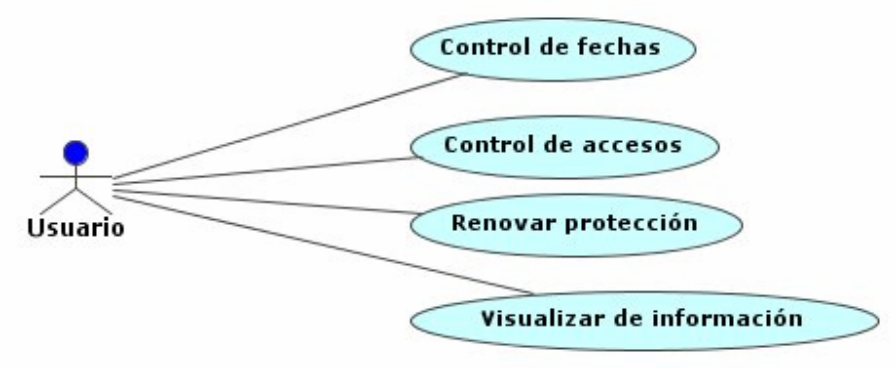


Figura 4.1 *Modelo de casos de uso del negocio*

- **Actividad: representar los requisitos como casos de uso**

☑ **Encontrar actores**

Actor	Descripción	Responsabilidades	Necesidades
Usuario	Representa a un individuo que usa la herramienta con el fin de proteger y renovar la aplicación software de un tercero	• Hacer un uso adecuado de la herramienta sacándole el máximo de rendimiento. • Recurrir a la misma para realizar las tareas de renovación y asignación de protección antes de ser entregada al usuario final. • Realizar labores de administración y mantenimiento del sistema.	El usuario utiliza la herramienta para: • Proteger por numero de accesos una aplicación software • Proteger por fecha una aplicación software • Renovar la protección de la aplicación cuando el cliente así lo requiera. • Obtener información de las aplicaciones protegidas mediante la base de datos.

Cliente	Representa a una persona o a una empresa que hace uso de la aplicación protegida.	✣ Usar la aplicación protegida dentro de los rangos permitidos por la herramienta ✣ Ponerse en contacto con los proveedores de la aplicación para realizar la renovación de la protección	El cliente utiliza el sistema para: ✣ Recurrir a este cuando necesita renovar la protección de la aplicación. ✣ Solicitar la clave de uso cuando ocurra algún problema con la aplicación protegida.

Tabla 4.2 *Actores del sistema*

☑ **Modelo general de casos de uso**

El diagrama de casos de uso que se presenta a continuación representa el gran grupo de funciones principales que va ha desarrollar la herramienta software. Con esto se busca dar una idea global del comportamiento de la herramienta.

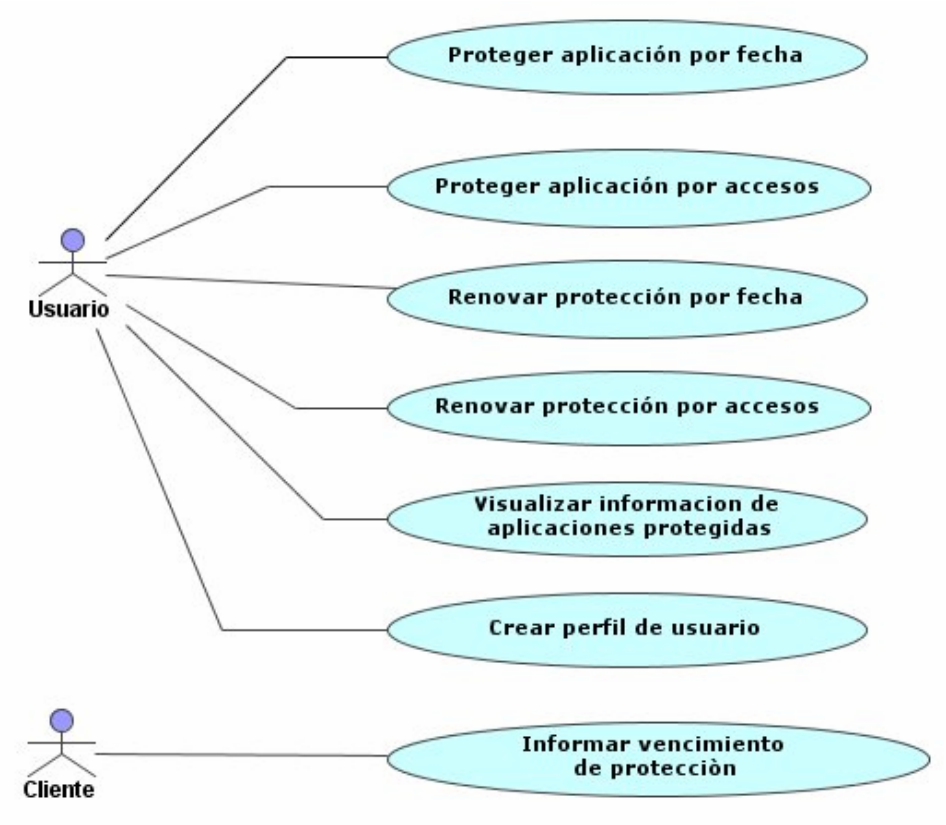


Figura 4.2 *Modelo general de casos de uso*

☑ Principales casos de uso

Proteger aplicación por fecha

Figura 4.3 *Caso de uso Proteger aplicación por fecha*

Descripción: permite al usuario de la herramienta establecer la protección por el tiempo de uso de la aplicación medido en días.

- **Proteger aplicación por número de accesos**

Figura 4.4 *Caso de uso Proteger aplicación por accesos*

Descripción: permite proteger la aplicación por número de accesos a la misma por parte del usuario.

- **Renovar protección por fecha**

Figura 4.5 *Caso de uso Renovar protección por fecha*

Descripción: permite al administrador luego de vencerse el plazo de protección renovar por un periodo de tiempo la aplicación antes protegida.

- **Renovar protección por accesos**

Figura 4.6 *Caso de uso Renovar protección por accesos*

Descripción: permite al administrador, luego de vencerse el plazo de protección renovar por un número determinado de accesos para la protección protegida.

- **Informar vencimiento protección**

Figura 4.7 *Caso de uso Informar vencimiento de protección*

Descripción: en el momento que este próximo el vencimiento de la protección se informará visualmente del vencimiento y este continuará hasta que sea renovada la licencia.

- **Visualizar información de aplicaciones protegidas**

Figura 4.8 *Caso de uso Visualizar información de aplicaciones protegidas*

Descripción: permite al administrador de la herramienta obtener información acerca de todas las aplicaciones protegidas y a su vez información particular de cada aplicación.

- **Crear perfil de usuario**

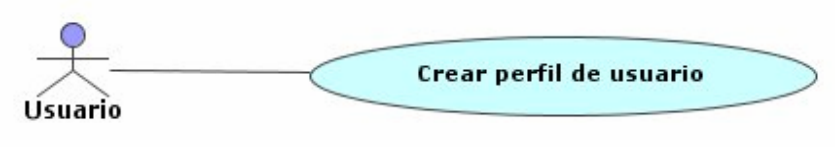


Figura 4.9 *Caso de uso Crear perfil de usuario*

Descripción: todo usuario que desee acceder a información de aplicaciones especificas debe tener un perfil, el cual irá acompañado de una contraseña.

☑ **Detallar un caso de uso: renovar protección por fecha**

Este caso de uso es lo suficientemente representativo como para observarlo en detalle:

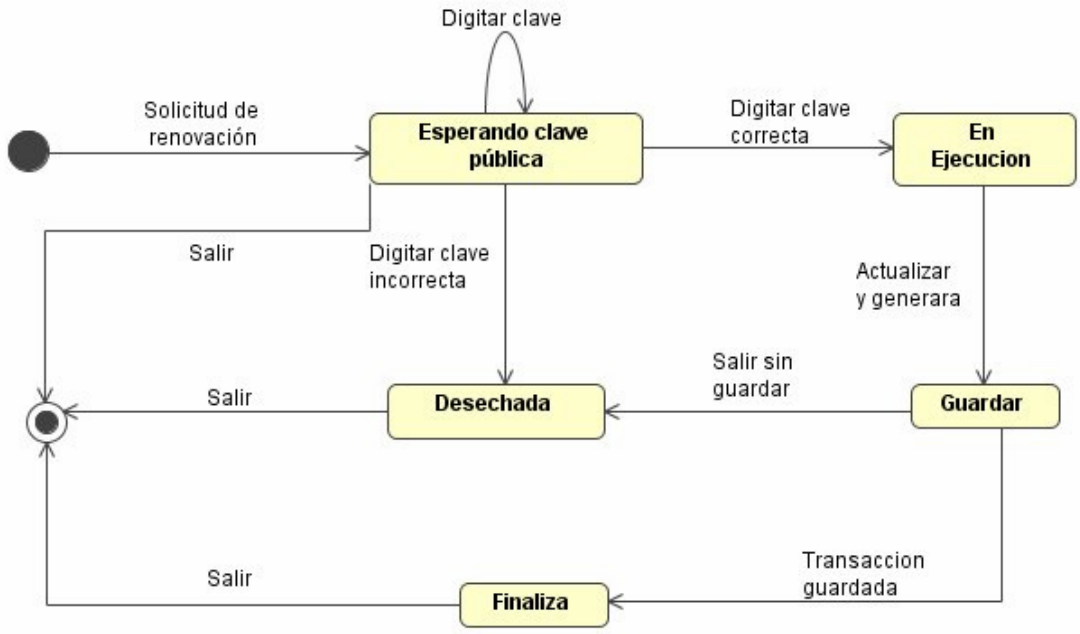


Figura 4.10 *Diagrama de estado del caso de uso: renovar protección por fecha*

Caso de uso: renovar protección por fechas	
Precondición	El usuario debe conocer la clave pública de proyecto protegido a renovar

Descripción	◆ El usuario luego de ingresar la clave publica al sistema, este entra en un estado de comprobación, sí coincide con la clave del registro del proyecto, se presentará la opción de actualizar. ◆ En la opción de actualizar el usuario presenta alternativas tales como: número de días de renovación, número de accesos y / o datos de interés; posterior a esto el sistema genera una nueva clave privada que incluye los cambios realizados anteriormente y con la cual se puede realizar la posterior renovación. ◆ Finaliza el proceso de renovación exitosamente, pueden ocurrir el caso que la clave pública sea errónea o no sea digitada, en este caso el sistema finaliza el proceso inmediatamente.
Caminos alternativos	◆ El usuario una vez inicie correctamente el proceso de renovación (digite correctamente la clave y sea comprobada) puede decidir si renueva la fecha de protección de su aplicación o si aborta. ◆ En el paso final puede escoger si guarda los cambios realizados y de esta manera finalizar correctamente el proceso o por el contrario no guarda los cambios y el proceso de renovación en este caso sería nulo.
Poscondiciones	◆ El proceso correcto de renovación finaliza cuando se renueva la fecha de protección de la aplicación, se genera la nueva clave privada y se han guardado los cambios o caso contrario si ha sido desechada.

Tabla 4.3 *Caso de uso Proteger aplicación por accesos*

4.2.2 Flujo de trabajo del análisis

Actividad: analizar la arquitectura

El propósito de analizar la arquitectura es esbozar el modelo de análisis y la arquitectura mediante la identificación de paquetes del análisis, clases del análisis evidentes y requisitos especiales comunes. Los paquetes se crean con base en los requisitos funcionales, representados en casos de usos y en las entidades del dominio del negocio.

Identificación de paquetes del análisis

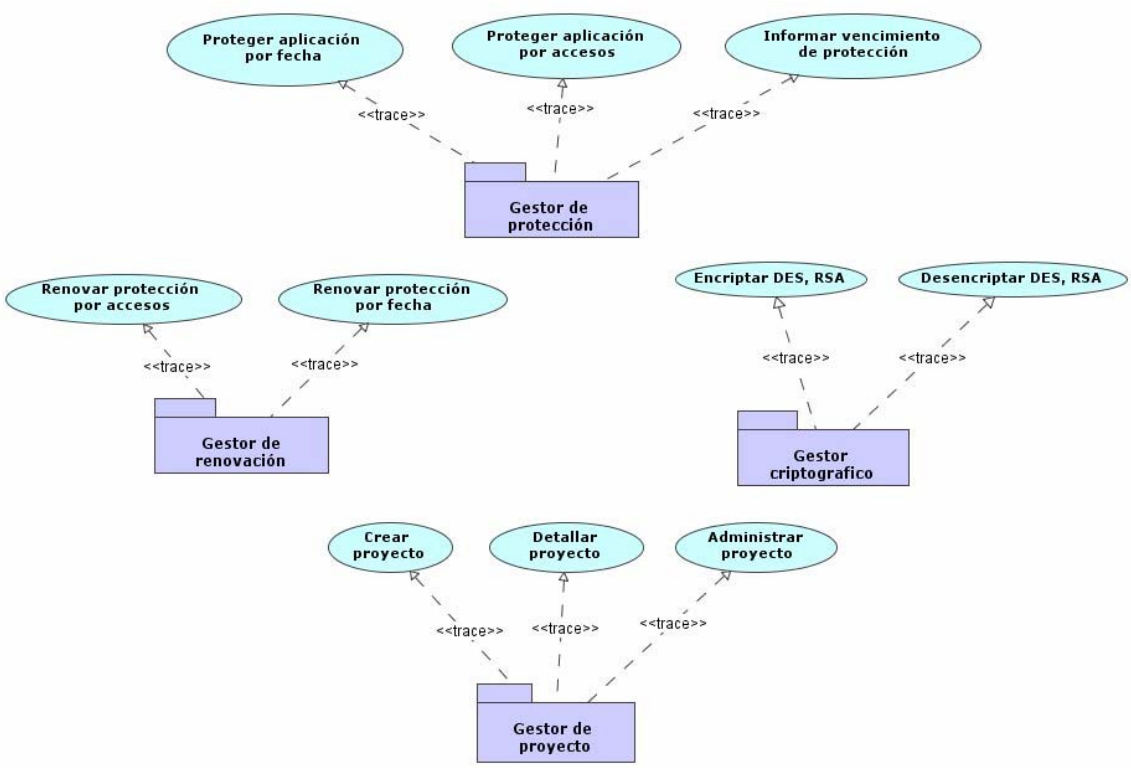


Figura 4.11 *Identificación de paquetes generales de análisis*

☑ **Identificación de clases de entidades obvias**

Actividad: analizar un caso de uso

Considerando que el caso de uso *Proteger aplicación por fechas* es representativo de la arquitectura candidata se analizará a continuación

☑ **Identificación de clases del análisis**

Elegiremos los casos de uso que sean relevantes para obtener una buena arquitectura. El primer paso es identificar las clases del análisis y la relación para interactuar hacia el flujo de la información

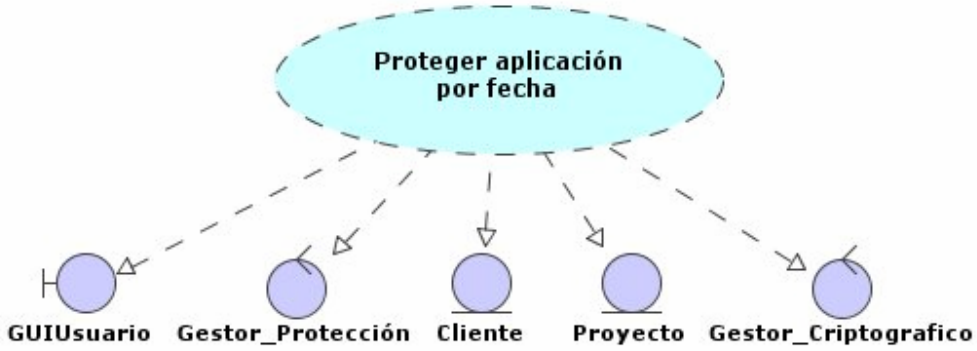


Figura 4.12 *Clases de análisis en el caso de uso Proteger aplicación por fecha*

Se identifican las clases de control, entidad e interfaz necesarias para el caso de uso *Proteger Aplicación por fechas***.**

Observando la figura encontramos dos clases de control como son cliente y proyecto que son tomados directamente de la base de datos, una interfaz gráfica de usuario que permite la integración y uso de los módulos.

☑ **Descripción de las interacciones entre objetos del sistema**

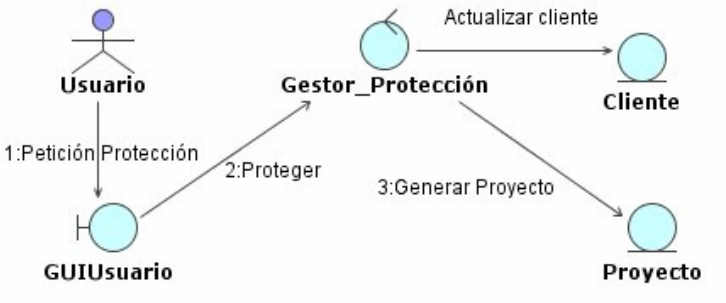


Figura 4.13 *Análisis del caso de uso Proteger aplicación por fecha*

4.3 EVALUACIÓN DE LA FASE DE INICIO

Al finalizar la fase inicio, se concluye que lo más importante es saber que el proyecto es viable, esto se puede apreciar al examinar el ámbito, riesgos críticos y arquitectura candidata. El producto de esta fase se encuentra resumido en la tabla 4.4.

Flujo	***Vista***	***Comentario***
Requisitos	**Tabla 4.1** *Riesgos Críticos* **Figura 4.1** *Modelo de casos de uso del negocio* **Figura 4.2** *Modelo general de casos de uso* **Figura 4.10** *Diagrama de estado del caso de uso: renovar protección por fecha* **Tabla 4.2** *Actores del sistema* **Tabla 4.4** *Caso de uso Proteger aplicación por accesos*	Se presenta la viabilidad del proyecto mediante la mitigación del riesgos y un breve esbozo de la arquitectura candidata mediante el análisis de estado de los casos de uso

Análisis	**Figura 4.11** *Identificación de paquetes generales de análisis* **Figura4.13** *Análisis del caso de uso Proteger aplicación por fecha*	Identificar paquetes identificados del sistema

Tabla 4.4 *Vistas de la fase de inicio*

4.4 PLANEACIÓN DE LA FASE DE ELABORACIÓN

El resultado de la fase de inicio es un modelo de casos de uso parcialmente completo y una descripción de la arquitectura candidata; ahora se recopilará la mayor parte de los requisitos, los cuales serán clasificados según su prioridad y analizados de acuerdo a su importancia en la arquitectura que busca.

Para el logro del objetivo fundamental de esta fase se desarrollara un porcentaje mayoritario de casos de uso en detalle, de tal modo que satisfagan la línea base y que esta sea lo suficientemente robusta para resistir la fase que le sigue (construcción); sin embargo se deberá abordar y mitigar los riesgos que pueden interferir en la consecución de este objetivo.

CAPITULO 5

FASE DE ELABORACIÓN

Los principales objetivos de esta fase son la recopilación de la mayor parte de los requisitos, expresando los requisitos funcionales como casos de uso y asociando a estos sus actividades y procesos, además de establecer una arquitectura base que guíe y fundamente el trabajo en las siguientes fases de desarrollo.

Para el logro de los objetivos se identificara y analizará de manera genérica los puntos claves del sistema y posteriormente se profundizará, para con ello empezar a formar una sólida línea base de la arquitectura, la cual implica desarrollar en gran parte los casos de uso y abordar los riesgos que obstaculicen la consecución del objeto de la fase.

5.1 IDENTIFICACIÓN DE RIESGOS

El modo en que se planifica el desarrollo de un nuevo sistema está influenciando por factores de riesgo que se perciben, por tanto es recomendable crear al inicio de cada fase de desarrollo una lista de riesgos asociado con su respectivo plan de contingencia. La identificación del riesgo se puede tornar difícil y mucho más su impacto, debido a la carencia o fiabilidad de las fuentes de información; sin embargo se conoce de la posibilidad de ocurrencia de eventos que desnudan los riesgos críticos y no críticos, que posteriormente deben ser mitigados y de esta manera realizar una exitosa planificación para el logro de objetivos.

Los tipos de riesgo que se presentan son los riesgos técnicos, cada uno de los cuales se transforma en esta etapa en un caso de uso; una vez implementado y estructurado correctamente mitiga el riesgo y de esta manera forman parte del proceso, aunque existen riesgos marginados que deben ser tratados en detalle independientemente y mitigados antes que su presencia afecte el proceso de desarrollo. Estos son:

- ***Riesgo 1****: establecimiento de una arquitectura no flexible:* uno de los riesgos más críticos que se pueden presentar es la construcción de una estructura arquitectónica que no pueda evolucionar en su ciclo de vida de desarrollo.

  Control: se identifica los casos de uso arquitectónicamente significativos, que corresponden a aquellos que cubren las tareas funcionales que el sistema ha de realizar, además de la identificación de los casos de uso de requisitos no funcionales, casos de uso secundarios, auxiliares y opcionales, que aunque no son importantes arquitectónicamente hablando permiten la mayor cobertura de requisitos y encontrar una base de la arquitectura.

- ***Riesgo 2****: recopilación incorrecta de requerimientos:* un sistema que no haga lo que realmente deseen los usuarios no contribuye al logro de objetivos inicialmente trazados y se estaría hablando de una planificación errada en la que se quería desarrollar un software y se obtuvo otro con errores y defectos.

  Control: una vez recopilada la información y definidos los requisitos se sigue con la identificación de casos de uso necesarios para asegurar que se esta desarrollando el sistema correcto que asegura la evolución del sistema en las siguientes fases. Todo esto se consigue con la realización de un flujo de trabajo de los requisitos de usuario.

5.2 FLUJOS DE TRABAJO

En los flujos de trabajo de esta fase se recopilan, analizan, diseñan, implementan y prueban solo los requisitos relevantes desde el punto de vista de la arquitectura. Lo que se implemente en esta fase servirá para probar lo que se diseña, pero puede que no sirva para construcción.

Los riesgos en esta fase no deben eliminarse sino reducirlos a un nivel aceptable para la fase de construcción

5.2.1 Flujo de trabajo de los requisitos

Se identificará y describirá en detalle el 80 por ciento de la totalidad de casos de uso, estableciendo prioridades y examinando de acuerdo a su importancia en la definición de la arquitectura.

Actividad: encontrar actores y casos de uso

Se identifican los casos de uso adicionales a los identificados en la fase de inicio, aunque en detalle se tratarán la totalidad, para que de esta manera el logro de los objetivos sea más satisfactorio.

En la etapa de inicio se identificaron algunos casos de uso y se describieron en forma general; ahora se volverán a analizarán en detalle descomponiéndolos o ampliándolos, para obtener los casos de uso que realmente reflejen los requisitos de la aplicación en desarrollo.

Se maneja el concepto de perfiles de usuario y se identifican correctamente los actores y sus actividades asociadas en pro de la obtención más correcta de los casos de uso.

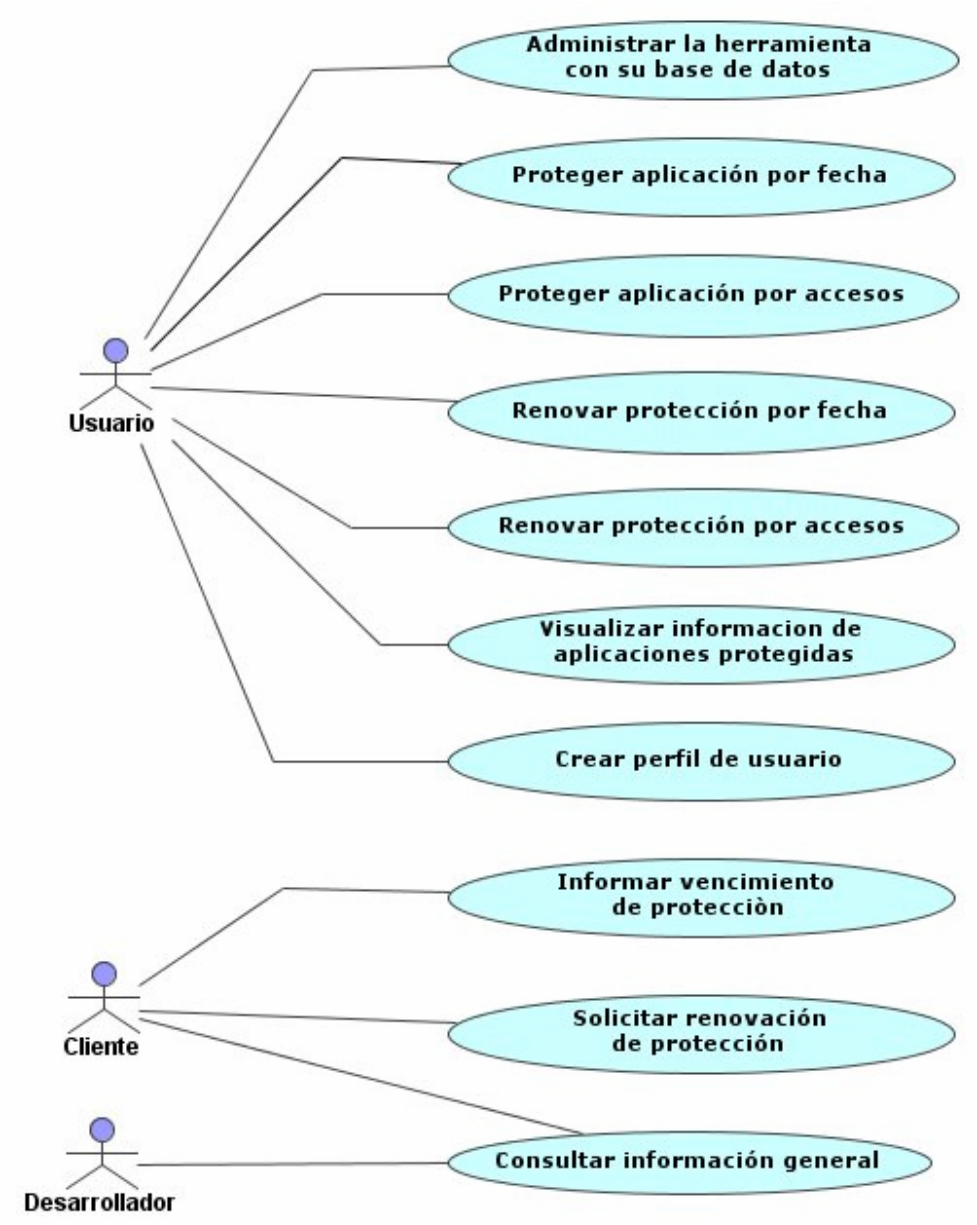


Figura 5.1 *Modelo de Casos de uso: diagrama General*

Descripción del Modelo General
El sistema permitirá dependiendo del perfil de usuario acceder a la totalidad de opciones que se presentan con previa validación de su clave identificadora.

El cliente de la aplicación solo puede consultar información referente a su aplicación protegida; es quien solicita renovación de uso y variación de información particular al usuario administrador. Usuario: persona o ente que interactúa con la herramienta y con sus funciones. Desarrollador: propietario moral de una aplicación, es quien ha desarrollado una aplicación protegida y en su calidad debe conocer acerca de sus aplicaciones; por ello tiene acceso a la opción de consulta no solo para conocer información particular sino información de aplicaciones de sus similares; Administrador: es un usuario privilegiado que tiene acceso a todas las opciones ofrecidas por la aplicación.

Tabla 5.1 *Tabla de descripción del modelo general*

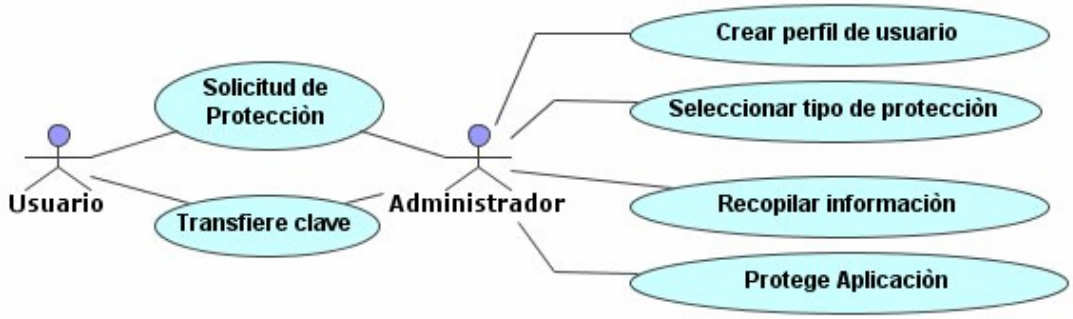


Figura 5.2 *Modelo de casos de Uso: diagrama de protección*

Descripción del Modelo Protección

El sistema cuenta con dos tipos de protecciones: protección por limite de tiempo en días, y número de accesos; también se ofrece la protección combinada de las anteriores. Adquirir el uso de la herramienta se obtiene realizando una solicitud formal o informal al administrador del sistema, si este último acepta la solicitud, se remitirá a recopilar cierta información de interés y proteger la aplicación, brindando al usuario la posibilidad de seleccionar el tipo de protección. Una vez realizada esta operación, el cliente lleva la aplicación ya protegida y al momento de la primera ejecución esta generará una clave de uso publico, la cual será comunicada al administrador o usuario encargado de la herramienta para que el le genere mediante la herramienta la clave privada que le permita usar su herramienta.

Tabla 5.2 *Tabla de descripción del modelo de protección*

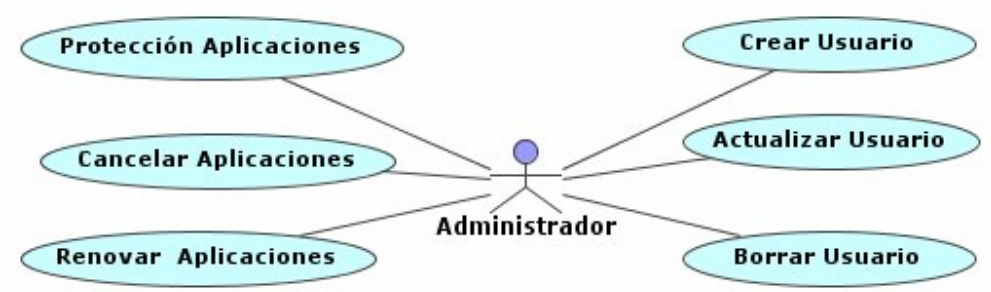


Figura 5.3 *Modelo de Casos de Uso. Diagrama de Administración*

Descripción del Modelo Administración
El administrador presenta diversas funciones que se pueden agrupar en: funciones de Usuario y funciones Operativas. La primera se refiere a las opciones para con el usuario del sistema como: crear, modificar y borrar, esta última opción se entiende como inactivar y no borrar físicamente, es decir si se desea conocer información de un determinado operario del sistema deberá estar en la capacidad de permitir

consultar información de este. La segunda se refiere a la parte operativa como:

1. **Protección.** Es una función exclusiva del administrador o delegada por este a un usuario de recibir, tramitar y responder las solicitudes de afiliación a la herramienta de protección.

2. **Cancelación.** En un momento dado un cliente puede solicitar renovación de uso, sin embargo el desarrollador de la aplicación previamente solicito no renovar la protección a un determinado cliente, por lo que se cancelará de manera temporal o definitiva la renovación de esta aplicación y se tendrá acceso a las opciones de consulta determinadas y configuradas anteriormente por el usuario o desarrollador de la aplicación.

3. **Consulta.** Se tiene acceso a todos los modos de consulta que ofrece la herramienta.

4. **Renovar.** Es el encargado de renovar licencia de uso a los clientes.

Tabla 5.3 *Tabla de descripción del modelo de administración*

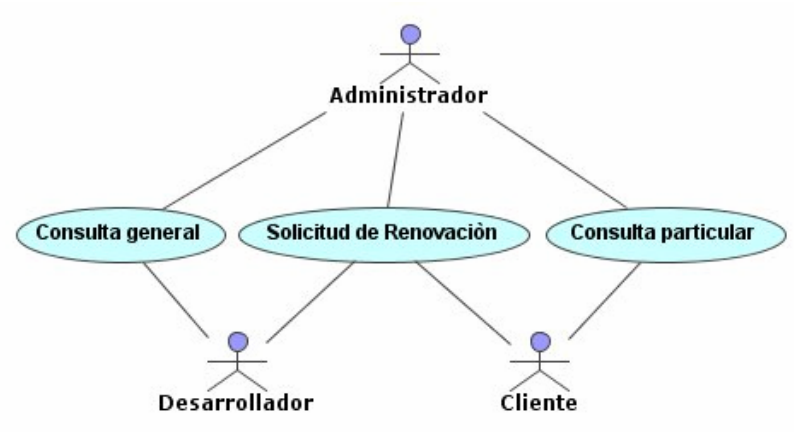


Figura 5.4 *Modelo de Casos de Uso. Diagrama de Consulta*

Descripción del Modelo Consulta

El cliente de la herramienta (quien usa la aplicación protegida) solo presenta dos opciones de consulta, identificando su proyecto por nombre o por código para acceder a la información referente a este; Esto no ocurre con el administrador a quien no se limita la consulta, sino por el contrario presenta todos los modos de consulta que ofrece la herramienta examinando por nombre o código a los usuarios, clientes, desarrolladores o aplicaciones protegidas.

Los actores que intervienen en el diagrama de consulta pueden tomar varios roles dependiendo de la utilización de la herramienta, por ejemplo si un desarrollador que pertenece a una empresa o institución educativa realiza un software para la misma, los derechos de autor en cuanto al derecho moral es de quien desarrolla la aplicación, sin embargo el derecho patrimonial pertenece a la entidad a la cual el desarrollador realizó la aplicación en calidad de empleado, profesor o estudiante, por lo que los derechos de ejecución pública esta a cargo de esta.

Tabla 5.4 *Tabla de descripción del modelo de consulta*

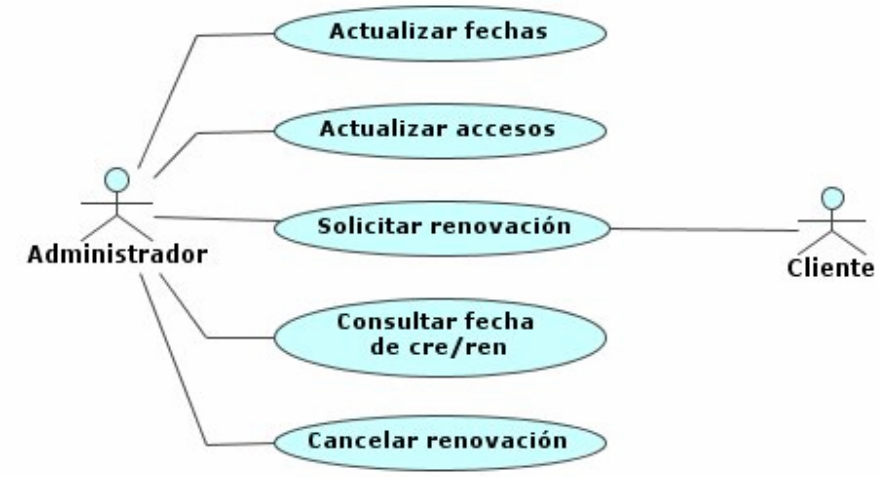


Figura 5.5 *Modelo de Casos de Uso: diagrama de renovación o actualización*

Descripción del Modelo Renovación o Actualización

El sistema permitirá a los clientes una vez finalice la fecha de protección de la aplicación, solicitar renovación de uso de la herramienta. El proceso consiste en digitar la clave pública (identificador de usuario) por parte del administrador, si esta coincide con la dada anteriormente en el momento de creación de la protección o en la renovación inmediatamente anterior, el sistema generará una nueva clave privada, con la cual se podrá continuar usando la herramienta. El proceso de renovación se puede considerar como un proceso Solicitud-Respuesta. Solicitud: cuando el cliente desea renovar y se pone en contacto con el administrador con su clave. Respuesta: cuando el sistema ha validado la clave, actualizado y renovado exitosamente una aplicación.

Tabla 5.5 *Tabla de descripción del modelo de administración*

Lo que se ha hecho hasta ahora es agrupar los requerimientos de acuerdo a su importancia y generando nuevos casos de uso, además de analizar en más detalle los casos de uso de la etapa de inicio, encontrando similitudes con los surgidos en esta fase, complementados y desglosados para de esta manera garantizar la mayor recopilación de requerimientos y la construcción de una línea de arquitectura sólida y flexible.

5.2.2 Flujo de trabajo del análisis

Durante la fase de inicio, se empezó a realizar un esquema del modelo de análisis, ahora este se refinará basado en la utilización de los casos de uso arquitectónicamente significativos para de esta manera comprender los detalles del sistema.

Las actividades que encierra el análisis de la arquitectura, estan en la identificación y análisis de los casos de uso significativos arquitectónicamente.

Actividad: análisis de la arquitectura

En la fase anterior, se desarrollo el análisis de la arquitectura solo hasta determinar que había una estructura factible; en esta etapa se extiende dicho análisis a un punto tal que pueda servir de base a una línea de la arquitectura ejecutable.

Para lograr este propósito se realiza una partición en paquetes de análisis representados en los casos de uso, identificando los paquetes específicos de la aplicación y los paquetes generales de la misma, luego se identifica con facilidad los paquetes de servicio y clases de análisis significativos arquitectónicamente y obvios.

Figura 5.6 *Paquetes de Análisis de Gestión de Consulta*

Figura 5.7 *Paquetes de Análisis de Gestión de Administración*

Este paquete está compuesto por gestión de usuario y gestión operativa, el primero de ellos describe las funciones del administrador para con el usuario y el segundo las funciones genéricas del administrador.

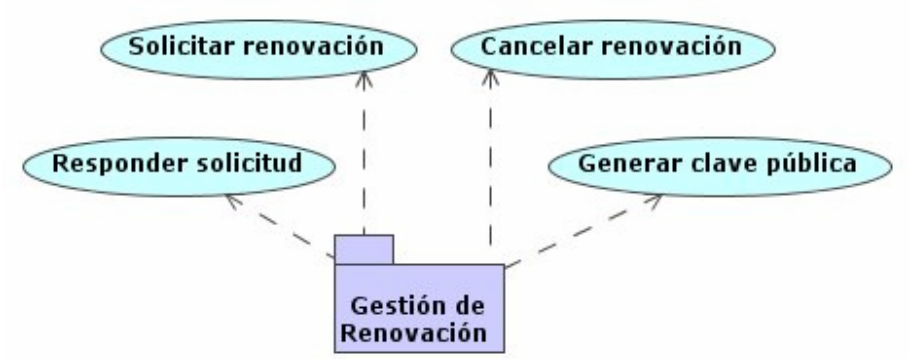


Figura 5.8 *Paquetes de Análisis de Gestión de Renovación*

Figura 5.9 *Paquetes de Análisis de Gestión de Operativa*

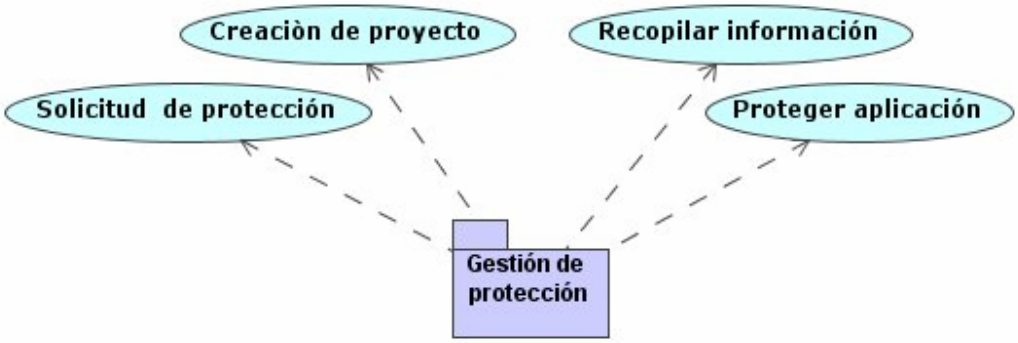


Figura 5.10 *Paquetes de Análisis de Gestión de Protección*

Actividad: análisis de casos de uso

Los casos de uso en ocasiones no son comprensibles tal y como están descritos en el modelo de casos de uso, por tanto debe ser refinados ya sea por su complejidad o por su significado en la arquitectura, analizando en detalle como clases de análisis en entidades, controles e interfaz y como se relacionan entre si para de esta manera llevar un flujo secuencial del caso de uso en análisis; para esto se ha de utilizar diagramas de colaboración que permitan una clara esquematización del proceso.

Paquete de gestión de protección

Para este caso se tienen dos entidades: tipo y modalidad. La primera contiene información sobre el tipo de protección de un determinado proyecto (por número de accesos, por tiempo o la combinación de las anteriores). La segunda (modalidad), se refiere a sí el proyecto es un trabajo de grado, proyecto semestral, software académico, etc.

La clase de interfaz IU de protección se encarga de permitir al usuario seleccionar de las opciones que se presentan de tipo de protección y modalidad, así como de mostrar información sobre el proyecto o aplicación protegida.

La clase de control Gestor de Protección se encarga de comprobar que el cliente anteriormente seleccionó un tipo y una modalidad, de no haberlo hecho no se permitirá la continuación del proceso de lo contrario almacenará en la correspondiente entidad esta información.

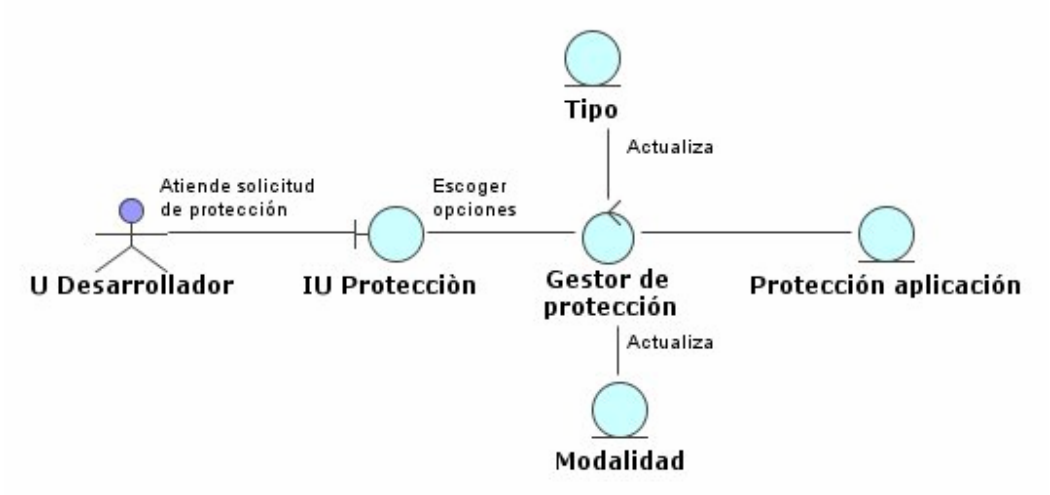


Figura 5.11 *Diagrama de Colaboración para el Caso de Uso Protección*

☑ Paquete de gestión de renovación

Cuando un cliente desee renovar su protección deberá realizar la solicitud al administrador, quien ingresará al módulo de renovación y digitará la clave pública del cliente, generada una vez expire la protección; esta clave será validada por el gestor de renovación que comparará esta, con la clave guardada en la entidad clientes, si efectivamente es correcta se presentarán las opciones de renovación en cuanto a tipo y de esta manera actualizar la información en las entidades y generación de la nueva clave privada de uso de la aplicación.

Figura 5.12 *Diagrama de Colaboración para el Caso de Uso Renovación*

5.2.3 Flujo de trabajo del diseño

En esta fase se modela el sistema y se encuentra su forma base para que de esta manera soporte todos los requisitos, incluyendo los requisitos no funcionales y otras restricciones.

Los objetivos esenciales con esta fase son: adquirir una comprensión en detalle de los aspectos relacionados con los requisitos no funcionales y restricciones de desarrollo en cuanto a lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de interfaz de usuario, entre otros; además de un punto de inicio para las actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.

En esta fase se diseñará e implementará los casos de uso, clases y subsistemas que arquitectónicamente son significativos. Los paquetes, durante el análisis, y los subsistemas, durante el diseño, son críticos para definir las vistas de la arquitectura.

Actividad: diseñar la arquitectura

Se diseña aquellos aspectos arquitectónicamente significativos del sistema, los cuales conforman el modelo de diseño que incluyen subsistemas, clases, interfaces y casos de uso importantes para la arquitectura que se busca.

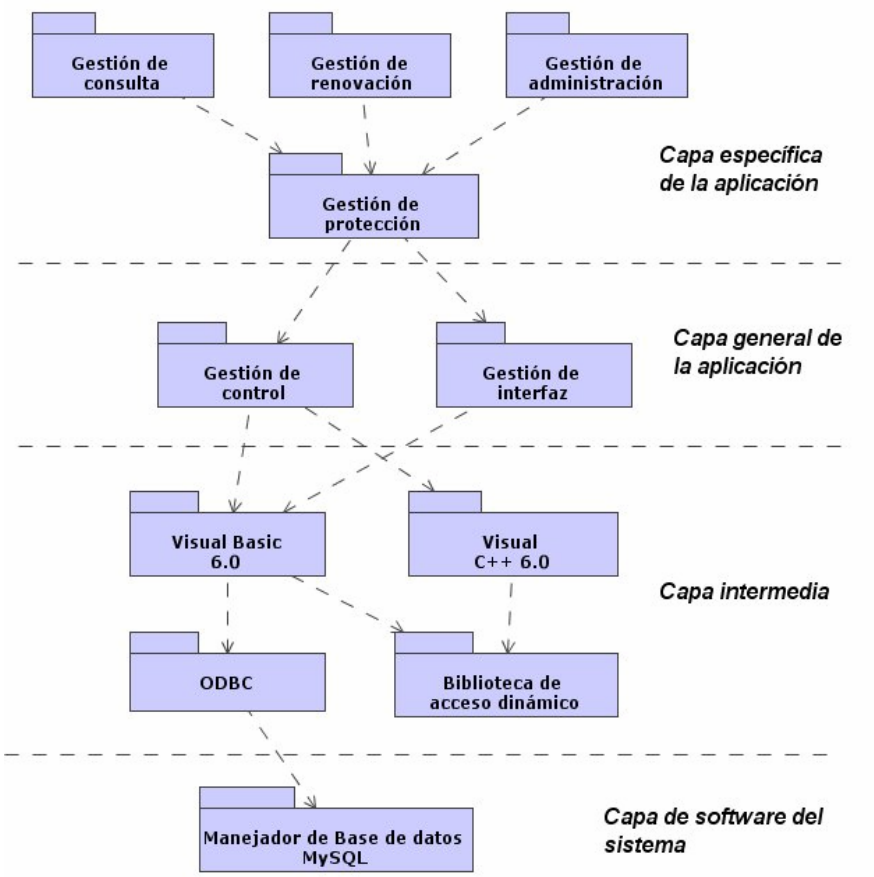


Figura 5.13 *Subsistemas distribuidos según capas de la arquitectura*

Actividad: diseñar clases

☑ Clases participantes en el caso de uso renovación

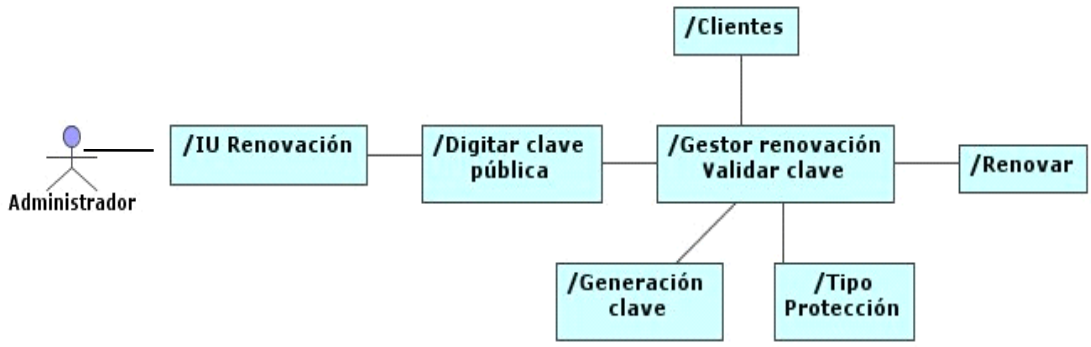


Figura 5.14 *Clases participantes en el caso de uso de renovación*

La **IU** protección, es la interfaz de la aplicación que solo esta disponible para el administrador del sistema; en el proceso de renovación comienza con la introducción de la clave publica del cliente que desea renovar. Esta clave es validada por parte del usuario-administrador confrontándola con la clave que se encuentra en la entidad cliente, si se comprueba su validez se activa la opción de renovación donde se presentan tres modos de protección que deben ser escogidos por el cliente. Luego de finalizar la selección, el gestor de protección generara una clave privada, con la cual el cliente podrá seguir usando la aplicación.

☑ Clases participantes en el caso de uso protección

El usuario de la herramienta solicita protección de una determinada aplicación realizando la solicitud al usuario-administrador, él cual analizara la petición y de aprobarse la misma se ingresará al sistema por medio de la **IU** protección; aquí ingresara información referente a la aplicación que será guardada en la base de datos para posteriores consultas y validación a la hora de renovar.

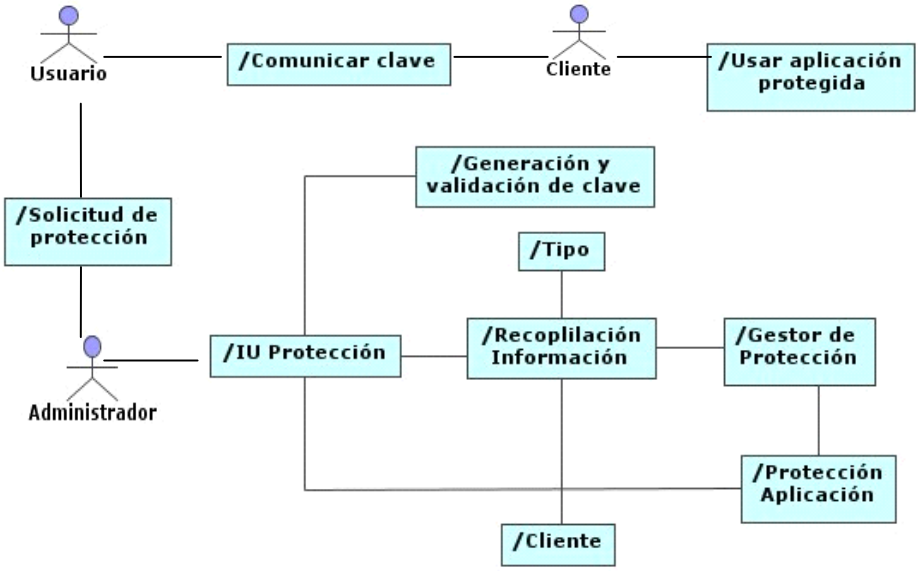


Figura 5.15 *Clases participantes en el caso de uso de protección*

Después de concluir la recopilación satisfactoria de la información se procederá a proteger la aplicación solicitada llamando a la biblioteca de vínculo dinámico protección, la cual cifrara la aplicación para finalizar el proceso de protección. Sin embargo para usar la aplicación protegida, se debe solicitar una clave de uso, la cual se obtiene ingresando nuevamente a la interfaz de protección e invocando la biblioteca de vínculo dinámico, por medio de la cual se genera una clave privada de uso de acuerdo al tipo de protección y al código asignado al proyecto; con esta clave se puede usar la aplicación protegida.

En primera instancia, la clave se le suministra al usuario (propietario de la aplicación) y este será responsable de comunicarla al cliente que usa la aplicación.

Clase participante en el caso de uso consulta

El proceso de consulta se inicia con la solicitud de consulta al administrador del sistema, quien es responsable de verificar el perfil del usuario y de acuerdo a esto, se activan las opciones de consulta asociadas al perfil; sin embargo el administrador negara la solicitud de consulta si encuentra que la persona que realiza la solicitud no es usuaria operativa[13] del sistema.

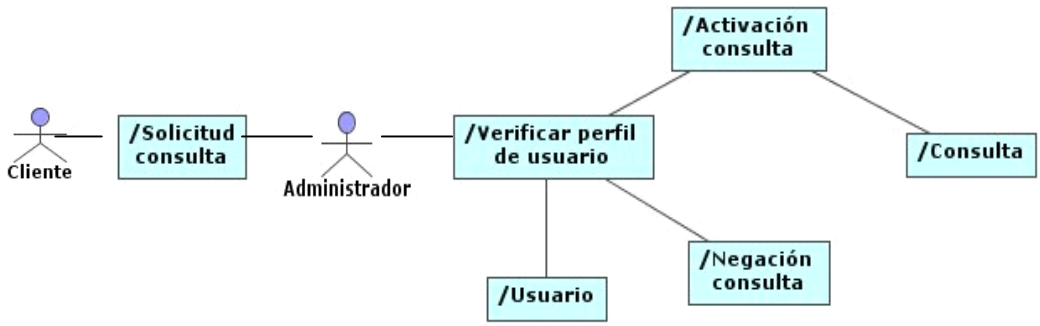


Figura 5.16 *Clases participantes en el caso de uso de consulta*

Actividad: diseñar un subsistema

[13] Persona que cuenta con un perfil de usuario en la aplicación.

Los subsistemas permiten organizar el modelo de diseño en partes significativas e identificables de modo que se puede trabajar independientemente en cuanto a diseño e implementación.

Los subsistemas son derivados de los paquetes de análisis identificados y analizados anteriormente, como son los subsistemas: actualización o renovación, administración y consulta, los cuales se denominan subsistemas de segundo orden, sumado a dos nuevos subsistemas: subsistema de gestión y subsistema de Gestión de Interfaz que corresponden a los subsistemas de primer orden.

El subsistema de Gestión se encarga de controlar el acceso a los demás subsistemas denominados de segundo orden, a los cuales para acceder se debe inicialmente estar en sesión o utilización de la aplicación, previa validación de clave de ingreso; el sistema maneja la administración de perfiles de usuario para restringir el acceso a los subsistemas de segundo nivel o orden; por su parte el subsistema de Gestión de interfaz es el encargado de controlar la consistencia, coherencia, diseño grafico de interfaces, colores, simbología, para que de esta manera contribuya a la utilización de la herramienta.

El subsistema de protección tiene una particularidad, la cual consiste en que solo esta activo cuando se va a utilizar la aplicación por primera vez, una vez solicitado el servicio y aprobado el mismo, el sistema generará una clave de uso para iniciar a utilizar la herramienta de protección y una vez allí dependiendo del perfil que le sea asignado por parte del administrador del sistema, acceder a los subsistemas que le sean permitidos.

☑ **Subsistema de gestión de control**

Las aplicaciones protegidas son reguladas por número de accesos o limitante de tiempo, por ello se hace necesario un mecanismo de control que permita la veracidad de este proceso; el proceso de consulta se enfrenta al problema de

autenticidad, por ello se implementa el control de acceso y de inicio de sesión, el cual solicita identificación de usuario o login respaldado de su correspondiente clave de acceso, una vez validado e identificado su perfil se remite a activar o restringir la totalidad de opciones de consulta.

En el caso del proceso de renovación es exclusivo del administrador del sistema quien actualizará la protección de una determinada aplicación a solicitud del cliente con su correspondiente clave asignada inicialmente o en la anterior renovación, la cual se validará y de inmediato se generará la nueva clave de uso, sin embargo el cliente deberá conocer la fecha de terminación o el número de accesos faltantes por esto se implementa un mecanismo que permita la visualización periódica de un mensaje que brinde tal información al cliente; los controles a estas situaciones y demás son unificados en el subsistema de gestión.

☑ **Subsistema de gestión de interfaz**

La interfaz es el medio de comunicación entre el usuario y el sistema; de esta depende el objetivo de la utilización múltiple.

Este subsistema es el encargado de todo lo relacionado con la interfaz de usuario y de que cumpla los requisitos básicos tales como: estándares de diseño, consistencia, coherencia, colorido, en pro de la satisfacción a los usuarios.

Cuando se habla de *consistencia* se refiere a una adecuada distribución de imágenes, texto y controles gráficos en cada una de las interfaces y opciones que componen el sistema.

Coherencia: el paso de una pantalla a otra debe ser acorde al trabajo que en ese momento se intente realizar, por ejemplo, si se escoge una opción se debe activar la interfaz que acompaña esta y no otra que no tenga relación alguna, o bien que la interfaz de la opción elegida no se preceda de interfaces que el usuario deba desactivar para llegar finalmente a la deseada.

No Sobrepoblación de Pantallas: la óptima división de controles gráficos permiten una buena comprensión en las pantallas, es decir en una sola interfaz no se acumula toda la información que se desea brindar sino que se divide en interfaces sucesivas y organizadas para la ilustración de la información de acuerdo a la opción elegida.

Control de Colores: si la gama de colores que componen la interfaz principal y las interfaces secundarias son muy intensas, será incomodo la utilización de la aplicación.

☑ **Subsistema de consulta**

Generador de información de aplicaciones, dependiendo del perfil o rol que posea una determinada persona estará disponible o no la totalidad de información que el sistema proporcione.

☑ **Subsistema de renovación**

Permite la renovación de uso de la herramienta, este proceso comienza una vez se acerca la expiración de la aplicación ya sea por número de accesos, limite de tiempo o las dos. El sistema periódicamente generará un mensaje de aviso al cliente, donde le proporcionará la información de su protección y una vez finalicé la licencia de uso generará una clave, con la cual el cliente deberá ponerse en contacto con el administrador quien continuará con el proceso actualizando el tipo de protección y generando la nueva clave de uso al cliente.

- ***Protección por accesos***: número de accesos que el cliente puede realizar a la aplicación.

- ***Protección por limite de tiempo***: número de días a partir de la fecha de renovación que el cliente cuenta para utilizar la herramienta de protección.

- ***Protección por accesos y tiempo***: protección alterna que se compone de las anteriores, sin embargo una vez expire cualquiera de las dos opciones de protección deberá renovarse la licencia de uso.

Estas opciones de renovación podrían ser violadas con facilidad, sin embargo se cuenta con controles que contrarrestan cualquier variación al número de accesos y a la fecha actual del sistema que realice una determinada persona.

☑ **Subsistema de protección**

Recopilador de información de usuarios del sistema como nombre, teléfono, empresa e información general de interés como: fecha de protección, fecha limite de uso, número de accesos, la cual posteriormente puede ser consulta ingresando al subsistema de consulta. La información recopilada allí solo puede ser variada por el administrador.

☑ ***Subsistema administración***

Administrador es la persona encargada de la regulación y control operativo de la aplicación; es quien tiene la exclusividad de acceso a todos los subsistemas de la aplicación, generador de perfiles de usuario, realiza la actualización de las licencias de uso. Por ello se considera el motor de la aplicación software.

Actividad: Diseño de subsistemas de renovación y consulta

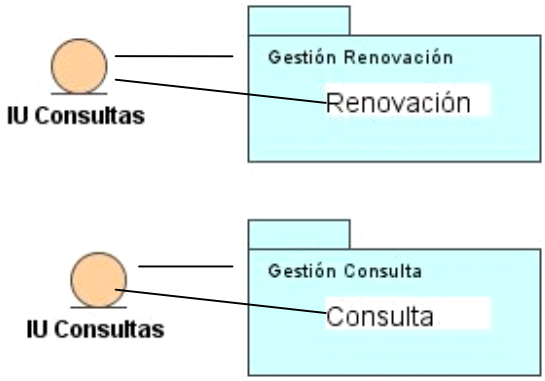


Figura 5.17 *Diseño de subsistema*

El subsistema de gestión de renovación proporciona la activación de la opción renovar en la interfaz de la aplicación. También brinda la interfaz de consulta perteneciente a la gestión de consulta.

5.2.4 Flujo de trabajo de la implementación

El objetivo fundamental de la implementación es desarrollar la arquitectura y el sistema como un todo, planificando y distribuyendo el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue, implementando clases y subsistemas encontrados durante el diseño.

Actividad: implementación de una clase e implementación de un

subsistema

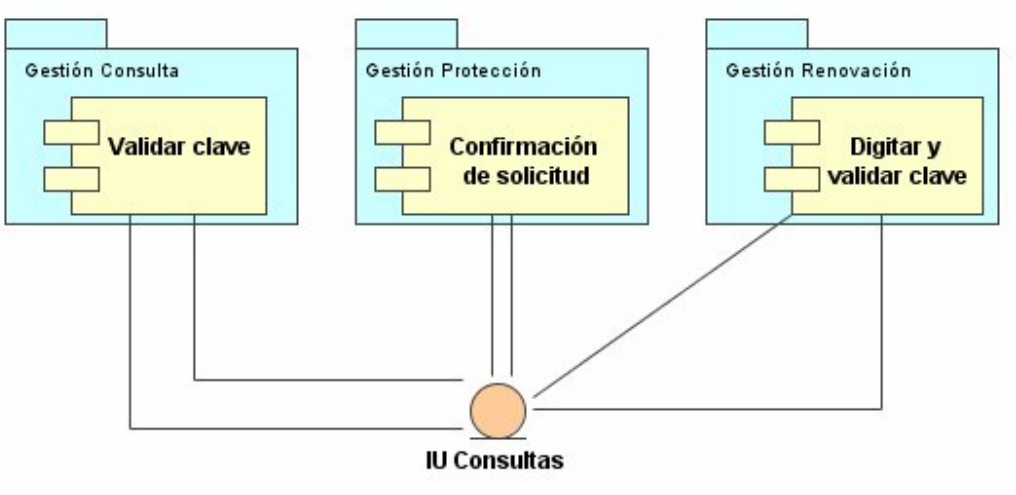


Figura 5.18 *Implementación de una clase y un subsistema*

Los subsistemas consulta, protección y renovación necesitan proporcionar la interfaz de la aplicación para realizar sus procesos; cada uno de estos con previas validaciones y confirmaciones, para de esta manera continuar la secuencia de procesos.

5.3 EVALUACIÓN DE LA FASE DE ELABORACIÓN

La evaluación debe centrarse en la determinación de la línea base de la arquitectura.

Para obtener la arquitectura se recopilará la mayoría de casos de uso y se analizan aquellos más significativos desde el punto de vista de la arquitectura. Estos son: renovación, protección, consulta y administración; los cuales junto a los subsistemas identificados y detallados y sus clases componentes permitieron la obtención del modelo de diseño: base de la arquitectura.

A continuación se mostrará una lista de casos de uso y de subsistemas que fueron analizados y diseñados.

Casos de Uso	Vista	Comentario
Renovación	Figura 5.5	Presenta los actores y casos de uso significativos
Protección	Figura 5.2	
Consulta	Figura 5.4	
Administración	Figura 5.3	

Tabla 5.6 *Casos de uso analizados*

Subsistema	Breve descripción
Gestión de Control	Control de inicio de sesión
Gestión de Interfaz	Control de interfaz
Gestión de Administración	Funciones y accesibilidad del usuario-administrador
Gestión de Protección	Control de Protección
Gestión de Renovación	Control de Renovación
Gestión de Consulta	Control de Consulta

Tabla 5.7 *Subsistemas analizados*

5.4 PLANEACIÓN DE LA FASE DE CONSTRUCCIÓN

Es importante comenzar a planificar de forma detallada la fase de construcción y determinar el número de iteraciones necesarias, si es el caso. Para el desarrollo del software a implementar, en cada fase realizada se han analizado en detalle la mayoría de los aspectos que define el proceso unificado; por ello se considera que si se continúa con esta línea de desarrollo bastara con una sola iteración para obtener la versión operativa del sistema.

Al final de la fase de construcción se debe haber identificado el 100% de los casos de uso del sistema debidamente detallado, realizado pruebas de unidad con

resultados satisfactorios, mitigados los riesgos identificados en esta fase y en las anteriores; y finalmente la versión terminada del sistema.

CAPITULO 6

FASE DE CONSTRUCCIÓN

El objetivo más importante de esta fase es enfatizar la implementación y las pruebas del software. Se espera obtener una versión de la herramienta que cumpla con los requerimientos establecidos al comienzo del proyecto. Existen algunos documentos que deben ir de la mano con la construcción tal como el manual de usuario y la documentación de la implementación, los riesgos deben estar casi mitigados en esta fase.

6.1 Flujos de trabajo

Los flujos de trabajo se centran en la construcción del sistema, es decir en completar la realización de los casos de uso, implementar los subsistemas y clases como componentes que será el producto de un desarrollo iterativo y guiado por los casos de uso. Las pruebas toman importancia ya que en las fases anteriores solo se implementaban para probar los casos de uso y en esta fase deben estar encaminadas hacia el correcto funcionamiento de la versión ejecutable.

6.1.1 El flujo de trabajo de los requisitos

Actividad: Desarrollo de prototipos de interfaz

Los prototipos de interfaz de usuario nos ayudan a comprender y especificar las interacciones entre actores humanos y el sistema durante la captura de requisitos.

☑ **Interfaz de prueba de los procesos de protección y renovación**

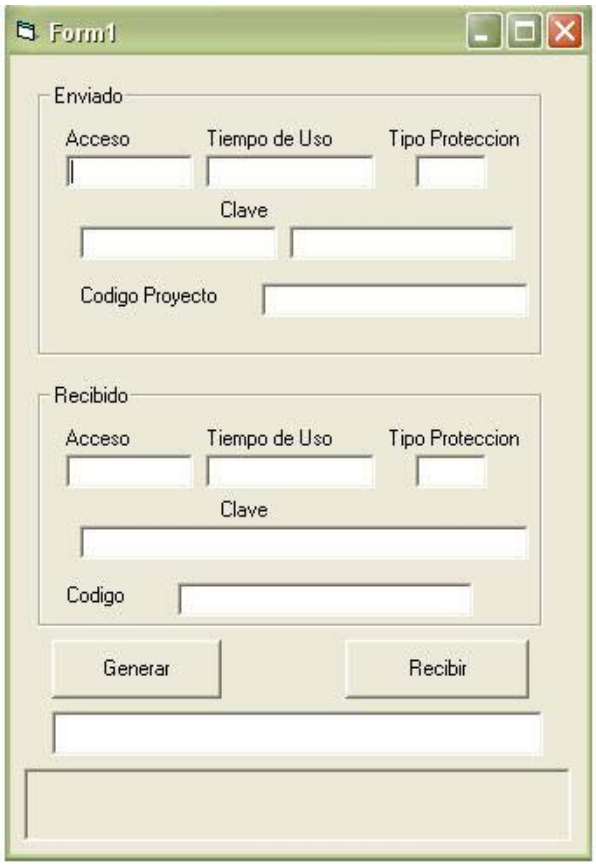


Figura 6.1 *Primer prototipo de la interfaz*

☑ **Interfaz final de la aplicación**

Figura 6.2 *Prototipo final de la interfaz*

Actividad: Determinar la prioridad de los casos de uso

Orden de prioridad	*Caso de Uso*	*Riesgo*	*Impacto*
1	Protección	Identificación de clases y subsistemas	No permitirá determinar una arquitectura correcta
2	Renovación	Disponible Usuario-Administrador	Desmotivación hacia el uso de la herramienta por parte del cliente
3	Consulta	Confiabilidad de la base de datos	Imposibilidad de renovación y consulta
4	Administración	Integridad del sistema de acuerdo al perfil de usuario	Deterioro de la confiabilidad de la herramienta

Tabla 6.1 *Prioridad de los casos de uso*

6.1.2 El flujo de trabajo del análisis

El objetivo de esta fase es obtener un producto software en su versión operativa inicial. Este debe contar con ciertas características en cuanto a calidad y aseguramiento de requisitos.

Actividad: Analizar un caso de uso

Se identifica y detalla el cien por ciento (100%) de requisitos en casos de uso, de manera que se busca satisfacer los requerimientos funcionales trazados al inicio del proceso de desarrollo y durante su evolución.

☑ **Caso de uso protección**

Descripción: El desarrollador solicita la protección al administrador que a su vez puede realizar las siguientes actividades:

- Recopilar información del usuario.
- Crear perfil de usuario.
- Seleccionar el tipo de protección.
- Proteger la aplicación.

Luego el administrador le transfiere la clave de uso directamente al cliente o por medio del desarrollador si este es el intermediario.

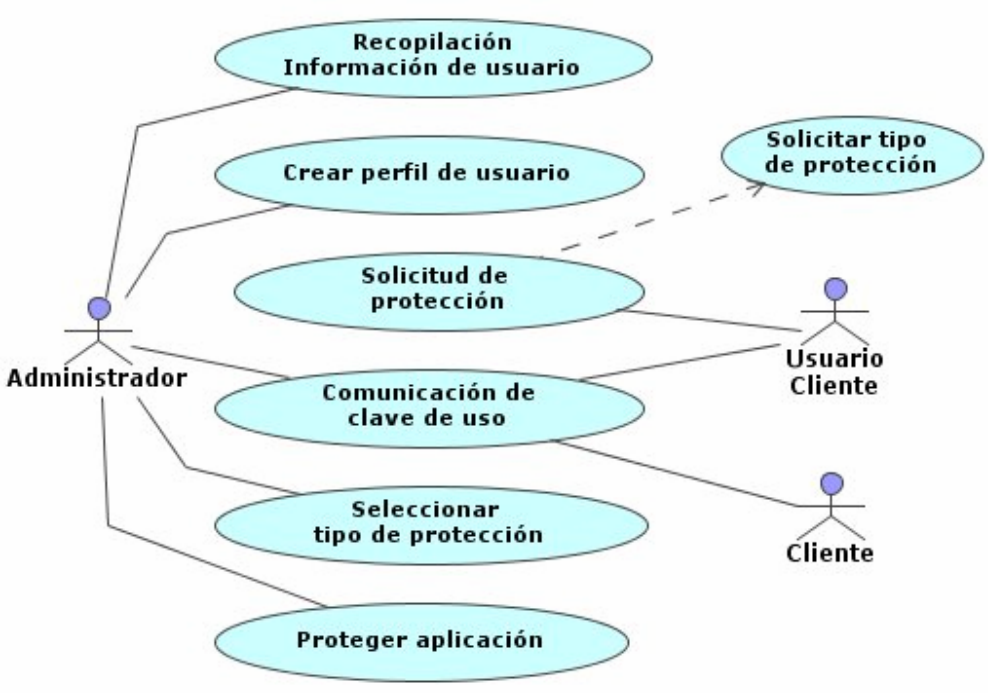


Figura 6.3 *Caso de uso protección*

☑ **Caso de uso renovación**

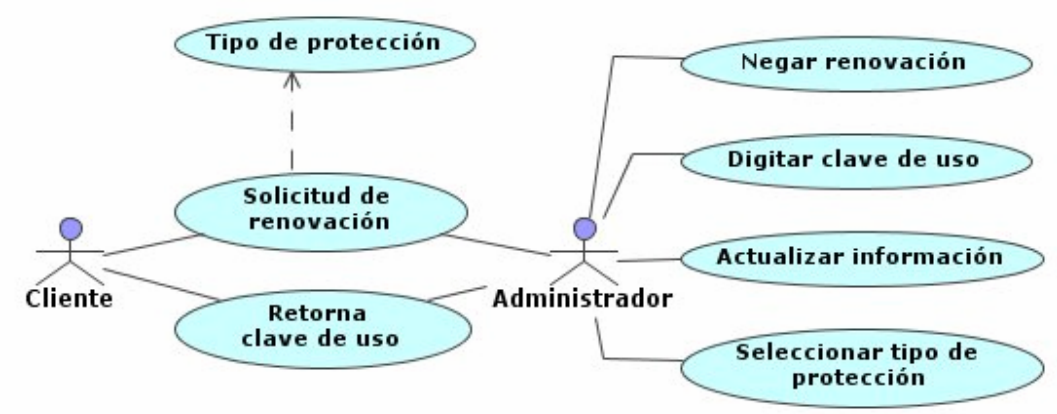


Figura 6.4 *Caso de uso renovación*

El cliente solicita la renovación. Se utiliza el caso de uso tipo-protección, cuando la aplicación ha sido corroborada, el administrador dispone de ciertas acciones que puede realizar como lo es actualizar la información, que puede ser referente al cliente o al desarrollador. El administrador devuelve la clave privada al cliente para que este la use.

☑ Caso de uso consulta

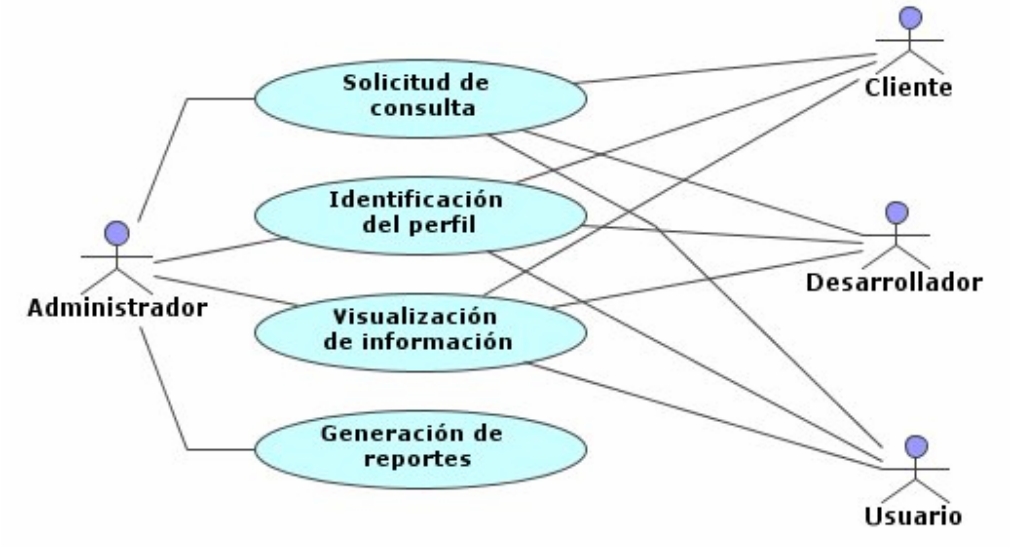


Figura 6.5 *Caso de uso consulta*

Descripción: Las consultas pueden ser realizadas por el cliente, usuario, desarrollador o administrador; sin embargo el administrador puede tener acceso a reportes de los proyectos protegidos.

6.1.3 El flujo de trabajo del diseño

En este flujo de trabajo nos centraremos en los casos de uso que no fueron utilizados para la obtención de la línea base de la arquitectura, teniendo en cuenta que en la fase anterior se diseñaron los casos de uso arquitectónicamente significativos como son: protección, consulta y renovación.
Se deja al margen el caso de uso administración ya que la prioridad que presenta es baja.

Actividad: representar un caso de uso en clases participantes

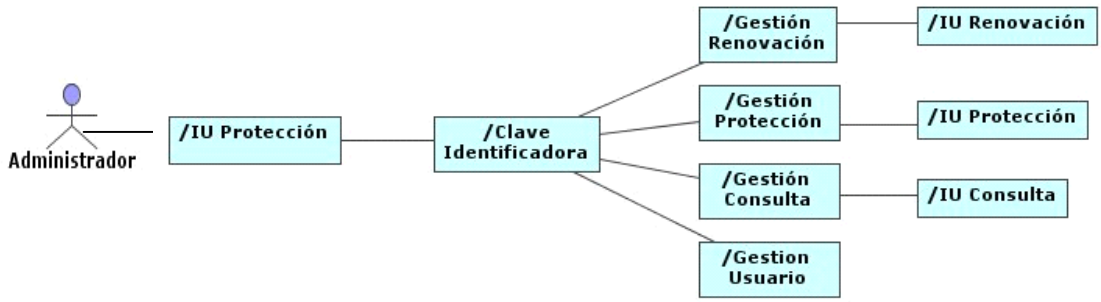


Figura 6.6 *Representación de las clases principales*

El administrador interactúa con la IU protección y utilizando su clave de identificación tiene acceso a la gestión de renovación que se activa cuando un proyecto ya ha sido creado y ha caducado. El gestor de protección se usa para proyectos nuevos. El proceso de consulta del administrador involucra los reportes y las consultas. El gestor de usuarios permite crear o eliminar usuarios.

6.1.4 El flujo de trabajo de la implementación

Este flujo representa la versión operativa inicial de la aplicación, la cual recopila el 100% de los casos donde se realiza la interacción de todos los componentes analizados en las fases anteriores.

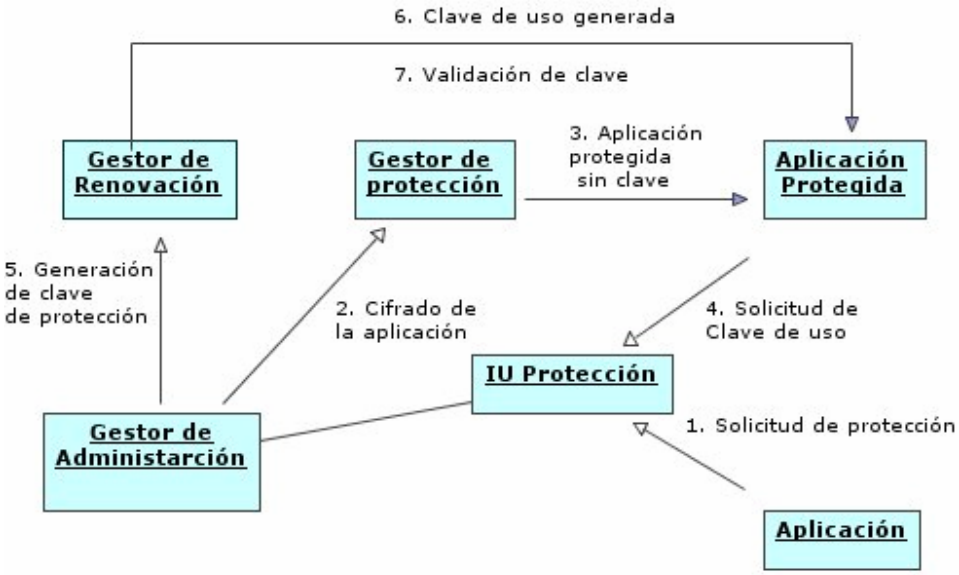


Figura 6.7 *Diagrama de colaboración de clases del sistema*

En el anterior diagrama se refleja la situación que se presenta cuando un usuario cliente solicita la protección de una aplicación. Cuando esto ocurre, se comunica con la IU protección, la cual le sirve de interfaz para establecer contacto con el gestor de administración. La aplicación es protegida por el gestor de protección, aunque en este punto aun no se cuenta con clave para acceder a la aplicación protegida. El cliente debe solicitar la clave para poder acceder a la aplicación, lo cual es realizado por el gestor de renovación quien proporciona la clave que será utilizada por el cliente.

Cuando el cliente aplica la clave proporcionada, es necesaria la validación de esta para poder acceder a la aplicación protegida.

Actividad: Implementar una clase e implementar un subsistema

Las clases se agrupan en sus correspondientes subsistemas. A continuación se implementará y detallará cada una de las clases de los subsistemas de gran importancia en la aplicación.

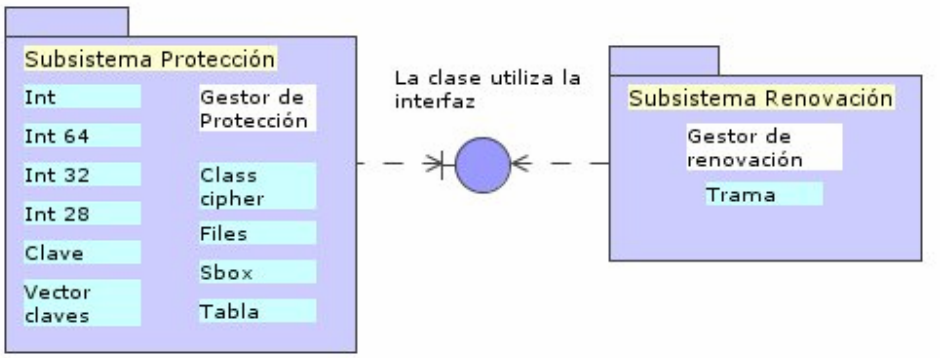


Figura 6.8 *Implementación de clases y subsistemas*

La línea de trazo discontinuo de una clase a una interfaz significa que la clase usa la interfaz; esta interfaz permite la asociación de los subsistemas.

6.1.4.1 Clases del algoritmo DES

Las clases en donde se encuentra el algoritmo DES implementado están contenidas en la librería de vinculo dinámico llamada desfin.dll. Estas clases están agrupadas en el subsistema de protección.

6.1.4.1.1 Clases para el manejo de los datos

El algoritmo DES contiene operaciones de sustitución y de transposición, cada una ellas fue desarrollada en diferentes tipos de clases con el fin de diferenciar al momento de manipular el código fuente.

6.1.4.1.1.1 Datos numéricos

En el siguiente código se encuentran implementada la definición de los diferentes tipos de objetos necesarios para la implementación del algoritmo DES.

```
class Int {
  public:
  virtual void writeBit(int nbit, bool valor) = 0;
  virtual bool readBit(int nbit) = 0;
  virtual Int* clone() = 0;
  void permutation(Tabla tabla);
};
```

Creada como una clase modelo que servirá para definir os tipos que se utilizan en el cifrado y descifrado del algoritmo DES.

```
class Int64: public Int {
  protected:
  int64 entero;

  public:
  void setInt(int64 ent){entero = ent;}
  int64 getInt(){return entero;}
  void writeBit(int nbit, bool valor);
  bool readBit(int nbit);
  Int* clone();
  Int32 get32Left();
  Int32 get32Right();
  static Int64 concatenate(Int32 high, Int32 low);
  static Int64 xor(Int64 a, Int64 b);
  void xor(Int64 a);
};
```

Clase creada para la clave y el texto a procesar.

Esta clase hereda del Int.

Se encapsulan las funciones básicas como lo es escribir, leer, asignar y operaciones lógicas

Clase para realizar las operaciones internas del DES.

Hereda de la clase Int.

Con esta clase se soluciona el problema que se presenta en el

```
class Int32: public Int{
  protected:
  int32 entero;

  public:
  void setInt(int32 ent){entero = ent;}
  int32 getInt(){return entero;}
  void writeBit(int nbit, bool valor);
  bool readBit(int nbit);
  Int* clone();
  static Int32 xor(Int32 a, Int32 b);
  void xor(Int32 a);
};
```

```
class Int28: public Int32 {

  public:

  void setInt(int32 ent){entero = ent; clean();}

  void clean(){entero &= 0x0FFFFFFF;}

  void lShift(int desp);
};
```

Clase para realizar las operaciones internas del DES.

Hereda de la clase Int32.

Cuenta con funciones como clean que limpia los bits sobrantes (más allá de los 28 de interés).

```
class Clave: public Int64 {

  public:

  Int28 get28Low();

  Int28 get28High();

  static Clave concatenate(Int28 high, Int28 low);
```

Clase para el manejo de la Clave del algoritmo DES.

Hereda de la clase Int64.

Cuenta con funciones que permiten obtener la parte alta y baja de las tramas analizadas. También permite concatenar cadenas (unir parte alta con la parte baja)

```
};
```

6.1.4.1.1.2 Tablas de datos.

Para aplicar el algoritmo DES, se deben realizar transformaciones y permutaciones con base a información consignadas en tablas predefinidas por el estándar DES.

Clase para alojar el vector de claves requerido en el proceso de cifrado y descifrado.

Se cuenta con funciones para el manejo de los datos almacenados en el interior del vector.

```
class VectorClaves {
  Clave vector[NCLAVES];

  public:
  void setItem(int pos, Clave clave){vector[pos]=clave;}
  Clave getItem(int pos){return vector[pos];}
};
```

Clase implementada para la creación, asignación y manejo de las tablas utilizadas en el DES

Se cuenta con funciones para el manejo de los datos almacenados en el interior de las tablas.

```
class Tabla {
  int *tabla;
  int tamano;
  public:
  Tabla(){tabla = NULL; tamano = 0;}
  Tabla(int *vector, int tam){set(vector, tam);};

  void set(int *vector, int tam);
  int getItem(int i);
  int getLength() {return tamano;}
};
```

Clase implementada para la creación, asignación y manejo de la tabla Sbox.

Se hace necesario implementar esta clase independiente de la clase tabla por la complejidad que reviste el tamaño.

```
class Sbox {
  static int32 tabla[8][4][16];

   public:
    int32 getItem(int i, int j, int k){ return (tabla[i][j][k]);}
```

```
  Int32 sustitute(Int64 data);
};
```

6.1.4.1.1.3 Clase de Agrupación

Para obtener el resultado final del algoritmo DES, se implementa la siguiente clase que utiliza todas las clases descritas anteriormente.

Clase que encapsula todas las funciones necesarias para el proceso de cifrado y descifrado.
Lo que se necesita ya esta implementado como clases, por lo cual se deben crear objetos de dichas clases e invocar las funciones.

```
class Cipher {
 static int vectorPc1[];
 static int vectorPc2[];
 static int vectorLs[];
 static int vectorIp[];
 static int vectorIpInv[];
 static int vectorBitSel[];
 static int vectorP[];
 Tabla pc1;
 Tabla pc2;
 Tabla ls;
 Tabla ip;
 Tabla ipInv;
 Tabla bitSel;
 Tabla p;
 VectorClaves keys;
 Sbox sbox;
 void generateKeyVec(Clave key, VectorClaves *keyVec);
 Int32 function(Int32 a, Clave key);
 Int64 cipher(Int64 input, VectorClaves *keyVec);
 Int64 decipher(Int64 input, VectorClaves *keyVec);
 public:
 Cipher() {
  pc1.set(vectorPc1, 56);
  pc2.set(vectorPc2, 48);
  ls.set(vectorLs, 16);
```

```
    ip.set(vectorIp, 64);
    ipInv.set(vectorIpInv, 64);
    bitSel.set(vectorBitSel, 48);
    p.set(vectorP, 32);
  }
  void generateKeys(Clave key);
  Int64 cipher(Int64 input);
  Int64 decipher(Int64 input);
};
```

6.1.4.2 Clase para el manejo de Tramas

Clase utilizada para armar las tramas que serán mostradas en el momento de vencimiento y de renovación.

También se utiliza para desarmar las tramas y poder obtener la información que se almacena internamente.

Los atributos char son utilizados como variables auxiliares para el proceso de armado y desarmado.*

```
class Trama {
        ulong parte1;
        int64 parte2;
        ulong parte3;
        ulong parte4;
        char *aux_trama;
        char *trama_1;
        char *trama_2;
        char *trama_21;
        char *trama_22;
        char *trama_3;
        char *trama_4;
        char *trama;
        char *cero;
        char* Armar_trama(ulong tipo_p, ulong tiempo, int64 clave, ulong acceso, ulong codigo);
        void Desarmar_trama(char *trama,ulong &tipo_p, ulong &tiempo, int64 &clave, ulong &acceso,
        ulong &codigo);
};
```

6.1.4.3 Clases para el manejo de archivos

Para poder realizar un manejo eficiente de la fecha, se crea la siguiente estructura de datos:

```
typedef struct {
        int anno;
        int mes;
        int dia;
        int hora;
        int minuto;
        int segundo;
}Fecha_Date;
```

Clase implementada para la creación, protección y manejo de archivos.

```
class File {
 public:
 int handle;
 int64 marca;
 int64 marca1;
 int64 marca2;
 int64 marca3;
 int64 marca4;
 int64 marca5;

 void CifrarFile(int handle_1, int handle_2);
 void DescifrarFile(int handle_1, int handle_2, int64 longitud);
 void ArmarFile(int handle_1,int handle_2,int handle_3,int acceso, int tiempo,int tipo);
 void DesarmarFile(int handle_1, int handle_2);
 void Ubicar_punto(int handle, int64 patron);
 void Ubicar_principio(int handle_1);
 void Ubicar_fin(int handle_1);
 bool Localizar(int64 info, int64 patron);
 int64 Establecer_marca(int64 key);
 void Colocar_fecha(int handle,Fecha_Date fecha,int64 patron);
 Fecha_Date Consultar_fecha(int handle,int64 patron);
 void Colocar_acceso(int handle, int64 acceso);
 int64 Consultar_acceso(int handle);
```

La clase File cuenta con funciones que le permiten localizar información que se encuentra dentro de los archivos protegidos.

```
  int64 Consultar_tamano(int handle);
  void ModificarTipoProteccion(int handle, int tipo);
  int ConsultarTipoProteccion(int handle);
};
```

6.2 PRUEBAS DE IMPLEMENTACIÓN

En esta subsección se realizaran pruebas a los subsistemas para comprobar el funcionamiento correcto del sistema. Para esto previamente la base de datos se ha poblado con datos ficticios.

Subsistema de Gestión Control

Estado del Sistema		
El Administrador no ha accedido al sistema, por lo cual no se ha inicializado la variable de sesión.		
Parámetros		
Nombre	**Valor**	**Descripción**
----------	----------	----------

Tabla 6.2. *Prueba I al subsistema de Gestión Control*

Prueba de validación de datos

Caso 1 Datos validos.

Datos de Entrada		
Datos necesarios para validarse en el sistema		
Nombre	**Valor**	**Descripción**
Login Usuario	Administrador	Identificación valida
Password Usuario	xxxxxxxx	Clave de acceso valida

Tabla 6.3. *Caso 1. Datos validos*

Caso 1.1 Datos validos para usuario-Administrador.

Datos de Entrada		
Accediendo al modulo Renovación		
Nombre	**Valor**	**Descripción**
Opción del sistema	Renovación	Acceso permitido al administrador para actualizar una determinada aplicación

Tabla 6.4. *Caso 1.1 Datos válidos para Usuario-Administrador*

Caso 1.2 Datos validos para usuarios.

No puede acceder directamente a los módulos del sistema. Se debe solicitar al administrador la realización de los procesos de protección, consulta y renovación.

Caso 2 Datos inválidos.

Datos de Entrada		
Datos para realizar la validación de acceso		
Nombre	**Valor**	**Descripción**
Login Usuario	Administrador	Identificación valida
Password Usuario	xxx	Clave invalida. La longitud debe ser mayor de 4 caracteres

Tabla 6.5 *Caso 2. Datos inválidos*

Estado del Sistema		
Inicio de sesión. Se ha accedido al sistema.		
Parámetros		
Nombre	**Valor**	**Descripción**

Identificador de perfil	1	identificador de perfil correspondiente al usuario administrador
Identificación de perfil	2	Identificador de perfil para el actual usuario que corresponde al usuario del sistema

Tabla 6.6 *Prueba II al subsistema de Gestión Control*

- **Caso 2.1 Datos Inválidos para el Administrador**

  En ningún caso presenta restricción de acceso al total de opciones que presenta el sistema.

- **Caso 2.2 Datos Inválidos para el Usuarios**

Datos de Entrada		
Accediendo al modulo Renovación		
Nombre	**Valor**	**Descripción**
Opción del sistema	Renovación	Proceso no permitido para el usuario. Exclusivo del administrador, quien lo realiza previa solicitud del usuario.

Tabla 6.7. *Caso 2.2 Datos inválidos para Usuarios*

- **Subsistema de Gestión de Interfaz**

Estado del Sistema		
El usuario ya ha accedido al sistema		
Parámetros		
Nombre	**Valor**	**Descripción**
Identificador de perfil ID_Usuario	2	Identificador de perfil de usuario correspondiente a un usuario del sistema

Tabla 6.8. *Prueba I al subsistema de Gestión de Interfaz*

☑ **Interfaz para una zona determinada**

Estado del Sistema		
El usuario accede al sistema a consultar información		
Nombre	**Valor**	**Descripción**
Indice Zona actual: TabStrip.SelectedItem.Index	Case 5	Indice que define la zona actual: 5 que corresponde a la opción Consultar
Resultado: Datos de Salida		
Información	**Valor esperado**	**Valor obtenido**
Zonas a las que tiene acceso	boton_Nuevo boton_Guardar boton_Proteger boton_Renovar boton_Ayuda boton_Acerca de boton_Salir 0. Proyecto 1. Cliente 2. Desarrollador 3. Aplicación a proteger 4. Reporte 5. Consulta	No tiene acceso No tiene acceso No tiene acceso No tiene acceso boton_Ayuda boton_Acerca de boton_Salir 0. Proyecto 1. Cliente 2. Desarrollador No tiene acceso No tiene acceso No tiene acceso 5. Consulta

Tabla 6.9. *Prueba I de la interfaz*

Subsistema renovación

Estado del Sistema		
El usuario-administrador ha accedido al sistema.		
Parámetros		
Nombre	**Valor**	**Descripción**
ID_Usuario	1	identificador de perfil correspondiente al usuario-administrador

Tabla 6.10. *Acceso al subsistema de Renovación*

☑ **Caso 1. Renovar una protección**

Datos de Entrada			
Datos del cliente que ha solicitado renovación de una determinada aplicación necesarios para continuar con el proceso de renovación.			
Nombre	**Valor Predeterminados**	**Valores Obtenidos**	**Observación**
Trama *1 | 2 | 3 | 4 | 5* ***1.Tipo de protección*** ***2. Acceso*** ***3. Calve publica*** ***4. Tiempo*** ***5. Código proyecto***	32868T-24512-3A000-52754-1B000-50A00-20040-00100	------	Trama generada por el sistema una vez expire la protección de una aplicación.
Tipo de protección	3	49302T-51245-7A000-95684-5B000-80A00-20040-00101	Selección de tipo de protección. Acceso y tiempo que desea el cliente para actualizar su protección. Una vez realizado lo anterior se obtiene la clave privada de uso de la aplicación renovada.

Tabla 6.11. *Renovar una protección*

☑ **Caso 2. Validación de datos**

Datos de Entrada		
Datos necesarios para realizar el proceso de renovación		
Nombre	**Valor**	**Observación**
Trama	49252T-12452-1A000-51246-	Clave invalida, no correspondiente a la longitud predeterminada: 41 caracteres. Además las letras que componen las claves son marcas de

	2B00C-50A00-20040-00	clase que han sido predeterminadas por tanto la trama solo estará compuesta por 4 letras a lo máximo
Resultado de salida		Ninguno. Debido a que no se pudo realizar el proceso de renovación.

Tabla 6.12. *Caso 2 Validación de datos*

Subsistema de Consulta

Estado del Sistema		
Un usuario accedió al sistema. Se ha iniciado sesión.		
Parámetros		
Nombre	**Valor**	**Descripción**
ID_Usuario	3	Identificador de perfil correspondiente al cliente de la aplicación

Tabla 6.13. *Acceso al sistema como cliente*

Caso 1. Consultar.

Datos de entrada		
Identificación de usuario de acuerdo al perfil que se maneja. De esta manera se activan las opciones de consulta.		
Parámetros		
Nombre	**Valor**	**Descripción**
Consulta particular	Frame3(5)	El cliente solo puede consultar información referente a la aplicación protegida que usa.

Tabla 6.14. *Caso1 Consultar*

Subsistema de Protección

Crear Protección

<table>
<tr><th colspan="3">Datos de Entrada</th></tr>
<tr><td colspan="3">Datos generales de la aplicación tales como: Desarrolladores, clientes (quienes usan las herramientas) y quien solicita protección (Usuario)</td></tr>
<tr><td>Nombre</td><td>Valor</td><td>Descripción</td></tr>
<tr><td>nombre_proy</td><td>proy_1</td><td rowspan="10">Recopilación de información referente a la aplicación a proteger.</td></tr>
<tr><td>modalidad_proy</td><td>1</td></tr>
<tr><td>nombre_cliente</td><td>cliente_1</td></tr>
<tr><td>dir_cliente</td><td>dir_1</td></tr>
<tr><td>tel_cliente</td><td>tel_1</td></tr>
<tr><td>correo_cliente</td><td>correo_1</td></tr>
<tr><td>nom_desarrollador</td><td>desarrollador_1</td></tr>
<tr><td>correo_desa</td><td>correo_des_1</td></tr>
<tr><td>inst_desa</td><td>institución_1</td></tr>
<tr><td colspan="3">Resultado recopilación. Se ofrece el tipo de protección, acceso y tiempo que se desea proteger.</td></tr>
<tr><td>Nombre</td><td>Valor</td><td>Descripción</td></tr>
<tr><td>ID_Proyecto</td><td>1</td><td>Identificador de proyecto</td></tr>
<tr><td>Fecha_crea_proy</td><td>05/03/2004</td><td>Fecha de creación del proyecto</td></tr>
<tr><td>ID_Cliente</td><td>3</td><td>Identificador de perfil de usuario Cliente</td></tr>
<tr><td>ID_Desarroll</td><td>4</td><td>Identificador de desarrollador</td></tr>
<tr><td>tipo_prot</td><td>1</td><td>Tipo de protección.
1. Accesos
2. Tiempo
3. Accesos y tiempo</td></tr>
<tr><td colspan="3">Resultado de Salida. protección de aplicación con su correspondiente clave uso</td></tr>
</table>

Tabla 6.15. *Creación de un proyecto*

☑ **Validación de datos**

Estado del Sistema				
Validación de datos para generar la clave de protección.				
Nombre	**Valor**	**Descripción**	**Observación**	
Tipo_proteccion	3	Protección por acceso y tiempo	Valido	Se debe escoger una de las tres opciones.
Acceso_txt	50	Protección por 50 acceso	Valido	Valor debe ser estrictamente mayor que cero (0).
Tiempo_txt	20	Protección por 20 días	Valido	Valor en días mayor que cero (0).

Tabla 6.16. *Validación de datos*

PARTE III

PROBANDO EL SISTEMA

En esta Parte III se muestra de una manera detallada la forma utilizada para llevar a cabo la protección de aplicaciones software; se hace un recorrido por las diferentes posibilidades de protección existentes, analizando las ventajas y desventajas que ofrecen, para definir por ultimo la forma implementada. También se realizan pruebas de auditoria al sistema, buscando reducir al mínimo los errores contenidos en la aplicación.

Los objetivos específicos de esta parte son:

- Analizar los diferentes tipos de mecanismos de protección existentes, especificando las ventajas y desventajas con que cuentan.
- Proponer un mecanismo de protección que se amolde a las condiciones con que se cuenta.
- Describir el flujo de ejecución de una aplicación protegida.
- Realizar prueba de auditoria al sistema.

CAPITULO 7

ASPECTOS RELEVANTES

7.1 COMO PROTEGER APLICACIONES

Proteger una aplicación es un proceso no tan fácil de concebir debido a que requiere conocimientos que no son muy manejados o son transparentes para la mayoría de las personas que están en contacto con el mundo de la informática; dentro de este tipo de conocimiento se encuentra la criptografía, manejo de memoria, procesos, hilos, manejo de archivos y programación avanzada para Win32. Para poder adquirir información sobre estos aspectos remítase al marco teórico.

No existe una única forma de proteger las aplicaciones, ya que se pude realizar mediante validaciones incluidas por el programador al momento de implementar la aplicación o mediante mecanismos externos que garanticen el bloqueo de la aplicación al momento de vencimiento de algunos de los términos establecidos por el distribuidor de la aplicación.

7.1.1 Protección durante la implementación por parte del programador

Consiste en comparar claves, información del cliente, información de la maquina residente y monitoreo por medio de Internet.

7.1.1.1 Comparación de claves

Consiste en incluir dentro del código fuente al momento de la programación una variable a la cual se le asigna un valor; este debe ser comparado al momento de ejecución con un dato ingresado por el usuario.

- ***Ventaja*:** Personalizar las contraseñas y las interfaces de protección
- ***Desventaja:*** Es una manera poco practica de realizar la protección, ya que el desarrollador debe generar un ejecutable por cada una de las distribuciones a realizar. Además, de esta manera solo se logra que la ejecución de la aplicación se haga con conocimiento de la contraseña, pero si esta es divulgada con el ejecutable de la aplicación cualquier persona podría tener acceso a esta desconociendo los derechos del productor de la aplicación.

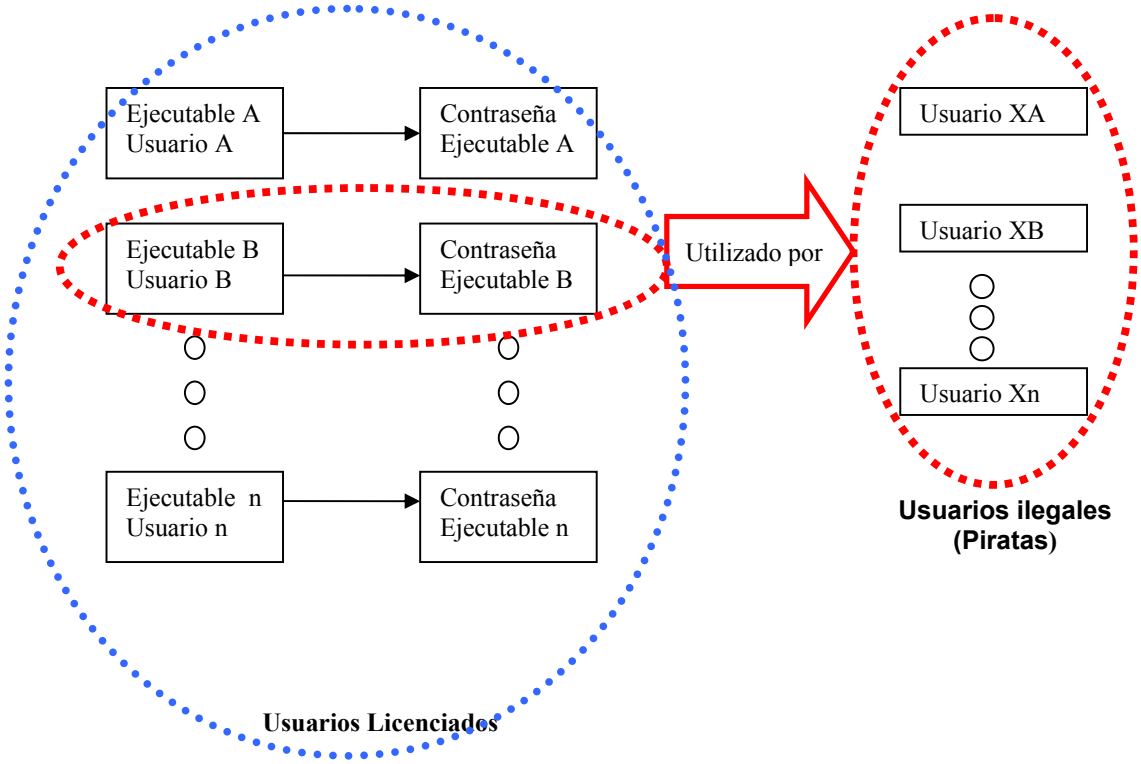


Figura 7.1 *Desventajas de la protección por comparación de claves*

7.1.1.2 Información del cliente

El programador puede incluir en el código fuente de la aplicación a realizar información concerniente al cliente como lo es: Nombre, dirección, teléfono, correo electrónico, nombre de la empresa y mucha más información personal acerca de la persona que usara la aplicación.

- ***Ventaja:*** Esto le puede garantizar que ningún usuario diferente al cliente utilice la aplicación, ya que una persona que obtenga la aplicación por otros medios diferentes a la compra, podría no contar con información concerniente a la persona que adquirió los derechos de ejecución de la aplicación por medio de la negociación legal.

- ***Desventaja:*** Se debe generar un ejecutable por cada distribución realizada como sucede en el caso anterior. Además, para evitar la distribución ilegal de la aplicación solo se tiene de garantía la buena fe de la persona que legalmente adquirió los derechos de ejecución de la aplicación.

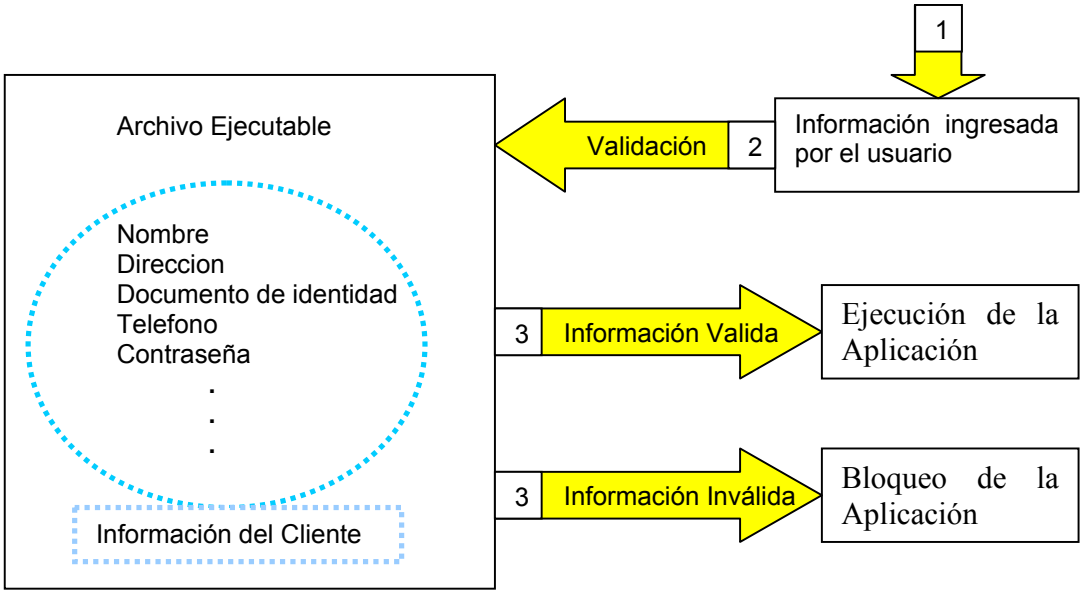


Figura 7.2 *Descripción de la protección por información del usuario*

7.1.1.3 Información de la maquina residente

Consiste en realizar comprobaciones de información propia de la maquina en la cual funciona la aplicación, como lo es el serial del disco duro, el tipo de procesador, el numero de sectores defectuosos del disco, el total de clusters del disco duro, etc.
Esta validación requeriría que el programador tenga contacto con la maquina residente para poder averiguar dicha información. Para esto se debe utilizar algunas API's de Windows, para que la aplicación en tiempo de ejecución pueda extraer la información que será utilizada como patrón para la comparación.

- **Ventaja:** Permite que la aplicación protegida solo funcione en la maquina del cliente al cual ha sido concedido el permiso de ejecución. En caso que el cliente intente instalar la aplicación en otra maquina, está no funcionara.

- **Desventaja:** Al utilizar este tipo de procedimientos se requiere un contacto previo con la maquina, y en caso que la configuración de la maquina cambie se debe modificar la información que se toma como patrón para la comparación. Si esta información fue colocada en el código fuente se debe generar otro archivo ejecutable.

Por ejemplo, con la API *GetDiskFreeSpace* podemos apreciar claramente el inconveniente que podría acarrear el usar este procedimiento. Esta función da información acerca del disco especificado

```
BOOL GetDiskFreeSpace ( LPCTSTR lpRootPathName,
                        LPDWORD lpSectorsPerCluster,
                        LPDWORD lpBytesPerSector,
                        LPDWORD lpNumberOfFreeClusters,
                        LPDWORD lpTotalNumberOfClusters);
```

La variable *lpTotalNumberOfClusters* referencia el numero total de *clusters* en el disco.

```
      Bool p = GetDiskFreeSpace ( root,sector,bytes,number,totl_cluster);
If(p){
        //El numero de cluster del disco es 11990
        If(total_cluster = = 11990){
                Ejecutar_aplicacion();  //Ejecuta la aplicación.
        }
}
```

Figura 7.3 *Archivo fuente en C++. Primera versión*

Si se cambia el disco duro, es posible que el número de cluster varié, lo cual requeriría un cambio en la validación. Esto ocasionaría la generación de un nuevo ejecutable.

```
Bool p = GetDiskFreeSpace ( root,sector,bytes,number,totl_cluster);
If(p){
        //El numero de cluster del nuevo disco es 12900
        If(total_cluster = = 12900){
Ejecutar_aplicacion();  //Ejecuta la aplicación.
        }
}
```

Figura 7.4. *Archivo fuente en C++. Segunda versión.*

7.1.1.4 Monitoreo por medio de Internet

Se usa en aplicación de entorno distribuido. Consiste en dejar un canal abierto para que el desarrollador pueda monitorear desde su servidor información referente al uso que se le esta dando a su aplicación, para detectar posibles violaciones a los permisos otorgados, por ejemplo:

1. Número de usuarios superior a las licencias otorgadas al cliente.
2. Número de peticiones de clientes superiores a las licencias clientes otorgadas.

Estas protecciones son utilizadas por grandes casa desarrolladores de software, que se dedican a distribuir aplicaciones especializadas como lo es: SAP, PeopleSoft, etc.

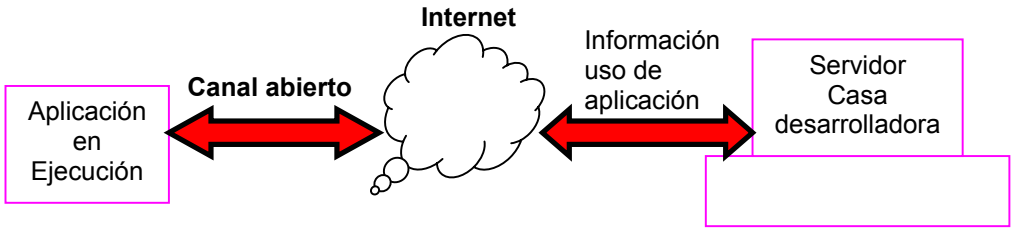


Figura 7.5 *Monitoreo por medio de Internet.*

7.1.2 Protección mediante mecanismos externos

Es otro tipo de protección que no requiere incluir instrucciones en el código fuente de la aplicación original, lo cual libera al programador de la responsabilidad de proteger su aplicación y lo deja en manos de otras aplicaciones; este tipo de aplicaciones trabaja solo con manipulación de archivos ejecutables (hexadecimal).

Estos mecanismos son: Realizar injertos al archivo ejecutable, ejecutar la aplicación protegida desde otra aplicación, implementar la protección por medio de funciones en una librería de vinculo dinámico (dll) y elaborar una aplicación estándar que verifique los permisos otorgados.

7.1.2.1 Injerto al archivo ejecutable

Se conoce como injerto a la inserción de instrucciones en lenguaje de maquina a un archivo ejecutable. El injerto se debe hacer en lenguaje de maquina debido a que el archivo ejecutable también se encuentra en este lenguaje.

Para realizar un injerto se debe localizar un espacio dentro del archivo ejecutable en el cual no se encuentren ninguna instrucción a ejecutar, lo cual se conoce como

hueco[14]. Estos huecos se generan debido al formato de los archivos ejecutables, ya que existe un estándar que establece las zonas que debe tener un archivo para ser reconocido por la maquina como un ejecutable; si el código fuente implementado es tan corto que no cubre estos espacios, el compilador debe rellenar de ceros (0) para cumplir con el estándar.

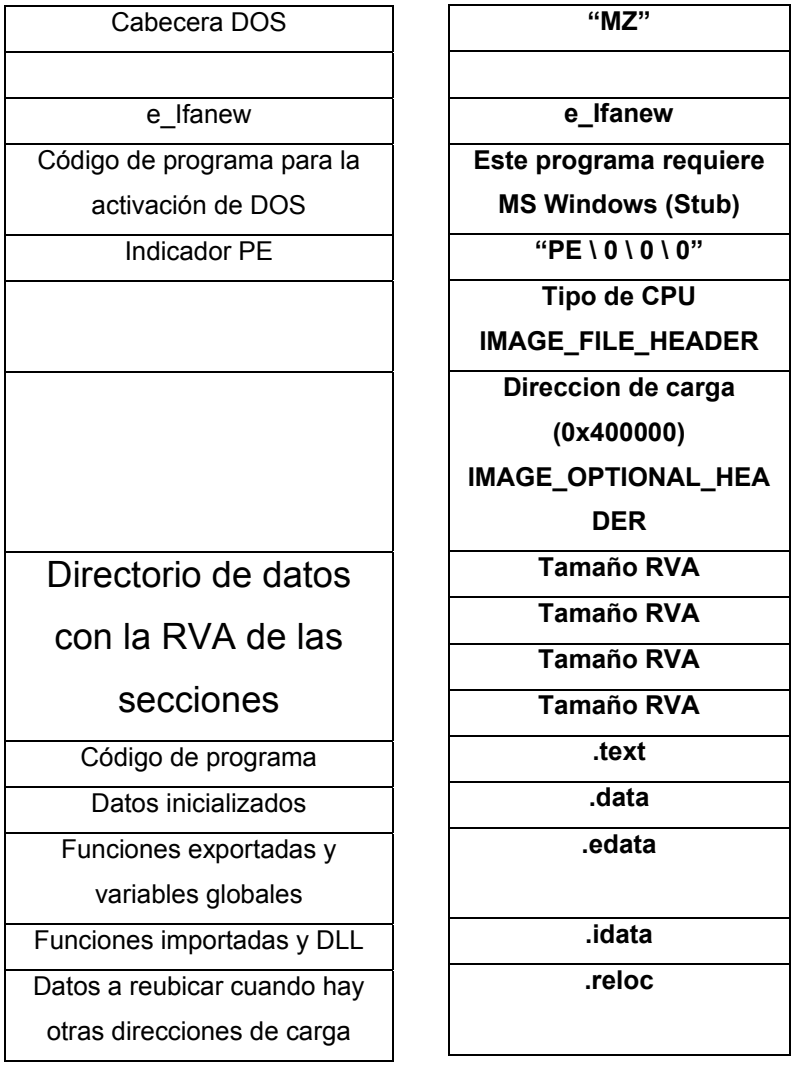


Figura 7.6 *Estructura de un archivo ejecutable (PE).*

[14] Hueco. Espacio dentro de un archivo ejecutable que el compilador rellena de ceros (0) al momento de la compilación.

La cabecera DOS (“MZ”) para los ejecutables es una etiqueta que por defecto debe encintrarse al principio de toda ejecutable. Stub indica sobre que plataforma puede correr el PE. Luego del stub se identifica el archivo como un Portable ejecutable, mediante el indicador PE. Además en el estándar también se incluye el directorio de datos con la RVA[15]; cada uno de estos RVA corresponde a las secciones del ejecutable (.text, .data, .edata, .idata, .reloc), en las cuales se almacena la información necesaria para la correcta ejecución del archivo. El RVA de cada una de las secciones corresponde a la dirección en memoria en la cual se localiza la sección para ser ejecutada

En el siguiente ejemplo se puede apreciar en resalto algunos huecos de un archivo ejecutable.

00000000 4c 01 07 00 57 e1 36 34 a0 02 00 00 1e 00 00 00 L...W.64.....
00000010 00 00 00 00 2e 64 72 65 63 74 76 65 00 00 00 00 .....drectve....
00000020 00 00 00 00 26 00 00 00 2c 01 00 **00 00 00 00 00** ....&...,.....
00000030 **00 00 00 00 00 00 00 00** 00 0a 10 00 2e 64 65 62 .....deb
00000040 75 67 24 53 00 00 00 00 00 00 00 00 5c 00 00 00 ug$S.....\...
00000050 52 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R...............
00000060 48 00 10 42 2e 74 65 78 74 00 00 00 00 00 00 00 H..B.text.....
00000070 00 00 00 00 0a 00 00 00 ae 01 00 00 b8 01 00 00 ....
00000080 c2 01 00 00 01 00 03 00 20 10 50 60 2e 64 65 62 .... .P`.deb
00000090 75 67 24 53 00 00 00 00 00 00 00 00 30 00 00 00 ug$S.....0...
000000a0 d4 01 00 00 04 02 **00 00 00 00 00 00** 02 00 00 00 ....
000000b0 48 10 10 42 2e 74 65 78 74 00 00 00 00 00 00 00 H..B.text.....
000000c0 00 00 00 00 05 00 00 00 18 02 00 00 00 00 00 00 ....
000000d0 1d 02 00 00 00 00 02 00 20 10 50 60 2e 64 65 62 .... .P`.deb
000000e0 75 67 24 53 **00 00 00 00 00 00 00 00** 2f 00 00 00 ug$S...../...
000000f0 29 02 00 00 58 02 00 00 00 00 00 00 02 00 00 00)...X...........
00000100 48 10 10 42 2e 64 65 62 75 67 24 54 **00 00 00 00** H..B.debug$T....
00000110 **00 00 00 00** 34 00 00 00 6c 02 00 00 00 00 00 00 ....4...l.......
00000120 00 00 00 00 00 00 00 00 48 00 10 42 2d 64 65 66 ....H..B-def
00000130 61 75 6c 74 6c 69 62 3a 4c 49 42 43 20 2d 64 65 aultlib:LIBC -de
00000140 66 61 75 6c 74 6c 69 62 3a 4f 4c 44 4e 41 4d 45 faultlib:OLDNAME
00000150 53 20 02 00 00 00 11 00 09 **00 00 00 00 00** 0a 68 S ....h
00000160 65 6c 6c 6f 32 2e 6f 62 6a 43 00 01 00 05 00 00 ello2.objC......
00000170 00 3c 4d 69 63 72 6f 73 6f 66 74 20 28 52 29 20 .<Microsoft (R)
00000180 33 32 2d 62 69 74 20 43 2f 43 2b 2b 20 4f 70 74 32-bit C/C++ Opt
00000190 69 6d 69 7a 69 6e 67 20 43 6f 6d 70 69 6c 65 72 imizing Compiler
000001a0 20 56 65 72 73 69 6f 6e 20 31 31 2e 30 30 55 8b Version 11.00U.
000001b0 ec e8 00 00 00 00 5d c3 04 00 00 00 13 00 00 00 ......].........
000001c0 14 00 08 **00 00 00 00 00** 03 00 00 00 01 00 08 00 ....
000001d0 00 00 02 00 2a 00 0b 10 **00 00 00 00 00 00 00 00** ....*...........

[15] Relative Virtual Address. Dirección virtual relativa

```
000001e0 00 00 00 00 0a 00 00 00 03 00 00 00 08 00 00 00 ................
000001f0 01 10 00 00 00 00 00 00 00 00 01 04 6d 61 69 6e ............main
00000200 02 00 06 00 20 00 00 00 08 00 00 00 0b 00 24 00 .... .........$.
00000210 00 00 08 00 00 00 0a 00 55 8b ec 5d c3 13 00 00 ........U..]....
00000220 00 00 00 03 00 00 00 01 00 29 00 0b 10 00 00 00 .........)......
00000230 00 00 00 00 00 00 00 00 00 05 00 00 00 03 00 00 ................
00000240 00 03 00 00 00 01 10 00 00 00 00 00 00 00 00 01 ................
```

Para realizar un injerto se necesitan las siguientes herramientas :

- Desensamblador.

  Herramienta utilizada para poder obtener las instrucciones en lenguaje de maquina que componen la aplicación. Existen en el mercado herramientas como el W32disasembler, MASM32.

  Se debe utilizar el desensamblador para obtener las instrucciones en lenguaje de maquina de la aplicación a proteger y de las instrucciones a injertar.

- Editor Hexadecimal.

  Herramienta utilizada para visualizar el contenido del archivo ejecutable de modo hexadecimal. Esto se utiliza para ubicar los huecos en el archivo ejecutable que se va a proteger.

  Una vez ubicado la dirección en la cual se encuentra el hueco, se procede a pegar las instrucciones que componen el injerto. Esta es la labor mas complicada, ya que si se llega a variar algún carácter correspondiente a una instrucción, el archivo ejecutable se corrompe, es decir el sistema operativo no reconocerá el archivo como un ejecutable.

```
jmp init
init:
mov esi, offset cod1     ; operando origen
mov edi, 0066e862h      ; operando destino (dir de la
instruccion a cambiar)
movsb                   ; y cambiamos la instruccion
mov esi, offset cod1
mov edi, 0066a8c5h
```

- ☑ **Ventaja:** Permite conservar el tamaño original del archivo a proteger, de manera que el usuario no percibiría que se han incluido nuevas instrucciones que se ejecutaran con la aplicación.

- ☑ **Desventaja:** No todas las aplicaciones tienen las mismas instrucciones, por lo cual en su ejecutable no se encontraran los huecos en la misma dirección. Esta característica no permite que el método se pueda usar en todas aplicaciones, razón por la cual es preferido por los Crackers para vulnerar las protecciones más que para proteger.

7.1.2.2 Ejecución de una aplicación protegida desde otra aplicación.

Consiste en elaborar una aplicación que solicite información que identifique al cliente que se le ha otorgado permisos para la ejecución de la aplicación. De manera muy sencilla se puede variar la información para cada distribución a realizar, ya que la aplicación que verifica esto es tan sencilla que solo consta de dos (2) funciones: una que verifica la información y otra que pone a la aplicación protegida en ejecución.

Para poder implementar este mecanismo se debe garantizar que la aplicación protegida solo entre en ejecución mediante la invocación de la aplicación que valida la información. Esto se puede lograr cifrando la aplicación a proteger; será descifrada luego de verificar la información del cliente, para poder ejecutarla como una aplicación normal.

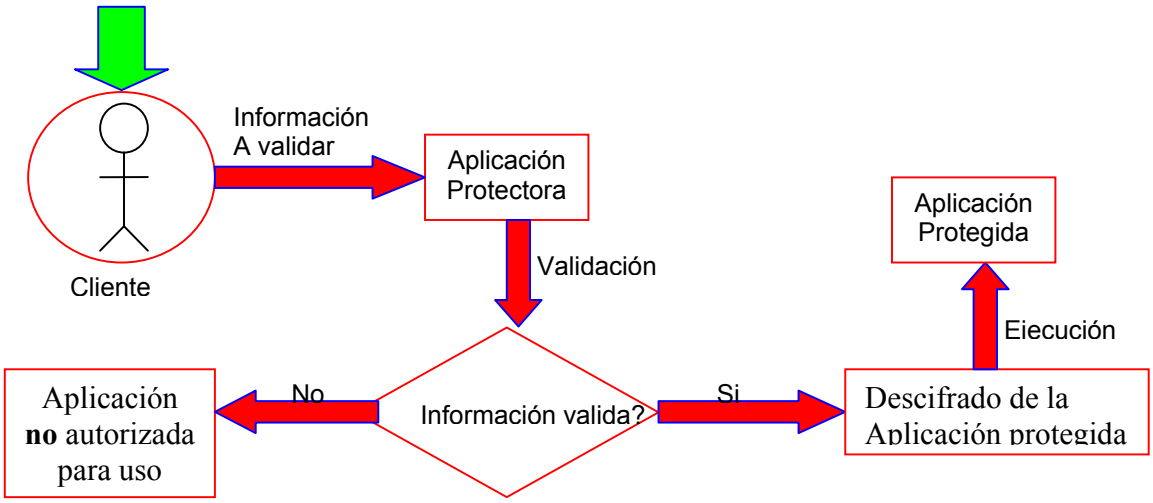


Figura 7.8 *Flujo de ejecución de una Aplicación protegida con el mecanismo 8.1.2.2*

- **Ventaja:** Permite que la aplicación protegida solo sea ejecuta luego de la validación de la información dada por el cliente al adquirir el producto.

- **Desventaja:** Si la persona que adquirió los derechos de ejecución de la aplicación suministra la información y el ejecutable que se le entregó, cualquier persona puede ejecutar la aplicación con solo digita la información suministrada. Genera incomodidad transportar dos aplicaciones (protectora y protegida) para una usuario que no sospecha como se lleva a cabo la protección.

7.1.2.3 Implementación de la protección por medio de funciones en una librería de vinculo dinámico (dll)

Consiste en implementar un grupo de funciones que permitan realizar el control y validación de los permisos otorgados a la aplicación. Estas funciones se encuentran almacenadas en una librería de vinculo dinámico (dll), de manera que puedan ser utilizadas por cualquier aplicación sin necesidad de rescribir el código de las funciones. Además, en caso que se requiera realizar algún cambio en las funciones, solo se debe modificar la librería de vinculo dinámico (dll) y no la aplicación que invoca a las funciones.

- **Ventaja:** Permite reutilizar código, de forma que la programación se haga más comprensible y depurada. Resulta difícil realizar un seguimiento de las funciones utilizadas por la aplicación; aun utilizando un visor de memoria.

- **Desventaja:** Las funciones implementadas deben ser invocadas por alguna aplicación para que puedan entrar en funcionamiento. Por lo cual, la única forma en que este tipo de protección puede funcionar es que la aplicación protegida invoque a las funciones, lo cual requeriría que al momento de escribir el código fuente de la aplicación, se incluyera el llamado a las funciones, o que se tomara el ejecutable de la aplicación y se desensamblara para introducirle el llamado de la funciones, con lo cual se volvería nuevamente en la primera opción ya analizada en el numeral 7.1.2.1.

Aunque invocar una librería dll mediante injertos suele ser sencillo, lo que se dificulta es el llamado a las funciones que componen la librería.

7.1.2.4 Elaboración de una aplicación estándar que verifique los permisos otorgados.

Consiste en elaborar una aplicación estándar que verifique los permisos otorgados y si estos son validos ejecute la aplicación protegida como un proceso hijo. Para nuestro caso, la ejecución de un proceso hijo requiere que exista un archivo ejecutable grabado en el disco; este archivo es invocado por un proceso (padre) que tiene el control sobre el proceso hijo (archivo que se ejecuta).

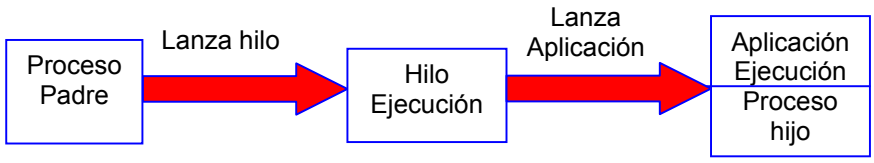


Figura 7.9 *Ejecución de una aplicación como un proceso hijo*

Como permisos otorgados se entienden los números de días de uso y el número máximo de acceso, dependiendo el tipo de protección otorgada.

- **Ventaja:** Se puede colocar todas las restricciones que se quieran en la aplicación estándar sin la limitante de espacio que se presenta cuando se hacen injertos.

- **Desventaja:** Es fácilmente identificable la protección mediante este mecanismo debido a que deben existir por cada protección, dos archivos ejecutables en el disco. Además, la aplicación protegida podría ejecutarse por si sola si se logra localizar su dirección en el disco.

7.2 SOLUCIÓN IMPLEMENTADA.

Tomando como base las posibles soluciones planteadas en el numeral anterior (7.1), se logró implementar una solución que cumple con los objetivos planteados por TRIONIX para garantizar una protección segura.

La solución implementada se basa en conceptos de manejo de archivos, manejo de procesos y criptografía. Para poder proteger una aplicación se requiere como condición inicial contar con las siguientes aplicaciones:

- Aplicación a proteger. Suministrada por la persona que solicita la protección de la aplicación.

- Protector. Aplicación suministrada por Trionix.
- Pprotector. Aplicación suministrada por Trionix.

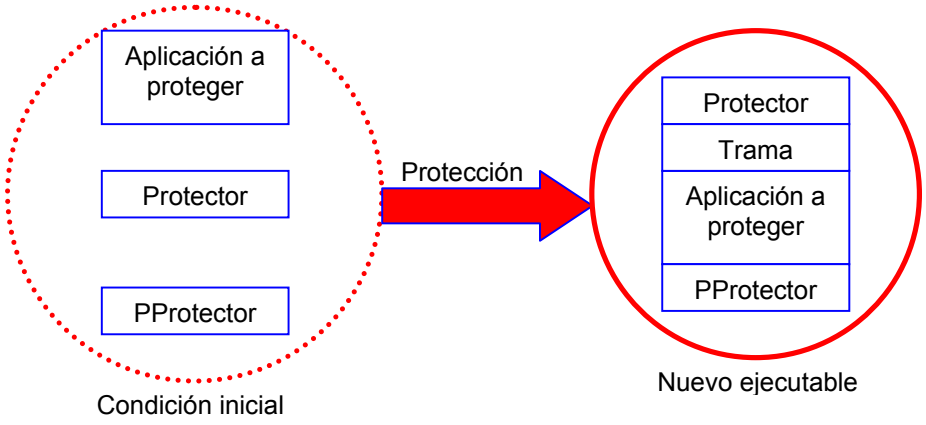


Figura 7.10 *Descripción general de la protección implantada*

7.4.1 Descripción.

7.4.1.1 Condición inicial

Inicialmente se encuentran las aplicaciones en su estado original, es decir como archivos ejecutables en el disco para poder ser sometidas al proceso de protección. Las tres aplicaciones son:

- **Aplicación a proteger**

  Archivo ejecutable de la aplicación que se desea proteger.

- **Protector**

  Archivo ejecutable de la aplicación estándar encargada de verificar los permisos otorgados

- **Pprotector**

  Archivo ejecutable de la aplicación encargado de monitorear el correcto flujo de eventos durante la ejecución de la aplicación protegida.

Deben existir estas dos últimas aplicaciones (protector y pprotector) para que al momento de la ejecución de la aplicación protegida por parte del cliente, protector verifique los permisos y pprotector asegure que protector nunca será finalizado estando en ejecución la aplicación a proteger. De ocurrir esto, pprotector debe finalizar y borrar la aplicación a proteger que se ha descifrado en el disco, para obtener más detalles ver el apartado 7.2.1.3.

7.4.1.2 Protección como proceso

Para poder llevar a cabo el proceso de protección se deben seguir una serie de pasos que permitan obtener los recursos necesarios para proteger la aplicación. Estos pasos son:

- Paso 1. Capturar información relacionada con:
  - ☑ Ruta de la aplicación a proteger. Dirección donde se localiza la aplicación a la cual se solicita proteger.
  - ☑ Ruta de destino. Dirección en la cual se desea grabar el nuevo ejecutable correspondiente a la aplicación protegida.
  - ☑ Tipo de protección. Protección que se le desea dar a la aplicación (1. Accesos, 2. Tiempo, 3. Accesos y tiempo).
  - ☑ Accesos. Numero de veces que se puede utilizar la aplicación protegida.
  - ☑ Tiempo. Numero medido en días de la duración del permiso de uso.
- Paso 2. Creación de un nuevo archivo ejecutable. Luego de cumplir con el paso 1 y las condiciones iniciales se procede a armar el nuevo archivo ejecutable de la siguiente manera:

- ☑ Abrir el protector para copiarlo bit a bit desde el principio, en el nuevo archivo ejecutable.

- ☑ Armar tramas con la información obtenida en el paso 1. Luego, se procede a copiar en el nuevo archivo ejecutable cada una de las tramas, colocando una bandera que las identifique. Estas banderas son números de 64 bits que se establecen de manera arbitraria, pero por seguridad se cifra con el algoritmo DES, con la misma clave utilizada para cifrar la aplicación a proteger, para mas detalles ver 8.4

- ☑ Abrir la aplicación a proteger para cifrarla con el algoritmo DES en paquetes de 64 bits en intervalos variables de paquetes; La razón por la cual no se cifran todos los paquetes es por motivo de tiempo de ejecución y recursos consumidos, tanto al momento de la protección como en la ejecución de la aplicación protegida por parte del cliente, siendo esta ultima uno de los puntos críticos del proceso, antes de copiar esta aplicación se debe colocar una bandera para identificarla.

- ☑ Abrir la aplicación protector para copiarla a continuación de la aplicaron a proteger byte a byte. Para poder identificarla dentro del nuevo archivo ejecutable se debe colocar una bandera.

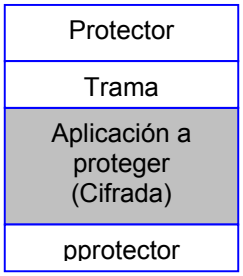


Figura 7.11 *Estructura de la aplicación protegida*

Este proceso de protección tiene un tiempo de duración proporcional al tamaño del archivo a proteger. Se puede apreciar un ejemplo en figura 8.14.

TRIONIX tiene incluido todos los algoritmos necesarios para el cifrado y elaboración de nuevos ejecutables, haciendo de la protección una labor transparente.

7.4.1.3 Ejecución de una aplicación protegida

De acuerdo a lo dicho en el numeral anterior, se obtiene un archivo ejecutable que consta del protector, tramas, aplicación a proteger y pprotector.

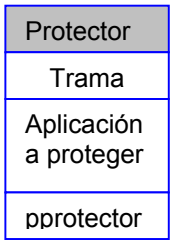


Figura 7.12 *Estructura de la aplicación protegida*

Este archivo al momento de ser ejecutado, empieza por la primera aplicación contenida en su interior (protector), la cual se encarga de realizar la validación de los permisos otorgados comparando con la información contenida en tramas almacenadas dentro del archivo de la aplicación protegida con la información del sistema como fecha y hora actual. Si estos permisos son validos, se descifra la aplicación a proteger que se encuentra en el archivo de la aplicación protegida y la graba como un archivo ejecutable en el directorio donde se encuentra la aplicación protegida, al igual que la aplicación pprotector. Esto se realiza para poder ejecutar ambas aplicaciones (aplicación a proteger, pprotector) como subprocesos.

La ejecución de las aplicaciones como subprocesos requiere que el proceso padre (protector) cree un hilo por cada subproceso a crear. Esto se realiza para poder tener algún control sobre los subprocesos hijos. En caso que el proceso padre sea eliminado, los subprocesos hijos se independizan y pasan a ser procesos sin padre. En este tipo de protección planteada esto traería graves consecuencias, ya que en caso de eliminar el protector, la aplicación a proteger que ha sido descifrada y

gravada en el disco como un archivo ejecutable no podría ser controlada para impedir que se ejecute por sí misma. Por esta razón se utiliza la aplicación pprotector, que tiene como función principal “proteger” a la aplicación protectora; esta protección se basa en que no puede permitir que el protector sea cerrado antes que la aplicación a proteger. También tiene como tarea actualizar la información contenida en tramas dentro de la aplicación protegida como lo es: numero de acceso, numero de días de uso de la aplicación y ultima fecha de acceso. Esta información es almacenada para que en la próxima ejecución de la aplicación protegida, el protector la pueda extraer y realizar la validación de los permisos otorgados.

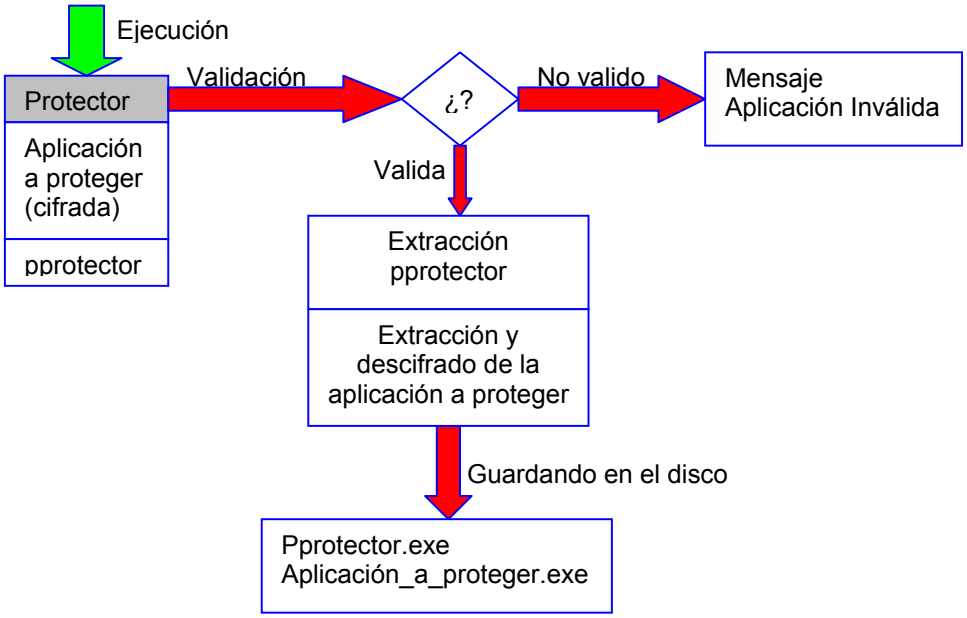


Figura 7.13 *Validación de la aplicación protegida y extracción de aplicaciones*

La aplicación pprotector permanece oculta en ejecución y solo reaccionara antes los eventos:

- **Finalización del proceso protector.**

  Pprotector debe finalizar el proceso de la aplicación a proteger y eliminarla del disco duro para que esta no pueda ser ejecutada sin protección.

- **Finalización de la ejecución de la aplicación a proteger.**

  Finaliza el proceso de la aplicación protector y elimina del disco el archivo correspondiente a la aplicación a proteger.

Figura 7.14 *TRIONIX en el proceso de protección*

De esta manera se espera que la aplicación a proteger no sea copiada cuando se encuentra descifrada en el disco; solo seria posible copiarla cuando se termina de grabar (fin del proceso descifrar), pero inmediatamente comienza su ejecución, lo cual por seguridad el sistema operativo impide copiar un archivo en ejecución. Cuando la aplicación a proteger deja de estar en ejecución es borrada por pprotector de manera que en ningún momento puede ser copiada. Además de esto

se toman medidas que mejoran la seguridad de la aplicación a proteger cuando se encuentra en disco descifrada, como son:

- ☑ El nombre de este archivo en el disco es generado de manera aleatoria. Su longitud es de mínimo cuatro (4) caracteres y máximo diez caracteres (10) que comprender mayúsculas y minúsculas sin incluir signos.

- ☑ Al terminar de descifrar el archivo, se coloca invisible al usuario.

Se espera dificultar la labor de cualquier persona que intente violar el esquema de seguridad aplicado.

7.4.1.3.1 Primera ejecución

La primera vez que se ejecuta la aplicación protegida aparece en un estado inactiva, mostrando un serial y solicitando otro para su comparación. El usuario de la aplicación debe suministrar este serial al administrador TRIONIX para que le sea devuelto el serial solicitado (clave privada).

El proceso de generación de la clave debe ser autorizado por el creador de la aplicación. Una vez se introduzca el serial y se presione iniciar, la aplicación debe empezar su ejecución y seguir el flujo de eventos expuesto anteriormente en el numeral 8.2.1.3.

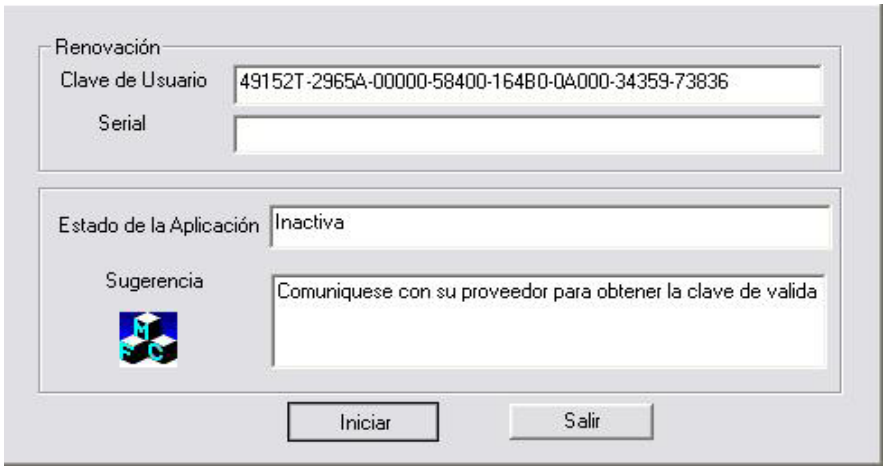


Figura 7.15 *Primera ejecución de la aplicación protegida.*

7.4.1.3.2 Vencimiento.

Cuando una aplicación se vence (por tiempo o acceso) la aplicación protector invoca funciones que le permiten generar la clave pública y la trama que le será suministrada al usuario al momento de iniciar la ejecución de la aplicación protegida. La generación de la clave (serial) se realiza mediante el algoritmo RSA[16] que esta implementado en una librería dll. Esta librería es invocada por la aplicación protector cuando alguno de los permisos otorgados se vence. Para más información acerca de la trama ver apartado 7.3

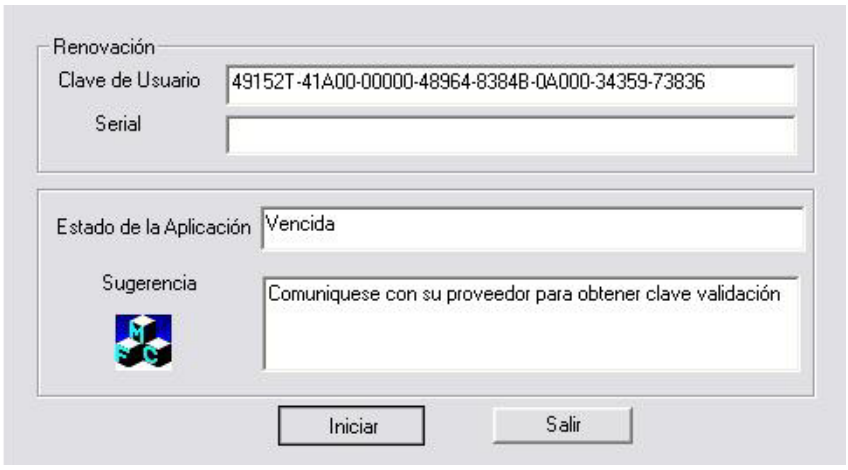


Figura 7.16 *Aplicación Vencida*

[16] El algoritmo RSA se encuentra implementado en la librería mdtrio.dll. Para mas información ver capitulo 6. Fase de construcción.

7.4.1.3.3 Renovación

Cuando la aplicación se vence se genera una clave (publica) que debe ser suministrada al administrador TRIONIX, quien procesa la información de actualización y genera la clave privada (serial).

Cuando la aplicación se vence se debe solicitar la renovación sin cerrar la aplicación vencida, ya que cada vez que se ejecuta, se genera una clave diferente. Esto se hace para evitar que la clave pública a suministrar sea alterada por medio de la copia o reemplazo.

7.5 TRAMA

La trama es una estructura de datos utilizada por la aplicación protector y por TRIONIX para el transporte de información. Su estructura esta definida según las necesidades presentadas en el proceso de protección y renovación.

La trama cuenta con información correspondientes a:

- **Tipo de protección.**

  Existen tres (3) tipos de protección:

  1. **Acceso:** se controla el número de acceso.
  2. **Tiempo:** controla el numero de días a utilizar establecido por el administrador.
  3. **Acceso y tiempo:** en caso de realizar este tipo de protección, se controla según lo descrito anteriormente en el numeral 1 y 2 (Acceso y Tiempo).

- **Acceso**

  Al momento de renovar la aplicación protegida, la trama transporta el número de accesos máximo para poder controlar. Al momento de vencimiento, la aplicación protector incluirá en la trama el número de acceso realizados durante la protección.

En caso de no renovar por acceso, la trama incluye un cero (0) en esta parte de la trama.

- **Tiempo**

Al momento de renovación, TRIONIX incluye en la trama el tiempo máximo de duración dado en días. Al momento de vencimiento, la aplicación protector incluirá en la trama el tiempo de uso dado en días; este tiempo es el número total de días en los cuales se uso la aplicación protegida.

En caso de no realizar una protección por tiempo se incluye en la trama un valor igual a cero.

- **Clave**

Cuando alguno de los permisos otorgados a la aplicación protegida se vence, mediante un llamado a la librería dll que contiene el algoritmo RSA para la generación de claves, la aplicación protector obtiene la clave publica y la incluye en la trama para que pueda ser recibida por TRIONIX mediante la comunicación entre el usuario y el administrador TRIONIX.

Si la trama es recibida correctamente, TRIONIX extrae la clave pública e invoca a la librería dll que contiene el RSA para poder generar la clave privada. Luego de obtener la clave privada, TRIONIX reúne la información necesaria para armar la trama que le será suministrada al usuario para lograr la validación y posterior actualización de los datos de protección de manera que la aplicación protegida vuelva a funcionar bajo los parámetros establecidos.

- **Código**

Al momento de proteger la aplicación se incluye en la trama el código del proyecto, el cual esta compuesto por el año, numero de proyecto y el numero de actualización, por ejemplo 2004000100 es el código del primer proyecto protegido en el año 2004 que no ha sido renovado; mientras que 2004000101 es

el código del primer proyecto protegido en el año 2004 que ha sido renovado una (1).

Al momento de renovar la aplicación, TRIONIX se encarga de extraer de la trama el código del proyecto y aumentarlo para obtener el código del nuevo proyecto. Este código es almacenado en la base de datos para llevar el registro de la última actualización, de manera que se permita generar los reportes de las actualizaciones.

7.5.1 Estructura de la Trama

La trama utilizada esta compuesta por números y letras; se agrupan los datos en ocho (8) paquetes separados por el signo menos (-); el primer bloque esta compuesto por 6 caracteres y los siguientes paquetes están formados por grupos de 5 caracteres.

La trama se construye por medio de una función que se encuentra en la librería dll para el manejo de tramas (mdtrio.dll). Se envía los parámetros a la función la cual retorna la trama como una cadena de caracteres.

```
char* Armar_trama ( ulong tipo_p, ulong tiempo, int64 clave, ulong acceso, ulong codigo)
```

Figura 7.17 Encabezado de la función utilizada para la generación de la trama

Las funciones que se encuentran en la librería para el manejo de tramas trabajan con desplazamiento de bits para poder armar las tramas que serán devueltas. Si enumeramos bit a bit la trama de izquierda a derecha se puede detallar como esta compuesta:

- **Tipo de protección.** Tiene como longitud 2 bit. Ocupa el bit 1 y 2.
- **Tiempo.** Tiene de longitud 16 bits, empieza en el bit 3 y termina en el bit 18.
- **Clave.** Tiene longitud de 64 bit. Va del bit 19 al 83 bit.

- **Acceso.** Longitud 16 bit. Ocupa del bit 84 al bit 100;
- **Código.** Longitud 32 bit. Ocupa del bit 101 al bit 132.

Un ejemplo de la trama es el siguiente:

49154T-0A000-00000-11559-B0000-3A000-20040-00100

7.3.2 Utilidad de la trama

La trama es de gran utilidad en los procesos de renovación y protección, ya que facilita el transporte de información. Cuando una aplicación se quiere renovar, se debe ingresar la clave devuelta por el administrador TRIONIX después de la solicitud realizada de renovación. Si esta clave es valida para la aplicación protegida, el protector extrae el número de acceso máximo, tiempo de uso máximo y tipo de protección, para sobrescribir la información que se encuentran en las tramas internas de la aplicación protegida.

7.6 BANDERAS

Son marcas que se colocan dentro del archivo ejecutable para poder demarcar donde se localiza cierta información. Su tamaño puede ser variable, pero en este caso se escogió un tamaño de 64 bit para disminuir la posibilidad de adivinarla.

El contenido de la bandera es asignado de manera arbitraria, ya que su confidencialidad se basa en el cifrado que posteriormente se le hace con el algoritmo DES, de manera que aunque el valor de la bandera es asignado inicialmente, la marca que se coloca en el archivo ejecutable es desconocida debido a la serie de operación que le aplica el algoritmo de cifrado.

```
marca1= Establecer_marca(5102404150460401);
marca2= Establecer_marca(5314101140326341);
marca3= Establecer_marca(5609606143656607);
marca4= Establecer_marca(5496404159460495);
marca5= Establecer_marca(5320247605349049);
marca6= Establecer_marca(5782956673432433);
marca7= Establecer_marca(5938729568673546);
marca8= Establecer  marca(5021840483984706);
```

Figura 7.18. *Banderas utilizadas por Trionix*

La función Establecer_marca es la encargada de cifrar la bandera que se ha establecido de manera arbitraria.

```
int64 File::Establecer_marca(int64
key){
/int64 marca;
return(cifrar(key));
```

Figura 7.19 *Funciona establecer_marca*

CAPITULO 8

AUDITORIA AL SISTEMA

La auditoria de sistemas consiste en la elaboración de un examen, por medio del cual se puede establecer que esta ocurriendo con el sistema y sugerir medidas a tomar con fines correctivos.

La auditoria a la herramienta fue realizada por el grupo software TRIONIX, debido a que no tiene sentido que los autores del proyecto sean juez y parte en la realización y prueba del funcionamiento de la herramienta software.

8.4 OBJETIVO

Analizar los riesgos a los que puede estar expuesta la herramienta software de protección contra instalación y uso no licenciado de aplicaciones software "TRIONIX" para desarrollar un plan de acción que permita disminuir los riesgos mientras se implementan los mecanismos de control definitivos en la siguiente versión.

8.5 METODOLOGÍA

Inicialmente se creó el cuestionario que se muestra en la Tabla 7.1, en el cual indaga sobre los posibles riesgos a los que podría estar sometida la aplicación software, según PRICE –WATHERHOUSE[17] (7 Niveles).

Nivel de Riesgo	***Preguntas***
1. Acceso a Funciones de Procesamiento	1. ¿Se validan los tipos y permisos para los usuarios?
	2. ¿Existe validación para la contraseña?
	3.¿Puede accederse directamente a la base de datos o a algún archivo físico en el disco donde se almacenen los datos?
2. Ingreso de Datos	4. ¿Qué campos se pueden editar? ¿Bajo qué criterio? ¿Por qué?
	5. ¿Se pueden dejar campos vacíos?
	6. ¿Existen campos obligatorios? ¿Cuales? ¿Se validan?
	7. ¿Se establecen límites para los campos?
	8. ¿Se utilizan dígitos verificadores para validar transacciones?
	9. ¿Se puede establecer correlación entre los campos?
	10. ¿Qué datos se guardan al ingresar?
	11. ¿Bajo qué criterios se puede realizar la toma de datos?
3.Items Rechazados / Suspenso	12. ¿El software cuenta con mecanismos para la recuperación de datos?
	13. ¿Se garantiza la integridad de los datos? ¿Cómo?

[17] GOMEZ F, Luis Carlos, Introducción a la Auditoria de Sistemas. Guía de Clase. Ediciones UIS. Bucaramanga. 1993.

	14. ¿Almacena información acerca de las transacciones realizadas (válidas / rechazadas)?
4. Procesamiento	15. ¿Se cuentan con rutinas generadoras de números secuénciales para el rastreo de transacciones?
	16. ¿Existen rutinas para comparar campos (Control de Balanceo)?
	17. ¿Se realizan backups? ¿Con qué frecuencia?
5. Estructura Organizacional de las funciones	18. ¿Existen manuales acerca del software? ¿Dan claridad acerca de su manejo?
	19. ¿Se cuenta con un cronograma para la toma de muestras (ingreso de datos)?
	20. ¿Se realizan chequeos para determinar la integridad de los datos ingresados en cada
	21. ¿Existe un formato establecido para registrar la información de las transacciones realizadas?
6. Cambios a los programas	22. ¿De qué manera se controla la instalación y el uso del software?
	23. ¿En la realización del software, tomó en cuenta la opinión de los usuarios?
	24. ¿Se programa adecuadamente el soporte y el mantenimiento?
	25. ¿Realizó un plan de pruebas antes de implementarlo?
	26. ¿Existe alguna persona encargada de la supervisión del correcto funcionamiento y la integridad de los datos?
7. Acceso General	27. ¿Se cuenta con lugares adecuados para almacenar los backups, archivos del sistema e instaladores?
	28. ¿Existen restricciones de horario a la hora de utilizar el software?

	30. ¿Se realizan informes periódicos acerca de las transacciones realizadas?
	31. ¿El software genera logs para el administrador? ¿Se realiza un chequeo periódico de los mismos?
	32. ¿Qué controles existen para el acceso a las instalaciones físicas donde opera el software?

Tabla 8.1 *Niveles de riesgos*

8.6 ANÁLISIS DE RIESGOS

Con base en las encuestas realizadas a los usuarios[18] y desarrolladores de la aplicación[19] , con la disponibilidad de las copias del ejecutable del sistema y su base de datos se realizó el análisis de riesgo especificado a continuación; sin embargo algunos riesgos se trataron de acuerdo a su posibilidad de aparición cuando este operando normalmente el sistema, por eso algunas observaciones, descripciones y recomendaciones están trazadas a un futuro inmediato al igual que los mecanismos de control planificados por el grupo software para mitigar los riesgos que puedan presentarse.

8.3.1 Acceso a controles de procesamiento.

El entorno de la aplicación cuenta con una estructura administrativa definida. En este estudio se dio a manera de prueba el rol de administrador a uno de sus desarrolladores, sin embargo cuando la aplicación inicie operaciones en la escuela de Ingeniería de sistemas, el administrador del sistema será el Director de Escuela o alguna persona designada por él.

Una de las funciones del administrador es asignar claves para el acceso al sistema; se presentan diferentes tipos de usuarios como son: Administrador, Usuario, Cliente

[18] Integrantes del grupo software TRIONIX.
[19] Autores del proyecto.

y Desarrollador. Cada uno de estos, cuenta con una clave que debe ser suministrada al inicio de la ejecución de la herramienta para su identificación. De acuerdo al tipo de usuario, la herramienta invoca un mecanismo que permite activar los menús asociados al perfil, restringiendo el acceso y garantizando que personas ajenas a la aplicación o perfil eliminen, modifiquen o procesen transacciones exclusivas del administrador.

8.3.4 Ingreso de datos

El ingreso de datos al sistema es exclusividad del usuario administrador. Cuando una persona desea proteger una aplicación y ha realizado la correspondiente solicitud al administrador, se inicia el proceso de protección con la recopilación de información referente a la aplicación a proteger y su entorno; inicialmente se solicita el nombre de la protección, nombre del desarrollador de la misma, correo, dirección e institución del mismo, modalidad de la aplicación. Estos campos deben ser llenados en su totalidad.

Una vez recopilada la información inicial, se debe seleccionar el tipo de protección: **1**. Por accesos, **2**. Por tiempo y **3**. Por accesos y Tiempo, en donde necesariamente se debe seleccionar una opción, pues el control utilizado para esto (lista) lo garantiza.

Se cuenta con controles que validan cada uno de los datos transferidos por el usuario al sistema.

El software no permite el ingreso de cantidades negativas.

Los códigos que se manejan están formados por el año, numero de proyecto y numero de actualizaciones. Con lo anterior se garantiza la protección de 9999 proyectos en un año y 99 actualizaciones de cada proyecto, lo cual se considera

suficiente. Por ejemplo el primer proyecto protegido tendrá el siguiente código 2004000100, luego de la primera actualización el código será 2004000101.

Los datos ingresados son compatibles con los datos del archivo maestro. Al comparar la información que ingresamos con la del archivo maestro podemos ver su correspondencia.

Solo se puede procesar independientemente una protección, renovación o consulta en un momento dado.

8.3.3 Ítems rechazados en suspenso

Ante la posibilidad de un corte del fluido eléctrico en el momento del procesamiento, puede que se borre la información que se este transfiriendo en ese momento y que no haya sido guardada, perdiéndose la información por completo sin un mecanismo que permita restablecerla.

Puede que solicitudes nunca sean atendidas por la dependencia estricta del administrador, lo cual implica que procesos como: Protección, renovación y consulta no se realicen, afectando de cierta manera la utilización de la herramienta de protección.

8.5.4 Procesamiento.

Se cuenta con rutinas que validan la autenticidad de las tramas que transfiere el usuario al administrador para renovar, la cual es proporcionada por el sistema una vez expire la protección realizada; sin embargo puede ser tomada por personas no autorizadas o expuesta a equivocaciones del usuario al transferir la trama.

El sistema controla las protecciones por tiempo tomando la fecha del sistema en el momento en que accede por primera vez el usuario a la aplicación protegida, esta fecha puede ser variada, sin embargo existen controles de fecha que impiden la ejecución de la aplicación protegida fuera del limite pactado.

En caso de que el servicio del sistema presente anomalías[20] en su funcionamiento, se debe contar con un equipo de respaldo de iguales especificaciones técnicas que continué brindando el servicio de protección normal.

8.5.5 Estructura organizativa

La selección del personal del grupo software se realiza por referencia e interés y no por medio de pruebas; sin embargo se brinda capacitación a los nuevos componentes en el manejo y buen funcionamiento de la aplicación. Cuando un usuario desea consultar e interactuar directamente con la aplicación, la interfaz esta bien documentada guiándolo a realizar la correspondiente consulta.

La comunicación entre el administrador y los usuarios puede ser formal o informal. La delegación de autoridad se realiza única y exclusivamente ante el grupo software o personas autorizadas; además se tiene establecido reemplazos en caso de ausencia del administrador del sistema.

El software utilizado cuenta con el correspondiente soporte por parte del grupo software que va evolucionando y variando a medida que se culmine con las siguientes versiones de la herramienta.

Los documentos fuentes se encuentran en la Escuela De Ingeniería de Sistemas e Informática y pueden ser consultados, previa solicitud y aprobación de la misma ante el Director de Escuela.

---

[20] Funcionamiento inestable o no arranque del programa, problemas de hardware.

8.5.6 Cambios a los programas

Si durante el procesamiento se presentan errores, el técnico encargado perteneciente al grupo software, puede solucionarlos modificando la base de datos, lo cual de uno u otro modo se prestaría a alteraciones indebidas.

Se pueden realizar modificaciones a la base de datos, con previa autorización por escrito por parte de: secretario del grupo software, administrador del sistema y Director de Escuela de Ingeniería de Sistemas.

8.5.7 Acceso general

El lugar en donde se encontrará el computador en que se ejecuta el software estará restringido el acceso; sin embargo el personal autorizado para acceder al sistema contará con verificador de login y password; lo cual se considera suficiente pues este identifica a la persona que ingreso y realizo alguna modificación.

8.6 PLAN DE CONTINGENCIA

Un plan de contingencia es aquel que debe ponerse en marcha una vez ocurra un evento de emergencia que atente al normal funcionamiento de la aplicación. A continuación se anuncian las posibles contingencias que pueden presentarse:

- Interrupciones del hombre: interrupciones por sabotaje o paros.
- Desastres naturales: eventos impredecibles del medio.
- Falla de Energía: conexiones eléctricas en mal estado o fallas del sistema eléctrico.
- Falla del software que involucre tiempo y costo: fallas que requieren de tiempo considerable y que acarrean un costo.
- La base de datos no permite transacciones.

- El equipo en donde se ejecuta la aplicación no arranca.

Pasos a Seguir bajo declaración de emergencia:

1. Informar de inmediato a la totalidad de integrantes del grupo software.
2. Análisis grupal de causas y consecuencias.
3. Puesta en marcha de procedimientos que normalicen el funcionamiento de la empresa.
4. Realización de actividades de los procedimientos en ejecución según responsabilidades.

Procedimientos asociados a la emergencia y sus responsables de tareas específicas.

Emergencia	***Procedimiento***	***Responsable***
Interrupciones	- Informar a las autoridades académicas, dado el caso de sabotaje por parte de un integrante de la comunidad académica, quien tomará las medidas del caso de acuerdo al Reglamento interno de la Universidad. - En caso de paro se suspenderá parcialmente el servicio debido a que la solución a esta situación es ajena.	Administrador e integrantes grupo software
Desastre Natural (Inundación, terremoto, tormenta)	- Informar a los organismos de socorro.	

Falla de Energía	✘ Puesta en marcha del equipo de respaldo si el corte es parcial. ✘ Evaluar las conexiones eléctricas y correcciones del caso. ✘ Cambio de lugar del equipo principal	Administrador Administrador y Director de Escuela. Administrador
La base de datos no permite transacciones	✘ Corrección de errores ✘ Implementación e implantación de una nueva BD. ✘ Puesta en marcha de programas de conversión. ✘ Ampliación de Espacio de memoria.	Grupo Software
No arranca el equipo	✘ Probar que está llegando el voltaje necesario 110v ✘ Tratar de ingresar con modo a prueba de fallos y corregir errores. ✘ Puesta en acción del equipo de respaldo.	Administrador y Grupo software

Tabla 8.2 *Plan de contingencias*

8.7 RECOMENDACIONES

De acuerdo con el análisis de riesgos realizado se recomienda a la Escuela de Ingeniería de Sistemas e Informática plantear un mecanismo que contrarreste la dependencia de tiempo estricta del Administrador del sistema. El desarrollo de un modulo Web para la aplicación podría solucionar esta situación y consolidar el producto en el ámbito local académico.

La ubicación física en donde se encontrará el computador en que se ejecutará la aplicación deberá tener su similar de respaldo, actualizado como el titular y con las mismas especificaciones técnicas, soporte técnico y efectividad, eficiencia,

integridad, y de información confiable. El lugar no solo deberá tener acceso restringido sino que deberá estar libre de humedad, sol y las instalaciones eléctricas en buen estado respaldadas por las especificaciones técnicas del caso.

Se sugiere que antes del desarrollo del modulo Web se asigne un horario de atención a los usuarios, en los cuales estos puedan solicitar la ejecución de algún proceso de manera que contrarreste la dependencia del administrador.

RECOMENDACIONES

- La aplicación desarrollada esta enfocada a la protección de diferentes tipos de software ejecutable. Esta labor es liderada por el administrador del sistema, él cual recibe la solicitud del cliente, recoge datos como son: tipo de protección, numero de acceso máximo, tiempo a utilizar, y lleva a cabo la protección.

  Se presenta un inconveniente en el sistema planteado como lo es la dependencia del administrador y la posible tramitología que se puede presentar. Aunque se debe aclarar que en la presente versión de la herramienta software Trionix esto no reviste un problema, ya que inicialmente Trionix esta enfocado a proteger e incentivar el software que se realiza al interior de la escuela, pero si se quiere en un futuro no lejano exponer al mercado Trionix se deben hacer ciertas mejoras.

  Para poder solucionar el inconveniente expuesto anteriormente, se recomienda automatizar el proceso de solicitud y renovación de la protección por medio de un modulo Web que sirva como interfaz entre Trionix y el cliente conectado a la Web.

- El sistema de protección se realiza por medio de manejo de archivo. Para poder ejecutar una aplicación protegida se debe manipular el archivo donde residen la aplicación a proteger y la aplicación protectora (protector y pprotector).

  Se plantea una mejora al sistema que incluye el código en la aplicación original, lo cual requiere manejo de lenguaje ensamblador y conocimiento de manejo de memoria.

Esta mejora que se plantea lograría una velocidad de ejecución mucho mayor que la obtenida en este proyecto, debido a que la interacción con el código original seria más directa.

Se debe aclarar que seria una investigación bastante larga, por lo cual se sugiere introducir al tema por medio de una asignatura electiva técnica profesional para seguir estudiando como una línea de investigación a desarrollar en un grupo software.

- La seguridad informática es un área que esta en auge a nivel mundial, debido al aumento que ha tenido en el mundo de las telecomunicaciones. La escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander debería incentivar su estudio y desarrollo, en búsqueda de obtener un campo que nos garantice un aporte a la sociedad.

- La protección de aplicaciones es un campo muy amplio. Este proyecto tiende a proteger aplicaciones software de escritorio, pero seria recomendable extender es tipo de protección a aplicaciones en la Web (protección a script), base de datos, aplicaciones en sistemas distribuidos y demás; con el fin de ofrecer un sistema de seguridad completo Trionix que permita proyectar a la escuela como productora de software sin temor a que las aplicaciones realizadas sean plagiadas.

CONCLUSIONES

- La creación de un mecanismo de protección tomando como base lo mejor de los mecanismos que existen y mejorando las deficiencias que estos presentan, es la forma mas apropiada de obtener la solución a la situación planteada; debido a que se ataca directamente los puntos críticos presentados como son: ubicación de archivos protegidos en el disco de la maquina residente, presencia de librerías dll en la maquina residente y control de procesos.

- Las API's de Windows permiten de una manera fácil manipular propiedades del sistema que antes eran muy difíciles de acceder y modificar. Esto hace posible que se pueda proteger una aplicación mediante funciones propias del sistema operativo (hilos, manejo de archivos, manejo de memoria, etc), pero este conocimiento también le abre las puertas para que cualquier persona que lo posea, pueda realizar accesos indebidos al sistema y a aplicaciones en ejecución. Esta es la razón por la cual el mundo de la seguridad informática se encuentra en constante evolución; si se observa lo que sucede con las aplicaciones de antivirus, se puede apreciar que cuando sale una nueva versión, ya existen virus que no son reconocidos por el antivirus, por lo cual se requieren continuas actualizaciones para la detección de los virus nuevos. De igual manera sucede con las soluciones que se plantean en cuanto a protección de aplicaciones software, en las cuales al momento de liberar una versión de la aplicación protectora, puede ser hallada la forma de vulnerar la protección.

  Por esta razón, la investigación de nuevas formas de protección debe darse de manera continua, para poder disminuir el margen de acción de las personas

interesadas en violar la protección de la aplicación. Para esto es importante una continua interacción con el medio en el cual se desenvuelve la aplicación para poder explorar con anterioridad las posibles formas en que puede vulnerarse la aplicación.

- Los algoritmos criptográficos utilizados en este proyecto como el DES y el RSA presentan ventajas como:

  - ☑ DES
    - Permite cifrar la aplicación protegida, garantizando que su descifrado sin conocer la clave (fuerza bruta) sea tan complejo que se requiera costos (tiempo, hardware y económicos) elevados, terminando por declinar en la intención del descifrado.
    - Su tiempo de ejecución es mayor que de los algoritmos de cifrado que incluyen transposición o sustitución. Pero esto se compensa con la seguridad brindada.
    - Luego de cifrar un archivo con este algoritmo se conserva el tamaño original, debido a que es un algoritmo simétrico de clave secreta.

  - ☑ RSA
    - Al ser un algoritmo de clave publica, puede ser utilizado como generador de claves (publica y privada) de manera que se garantice que para cada clave publica exista una clave privada única.
    - Cifrar un archivo con el RSA implica un aumento en el tamaño original, debido a su naturaleza de algoritmo de clave publica. Por esta razón no se utilizo al momento de cifrar la aplicación a proteger[21] para incluirla en el archivo de la aplicación protegida[22].

---

[21] Aplicación a proteger. Aplicación original suministrada al administrador Trionix para su protección. Ver capitulo 7.

[22] Aplicación protegida. Archivo ejecutable obtenido luego de realizar la protección. Ver capitulo 7 para mas información.

Se puede concluir que para el cifrado de archivos se recomienda los algoritmos simétricos de calve secreta y para el manejo de claves y transacciones seguras se recomienda el algoritmo RSA.

- Existen en el mercado herramientas que permiten proteger aplicaciones con una única clave de cifrado. Pero estas no permiten almacenar la información de las aplicaciones protegidas como lo es: nombre de quien usa la aplicación, tipo de uso, institución, y mucha más información que permite llevar un control sobre las distribuciones realizadas. Trionix cuenta con esta característica, la cual es implementada mediante una base de datos que permite almacenar información, realizar consultas y generar reportes; facilitando de esta manera la administración de las distribuciones realizadas.

  El proteger la aplicación no es suficiente protección, ya que si se cuenta con información sobre el estado y lugar donde se encuentra la aplicación, se puede hacer un monitoreo a clientes que tengan vencida la aplicación y sigan usándola.

- El formato con que cuenta Windows para los archivos ejecutables hace que la labor de protección pueda garantizarse para cualquier archivo Win32 ejecutable; esto se debe a que en Win 9x y superior, se trabaja con un formato estándar para archivos PE (portable ejecutable), cualquier lenguaje que trabaje sobre Win32 debe generar archivos ejecutables que cumpla con dicho formato.

- La creación de perfiles de usuario en la aplicación Trionix permite que solo el administrador sea el responsable de llevar a cabo los proceso de protección y renovación. Se diseño de esta manera, debido a que los procesos de protección y renovación son puntos críticos, que de no ser controlados puede llegarse a una perdida de control que pondría en duda la protección de las aplicaciones.

- La herramienta permite la protección de las aplicaciones por tiempo (días) y/o usos (numero de acceso) lo cual puede definirlo el distribuidor de la aplicación. Para esto se establecen los certificados que le informan a la aplicación el tipo de protección y el limite de uso que tendrá. Este método permite que el distribuidor de la aplicación pueda personalizar cada distribución dependiendo del tipo de acuerdo a que se llegue con el usuario final facilitando los procesos de control.

- Aplicar una metodología adecuada permite que el proceso de desarrollo de software se lleve a cabo dentro de los márgenes establecidos y con la calidad esperada. En este proyecto se trabajó con el proceso unificado de desarrollo. El cual es guiado por casos de uso, iterativo e incremental y centrado en la arquitectura. Aunque existen muchas metodologías, se escogió esta para establecerla como base en el proceso de investigación para el grupo software Trionix, debido a que favorece la filosofía de grupo de trabajo en cuanto al manejo de roles para la distribución de actividades en el desarrollo del proyecto software.

- Este proyecto es el inicio de la investigación en Seguridad Informática en la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander, liderado por el grupo software Trionix[23]. El objetivo de este grupo es garantizar la evolución de la aplicación software a medida que se detecten posibles formas de vulnerar la protección, llevando al enriquecimiento y realimentación que garanticen el ciclo de vida del producto.

- Aunque la criptografía brinda poderosas herramientas para la protección, por si sola no garantiza una protección integral, debido a que se requiere combinar con

[23] Grupo Software Trionix. Grupo dedicado a la investigación y desarrollo de mecanismos de protección de aplicaciones software. El proyecto bandera de este grupo es Trionix Versión 1.0, descrito en este trabajo de grado.

estrategias que contrarresten la astucia y forma de actuar del criptoanalista. Por esta razón en este proyecto se combinan las técnicas criptográficas con manejo de archivos, manejo de memoria y conceptos de funcionamiento del sistema operativo para llevar a cabo la protección.

Se debe recordar que un sistema es tan fuerte como el mas débil de sus componentes.

Bibliografía

[1] MENEZES, Alfred : OURSCHOT, Paul y VANSTONE, Scout. Handbook of applied cryptography. Boca Ratón : Editorial CRC Press, 1997. 780 p.

[2] PINO, Gil. Seguridad Informática: Técnicas Criptográficas. México: Editorial Alfaomega, 1997. 137 p.

[3] ANDREASSEN, Kart. Computer Criptology, Beyond decoder rings. New Jersey : Editorial Prentice, 1988. 268 p.

[4] PFLEEGER, Charles. Security in computing. Editorial Prentice, 1989. 538 p.

[5] SCHNEIER, Bruce. Applied Cryptography: Protocols, algorithms and Source code in C. New York: Editorial John Wiley, 1996. 758 p.

[6] JACOBSON, Ivar : BOOCH, Grady y RUMBAUGH James. El Proceso Unificado de Desarrollo de Software. Madrid : Editorial Addison Wesley,1999. 438 p.

[7] JACOBSON, Ivar : BOOCH Grady y RUMBAUGH James. El Lenguaje Unificado de Modelado. Madrid : Ed. Addison Wesley,2000.

[8] Pressman, Roger. Ingeniería del software : Un enfoque práctico. Tercera Edición. México: Editorial McGraw – Hill,1993.

[9] CEBALLOS, Francisco Javier. Java 2 : Curso de Programación. México: Editorial Alfaomega Ra_ma, 2000.

[10] CEBALLOS, Francisco Javier. Visual C++ : Programación Avanzada en Win32.México: Editorial Alfaomega,1999.

[11] _____, _____. Visual C++ 6: Aplicaciones para Win32. Segunda edición. México: Editorial Alfaomega Ra_ma. 2000.

[12] TISCHER, Michael y JENNRICH, Bruno. PC Interno 5: Programación de sistemas. México: Grupo editor Alfaomega,1998.

[13] NASH, Andrew et al. Pki : Infraestructura de claves publicas. Colombia: Mc graw-Hill, 2002.

[14] INSTITUTO COLOMBIANO DE NORMAS TÉCNICAS Y CERTIFICACIÓN. Tesis y otros trabajos de grado: Compendio. Bogota: Icontec, 2002. 23p.

[15] ECHENIQUE G. José A. Auditoria informática. Editorial McGraw Hill. 2ª. Edición. México, 2001.

Direcciones electrónicas:

- http://www.hispasec.com/. Hispasec Sistemas. Seguridad y Tecnologías de la información
- http://www.fit.qut.edu.au/. Computer Science & Electrical Engineering
- http://webs.ono.com/usr016/Agika/index.html. Astruc, guía para principiantes.
- http://www.cryptored.edu.es Red Iberoamericana de Criptografía.

GLOSARIO

Administrador

Persona encarga del mantenimiento y correcto uso de las funciones e información contenida en la herramienta Trionix.

Casos de uso

Conjunto de acciones que lleva a cabo el sistema con el fin de proporcionar al usuario un resultado importante y representa un requisito funcional.

Centrado en la arquitectura

La arquitectura incluye los aspectos más significativos del sistema: su estructura, comportamiento, restricciones, plataforma en la que debe funcionar el software, sistemas heredados, reutilización de componentes y requisitos no funcionales.

Cliente

Persona o entidad que usa las aplicaciones que han sido protegidas con la herramienta Trionix.

- **Componente**

Partes físicas del sistema. Pueden ser reemplazadas.

- **Criptoanalista**

Persona dedicada a descifrar mensajes cifrados.

- **Criptograma**

Texto cifrado mediante algún algoritmo criptográfico.

- **Desarrollador**

Persona con derecho moral sobre la aplicación que va ha ser protegida por Trionix.

- **Modalidad**

Forma bajo la cual se desarrolla o implementa el proyecto que va a producir como resultado una aplicación.

- **Protección**

Proceso mediante el cual se garantiza que una aplicación funcionara bajo una condiciones establecidas.

ANEXO 1 DES[24]

Introducción

DES (*Data Encryption Standard*, estándar de cifrado de datos) es un algoritmo desarrollado inicialmente por IBM a requerimiento del NBS[25] de EE.UU. y posteriormente modificado y adoptado por el gobierno de EE.UU. en 1977, como estándar de cifrado de todas las informaciones sensibles no clasificadas. Posteriormente, en 1980, el NIST estandarizó los diferentes modos de operación del algoritmo. Es el más estudiado y utilizado de los algoritmos de clave simétrica.

El nombre original del algoritmo, era Lucifer. Trabajaba sobre bloques de 128 bits, teniendo la clave igual longitud. Se basaba en operaciones lógicas booleanas y podía ser implementado fácilmente, tanto en software como en hardware. Tras las modificaciones introducidas por el NBS, consistentes básicamente en la reducción de la longitud de clave y de los bloques, DES cifra bloques de 64 bits, mediante permutación y sustitución y usando una clave de 64 bits, de los que 8 son de paridad (esto es, en realidad usa 56 bits), produciendo así 64 bits cifrados.

DES tiene 19 etapas diferentes. La primera etapa es una transposición, una permutación inicial (IP) del texto plano de 64 bits, independientemente de la clave.

---

[24] Tomado de *http://www.aci.net/kalliste/des.htm*. Traducción de Jorge Sánchez Arriazu diciembre de 1999.

[25] National Bureau of Standards, Oficina Nacional de Estandarización, en la actualidad denominado NIST, National Institute of Standards and Technology, Instituto Nacional de Estandarización y Tecnología.

La última etapa es otra transposición (IP-1), exactamente la inversa de la primera. La penúltima etapa intercambia los 32 bits de la izquierda y los 32 de la derecha. Las 16 etapas restantes son una Red de Feistel de 16 rondas. En cada una de las 16 iteraciones

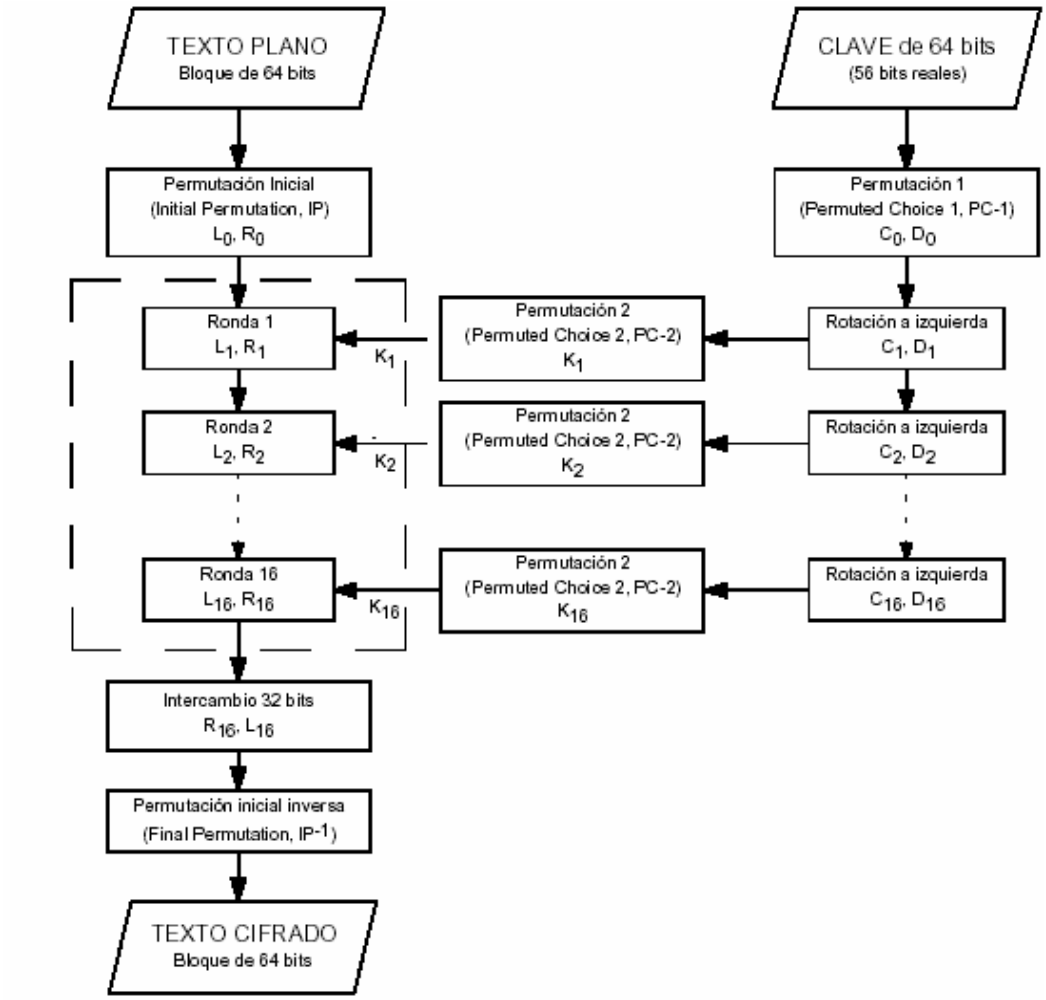


Figura A1 .Esquema general del algoritmo DES

se emplea un valor, Ki, obtenido a partir de la clave de 56 bits y distinto en cada iteración.

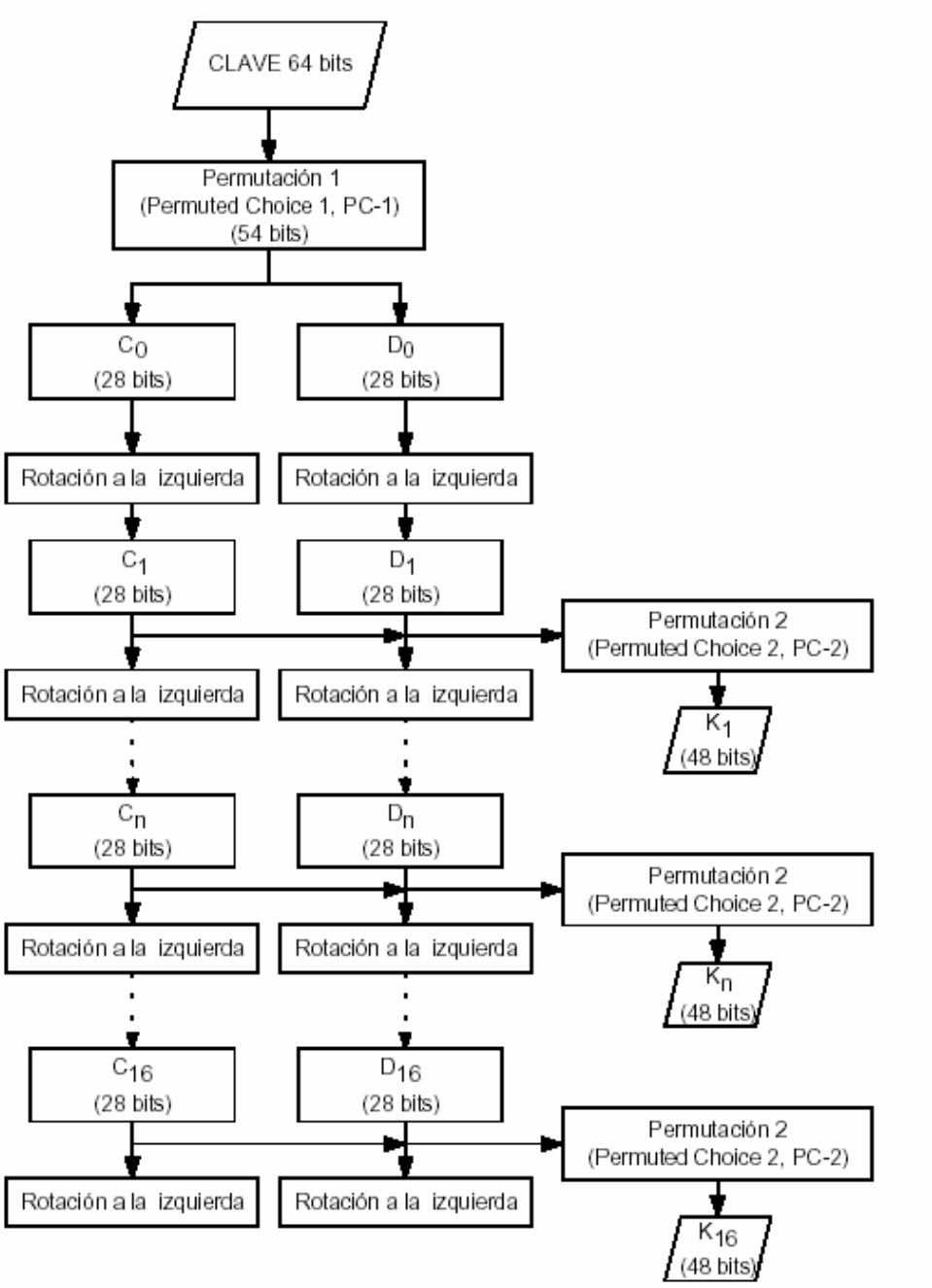


Figura A2. Calculo de la Subclave, Ki

Se realiza una permutación inicial (PC-1) sobre la clave, y luego la clave obtenida se divide en dos mitades de 28 bits, cada una de las cuales se rota a izquierda un

número de bits determinado que no siempre es el mismo. Ki se deriva de la elección permutada (PC-2) de 48 de los 56 bits de estas dos mitades rotadas.

La función f de la red de Feistel se compone de una permutación de expansión (E), que convierte el bloque correspondiente de 32 bits en uno de 48. Después realiza una or-exclusiva con el valor Ki, también de 48 bits, aplica ocho S-Cajas de 6*4 bits, y efectúa una nueva permutación (P).

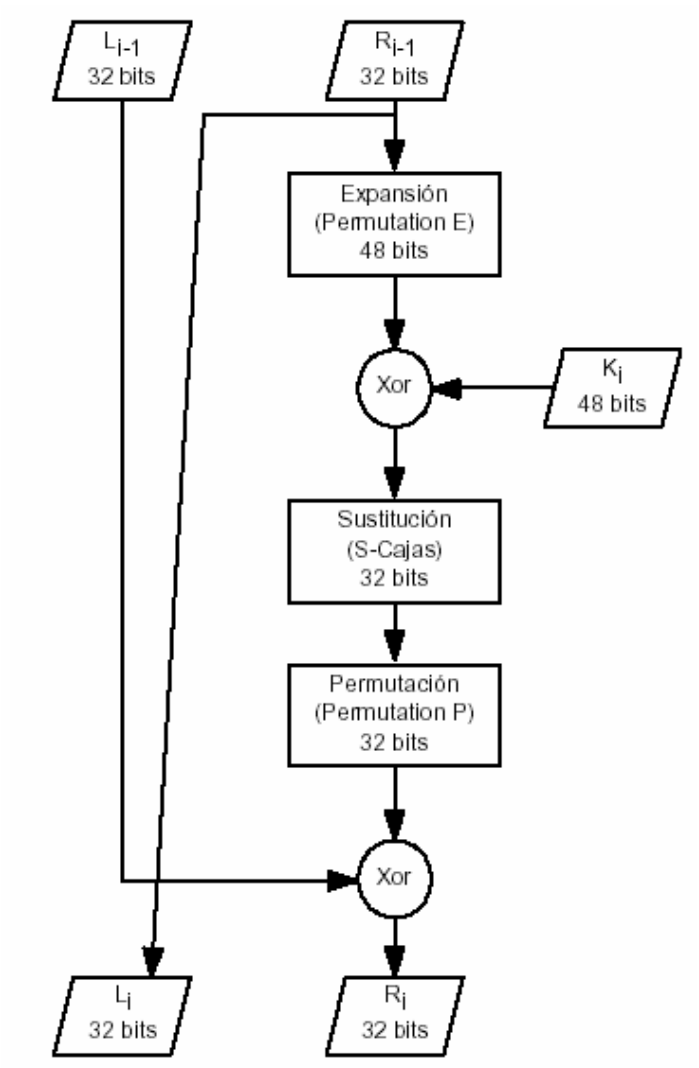


Figura A3 Ciclo de algoritmo DES

Para descifrar basta con usar el mismo algoritmo empleando las Ki en orden inverso.

Descripción paso a paso del algoritmo DES

Se describen los pasos necesarios para implementar este algoritmo.

(i) Encriptación:

1.*- Procesar la clave.*

1.1.- Solicitar una clave de 64 bits al usuario.

La clave se puede introducir directamente o puede ser el resultado de alguna operación anterior, ya que no hay ninguna especificación al respecto.

De cada uno de los ocho bytes se elimina el octavo bit (el menos significativo).

1.2.- Calcular las subclaves.

1.2.1.- Realizar la siguiente permutación en la clave de 64 bits reduciéndose la misma a 56 bits (El bit 1, el más significativo, de la clave transformada es el bit 57 de la clave original, el bit 2 pasa a ser el bit 49, etc.).

57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

Tabla A1. Permutación PC-1

1.2.2.- Dividir la clave permutada en dos mitades de 28 bits cada una. C(0) el bloque que contiene los 28 bits de mayor peso y D(0) los 28 bits restantes.

1.2.3.- Calcular las 16 subclaves (Empezar con i=1)

1.2.3.1.- Rotar uno o dos bits a la izquierda C(i-1) y D(i-1) para obtener C(i) y D(i), respectivamente. El número de bits de desplazamiento está dado por la tabla siguiente:

Ronda	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits despl.	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabla A2. Desplazamiento de bits

1.2.3.2.- Concatenar C(i) y D(i) y permutar. Se obtiene K(i), que tiene una longitud de 48 bits.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Tabla A3. Permutación PC-2

1.2.3.3.- Ir a 1.2.3.1. Hasta que se haya calculado K(16).

***2.**- Procesar el bloque de datos de 64 bits.*

2.1.- Obtener un bloque de datos de 64 bits. Si el bloque contiene menos de 64 bits debe ser completado para poder continuar con el siguiente paso.

2.2.- Realizar la siguiente permutación del bloque:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabla A4. Permutación inicial IP

2.3.- Dividir el bloque resultante en dos mitades de 32 bits. L(0) el bloque que contiene los 32 bits de mayor peso y R(0) el resto.

2.4.- Aplicar las 16 subclaves obtenidas en el paso 1

2.4.1.- E(R(i)). Expandir R(i) de 32 a 48 bits de acuerdo con la siguiente tabla.

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

Tabla A5. Expansión

2.4.2.- E(R(i-1)) Xor K(i). Or-exclusiva del resultado del paso 2.4.1. con K(i).

2.4.3.- B(1), B(2),..., B(8). Partir E(R(i-1)) Xor K(i) en ocho bloques de seis bits. B(1) representa a los bits 1-6, B(2) representa a los bits 7-12,..., B(8) representa a los bits 43-48.

2.4.4.- S(1)(B(1)), S(2)(B(2)),..., S(8)(B(8)). Sustituir todos los B(j) por los valores correspondientes de las S-Cajas o tablas de sustitución (Substitution Boxes, S-Boxes) de 6*4 bits, según se indica en los subapartados que siguen. Todos los valores de las S-Cajas se consideran de 4 bits de longitud.

Fila	Columna																S-Caja
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Tabla A6. S_box

2.4.4.1.- Tomar los bits 1º y 6º de B(j) y formar un número de 2 bits que llamaremos m. Este valor nos indicará la fila en la tabla de sustitución correspondiente S(j). Obsérvese que m=0 representa la 1ª fila y m=3 la última.

2.4.4.2.- Con los bits 2º a 5º de B(j) formar otro número, n, de cuatro bits que indicará la columna de S(j) en la que buscar el valor de sustitución. En esta ocasión n=0 representa la 1ª columna y n=15 la última columna.

2.4.4.3.- Reemplazar B(j) con S(j)(m,n), m fila y n columna.

2.4.4.4.- Ejemplo. Sea B(3)=42, en binario B(3)=101010. Buscaremos el nuevo valor de B(3) en S(3). Fila m y columna n, según lo expuesto anteriormente

m=10, n=0101, y en decimal m=2 y n=5. Por tanto, B(3) será S(3)(2,5)=15

2.4.4.5.- Volver a 2.4.4.1. hasta que todos los bloques B(j) hayan sido reemplazados por el valor de S(j) adecuado.

2.4.5.- P[S(1)(B(1))... S(2)(B(8))]. Concatenar los bloques B(1) a B(8) y permutar los 32 bits (cuatro bits cada B(j)) en función de esta tabla:

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Tabla A7. Permutación P

2.4.5.1.- Volver a 2.4.4.1. Hasta que todos los bloques B(j) hayan sido reemplazados por el valor de S(j) adecuado.

2.4.6.- P[S(1)(B(1))... S(2)(B(8))]. Concatenar los bloques B(1) a B(8) y permutar los 32 bits (cuatro bits cada B(j)) en función de la **Tabla A7**.

2.4.7.- R(i). Realizar una or-exclusiva entre el valor resultante y L(i-1). Este valor será R(i). Por tanto, R(i)=L(i-1) Xor P[S(1)(B(1))... S(2)(B(8))]

2.4.8.- L(i). L(i)=R(i-1)

2.4.9.- Repetir desde 2.4.1. hasta que se hayan aplicado las 16 subclaves.

2.5.- Hacer la siguiente permutación del bloque R(16)L(16). Obsérvese que esta vez R(16) precede a L(16)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tabla A8. Permutación final

(ii) Desencriptación:

Usar el mismo proceso descrito con anterioridad pero empleando las subclaves en orden inverso, esto es, en lugar de aplicar K(1) para la primera iteración aplicar K(16), K(15) para la segunda y así hasta K(1).

ANEXO 2 MYSQL. ESPECIFICACIONES

- Para la Creación de la base de datos con MySQL[26] se debe tener en cuenta el directorio donde deben estar las bases de datos, para esto debemos identificar el archivo My.ini ubicado en la carpeta Windows, editarlo agregándole las siguientes líneas:

  basedir =C:/mysql
  datadir =C:/mysql/data

  Como se puede notar MySQL debe estar instalado en el directorio raíz; otra forma de editar este archivo es mediante la aplicación WinMySQLAdmin 1.4 que se encuentra en C:\mysql\bin\., dirigiéndonos a la pestaña my.ini Setup y agregándole las líneas anteriormente citadas.

- Cuando la base de datos se crea con el DBDesigner 4[27] se debe conectar el modelo a una base de datos, pero hay que tener cuidado si se va a crear la base de datos en el localhost; se debe escoger el localhost del Network Hosts para ubicar o crear su base de datos, de tal forma que cuando use MySQL modo consola se facilite el acceso a las bases de datos. Este factor también se debe tener en cuenta al momento de transporta la base de datos de una maquina a otra.

[26] MySQL 4.0.18
[27] Ver Anexo

- Cuando se diseña la base de datos se debe tener en cuenta la integridad referencial y en MySQL el tipo de tabla que la soporta debe ser InnoDB; en la tabla donde se hace la referencia debe existir un índice con las llaves foráneas.

- Si se usa el DBDesigner 4 para crear la base de datos y desea trasladarla a otra maquina es recomendable guardar el modelo XML dentro de la misma base de datos y copiar la carpeta de la base de datos que contiene archivos *.FRM, *.MYI, *.MYD que se encuentra en C:\mysql\data y se copia en el mismo destino de la otra máquina.

- Para enlazar Visual Basic 6.0 con la base de datos se usa un ODBC llamado MySQL ODBC 3.51 el cual debe ser descargado del sitio www.mysql.com/downloads/.

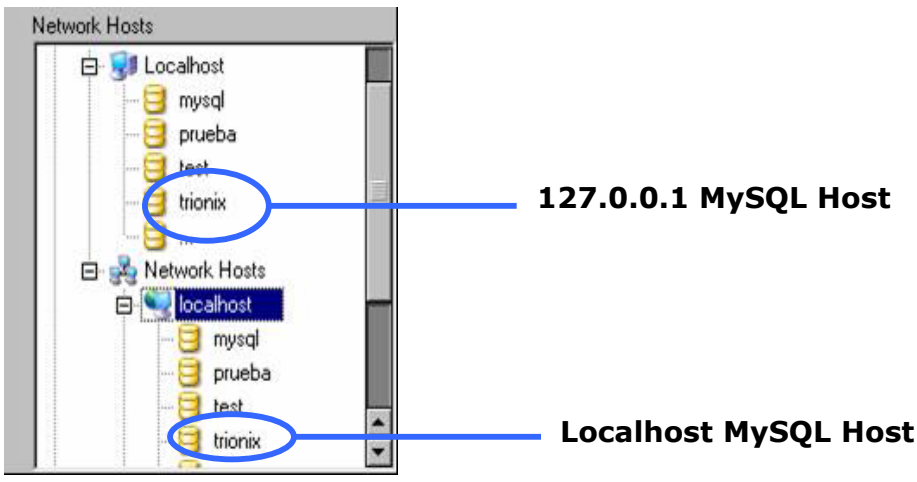


Figura. A4 *Uso del DBDesigner 4*

- El tipo de servidor de MySQL que se escogió fue el mysqld que es un compilador normal con todas las funcionalidades; en primera instancia se pensó en el Mysqld-nt el cual soporta conexiones temporales (pipes) y no presentó problemas con Windows XP. Pero al momento de hacer las pruebas se encontró que Windows 9X,Milenium no soportan este tipo de servidor, lo cual afectaba nuestra migración de plataforma.

- Las tablas de la Base de datos deben ser de tipo InnoDB para soportar transacciones e integridad referencial, pero el tipo de servidor mysqld que se escogió no las soporto en la pruebas realizadas, por lo cual se opto por el tipo de tabla ISAM que no posee algunas bondades pero que funcionó correctamente en todas las máquinas.

El transporte de la base de datos de una máquina a otra de similar arquitectura consiste en tomar la carpeta con el nombre de la base de datos y copiarla a la otra máquina, dentro de esta carpeta se encuentra tres archivos por cada tabla de la base de datos; las extensiones de estos archivos varia de acuerdo al tipo de tabla que sea. En este caso la extensiones son: .frm, .ISM, .ISD

ANEXO 4. BANCO DE DATOS

TABLAS DE LA BASE DE DATOS

TABLA	*INFORMACIÓN*
Actualizaciones	Información del proyecto luego de que se produce la renovación
Clientes	Clientes que usan aplicaciones protegidas
Desarrollador	Desarrolladores de las aplicaciones que van a ser protegidas
Protección	Describe el tipo de protección con la cual se protege
Proyectos	Proyectos creados apartir de aplicaciones protegidas
Usuarios	Usuarios del sistema, actores de los casos de uso
Proyectos_has_Desarrollador	Desarrolladores

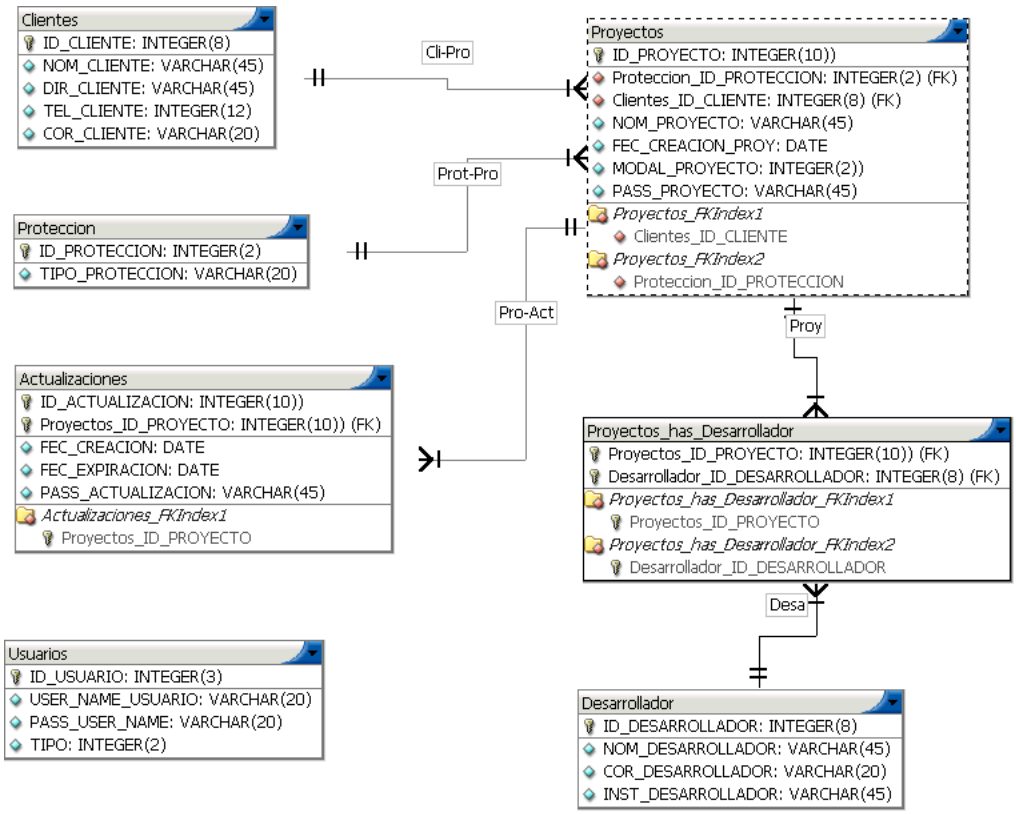


Figura A5. Modelo entidad relación

INFORMACION DE LAS TABLAS

TABLA ACTUALIZACIONES

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
ID_ACTUALIZACION	INTEGER	10	SI	SI	identificador de la actualización
FEC_CREACION	DATE	8	SI	NO	Fecha de inicio del periodo de la actualización
FEC_EXPIRACION	DATE	8	SI	NO	Fecha de fin de periodo de la actualización
PASS_ACTUALIZACION	VARCHAR	45	SI	NO	Password o clave del proyecto protegido de la actualización

Llave foránea	*Referencia a:*	
	Tabla	*Llave Primaria*
ID_PROYECTO	Proyectos	ID_PROYECTO

TABLA CLIENTES

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
ID_CLIENTE	INTEGER	8	SI	SI	identificador del cliente
NOM_CLIENTE	VARCHAR	45	SI	NO	Nombre del cliente que usara la aplicación protegida
DIR_CLIENTE	VARCHAR	45	NO	NO	Direccion del domicilio del cliente
TEL_CLIENTE	INTEGER	12	NO	NO	Teléfono del cliente
COR_CLIENTE	VARCHAR	45	NO	NO	Correo electrónico del cliente

TABLA PROTECCION

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
ID_PROTECCION	INTEGER	2	SI	SI	identificador de la protección
TIPO_PROTECCION	VARCHAR	20	SI	NO	Especificación del tipo de protección

TABLA USUARIOS

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
ID_USUARIO	INTEGER	3	SI	SI	identificador de usuario
USER_NAME_USUARIO	VARCHAR	20	SI	NO	Nombre del usuario para efecto de identificación en la herramienta
PASS_USER_NAME	VARCHAR	20	SI	NO	Password para garantizar la integridad
TIPO	INTEGER	2	SI	NO	Tipo de usuario

TABLA PROYECTOS

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
ID_PROYECTO	INTEGER	10	SI	SI	identificador del proyecto
NOM_PROYECTO	VARCHAR	45	SI	NO	Nombre que recibe el proyecto
FEC_CREACION	DATE	8	SI	NO	Fecha de creación del proyecto
MODAL_PROYECTO	INTEGER	2	SI	NO	Involucra cuatro tipo de modalidades de proyecto
PASS_PROYECTO	VARCHAR	45	SI	NO	Password del proyecto

Llave foránea	*Referencia a:*	
	Tabla	*Llave Primaria*
ID_CLIENTES	Clientes	**ID_CLIENTES**
ID_PROTECCION	Proteccion	**ID_PROTECCION**

TABLA DESARROLLADOR

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
ID_DESARROLLADOR	INTEGER	8	SI	SI	identificador de la actualización
NOM_DESARROLLADOR	VARCHAR	45	SI	NO	Fecha de inicio del periodo de la actualización
COR_DESARROLLADOR	VARCHAR	20	SI	NO	Fecha de fin de periodo de la actualización
INST_DESARROLLADOR	VARCHAR	45	NO	NO	Password o clave del proyecto protegido de la actualización

TABLA PROYECTOS_HAS_DESARROLLADOR

Campo	*Tipo*	*Longitud*	*Requerido*	*Llave primaria*	*Descripción*
Proyectos_ID_PROYECTO	INTEGER	10	SI	SI	identificador del proyecto
Desarrollador_ID_DESARROLLADOR	INTEGER	8	SI	SI	identificador del desarrollador que participa en varios proyectos

Llave foránea	***Referencia a:***	
	Tabla	***Llave Primaria***
ID_PROYECTO	Proyectos	**ID_PROYECTO**
ID_DESARROLLADOR	Desarrollador	**ID_DESARROLLADOR**

HERRAMIENTA SOFTWARE PARA LA PROTECCION CONTRA INSTALACION Y USO NO LICENCIADO DE SOFTWARE

(TRIONIX)

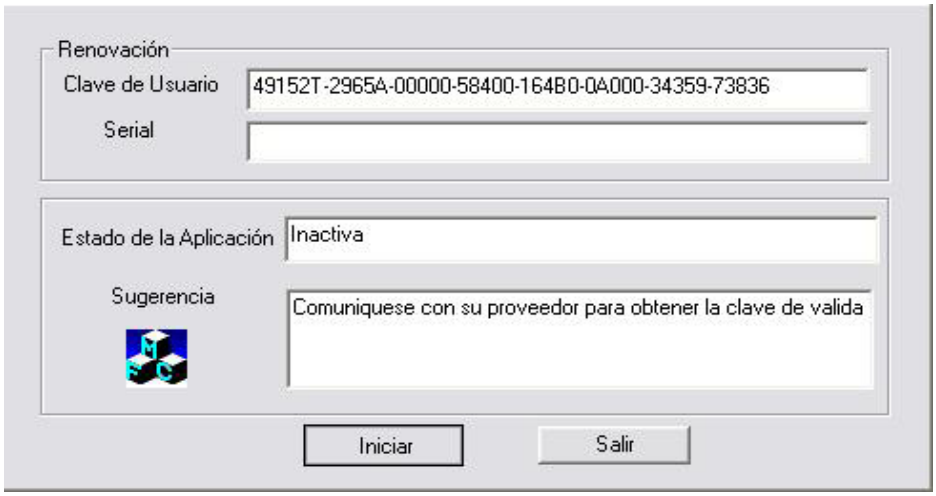
MANUAL DEL USUARIO

Juan Gabriel Quintero Peña
William Villamizar Vera

Universidad Industrial de Santander
Facultad de Ciencias Físico-mecánicas
Escuela de Ingeniería de Sistemas
Bucaramanga, 2004

APLICACIÓN PROTEGIDA

Luego de que una aplicación ha sido protegida debe ser ejecutada en la maquina del cliente. En la primera ejecucion se presenta la siguiente visualizacion.



La clave de usuario que genera la proteccion, es la misma que debe ser introducida al momentor de ejecutar la opcion de renovar, este dara como producto el serial que sera devuelto al cliente quien lo introducira y continuara con la ejecucion normal de su aplicación.

Cuando la proteccion se venza sacara un pantallita igual que permita ponerse en contacto con el proveedor o con el administrador de trionix

ZONA DEL SISTEMA

Las principales zonas son producto de la funcionalidad mas no de la distribucion de la interfaz

Zona	Descripción
Proyecto	Visualiza y administra la información involucrada en un proyecto, incluyendo reportes
Administración	Involucra el manejo de usuario y calidad de la información contenida en la base de datos
Protección	Involucra la proteccion y la renovacionde protecciones

Estos son los tipos de usuarios que pueden acceder a la aplicaion con sus roles

ACTOR	DESCRIPCION	RESPONSABILIDADES	NECESIDADES
Usuario ***(admin, Desarrollador)***	Representa a un individuo que usa la herramienta con el fin de proteger y renovar la aplicación software de un tercero	✣ Hacer un uso adecuado de la herramienta sacándole el máximo de rendimiento. ✣ Recurrir a la misma para realizar las tareas de renovación y asignación de protección antes de ser entregada al usuario final. ✣ Realizar labores de administración y mantenimiento del sistema.	El usuario utiliza la herramienta para: ✣ Proteger por numero de accesos una aplicación software ✣ Proteger por fecha una aplicación software ✣ Renovar la protección de la aplicación cuando el cliente así lo requiera. ✣ Obtener información de las aplicaciones protegidas mediante la base de datos.
Cliente	Representa a una persona o a una empresa que hace uso de la aplicación protegida.	✣ Usar la aplicación protegida dentro de los rangos permitidos por la herramienta ✣ Ponerse en contacto con los proveedores de la aplicación para realizar la renovación de la protección	El cliente utiliza el sistema para: ✣ Recurrir a este cuando necesita renovar la protección de la aplicación. ✣ Solicitar la clave de uso cuando ocurra algún problema con la

			aplicación protegida.

MAS AYUDA

Si desea consultar el menual mas completo puede acceder la ayuda de la herramienta que se identifica por un logo como el siguiente