Universidad de las Ciencias Informáticas

FACULTAD 1

Herramienta para manipulación de archivos de forma segura

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

Autor(es): Olber Rios Cutiño
Frank Geiler Vega Duverger

Tutor(es): Ing. Wilhem Manuel Verano
Ing. Félix Alejandro Prieto Carratala

20 Junio de 2013

DECLARACIÓN DE AUTORÍA

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

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

Olber Rios Cutiño

_____

Firma del Autor

Frank Geiler Vega Duverger

_____

Firma del Autor

Ing. Wilhem Manuel Verano

_____

Firma del Tutor

Ing. Félix Alejandro Prieto Carratala

_____

Firma del Tutor

DATOS DE CONTACTO

AGRADECIMIENTOS

DEDICATORIA

RESUMEN

En el presente trabajo de diploma se explican conceptos claves tales como Criptografía, Firma Digital, Certificado Digital, PKCS #7 y se realiza un estudio del arte identificando aplicaciones que implementen funcionalidades similares al producto desarrollado, además especificando las características de las herramientas, tecnologías y arquitectura utilizadas en el proceso de desarrollo y se generan los artefactos definidos por la metodología *Extreme Programming* (XP).

Se desarrolló una aplicación multiplataforma que utilizando las estructuras de datos del estándar permita el cifrado y la firma de archivos, además crear una lista de control de acceso para controlar que usuarios tienen permisos para modificar el fichero de extensión .saedzip que se genera al encriptar; garantizando así la confidencialidad, autenticidad e integridad de los datos.

Teniendo en cuenta los requerimientos de los clientes plasmados en las historias de usuario se desarrollaron las funcionalidades de la propuesta de solución y se prueba su correcto funcionamiento realizando las pruebas unitarias y de aceptación en cada iteración del producto concebida en el ciclo de desarrollo del *software*.

Palabras claves: PKCS #7, Criptografía, Firma Digital, Certificado Digital, ACL

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

La seguridad y el control de acceso de la información de los archivos informáticos son de gran valor para cualquier entidad. Actualmente se vive en una sociedad donde el intercambio de todo tipo de datos es algo común. En este intercambio los datos se envían a través de un soporte físico no seguro, el cual puede ser leído por una tercera entidad que intercepte el mensaje. En ocasiones se desea que el mensaje sea leído solo por el destino y en caso de que terceros lo intercepten, los datos no sean legibles. Además, se desea proteger los documentos que se encuentran almacenados en la computadora de personas que no tengan permiso sobre los mismos.

Existen varios mecanismos de protección de la información contenida en los archivos, uno de ellos es enviar la información a través de un canal protegido físicamente, otro, es enviar el mensaje a través de sistemas que lo defiendan durante el camino, como es el caso de la protección de mensajes mediante personal de seguridad, pero estos mecanismos de protección de mensaje han demostrado a lo largo de la historia que no son del todo eficientes.

La Criptografía es un conjunto de técnicas que han demostrado que solucionan el mecanismo de protección de la información de los datos. Esta es capaz de convertir una entrada de datos legibles, que solo es entendible para los receptores, basado en métodos computacionales y matemáticos, garantizando la confidencialidad, integridad y autenticidad de la información. Dentro de las aplicaciones de la criptografía se pueden mencionar: el cifrado de la información y la firma digital.

Por otro lado, no solo es necesario proteger la información, sino tener un control de acceso sobre la misma. De esta forma, se tiene un control de quienes tienen permisos sobre un archivo y las acciones que pueden realizar sobre él (lectura, lectura-escritura). Un mecanismo para gestionar los permisos sobre los archivos son las listas de control de acceso (ACL), las cuales determinan los roles y los permisos de cada usuario.

Muchas personalidades y empresas en el mundo se han dedicado al estudio y desarrollo de aplicaciones informáticas con el fin de obtener comunicaciones seguras, lo cual ha generado una serie de *software* con características específicas y que utilizan diferentes combinaciones

de técnicas criptográficas, proporcionando al sistema el nivel de seguridad necesario según los requerimientos de estos. Estas combinaciones de algoritmos se estructuran finalmente en forma de protocolos, para proporcionar métodos de comunicación segura normalizados.
Entre los protocolos y aplicaciones desarrolladas se pueden encontrar:

- ✓ **Protocolos**
  - **SSL**
  - **TSL**
  - **IPsec**
- ✓ **Aplicaciones**
  - **PGP**
  - **GPG**

En Cuba existen leyes para la protección de la información a través del uso de técnicas criptográficas tales como: el Decreto Ley número 199 por el que se regula la Seguridad y Protección de la Información Oficial, del 25 de noviembre de 1999 y la Resolución del Ministerio del Interior que pone en vigor el Reglamento para la criptografía y el servicio central cifrado en el exterior, del 2 de julio de 2002, siendo el MININT (Ministerio del Interior) el organismo que rige el cumplimiento y control de estas legislaciones. Estas leyes regulan los trámites de información y almacenamiento de la misma, usados fundamentalmente en el comercio electrónico y transacciones bancarias.
En la Universidad de las Ciencias Informáticas (UCI) se encuentra el Centro de Identificación y Seguridad Digital (CISED) en el cual se han realizado investigaciones y desarrolladas diversas aplicaciones para la protección de la información.

Partiendo del análisis anterior surge la siguiente ***situación problemática***: existen en la actualidad mecanismos para la protección de la información y uno de ellos es el cifrado de archivos mediante técnicas criptográficas que garantizan la confidencialidad, la autenticidad y la integridad. Estos mecanismos aunque garantizan el control de acceso a la información a nivel de sistemas operativos no lo incluyen de manera intrínseca, lo que impacta la portabilidad de los elementos de control de acceso.

Con el análisis de lo antes expuesto y con el fin de dar solución a la situación problemática, se plantea el siguiente ***problema de la investigación***: ¿Cómo mejorar el control de acceso a los archivos protegidos con técnicas criptográficas?

Para dar solución al problema científico queda definido como ***objeto de estudio***: La criptografía como mecanismo de protección de información.

Para dar solución al problema existente esta investigación se plantea como ***objetivo general***: desarrollar una aplicación informática que permita controlar el acceso a archivos cifrados para incrementar la seguridad de los datos almacenados, a partir de la utilización de técnicas criptográficas.

A partir del análisis del ***objetivo general*** se derivaron los siguientes ***objetivos específicos***:

- Identificar las herramientas y estándares disponibles para la manipulación de archivos cifrados, ventajas y limitaciones.
- Elaborar una propuesta de solución que cumpla con las funcionalidades necesarias para resolver el problema de la investigación.
- Implementar una aplicación informática que permita el control de acceso asociado a los archivos cifrados.
- Probar la aplicación desarrollada y comprobar que cumple con los requerimientos planteados.

Queda definido para esta investigación como ***campo de acción***, el control de acceso en archivos protegidos mediante técnicas de cifrado y firma digital.

Para dar cumplimiento a los objetivos antes mencionados se definieron las siguientes ***tareas de investigación***:

- Procesamiento y evaluación de la información obtenida de las referencias bibliográficas.
- Selección de los estándares más convenientes para esta investigación.
- Estudio del arte en el desarrollo del *software* para encontrar ventajas y limitaciones de los sistemas criptográficos.
- Selección de las tecnologías, herramientas, metodología y lenguaje necesarios para el desarrollo del producto.
- Propuesta de requisitos funcionales y no funcionales del sistema.

- Propuesta arquitectónica que cumpla con los requerimientos de la aplicación y perfeccione las limitaciones anteriores.
- Implementación de la aplicación informática.
- Pruebas a la aplicación para validar que cumple con los objetivos trazados en esta investigación.

Para el avance exitoso de la investigación se propone la utilización de los ***métodos investigativos*** siguiente:

- Teóricos
  - Histórico-Lógico: Permite profundizar en el estudio del fenómeno, su evolución y su desarrollo en diferentes períodos de la historia. Investiga las leyes generales del funcionamiento y desarrollo, descubriendo la lógica objetiva del desarrollo histórico basado en el objeto de estudio.
  - Analítico-Sintético: Implica la descomposición del fenómeno en sus partes constitutivas y obtener las características generales, organizando y sintetizando sus partes para desarrollar una correcta estructura de la investigación.

Con esta investigación se define como ***justificación de la investigación:*** un *software* que permita la manipulación de archivos de forma segura, a través de técnicas criptográficas y garantice control de acceso a nivel de fichero.

El desarrollo de esta investigación está estructurado en tres capítulos los cuales se describen a continuación:

Capítulo 1: Fundamentación teórica

En este capítulo se definirán los principales aspectos de la criptografía, definiendo los principales estándares y algoritmos criptográficos de firma digital, función resumen y cifrado de archivos, agregándole una lista de control de acceso; además se realizará un estudio del arte de las principales herramientas similares a este, así como la selección de la metodología de desarrollo de *software*, herramientas y tecnologías a utilizar para esta investigación.

Capítulo 2: Propuesta de solución

En este capítulo se describen los diferentes artefactos de las primeras fases de la metodología programación extrema, planificación y diseño; se hace el levantamiento de requisitos para la realización de las historias de usuario y una propuesta de solución que cumpla con los requerimientos planteados; se realiza el plan de iteraciones y se define la arquitectura, además se definen los patrones de diseño a utilizar en esta investigación.

Capítulo 3: Implementación y prueba

En este capítulo se da cumplimiento al plan de iteración a través de las fases de implementación y prueba describiendo los principales artefactos de las mismas, obteniendo el correcto funcionamiento de los requerimientos especificados.

Capítulo 1: Fundamentación teórica

Introducción

A continuación, en este capítulo se realizará un estudio sobre los principales conceptos sobre criptografía, el estado del arte de varias aplicaciones que utilizan sus algoritmos y por último un análisis sobre las herramientas, tecnologías y metodologías a utilizar en esta investigación. La criptografía se remonta a miles de años, desde sus inicios hasta la actualidad la criptografía ha tenido un avance tecnológico extraordinario, proporcionando métodos de cifrados cada vez mejores.

La palabra “Criptografía” proviene de las palabras griegas “kryptos” (oculto) y “gráphein” (escritura). La criptografía es una rama de las matemáticas capaz de transformar un mensaje legible (texto en claro), en uno no legible (cifrado) al cual solo puedan tener acceso personas autorizadas (Lezcano, 2003). La criptografía garantiza confidencialidad, integridad y autenticidad de los datos, lo que ha hecho que sea la técnica de protección de la información más utilizada. A continuación se presentan las dos grandes ramas de esta, criptografía simétrica y asimétrica o de clave pública, exponiendo sus principales características, su funcionamiento, principales algoritmos y desventajas.

1.1 Criptografía Simétrica

La criptografía convencional es también llamada de clave secreta, privada o sistema de cifrado simétrico. Esta consiste en el envío de un mensaje cifrado de un emisor, con una clave secreta mediante un algoritmo simétrico, a través de un canal físico inseguro, a un receptor que descifra mediante el mismo algoritmo de cifrado y la misma clave secreta (ver Figura 1). Su característica fundamental es que la misma clave que se utiliza para cifrar se usa para descifrar (Lezcano, 2003). La principal desventaja del cifrado simétrico es la distribución de las claves, ya que muchas personas deben conocer la misma clave. La criptografía de clave secreta fue el primer tipo de cifrado que surgió, el cual se clasifica en dos grupos, cifrado por bloques y cifrado por flujo.

Figura 1: Sistema criptográfico simétrico. Fuente (Laboratorio de Redes y Seguridad. UNAM - Facultad de Ingeniería, División de Ingeniería Eléctrica, 2012)

Cifrado en Bloques

Operan dividiendo el mensaje que se pretende codificar en bloques de tamaño fijo, y aplican sobre cada uno de ellos una combinación más o menos compleja de operaciones de confusión-sustituciones y difusión- transposiciones. Existen algunos algoritmos que aplican este tipo de cifrado como DES, Triple DES e IDEA en aplicaciones bancarias y comercio electrónico por su gran velocidad de cifrado, pero tiene varios puntos débiles como, la mala distribución y gestión de las claves, además no presenta firma digital (Lezcano, 2003).

Cifrado por Flujo

En 1917, Joseph Oswald Mauborgne y Gilbert Sandford Vernam inventaron un criptosistema perfecto según el criterio de Claude Shannon[1]. Dicho sistema consistía en emplear una secuencia aleatoria de igual longitud que el mensaje, que se usaría una vez y las claves deben ser equiprobables. Este método presenta el grave inconveniente que la clave es tan larga como el propio mensaje. Existe una debilidad intrínseca a todos los métodos de cifrado de flujo que vale la pena destacar, si un atacante conoce parte del texto claro, podrá sustituirlo por otro sin que lo advierta el legítimo destinatario (López, 2004).

[1] Claude Elwood Shannon (1916-2001), ingeniero eléctrico y matemático, recordado como "el padre de la teoría de la información."

Algoritmos Simétricos

DES (*Data Encryption Standard*)

Es un esquema de cifrado simétrico desarrollado en 1977 por el Departamento de Comercio y la Oficina Nacional de Estándares de EE.UU en colaboración con la empresa IBM[2] .Tiene como entrada un bloque de 64 bits del mensaje y lo somete a 16 iteraciones, el descifrado se realiza mediante la misma clave, haciendo el proceso inverso. Dependiendo de la naturaleza de la aplicación DES tiene 4 modos de operación para poder implementarse:

- ECB (*Electronic Codebook Mode*) para mensajes cortos, de menos de 64 bits.
- CBC (*Cipher Block Chaining Mode*) para mensajes largos.
- CFB (*Cipher Block Feedback*) para cifrar *bit* por *bit* o *byte* por *byte*.
- OFB (*Output Feedback Mode*) el mismo uso que CFB, pero evitando propagación de error.

DES fue el algoritmo más utilizado en la década de los 90 del siglo XIX. Hoy en día es vulnerable por la longitud de su clave, con ataques de fuerza bruta (Lezcano, 2003).

IDEA (*International Data Encryption Algorithm*)

Es más joven que DES, pues data de 1992. Para muchos constituye el mejor y más seguro algoritmo simétrico disponible en la actualidad. Trabaja con bloques de 64 bits de longitud y emplea una clave de 128 bits. Como en el caso de DES, se usa el mismo algoritmo tanto para cifrar como para descifrar. Dentro de las características del algoritmo es que no tiene claves débiles por lo que, hace casi imposible el ataque de fuerza bruta (López, 2004).

AES (*Advanced Encryption Standard*)

También conocido como Rijndael[3], es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Fue el resultado de un concurso lanzado por el Instituto Nacional de Estándares y Tecnología (NIST), publicado a finales de 1997. Es un algoritmo simétrico de cifrado por bloques, donde las longitudes del bloque y de la llave son variables (López, 2004).

[2] IBM(International Business Machines), empresa multinacional estadounidense de tecnología y consultoría con sede en Armonk, Nueva York

[3] Rijndael fue un refinamiento de un diseño anterior de Daemen y Rijmen. Consiste en una red de sustitución-permutación.

1.2 Criptografía Asimétrica

Los algoritmos asimétricos o de llave pública fueron introducidos por Whitfield Diffie[4] y Martin Hellman[5] a mediados de los años 70. La diferencia fundamental con los algoritmos simétricos es que las claves no son únicas sino que se generan en pares (una pública y una privada) para una misma persona, una clave para cifrar y la otra descifrar indistintamente. Otra diferencia es que las claves generalmente son mucho más grandes que los criptosistemas simétricos, lo que hace que la complejidad de cálculo de estos sea considerablemente más lentos que los algoritmos de cifrado simétricos, por lo que se utilizan generalmente para cifrar una clave de sesión (simétrica).Para la seguridad de este tipo de criptosistema debe cumplirse que a partir de una llave sea extremadamente difícil conocer la otra (López, 2004).Este procedimiento consiste en el envío de un mensaje cifrado mediante un algoritmo asimétrico, con la clave pública del receptor a través de un canal inseguro, el cual el receptor descifra mediante el mismo algoritmo de cifrado, con su clave secreta (ver Figura 2).

Figura 2: Sistema criptográfico asimétrico. Fuente (Laboratorio de Redes y Seguridad. UNAM - Facultad de Ingeniería, División de Ingeniería Eléctrica, 2012)

[4] Whitfield Diffie, en 1965 se graduó de matemático en el Instituto Tecnológico de Massachusetts. En 1976 publicó junto a Martin Hellman "New Directions in Cryptography", convirtiéndose en pionero de la criptografía asimétrica.

[5] Martin Hellman, se graduó en la Universidad de New York en 1966 con el título de Ingeniería Eléctrica. Hellman es famoso por ser el inventor junto a Diffie de la criptografía de clave pública.

Algoritmos de Cifrado Asimétrico

RSA

Este algoritmo fue propuesto por Ronald **R**ivest[6], Adi **S**hamir[7] y Leonard **A**dleman[8] en 1978, es el más conocido y versátil de los algoritmos de clave pública usados actualmente. RSA se basa en la dificultad para factorizar grandes números. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos primos grandes (López, 2004).

El Gamal

Fue diseñado en un principio para producir firmas digitales en 1984, pero posteriormente se extendió también para codificar mensajes. Se basa en el problema de los logaritmos discretos, que está íntimamente relacionado con el de la factorización y en el de Diffie-Hellman, es usado en *software* libre como en GNUPG (*GNU Privancy Guard*)[9] y además este algoritmo no está bajo ninguna patente (López, 2004).

1.3 Función *Hash*

Una función *hash* (resumen) es un algoritmo que crea una representación digital en la forma de un valor *hash* o resultado *hash* de una longitud estándar, el cual representa un resumen del documento. Este resultado es usualmente mucho más pequeño que el documento, pero sin embargo sustancialmente único (Braicovich, 1999).

Las funciones hash deben tener ciertas propiedades para hacerlas suficientemente seguras para su uso criptográfico.

Según (Braicovich, 1999) sea M el mensaje y h (M) el resultado de aplicar la función hash al mensaje M

- Debe poder convertir un mensaje de longitud arbitraría M en uno de longitud fija h (M)

[6] Ronald L. Rivest, nacido en 1947 en Schenectady, (Nueva York). Criptógrafo y profesor de ciencias de la computación en el departamento de ingeniería eléctrica y ciencias de la computación del MIT, crea el algoritmo RSA. Además es creador de otros algoritmos como RC2, RC4, RC5 y las funciones hash MD2, MD5, SHA y SHA-1.

[7] Adi Shamir es un criptógrafo Israelí. Fue uno de los inventores del algoritmo RSA (junto con Rivest y Adleman), y ha hecho numerosas contribuciones a los campos de la criptografía y las ciencias de la computación.

[8] Leonard Adleman es profesor en ciencias de la computación y biología molecular de la Universidad del Sur de California. Conocido por ser uno de los inventores de algoritmo RSA en 1977 y de la computación por ADN.

[9] GnuPG es un *software* libre alternativo, que permite cifrar y firmar datos.

- Debe ser unidireccional, es decir, dado un valor Y debe ser computacionalmente imposible encontrar M tal que h (M) =Y

Se debe poder asegurar que es computacionalmente imposible encontrar dos mensajes M y M' tales que h (M) = h (M ').

1.4 Estándar Criptográfico de Clave Pública (PKCS)

PKCS #7 es un estándar sobre la sintaxis del mensaje criptográfico (RFC - Petición de Comentarios - 2315), usado para firmar y/o cifrar mensajes en PKI (Infraestructura de Clave Pública) y para la diseminación de certificados, además es la base para el estándar S/MIME[10]. El estándar está actualmente basado en la RFC 5652, una actualización del PKCS #7 al CMS (Sintaxis de Mensajes Criptográficos), utilizado para firmar digitalmente, obtener el resumen, autenticar, o cifrar arbitrariamente el contenido de un mensaje, es un formato para representar mensajes protegidos criptográficamente. Cuando la protección está basada en criptografía de clave pública, en PKCS #7 se utilizan certificados X.509 para garantizar la autenticidad de las claves (Perramon, 2004).

Tabla 1: Estructura general de un mensaje PKCS #7. Fuente (Perramon, 2004)

Campo	Tipo
contentType	identificador único
content (opc.)	*Data*
	SignedData
	EnvelopedData
	SignedAndEnvelopedData
	DigestedData
	EncryptedData

El campo *contentType* es un identificador que indica cuál de las seis estructuras posibles hay en el campo *content*. Estas estructuras son:

[10] S/MIME (*Secure / Multipurpose Internet Mail Extensions*) es un estándar para criptografía de clave pública y firmado de correo electrónico.

1. *Data*: sirve para representar datos literales, sin aplicarles ninguna protección criptográfica.
2. *SignedData*: representa datos firmados digitalmente.
3. *EnvelopedData*: representa un mensaje con sobre digital (es decir, un mensaje cifrado simétricamente al que se añade la clave simétrica cifrada con la clave pública de cada destinatario).
4. *SignedAndEnvelopedData*: representa datos firmados y "cerrados" en un sobre digital. Esta es la estructura escogida para realizar el esquema de la solución de esta investigación.
5. *DigestedData*: representa datos a los cuales se les añade un resumen o *hash*.
6. *EncryptedData*: representa datos cifrados con clave secreta.

El campo *content* es opcional, porque en ciertos casos existe la posibilidad de que los datos de un mensaje no estén dentro del propio mensaje, sino en otro lugar o no posean (Perramon, 2004).

1.5 Certificado digital

El certificado digital es un documento creado y firmado por una autoridad certificadora (CA por sus siglas en inglés), el cual contiene datos significativos del propietario como el nombre del sujeto, la fecha de expiración y su clave pública, la cual se usa para cifrar mensajes. Si el certificado es auténtico y se confía en la CA entonces se puede confiar en el sujeto y autenticar el documento conociendo la clave pública de la CA.

Certificado digital X509v3.

Es un estándar de la Unión Internacional de Telecomunicaciones (UIT), para infraestructura de clave pública. X509 especifica un formato estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación.

Los formatos de codificación más comunes son *Distinguished Encoding Rules* (DER) o *Privacy Enhanced Electronic Mail* (PEM). Se definen un conjunto de atributos los cuales nos brindan la posibilidad de estandarizar estos atributos a las necesidades de los usuarios. Entre las extensiones de archivo de certificados X.509v3 se encuentran:

- ***.PEM*** - Certificado codificado en Base64.

- ***.P12*** – PCKC #12, puede contener certificados públicos y claves privadas protegidas con clave.
- ***.DER*** - Certificado codificado en DER.

Campos de X.509v3.

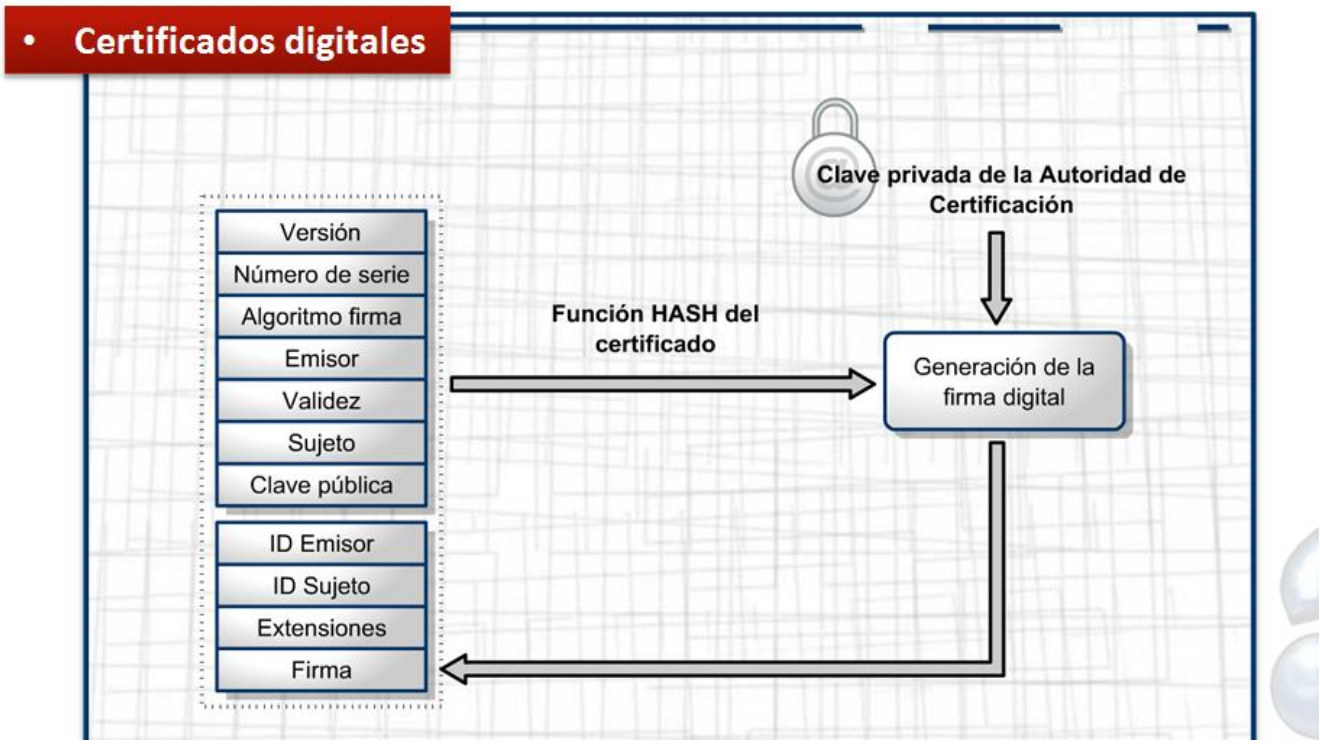


Figura 3: Campos de un certificado X.509v3. Fuente (Penalver, 1998)

- **Versión:** versión del estándar X.509, generalmente versión 3, que es la más actual.
- **Número de serie:** número identificador del certificado, único para cada certificado expedido por una autoridad certificadora determinada.
- **Algoritmo de firma:** algoritmo criptográfico usado para la firma digital.
- **Autoridad Certificadora:** datos sobre la autoridad que expide el certificado.
- **Fechas de inicio y de fin de validez del certificado**: Definen el tiempo de validez del mismo, que generalmente es de un año.
- **Propietario:** persona o entidad vinculada al certificado. Dentro de este apartado se usan una serie de abreviaturas para establecer datos de identidad. Entre ellas:

CN (nombre común del usuario), O (organización), L (ciudad), S (estado o provincia), C (país), E (correo electrónico), UID (ID de usuario).

- **Clave pública:** representación de la clave pública vinculada a la persona o entidad (en hexadecimal), junto con el algoritmo criptográfico para el que es aplicable.
- **Extensiones:** permite al administrador definir políticas específicas de aplicación, las políticas de emisión de certificados, tipos de materias, y los atributos clave de uso para una plantilla de certificado.
- **Firma de la Autoridad Certificadora**: asegura la autenticidad del mismo (Perramon, 2004).

1.6 Almacén de llaves

Es una tienda de gestión de claves y certificados almacenados, permite a los usuarios administrar sus propios pares de claves privadas, públicas y los certificados asociados a estas últimas, para el uso de los servicios de integridad de datos y autenticación, en el uso de cifrado y firmas digitales de mensajes.

1.7 *Request for Comments* (RFC)

La Petición de Comentarios es una serie de notas sobre internet, que incluye varias categorías (informativos, nuevos, históricos), con una propuesta oficial, que para su utilización debe estrictamente seguir el proceso del RFC, para alcanzar la calidad requerida.
A continuación, el RFC utilizado para el desarrollo de esta investigación.
RFC 5652 – CMS (*Cryptographic Message Syntax*)

- Este RFC describe la sintaxis de mensajes criptográficos (CMS), la cual se utiliza para firmar digitalmente, autenticar o cifrar contenido. CMS se deriva a partir de PKCS #7 siendo totalmente compatible con el mismo.
- CMS describe una sintaxis de encapsulación para la protección de datos, permitiendo realizar múltiples empaquetados, uno dentro de otro.

1.8 Librería Criptográfica *BouncyCastle*

BouncyCastle (BC, por sus siglas en inglés) es una implementación de algoritmos criptográficos. Está organizado de forma que sus interfaces de programación de aplicaciones

(API, por sus siglas en inglés) sean adaptables para su uso bajo cualquier entorno de infraestructura adicional. *BouncyCastle* nació como resultado del esfuerzo de dos colaboradores que sufrieron la necesidad de recrear librerías criptográficas en cada cambio de empleo. Un requisito de diseño inicial es que existieran versiones de la librería para el entorno Java por lo que existen 2 juegos de librerías. La librería compatible JCE se basa en API de bajo nivel, de modo que el código fuente es un ejemplo de implementación de problemas criptográficos comunes con dichas API. Estas están optimizadas para gestionar eficientemente los algoritmos criptográficos, de forma que se puedan usar en entornos de bajos recursos o no es posible usar las librerías JCE (por ejemplo en un *applets*). La API de peso ligero trabaja con todo, desde el J2ME para el JDK 1.6 y también hay una API en C # que proporciona una funcionalidad equivalente (Winters, 2000).

La librería *BouncyCastle* tiene como características según (Tau Ceti Co-operative Ltd, 2000):

- Un peso ligero de la API de criptografía de Java y C #.
- Un proveedor de *Java Cryptography Extension(JCE)* y la arquitectura *Java Cryptography*.
- Una biblioteca para leer y escribir objetos codificados ASN.1.
- Generadores para la versión 1 y versión 3 de los certificados X.509, versión 2 CRL y archivos PKCS #12.
- Generadores para la versión 2 X.509 certificados de atributos.
- Generadores de S / MIME y CMS (PKCS #7/RFC 5652).
- Generadores de OCSP (RFC 2560).
- Generadores de TSP (RFC 3161yRFC5544).
- Generadores de CMP y CRMF (RFC 4210 y RFC 4211).
- Generadores de OpenPGP (RFC 2440).

1.9 Firma Digital

Una firma digital es una secuencia de *bits* que se añade a un fragmento de información cualquiera, y que permite garantizar su autenticidad. Una firma digital debe cumplir con los siguientes requisitos:

1. Una firma digital válida para un documento no puede ser válida para otro distinto.
2. Solo puede ser generada por su legítimo titular.

3. Es públicamente verificable, cualquiera puede comprobar su autenticidad en cualquier momento, de forma sencilla (López, 2004).

Las normas TS733 y TS903 propuestas por el ETSI (*European Telecommunications Standards Institute*) definen los formatos técnicos de la firma electrónica. La primera se basa en el formato clásico PKCS #7 y la segunda en XMLDsig (*Extensible Markup Language Digital Signature*).

Bajo estas normas se definen tres modalidades de firma:

1. **Firma básica:** incluye el resultado de una operación de *hash* y clave privada, identificando los algoritmos utilizados y el certificado asociado a la clave privada del firmante.
2. **Firma fechada o avanzada:** a la firma básica se añade un sello de tiempo calculado a partir del *hash* del documento firmado por una TSA (*Time Stamping Authority*).
3. **Firma reconocida o completa:** a la firma avanzada, se añade información sobre la validez del certificado procedente de una consulta de CRL[11] o de OCSP[12] realizada a la Autoridad de Certificación (Ramos, 2010).

Características de la Firma Digital

1. **Integridad:** es necesario estar seguro de que los datos que se envían llegan íntegros, sin modificaciones, a su destino final. La integridad se consigue combinando criptografía, funciones hash y firmas digitales.
2. **Autenticidad:** todas las entidades participantes en la transacción deben estar perfecta y debidamente identificadas antes de comenzar la misma. La autenticidad se consigue mediante el uso de los certificados y firmas digitales.
3. **No Repudio:** se debe estar seguros de que una vez enviado un mensaje con datos importantes o sensibles el destinatario de los mismos no pueda negar el haberlos recibido. El no repudio se consigue mediante los certificados y la firma digital.

[11] CLR (*Certificates Revocation List*) lista con los números de serie de los certificados revocados por una entidad certificadora.

[12] OCSP (*Online Certificate Status Protocol*) protocolo que permite determinar si un certificado está revocado o no, a través de un servicio expuesto por la autoridad certificadora que lo emitió.

Proceso de Firmado

El proceso de firmado consiste en proveer a un sistema de cómputo con el documento a firmar y su clave privada (que solo él conoce). El programa cifra con la clave privada el resumen realizado al documento, produciendo como resultado un mensaje encriptado denominado firma digital. Juntos, el documento y la firma constituyen el documento firmado (ver Figura 4).

Generalmente, la firma digital es ligada a su mensaje y almacenada o trasmitida con este. Sin embargo, puede también ser enviado o almacenado como un elemento de datos separados, siempre que este mantenga una asociación confiable con su mensaje. Puesto que una firma digital es única para un mensaje, es inútil si está totalmente desconectada de su mensaje (Braicovich, 1999).

Proceso de Verificación

El proceso de verificación (algunos lo llaman proceso de autenticación) consiste en proveer a un programa, un documento firmado y la clave pública del supuesto firmante, el programa descifra con la clave pública la firma y lo compara con el documento indicando si es auténtico o no (ver Figura 4).

Algoritmos de Firma Digital

SHA-1/RSA

Este método combina SHA-1 como función hash y RSA como algoritmo de cifrado, y el proceso de firma y verificación es de manera semejante al descrito en este epígrafe. Además, el proceso de verificación resulta más rápido que el proceso de firmado, característica que lo hace superior a otros métodos como DSA, debido que lo más común es que un mensaje o documento electrónico sea firmado una vez y verificado varias veces. Actualmente es el método de firma digital más utilizado por las aplicaciones criptográficas y es soportado por muchos protocolos de seguridad (López, 2004).

DSA (*Digital Signature Algorithm*)

Este algoritmo fue diseñado, por David Kravitz[13] en 1991, exclusivamente para el firmado y la verificación de esta, asegurando con esto la integridad de los datos. La seguridad de DSA se basa en la dificultad de calcular logaritmos en un campo finito (López, 2004).

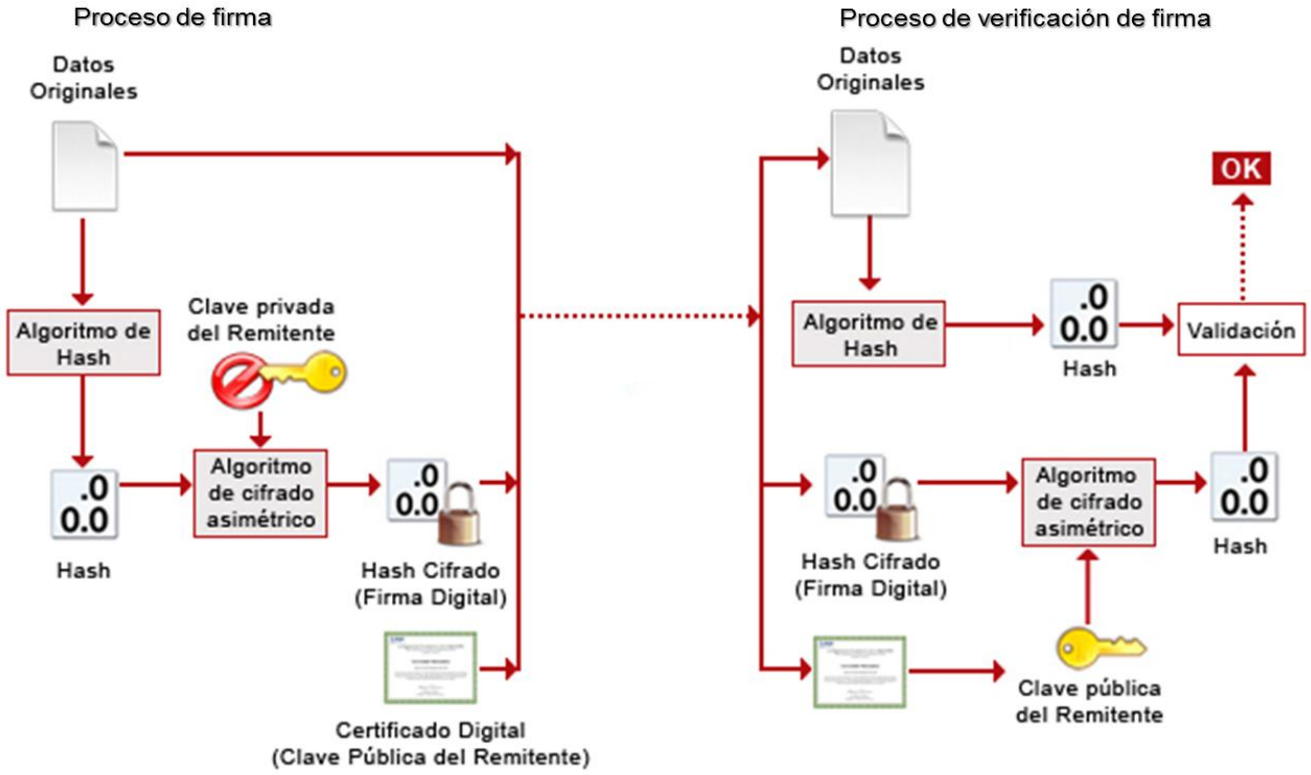


Figura 4: Proceso de creación y comprobación de firma. Fuente (INTECO, 2009)

1.10 Control de Acceso

Las listas de control de acceso permiten gestionar detalladamente los permisos de una aplicación, además manejan principalmente dos sucesos, entidades que solicitan el control de un recurso y las entidades que se quiere controlar, los usuarios son las entidades que quieren controlar un recurso, y en la jerarquía de ACL se les denominan como ARO (por sus siglas en inglés *Access Request Objects*) y las operaciones o datos son las entidades del sistema que se quieren controlar y en la jerarquía de ACL se les denominan como ACO (por sus siglas en inglés *Access Control Objects*).

[13] David Kravitz, es el inventor del algoritmo de firma digital (DSA) estandarizado en FIPS Publicación 186, Estándar de Firma Digital.

El control de acceso es el proceso de decidir quién puede tener permisos o controlar los recursos o sus aplicaciones. Existen varios modelos de control de acceso entre los cuales se encuentran:

- RBAC (Control de Acceso Basado en Roles)

Los usuarios son asignados a uno o varios roles mientras que los permisos y privilegios se asignan a estos roles. Las políticas de control de accesos basado en roles regulan el acceso de los usuarios a la información en términos de sus actividades y funciones de trabajo.

- MAC (Control de Acceso Obligatorio)

El sistema es quién protege los recursos, comparando las etiquetas del sujeto que accede frente al recurso accedido. La autorización para que un sujeto acceda a un objeto depende de los niveles de seguridad que tengan. En este modelo, los usuarios no gestionan el acceso a los objetos.

- DAC (Control de Acceso Discrecional)

Los usuarios deciden como proteger un recurso creado por ellos o del cual son propietarios configurando los permisos establecidos por el sistema para determinar quiénes y con qué nivel de acceso accederán al mismo; la característica principal en este modelo es que el propietario del recurso puede cederlo a un tercero.

- Selección del modelo de control de acceso a utilizar

Se selecciona el control de acceso discrecional como modelo a utilizar en el presente trabajo ya que en la aplicación a implementar el usuario propietario o creador de un recurso utilizando los permisos establecidos por el sistema permite a otros usuarios hacer uso del recurso según el nivel de acceso concedido.

1.11 Metodologías de desarrollo de software

En la actualidad existen diferentes metodologías para el desarrollo de *software* y seleccionar la adecuada para lograr un producto de gran calidad y que cumpla con los requisitos anhelados por el cliente es tarea esencial para lograr el éxito al afrontar un proyecto informático. Las metodologías de desarrollo de *software* según (Roberth G. Figueroa, 2011) se dividen en dos grandes enfoques o ramas que se conocen como metodologías tradicionales y metodologías ágiles, siendo las tradicionales concebidas para hacer un uso exhaustivo de documentación durante todo el ciclo del proyecto y las ágiles hacen énfasis en

la capacidad de respuesta a los cambios en los requisitos, la confianza en las habilidades del equipo y mantener una buena relación con el cliente.

Metodología Tradicional

Las metodologías tradicionales según (Roberth G. Figueroa, 2011) hacen énfasis en la documentación, planificación y procesos (plantillas, técnicas de administración, revisiones), presentan altos costos al implementar un cambio y no ofrecen una buena solución para proyectos donde el entorno es volátil. Una de las principales metodologías tradicionales es RUP[14] la cual centra su atención en cumplir con un plan de proyecto definido en la fase inicial del desarrollo del proyecto.

Metodología Ágil

El término ¨ágil¨ aplicado al desarrollo de *software* nace en una reunión realizada en Febrero de 2001 en Utah, Estados Unidos en la cual se crea La Alianza Ágil[15] que redacta lo que se conoce como el Manifiesto Ágil[16] que resume la filosofía ¨ágil¨.

Las metodologías ágiles según (Calo, Estevez, & Fillottrani, 2010) son modelos que permiten hacer entrega del *software* en partes pequeñas y utilizables conocidas como incrementos, donde el factor humano es fundamental para el éxito del proyecto y donde cada iteración se puede considerar como un pequeño proyecto en el que las actividades de requerimiento, diseño, implementación y pruebas son llevadas a cabo con el fin de producir un subconjunto del sistema final; el proceso se repite varias veces produciendo un nuevo incremento en cada ciclo, por eso el ciclo de desarrollo de las metodologías ágiles se conoce como iterativo e incremental.

Entre las principales metodologías ágiles se encuentran XP, Scrum, ASD, *Crystal Methodologies* las cuales ponen como relevancia que la capacidad de respuesta a un cambio, es más importante que el seguimiento estricto de un plan.

Comparación y selección de la metodología

- Comparación

[14] Rational Unified Process (RUP): es un proceso formal, que provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo.
[15] The Agile Alliance: organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos.
[16] Manifiesto Ágil, disponible en http://www.agilemanifesto.org/iso/es/.

Las diferencias entre las metodologías tradicionales y las metodologías ágiles se representará en la siguiente tabla obtenida de (Letelier, 2006) tomando como referencia no solo el proceso en sí, sino también al contexto de equipo y organización que es más favorable a cada una de estas filosofías de procesos de desarrollo de *software*.

Tabla 2: Comparación entre metodologías agiles y tradicionales. Fuente (Letelier, 2006)

Metodología Ágil	Metodología Tradicional
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos.
Pocos Roles, más genéricos y flexibles.	Roles más específicos.
No existe un contrato tradicional, debe ser bastante flexible.	Existe un contrato prefijado.
Cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Orientada a proyectos pequeños. Corta duración (o entregas frecuentes), equipos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio.	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas usadas en proyectos grandes y con equipos posiblemente dispersos.
La arquitectura se va definiendo y mejorando a lo largo del proyecto.	Se promueve que la arquitectura se defina tempranamente en el proyecto.
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo.	Énfasis en la definición del proceso: roles, actividades y artefactos.
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Se esperan cambios durante el proyecto	Se espera que no ocurran cambios de gran impacto durante el proyecto

- Selección

Luego de realizar un análisis sobre las diferencias entre las metodologías tradicionales y ágiles se ha llegado a la conclusión de que se hará uso de las metodologías ágiles para obtener el producto final debido a los siguientes aspectos:

- El equipo de desarrollo está compuesto por 3 integrantes que trabajan en el mismo sitio y el proyecto es de corta duración.

- Se necesitan pocos roles ya que el equipo de desarrollo es pequeño.
- La arquitectura del producto se va definiendo y mejorando a lo largo del proyecto.
- Se esperan cambios durante el desarrollo del proyecto en los requisitos.
- No existe un contrato firmado con un cliente.
- El cliente es parte del equipo de desarrollo.

Comparación entre metodologías ágiles

Todas las metodologías ágiles se rigen por los principios suscritos por los firmantes del Manifiesto Ágil pero cada una de estas presenta propiedades muy particulares que las diferencia una de otras, por eso en la tabla 3 se representará una comparación entre las metodologías ágiles más utilizadas en el mundo basándose en: vista del sistema como algo cambiante, colaboración entre los miembros del equipo y características propias de la metodología tales como: simplicidad, excelencia técnica, adaptabilidad, prácticas de colaboración y resultados, donde los valores más altos representan una mayor agilidad.

Tabla 3: Comparación entre metodologías ágiles. Fuente (Highsmith, 2002)

	ASD	Crystal [17]	DSDM [18]	FDD[19]	LD[20]	Scrum[21]	XP
Sistema como algo cambiante	5	4	3	3	4	5	5
Colaboración	5	5	4	4	4	5	5
Características Metodología (CM)							

[17] *Cristal Methodologies*: desarrolladas por Alistair Cockburn, se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos.

[18] *Dynamic Systems Development Method* (DSDM): creada en 1994, define el marco para desarrollar un proceso de producción de software y propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente implementación.

[19] *Feature Driven Development* (FDD): sus impulsores son Jeff De Luca y Peter Coad, se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software.

[20] *Lean Development* (LD): creada por Bob Charette's, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente.

[21] *Scrum*: desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle, define un marco para la gestión de proyectos y está especialmente indicada para proyectos con un rápido cambio de requisitos.

Adaptabilidad	5	5	3	3	4	4	3
Simplicidad	4	4	3	5	3	5	5
Excelencia técnica	3	3	4	4	4	3	4
Prácticas de colaboración	5	5	4	3	3	4	5
Resultados	5	5	4	4	4	5	5
Media CM	4.4	4.4	3.6	3.8	3.6	4.2	4.4
Media Total	4.8	4.5	3.6	3.6	3.9	4.7	4.8

Después de realizar un análisis con los datos observados en la tabla 3 las dos metodologías ágiles a tener en cuenta para el desarrollo del producto son ASD y XP.

Adaptive Software Development (ASD)

La metodología ASD según (Highsmith J., 2000) es un modelo de implementación de patrones ágiles para desarrollo de *software* que su funcionamiento es cíclico y reconoce que en cada iteración se producirán cambios e incluso errores.

Características

- Orientado a los componentes (la funcionalidad del producto, características) más que a las tareas en las que se va a alcanzar dicho objetivo.
- La revisión de los componentes se usa para aprender de los errores y volver a iniciar el ciclo de desarrollo.
- Tolerante a los cambios.
- Guiado por los riesgos.
- Iterativo.

Ciclo de vida

- Especular: es donde se inicia y se planifican las características del *software*.
- Colaborar: se desarrollan las características del *software*.
- Aprender: se revisa la calidad y si no se encuentran errores se entrega al cliente.

Ventajas

- Apunta hacia el *Rapid Application Development* (RAD), el cual enfatiza velocidad de desarrollo para crear un producto de alta calidad bajo mantenimiento involucrando al usuario lo más posible.
- Promulga colaboración, la interacción de personas.
- La tercera fase del ciclo de vida, revisión de los componentes, sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.
- Utiliza información disponible acerca de cambios para mejorar el comportamiento del *software*.
- Anticipa cambios y trata automáticamente con ellos dentro de un programa en ejecución, sin la necesidad de un programador.

Desventajas

- Aunque el ciclo entre el aprendizaje y la especulación es bueno permitiendo entregar productos con alta calidad, la prolongación de dicho ciclo por errores o cambios que no son detectados en reuniones anteriores influye tanto en la calidad del producto como en su costo total.

Extreme Programming (XP)

La metodología XP según (Kent, 2000) está centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.

Características

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Programación por parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone una mayor calidad del código escrito de esta manera, el código es revisado y discutido mientras se escribe, es más importante que la posible pérdida de productividad inmediata.
- Frecuente interacción del equipo de programación con el cliente o usuario: se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

- Corrección de todos los errores antes de añadir una nueva funcionalidad: hacer entregas frecuentes.
- Refactorización del código: reescribir ciertas partes del código para aumentar su legibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrán añadir funcionalidades si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Ciclo de vida

- Exploración: los clientes plantean a grandes rasgos los requisitos que son de interés para la primera entrega del producto.
- Planificación de la entrega: los clientes establecen la prioridad de cada requisito y los programadores realizan una estimación del esfuerzo necesario de cada uno.
- Iteraciones: incluye varias iteraciones sobre el sistema antes de ser entregado.
- Producción: requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente.
- Mantenimiento: requiere de tareas de soporte para el cliente.
- Muerte del proyecto: el cliente no tiene más requisitos para ser incluidas en el sistema, por lo que se requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema.

Ventajas

- Apropiado para entornos volátiles.
- Estar preparados para el cambio, significa reducir su coste.

- Planificación más transparente para los clientes, conocen las fechas de entrega de funcionalidades, vital para su negocio.
- Permite definir en cada iteración cuales son los objetivos de la siguiente.
- Permite tener realimentación de los usuarios.
- La presión está a lo largo de todo el proyecto y no en una entrega final.

Desventajas

- Altas comisiones en caso de fallar.
- Delimitar el alcance del proyecto con el cliente.

Selección de la metodología a utilizar

Después de realizar un análisis observando las características, ciclo de vida, ventajas y desventajas de las metodologías ASD y XP se ha llegado a la conclusión de que la metodología a usar para el desarrollo del producto es *Extreme Programming* (XP) debido a los siguientes aspectos:

- XP se centra más en la programación o creación del producto que en la administración del proyecto.
- XP se centra en la propiedad de código compartido lo que permitirá al equipo de desarrollo tener total conocimiento y responsabilidad del producto final a diferencia de ASD donde varios grupos de desarrollo se encargan de desarrollar módulos del producto final.
- XP recomienda que las tareas a desarrollar sean implementadas por 2 personas en vez de una como en la metodología ASD ya que se logra una mayor calidad del código al ser revisado y discutido mientras se escribe.

1.12 Herramientas y Tecnologías

IDE NetBeans 6.9 como entorno de desarrollo

NetBeans es un entorno de desarrollo hecho principalmente para el lenguaje de programación Java. Es un producto libre y gratuito sin restricciones de uso. *Sun MicroSystems* fundó el proyecto de código abierto *NetBeans* en junio de 2000.

Java

Es un lenguaje de alto nivel sencillo, orientado a objetos, que brinda gran nivel de seguridad y rendimiento. Está diseñado para que cualquier programa que esté elaborado con él sea ejecutado autónomo del sistema operativo, solamente necesita el uso de la máquina virtual de java.

Herramientas CASE para modelado UML

Las herramientas CASE (*Computer-Aided Software Engineering*) para UML (*Unified Modeling Language*) según (Gonzalo Génova Fuster, Miguel Fuentes Torres, & Cruz Valiente, 2006) surgen debido a la necesidad de ingenieros, analistas y arquitectos informáticos de modelar sistemas de *software* mediante diagramas UML para representar, detallar, construir y documentar los artefactos generados en la implementación de una aplicación a lo largo del ciclo de vida de desarrollo del producto.

- ✓ Visual Paradigm 5.0

Herramienta CASE para modelado UML creada por *Visual Paradigm International* para acelerar y automatizar el ciclo vital completo de desarrollo de *software*, permitiendo crear esquemas en las etapas de captura de requisitos, análisis, diseño e implementación brindando a analistas, arquitectos y desarrolladores grandes ventajas al implementar aplicaciones informáticas.

Características

- ➢ Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML creando de forma sencilla toda la documentación UML necesaria.
- ➢ Soporte para análisis textual lo que permite la captura de los requisitos del sistema y la identificación de clases candidatas.
- ➢ Soporte para tarjetas CRC.
- ➢ Soporte para múltiples plataformas tales como Windows, Linux y Mac OS.
- ➢ Integración total con Microsoft Office permitiendo editar directamente en el documento el contenido del diagrama creado.

- Perfecta integración con los principales entornos de desarrollo tales como Microsoft Visual Studio, Borland JBuilder, Eclipse / IBM WSAD, NetBeans / Sun ONE, IntelliJ IDEA y JDeveloper.

✓ Rational Rose Modeler 7.0

Herramienta CASE para modelado UML creada por *Rational Software Corporation* (actualmente parte de IBM), para producir modelos visuales utilizando UML permitiendo así el diseño de las soluciones de *software* requeridas por analistas, arquitectos y desarrolladores de sistemas informáticos.

Características

- La ingeniería de código directa e inversa es posible para los lenguajes de programación Ada, ANSI C++, C, C++, CORBA, Java, Microsoft Visual Basic, Microsoft Visual C++, así como diseños de base de datos y aplicaciones web.
- Permite la creación de los siguientes tipos de diagramas basados en UML: diagramas de actividad, clase, componente, despliegue, secuencia, diagrama de estado, casos de uso, colaboración.
- Integración del diseño de aplicaciones con el desarrollo, lo cual unifica el equipo del proyecto proporcionando una ejecución y una notación de modelos UML comunes.
- Provee plantillas de código que pueden aumentar significativamente la cantidad de código fuente generado.
- Se pueden aplicar los patrones de diseño, ya que provee 20 de los patrones de diseño GOF para Java.
- Soporte para las plataformas Windows y Linux.

Selección de la herramienta CASE de modelado UML a utilizar

Después de realizar un análisis observando las características de dos de las herramientas CASE de modelado UML más usadas mundialmente se ha llegado a la conclusión de que la herramienta a usar para el desarrollo del producto es *Visual Paradigm 5.0* debido a los siguientes aspectos:

- Visual Paradigm permite una perfecta integración con el IDE de desarrollo Netbeans, no siendo así con el Rational Rose, lo que permitirá invertir código para generar documentación de una manera sencilla.
- Visual Paradigm es multiplataforma mientras Rational Rose solo soporta las plataformas Windows y Linux.
- Visual Paradigm incluye soporte para análisis textual.
- Visual Paradigm incluye soporte para las tarjetas CRC que son utilizadas en la metodología XP.

Sistemas que implementan cifrado y firma digital con control de acceso.

En la actualidad los usuarios y las empresas están en constante intercambio de información sobre soportes inseguros. Con el uso de técnicas de cifrado y firmas digitales pueden intercambiar información de forma segura. Existen en la informática numerosas aplicaciones de cifrado y firma digital, a continuación características de las más usadas por los usuarios:

PGP Desktop[22]**:** es un sistema creado para el cifrado de comunicación a través de internet mediante el uso de criptografía de clave pública, el cual fue creado por Philip R. "Phil" Zimmermann[23] en 1991 (Philip Zimmermann), actualmente es un sistema de criptografia híbrida. PGP provee diversas funcionalidades de uso, que son las siguientes:

- Cifrado de mensajes
- Firma digital
- Compresión
- Compatibilidad de correo

La principal desventaja de este sistema es que se necesita licencia para su uso y en correspondencia con la licencia que tenga serán las funcionalidades que tendrá habilitada.

GPG**:** es un motor de cifrado sin ninguna característica gráfica, lo cual no impide realizar todas las funciones necesarias, pero dificulta su uso; este incluye un sistema interactivo de menú para el usuario. GPG posee una serie de características mencionada a continuación (Koch, 2002).

[22] PGP: es un programa cuya finalidad es proteger la información distribuida a través de *Internet* mediante el uso de criptografía de clave pública.

[23] Philip R. Zimmermann ,es el creador del software más utilizado encriptación de correo electrónico en el mundo, Pretty Good Privacy

- No utiliza algoritmos patentados
- Posee licencia GPL, escrito desde cero.
- Descifra y verifica mensajes de PGP 5, 6 y 7.
- Facilidad de implementación de nuevos algoritmos utilizando módulos
- Soporta fechas de caducidad de claves y firmas.

Seahorse: Es una interfaz gráfica de GnuPG (*GNU Privacy Guard*) que se integra en el escritorio GNOME[24] para la gestión de claves de PGP y SSH. Es una herramienta para realizar comunicaciones seguras y almacenamiento de datos seguros. Con las siguientes características:

- Creación y gestión de claves PGP.
- Creación y gestión de claves SSH[25].
- Publicar y recuperar las claves de los servidores de claves.
- Guarda sus contraseñas en una caché, por lo que no tienes que volver a escribirla.
- Copia de seguridad de sus claves.

En resumen, se han analizado varias aplicaciones de cifrado y firma digital de archivos de las más usadas por los usuarios en el mercado de seguridad informática, que garanticen confidencialidad, integridad y autenticidad de los datos. Pero hasta el momento en la bibliografía consultada no se identificó ningún sistema que posea la inclusión de listas de control de acceso.

Basándose en el estudio anterior esta investigación propone como solución una aplicación de escritorio que incluya:

- Cifrado y descifrado de archivos que permita la inclusión de una lista de control de acceso, para uno a varios destinatarios.
- Creación y comprobación de firma digital básica.
- Soporte para múltiples sistemas operativos.
- Interfaz de usuario.

[24] GNOME: entorno de escritorio e infraestructura de desarrollo para sistemas operativos Unix.

[25] SSH (***S**ecure **SH**ell*),es un protocolo seguro para acceder a las máquinas remoto a través de la red

1.13 Conclusiones

1. El análisis de conceptos claves enmarcados en el objeto de estudio permitió definir como algoritmo de cifrado simétrico al AES, algoritmo de cifrado asimétrico al RSA y como algoritmo de firma digital al SHA-1/RSA; además de identificar que las aplicaciones similares a la propuesta de solución no cumplen con las funcionalidades necesarias para resolver el problema planteado.
2. La comparación de las herramientas y tecnologías de acuerdo a las características, ventajas, desventajas y limitaciones favoreció la selección de XP como metodología de desarrollo de *software*, Visual Paradigm 5.0 como herramienta CASE de modelado UML, el estándar criptográfico PKCS #7 y la plataforma de desarrollo Java.
3. El análisis de los modelos de control de acceso contribuyó a definir al discrecional (DAC por sus siglas en inglés) como modelo a utilizar en la propuesta de solución.

Capítulo 2: Propuesta de Solución

Introducción

En este capítulo se hará la propuesta de solución para la creación de una aplicación de cifrado y firma digital con control de acceso, utilizando la metodología de desarrollo de *software* XP (*Extreme Programming*). Esta metodología propone seis fases, de las cuales en este capítulo se mostrara la evolución de la solución durante las primeras fases, Exploración, Planeación y Diseño, además se presentaran los diferentes artefactos generados durante el ciclo de vida de desarrollo del *software*.

2.1 Modelo de Dominio

El modelo de dominio o modelo Conceptual es un artefacto que representa conceptos de la realidad física que se manejan en el sistema, los cuales son bien descritos, ya que estos conceptos son los que deberá trabajar la aplicación. El modelo de dominio para cualquier solución, utilizará los mismos conceptos que tenga la misma idea de solución de los casos de usos que se hayan elegido (Craig, 1999).

Glosario de Conceptos del Modelo de Dominio (ver Figura 5)

Emisor: representa el usuario que realiza todo el proceso de firma y cifrado del mensaje con ACL, el cual esta se traslada por un CFI a través de algoritmos criptográficos.

Receptor: representa el usuario que recibe el mensaje firmado y cifrado con la ACL, que a través de algoritmos criptográficos obtiene el mensaje original.

Mensaje: representa cualquier tipo de archivo.

Mensaje cifrado: representa cualquier tipo de archivo cifrado con una extensión proporcionada por la aplicación.

Criptografía: representa la ciencia que se encarga de convertir el mensaje, en un mensaje cifrado y viceversa, a través de algoritmos criptográficos.

Algoritmos criptográficos: representan funciones matemáticas para realizar el proceso de cifrado y descifrado de los mensajes.

CFI: representa memorias flash, discos duros, disquetes y el envío por la red.

ACL: representa un fichero al cual el sistema tendrá acceso, para dar permisos a los receptores.

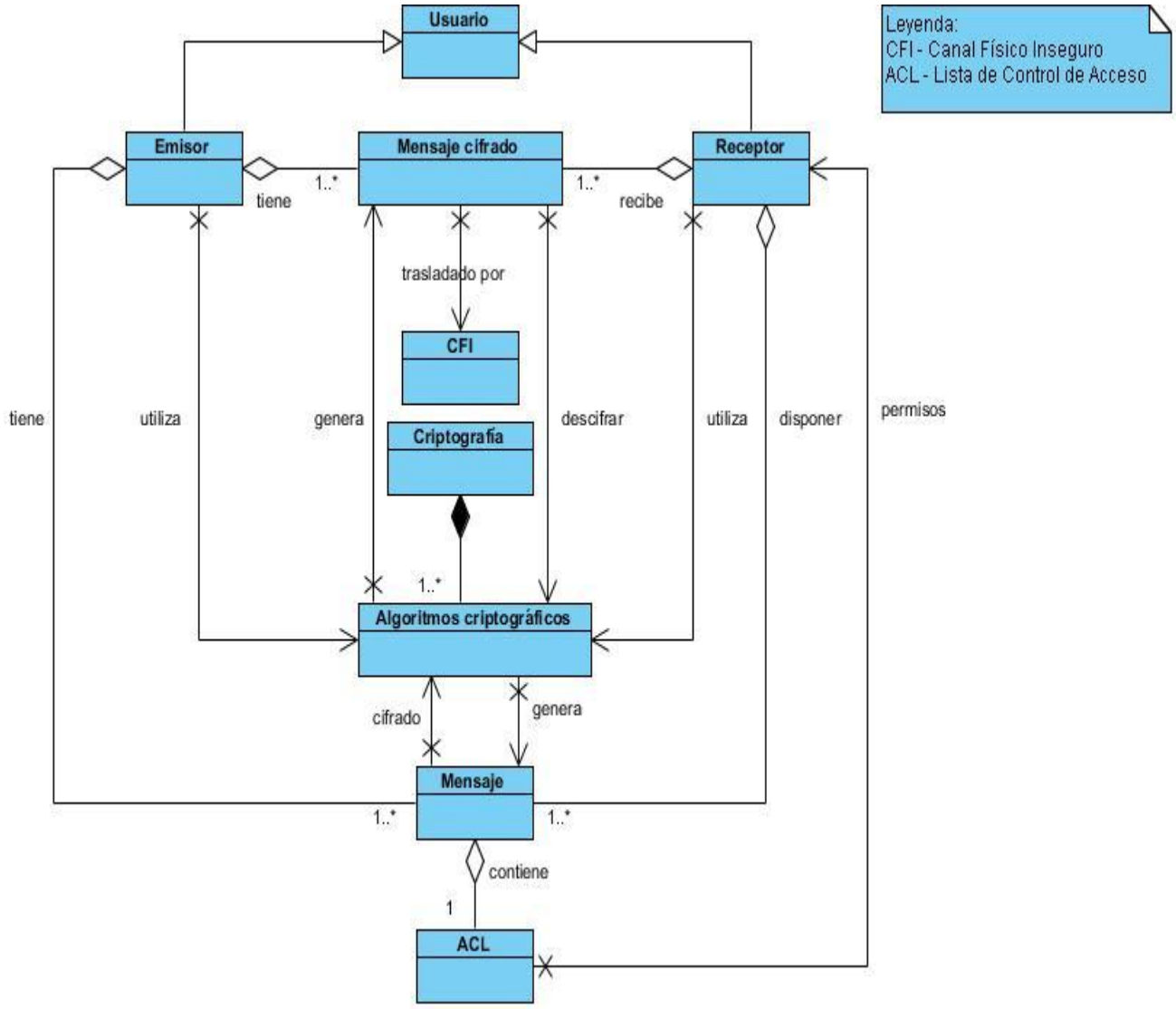


Figura 5: Modelo de Dominio

2.2 Requerimientos del software

Los requerimientos de *software* son especificaciones completas de los que el sistema va a desarrollar. Aquí se describen todas las interacciones que tendrán los usuarios sobre la aplicación. Estos requerimientos contienen las historias de usuario y los requisitos no funcionales, los cuales se muestran a continuación.

Historias de usuario

Las historias de usuarios son especificaciones que el sistema debe cumplir, además describen con detalle la función de éste, sus entradas y salidas, excepciones, entre otras. Las historias de usuarios definidas para este sistema son los siguientes:

HU1: Generar estructura RecipientInfo.

HU2: Generar estructura SignerInfo.

HU3: Generar mensaje cifrado con ACL.

HU4: Generar estructura SignedAndEnvelopedData

HU5: Gestionar contenido de la estructura SignedAndEnvelopedData.

HU6: Descifrar contenido de la estructura RecipientInfo.

HU7: Descifrar contenido del mensaje.

HU8: Descifrar contenido de la estructura SignerInfo.

HU9: Gestionar Certificados

Requerimientos no funcionales

Los requerimientos no funcionales se refieren a propiedades procedentes de los requisitos funcionales. Los requerimientos no funcionales especifican o restringen las propiedades emergentes del sistema como usabilidad, rendimiento, protección, disponibilidad entre otros. El incumplimiento de un requerimiento no funcional puede inutilizar el sistema entero.

Requerimientos mínimos de *hardware*

- Microprocesador con velocidad de procesamiento de 1.0 GHz o superior.
- Al menos 20 Mb de espacio en disco duro.
- Memoria RAM de 512 Mb o superior.

Requerimientos de *software*

- El sistema requiere para su funcionamiento la máquina virtual de java JDK versión 7.0 o superior.
- El sistema será multiplataforma.
- El sistema requiere un almacén de llaves.

Requerimientos de diseño e implementación

- El sistema será una aplicación de escritorio.
- La implementación de la aplicación será en lenguaje Java.

- La aplicación debe ser diseñada utilizando el estándar criptográfico PKCS #7.

2.3 Planificación

Historias de usuario

Las historias de usuario en la metodología de XP se utilizan para especificar los requisitos funcionales del *software*. Es una forma abreviada de administrar estos requisitos desde la perspectiva del cliente. A continuación se muestran 9 tablas que describen las historias de usuario.

Tabla 4: Historia de usuario generar estructura RecipientInfo.

Historia de Usuario	
Número: HU_1	**Nombre:** Generar estructura RecipientInfo
Usuario: Programador	
Prioridad del Negocio: Media	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 1
Descripción: Se generará una llave con un algoritmo criptográfico simétrico, para cada receptor será cifrada esta con la llave pública de cada uno, además posee en la estructura información adicional de los receptores.	
Observaciones:	

Tabla 5: Historia de Usuario Generar estructura SignerInfo.

Historia de Usuario	
Número: HU_2	**Nombre:** Generar estructura SignerInfo
Usuario: Programador	
Prioridad del Negocio: Media	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 2
Descripción: Se cifra un resumen del mensaje a enviar con la llave privada del emisor (firma digital) y luego el resumen cifrado es encriptado con la llave de sesión (simétrica). La estructura tendrá el mensaje doblemente cifrado además de información sobre el firmante.	

Observaciones:

Tabla 6: Historia de Usuario Generar mensaje cifrado.

Historia de Usuario	
Número: HU_3	**Nombre:** Generar mensaje cifrado
Usuario: Programador	
Prioridad del Negocio: Media	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 1
Descripción: Se cifrará con la llave de sesión de los receptores el mensaje y la lista de control de acceso con los permisos de los receptores sobre el mensaje.	
Observaciones:	

Tabla 7: Historia de Usuario Generar estructura SignedAndEnvelopedData.

Historia de Usuario	
Número: HU_4	**Nombre:** Generar estructura SignedAndEnvelopedData
Usuario: Programador	
Prioridad del Negocio: Alta	**Riesgo de Desarrollo:** Alta
Puntos Estimados: 1	**Iteración Asignada:** 2
Descripción: Se generará un paquete que contenga las estructuras RecipientInfo, SignerInfo, además del mensaje con la lista de control de acceso.	
Observaciones:	

Tabla 8: Historia de Usuario Gestionar contenido dela estructura SignedAndEnvelopedData.

Historia de Usuario	
Número: HU_5	**Nombre:** Gestionar contenido de la estructura SignedAndEnvelopedData
Usuario: Programador	
Prioridad del Negocio: Alta	**Riesgo de Desarrollo:** Alta
Puntos Estimados: 1	**Iteración Asignada:** 2
Descripción: Se obtendrán las estructuras RecipientInfo y SignerInfo, además del mensaje cifrado que contiene la	

lista de control de acceso.
Observaciones:

Tabla 9: Historia de Usuario Descifrar contenido de la estructura RecipientInfo.

Historia de Usuario	
Número: HU_6	**Nombre:** Descifrar contenido de la estructura RecipientInfo
Usuario: Programador	
Prioridad del Negocio: Media	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 1
Descripción: Se obtendrá la llave de sesión al descifrar con la llave privada de cada receptor.	
Observaciones:	

Tabla 10: Historia de Usuario Descifrar contenido del mensaje.

Historia de Usuario	
Número: HU_7	**Nombre:** Descifrar contenido del mensaje
Usuario: Programador	
Prioridad del Negocio: Alta	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 1
Descripción: Se obtendrá el mensaje descifrado con la llave de sesión y la lista de control de acceso con los permisos asignados a cada receptor.	
Observaciones:	

Tabla 11: Historia de Usuario Descifrar contenido de la estructura SignerInfo.

Historia de Usuario	
Número: HU_8	**Nombre:** Descifrar contenido de la estructura SignerInfo
Usuario: Programador	
Prioridad del Negocio: Media	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 2
Descripción: Se obtendrá el resumen del mensaje al descifrar con la llave de sesión primero y luego con la llave	

pública del emisor.
Observaciones:

Tabla 12: Historia de Usuario Gestionar Certificados.

Historia de Usuario	
Número: HU_9	**Nombre:** Gestionar Certificados
Usuario: Programador	
Prioridad del Negocio: Alta	**Riesgo de Desarrollo:** Media
Puntos Estimados: 1	**Iteración Asignada:** 1
Descripción: Se importarán el/los certificado(s) digital(es) por cada destinatario(s).	
Observaciones:	

2.4 Propuesta de solución

A continuación se define un esquema para dar solución al problema de la investigación.

- Esquema SignedAndEnvelopedData

La estructura SignedAndEnvelopedData está compuesta por tres estructuras las cuales se describen a continuación.

RecipienInfo: Esta estructura posee todas las claves de sesión cifrada con la clave pública de cada destinatario.

EnvelopedData: Esta estructura es el mensaje cifrado con la clave de sesión, en este caso además con la ACL.

SignerInfo: Esta estructura contiene todas las firmas cifradas con la clave de sesión.

- Descripción del esquema SignedAndEnvelopedData

Creación del paquete (ver Figura 6): se genera una clave de sesión mediante un algoritmo simétrico, esta clave de sesión se cifra para cada destinatario con su clave pública y se almacena en la estructura RecipientInfo, con la clave de sesión se cifra el mensaje con la ACL y la estructura SignerInfo, esta última está compuesta por la firma de cada uno de los firmantes cifrada con la clave de sesión.

Recepción del paquete (ver Figura 7): se descifra la clave de sesión del receptor de la estructura RecipientInfo, con esa clave se descifra el mensaje cifrado, obteniendo el mensaje original con la ACL y la estructura SignerInfo obteniendo las firmas, esta firma se

descifra con la clave pública del emisor obteniendo el resumen del mensaje original, se le aplica una función resumen al mensaje original y se compara con el obtenido a través la clave pública y se valida si es correcta la firma.

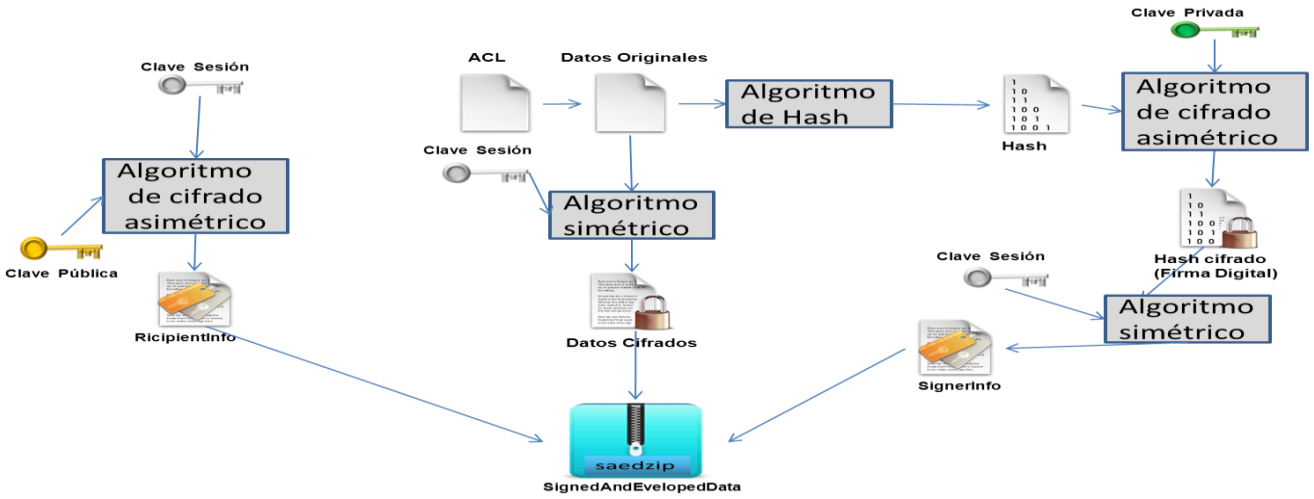


Figura 6: Creación del empaquetado SignedAndEnvelopedData

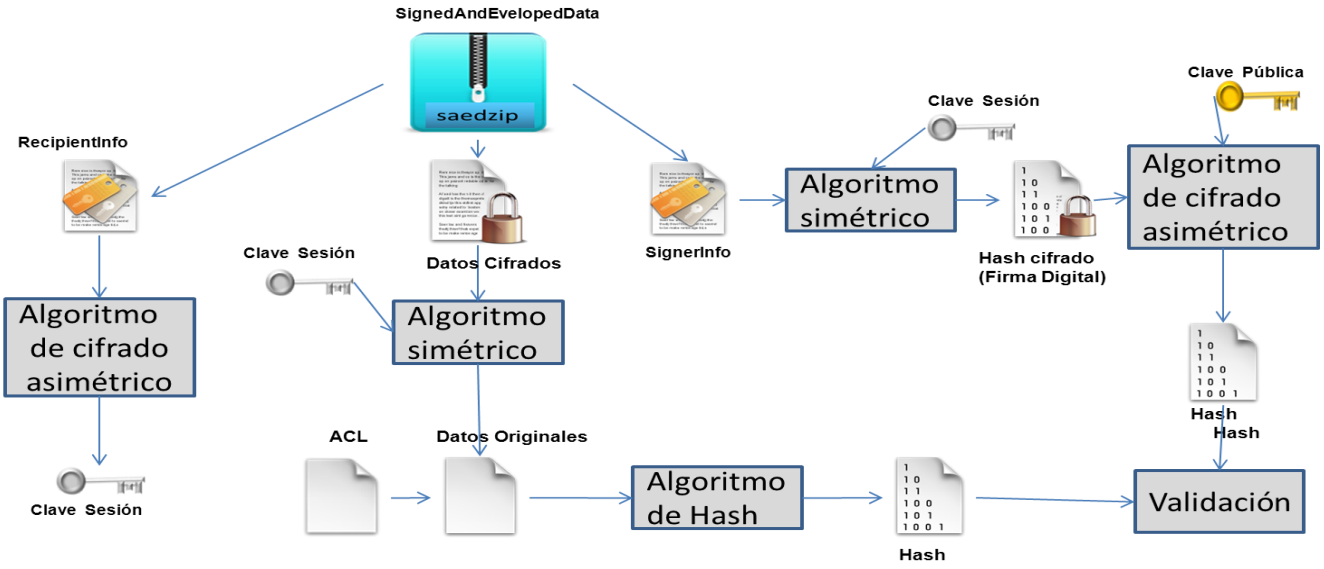


Figura 7: Recepción del empaquetado SignedAndEnvelopedData

2.5 Estimación de Tiempo

Para el desarrollo de la aplicación se realizó una estimación del tiempo que necesitan los programadores para cada historia de usuario. El tiempo se expresa en semana; la estimación de una historia de usuario no debe desarrollarse en menos de 1 semana, ni más de 3 semanas.

Tabla 13: Estimación de tiempo para el desarrollo de la aplicación.

Historia de Usuarios	Estimación (semanas)
Generar estructura RecipientInfo	2
Generar estructura SignerInfo	2
Generar mensaje cifrado	2
Generar estructura SignedAndEnvelopedData	2
Gestionar contenido de la estructura SignedAndEnvelopedData	2
Descifrar contenido de la estructura RecipientInfo	2
Descifrar contenido del mensaje	2
Descifrar contenido de la estructura SignerInfo	2
Gestionar Certificados	2
Total	18

Plan de Iteraciones

Tabla 14: Plan de iteraciones.

Iteración	No.HU	Historia de usuario	Duración estimada
Iteración 1	HU1	Generar estructura RecipientInfo	10 semanas
	HU3	Generar mensaje cifrado con la ACL	
	HU6	Descifrar contenido de la estructura RecipientInfo	
	HU7	Descifrar contenido del mensaje	
	HU9	Gestionar Certificados	
Iteración 2	HU2	Generar estructura SignerInfo	8 semanas
	HU4	Generar estructura SignedAndEnvelopedData	
	HU5	Gestionar contenido de la estructura	

		SignedAndEnvelopedData	
	HU8	Descifrar contenido de la estructura SignerInfo	

2.6 Diseño

Metáfora

Para la creación de una estructura SignedAndEnvelopedData, se necesita cargar uno o varios documentos los cuales serán firmados y cifrados con una clave de sesión la cual será cifrada con la clave pública de cada destinatario y guardada en una estructura RecipientInfo, las claves públicas estarán contenidas en certificados importados previamente o en el momento de la creación de la estructura, los documentos firmados y cifrados estarán en las estructuras SignerInfo y EnvelopedData respectivamente. Estas dos estructuras serán gestionadas con la clave de sesión y encapsuladas y exportadas en la estructura SignedAndEnvelopedData con la extensión .saedzip. Los datos cifrados tendrán un control de acceso sobre cada destinatario.

Arquitectura del sistema

El patrón de arquitectura por capas es una de las técnicas más comunes que los arquitectos de *software* utilizan para dividir sistemas de *software* complicados. Donde cada una de estas capa descansa sobre la inferior, la capa más alta utiliza las funcionalidades de la capa que se encuentra por debajo de esta, siendo esta última inconsciente de la capa superior. En esta arquitectura cada capa tiene bien definido que responsabilidad tiene cada una de ellas y que implementan.

Un sistema en capas según (Marquina, 2008) tiene una cantidad importante de beneficios:

- Se puede entender una capa como un todo sin considerar las otras.
- Las capas se pueden sustituir con implementaciones alternativas de los mismos servicios básicos.
- Se minimizan dependencias entre capas.
- Las capas posibilitan la estandarización de servicios.
- Luego de tener una capa construida, puede ser utilizada por muchos servicios de mayor nivel.

Descripción de la arquitectura

La propuesta de solución de la aplicación de cifrado y firma digital con ACL basa el desarrollo de su implementación en la arquitectura n-capas, donde se utilizarán dos capas para el desarrollo de la solución, una capa de presentación y una de lógica del negocio. Dado que el objetivo principal de esta arquitectura es separar la lógica del negocio de la lógica del diseño. Además que este estilo se puede llevar a cabo en varios niveles, siendo así, si ocurre un error en algunos de ellos solo afectará ese nivel, propiciando esto que las demás capas permanezcan intactas.

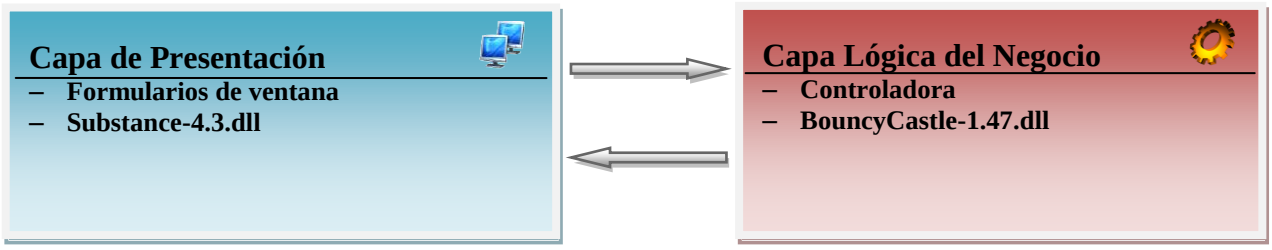


Figura 8: Arquitectura del sistema

Patrones de diseño

Los patrones son una disciplina de resolución de problemas de la ingeniería del *software* que emergió en mayor medida de la comunidad de orientación a objetos, aunque pueden ser aplicados en cualquier ámbito de la informática y las ciencias en general.

Según (Alexander, 1979) cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces.

Según (Gamma Erich, 1995) un patrón de diseño describe una estructura recurrente de componentes que se comunican para resolver un problema general de diseño en un contexto particular. Nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizables. Identifica las clases e instancias participantes, sus roles y colaboraciones y la distribución de responsabilidades.

Los patrones de diseño no son dogmas que deben ser aceptados. ¿Qué es? y ¿Qué no?, es un patrón de diseño, es una cuestión que depende del punto de vista de cada desarrollador y del nivel de abstracción en que se trabaje.

Después de haber analizado los conceptos anteriores que definen de una manera no muy diferente que son los patrones de diseño se puede llegar a la reflexión de que un patrón de diseño no es una solución en sí misma, sino la documentación de la forma en que construyeron soluciones a problemas similares en el pasado, lo cual permite una mejor gestión de la experiencia y trasmisión de conocimientos.

- ✓ Patrones GRASP

Los patrones GRASP (acrónimo en inglés de *General Responsibility Asignment Software Patterns*) según (Craig, 1995) son patrones generales de *software* para asignación de responsabilidades, aunque se consideran más que patrones como una serie de "buenas prácticas" de aplicación recomendable en el diseño de *software*.

Se definieron como patrones GRASP a utilizar en la aplicación los siguientes:

Bajo acoplamiento:

Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que desarrollan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como Experto o Alta Cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades.

Alta cohesión:

Una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla, además puede destinarse a un propósito muy específico. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos.

Experto:

El uso de este patrón permitirá a los objetos valerse de su propia información para hacer lo que se les pide, esto favorece la existencia de mínimas relaciones entre las clases permitiendo contar con un sistema sólido y fácil de mantener. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas siendo más fáciles de comprender y de mantener.

Creador:

Se considera para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto solo pueda ser creada por el objeto que contiene la información necesaria para ello. El uso de este patrón admite crear las dependencias mínimas necesarias entre las clases, beneficiando el mantenimiento del sistema y brindando mejores oportunidades de reutilización.

- ✓ Patrones GOF (Gang of Four)

Los Patrones GOF (acrónimo en inglés de *Gang of Four*) según (Gamma Erich, 1995) describen las formas comunes en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros, además tratan la relación entre clases, la combinación de clases y la formación de estructuras de mayor complejidad.

Los patrones de diseño GOF se clasifican en tres categorías basadas en su propósito: creacionales, estructurales y de comportamiento:

- ➢ Creacionales: describen las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
- ➢ Estructurales: describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.
- ➢ Comportamiento: definen la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Se definió como patrón GOF a utilizar en la aplicación el siguiente:

- ➢ Delegation (delegación): se quiere extender y reutilizar la funcionalidad de una clase sin utilizar la herencia. En el siguiente código la clase File_CMS tiene como uno de sus atributos a la clase ACL_L lo que le permitirá a la primera utilizar todas las funcionalidades de la última.

```
public class File_CMS implements Serializable {
    private ACL_L control_acceso;
    private LinkedList<String> files_nombre;
    private LinkedList<byte[]> files_bytes;
    private LinkedList<File> files_cms;

    public File_CMS() {
        this.files_nombre = new LinkedList<String>();
        this.files_bytes = new LinkedList<byte[]>();

    }
```

Figura 9: Ejemplo de uso del patrón delegación

Diagrama de Clases

Según (Craig, 1995) un diagrama de clases es un esquema estático que detalla la estructura de un sistema exponiendo sus clases, atributos y las relaciones entre ellos; estos son utilizados durante el proceso de análisis y diseño de las disímiles metodologías, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro. En Figura 10 se muestra el diagrama de clases confeccionado para esta investigación.

Tarjetas CRC (Class-Responsibility-Collaboration)

Las tarjetas CRC según Kent Beck (Kent, 2000) son una técnica de diseño orientado a objetos cuyo objetivo es realizar un inventario de las clases a implementar en el sistema y la forma en que interactúan, para de esta forma facilitar el análisis y discusión de las mismas por parte del equipo de desarrollo y hacer que el diseño sea lo más simple posible verificando las especificaciones del sistema.

Tabla 15: Tarjeta CRC Clase ACL.

ACL_L	
Funcionalidades	Colaboración
Agregar_Destinatarios Obtener_Permisos Comprobar_Permisos	Destinatario Principal
Descripción: Esta clase es la encargada de gestionar la lista de destinatarios, determinando que permisos tiene cada destinatario, comprobando cuales acciones (lectura/lectura-escritura de archivos) puede realizar cada destinatario.	

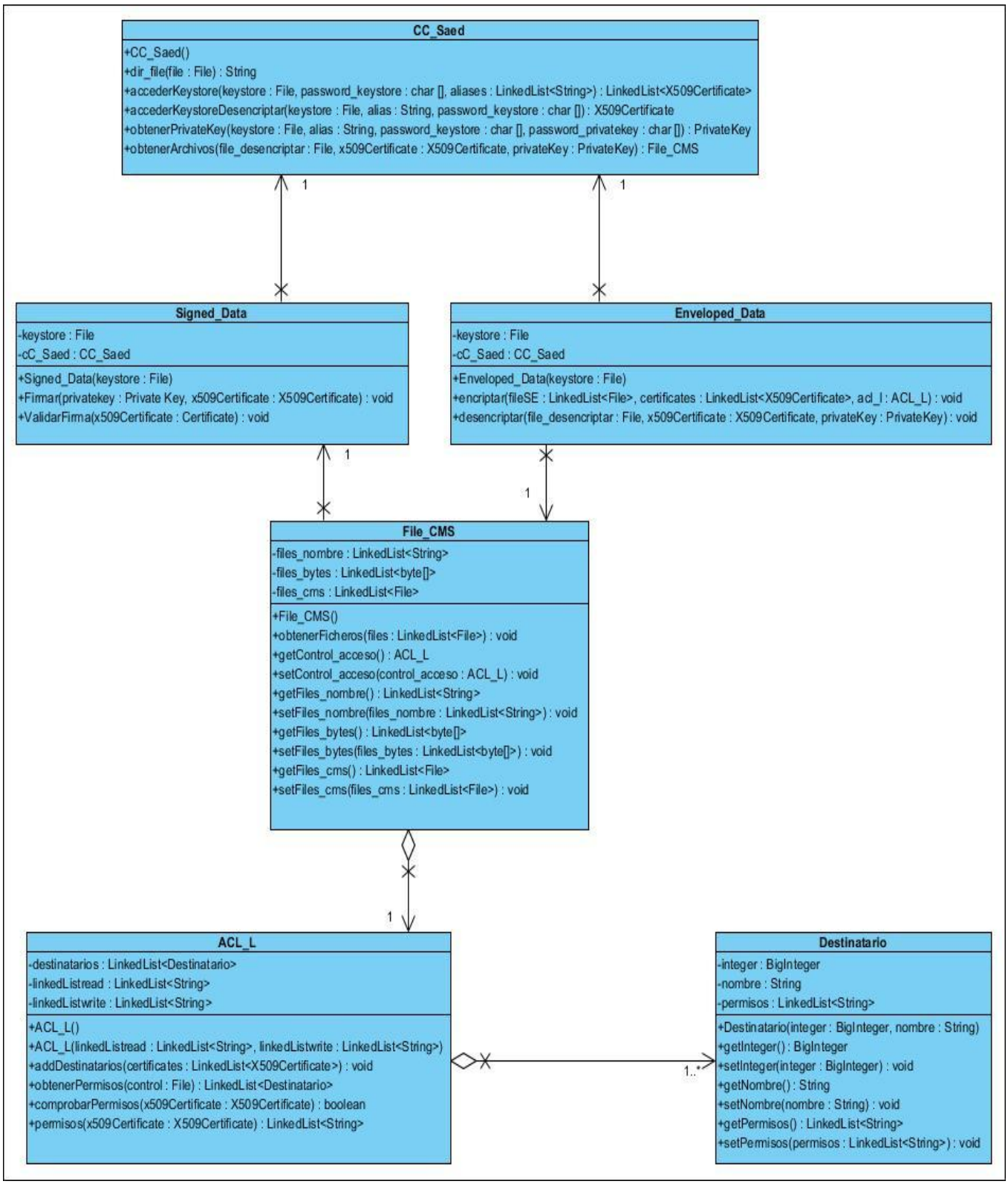


Figura 10: Diagrama de clases

Tabla 16: Tarjeta CRC Clase Enveloped_Data.

Enveloped_Data	
Funcionalidades	Colaboración
Encriptar_Archivos Desencriptar_Archivos	CC_Saed File_CMS Principal ACL_L
Descripción: Esta clase es la encargada de realizar el proceso de cifrado y descifrado de ficheros. La cual recibe toda la información desde las interfaces del sistema. La clase Enveloped_Data posee dos métodos un método Encriptar_Archivos el cual a través de una cadena de certificados y una lista de archivos realiza el proceso de cifrado, devolviendo un empaquetado con todos los archivos cifrados incluyendo los recipiente para cada uno de los destinatarios, la firma digital y la lista de control de acceso.	

Tabla 17: Tarjeta CRC Clase Signed_Data.

Signed_Data	
Funcionalidades	Colaboración
Firmar_Archivos Comprobar_Firma	File_CMS Principal
Descripción: Esta clase se encarga realizar la firma digital a los ficheros importados en la aplicación .Posee dos métodos Firmar_Archivo y el método Comprobar_Firma el cual realiza un resumen del los archivos, descifra la firma digital obteniendo el resumen de los datos y compara si los resúmenes son iguales.	

Tabla 18: Tarjeta CRC Clase Destinatario

Destinatario	
Funcionalidades	Colaboración
get_Integer get_Nombre get_Permisos	

Descripción: Esta clase es la encargada de crear cada destinatario que será agregado en la lista de control de acceso.

Tabla 19: Tarjeta CRC Clase File_CMS

File_CMS	
Funcionalidades	Colaboración
Obtener_Ficheros	Principal
Descripción: Esta es la clase encargada de gestionar los ficheros .Cuando se importa los ficheros son convertidos en arreglos de bytes y almacenados en una lista de bytes para ser exportados posteriormente en la estructura SignedAndEnvelopedData.	

Tabla 20: Tarjeta CRC Clase CC_Saed

CC_Saed	
Funcionalidades	Colaboración
Dir_File Acceder_Keystore Obtener_Private_Key Obtener_Archivos	File_CMS Principal
Descripción: Esta es la clase controladora que se encarga de realizar el proceso de obtención de las llaves y los certificados para firmar y cifrar los archivos y proveerle toda la información necesaria a las clases Signed_Data y Enveloped_Data para realizar sus funcionalidades.	

2.7 Conclusiones

1. La modelación del dominio contribuyó a la elaboración de las HU y de esta manera se definieron las principales funcionalidades que debe cumplir el sistema.
2. La definición de la arquitectura basa en el patrón n-capas permitió establecer la estructura lógica de la solución conformada por las capas de negocio y presentación, unido a esto y aplicando los patrones de diseño GOF y GRASP se definieron la jerarquía de clases así como sus relaciones y dependencias con lo que se alcanza una vista general del sistema a implementar.

Capítulo 3: Implementación y prueba

Introducción

Una vez culminado la planificación y el diseño del sistema continúa esta investigación con la implementación de la aplicación realizando cada una de las liberaciones de las iteraciones propuestas en el diseño y el estándar de codificación propuesto, además se muestran las pruebas de aceptación originadas por el cliente para el correcto funcionamiento de la aplicación.

3.1 Implementación del Sistema

Iteración 1

En esta primera iteración se implementan las historias de usuarios principales, para obtener una aproximación del producto con sus características y funcionalidades más importantes.

Tabla 21: Iteración 1

Iteración	No.HU	Historia de usuario	Duración estimada
Iteración 1	HU1	Generar estructura RecipientInfo	10 semanas
	HU3	Generar mensaje cifrado con la ACL	
	HU6	Descifrar contenido de la estructura RecipientInfo	
	HU7	Descifrar contenido del mensaje	
	HU9	Gestionar Certificados	

Tareas de la Ingeniería Iteración 1

Tabla 22: Tareas de la Ingeniería Iteración 1

Iteración	No	Historia de usuario	Tareas
	HU1	Generar estructura RecipientInfo	1. Diseñar una interfaz de usuario para la gestión de la estructura RecipientInfo 2. Importar certificado

Iteración 1	HU3	Generar mensaje cifrado	1. Diseñar una interfaz de usuario para la gestión de archivos a cifrar 2. Importar claves desde un almacén de llaves 3. Cifrar un mensaje
	HU6	Descifrar contenido de la estructura RecipientInfo	1. Importar clave desde un almacén de llaves 2. Descifrar estructura RecipientInfo
	HU7	Descifrar contenido del mensaje	1. Diseñar una interfaz de usuario para la gestión de archivos a descifrar 2. Importar claves desde un almacén de llaves 3. Descifrar un mensaje cargado a la aplicación
	HU9	Gestionar Certificados	1. Importar certificados digitales

Tareas de la Ingeniería Detalladas Iteración 1

A continuación se presenta la Tabla 23 con los detalles de las Tareas de la Ingeniería Iteración 1 las demás tablas están en el Anexo 1.

Tabla 23: HU1_T1

Tarea	
Número de la Tarea: HU1_T1	**HU:** Generar estructura RecipientInfo
Nombre: Diseñar una interfaz de usuario para la gestión de la estructura RecipientInfo	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: La interfaz debe tener un campo para adicionar Recipientes a la estructura a través un certificado.	

Iteración 2

Tabla 24 : Iteración 2

Iteración	No	Historia de usuario	Duración estimadas
Iteración 2	HU2	Generar estructura SignerInfo	8 semanas
	HU4	Generar estructura SignedAndEnvelopedData	
	HU5	Gestionar contenido de la estructura SignedAndEnvelopedData	
	HU8	Descifrar contenido de la estructura SignerInfo	

Tareas de la Ingeniería Iteración 2

Tabla 25: Tareas de la Ingeniería Iteración 2

Iteración	No	Historia de usuario	Tareas
Iteración 2	HU2	Generar estructura SignerInfo	1. Crear estructura SignerInfo 2. Importar clave privada desde un almacén de llaves
	HU4	Generar estructura SignedAndEnvelopedData	1. Crear estructura SignedAndEnvelopedData
	HU5	Gestionar contenido de la estructura SignedAndEnvelopedData	1. Gestionar contenido de la estructura SignedAndEnvelopedData
	HU7	Descifrar contenido de la estructura SignerInfo	1. Descifrar estructura SignerInfo 2. Importar certificados desde un almacén de llaves

Tareas de la Ingeniería Detalladas Iteración 2

A continuación se presenta la Tabla 26 con los detalles de la: Iteración 2 las demás tablas están en el Anexo 2.

Tabla 26: HU2_T1

Tarea	
Número de la Tarea: HU2_T1	**HU:** Generar estructura SignerInfo
Nombre: Crear estructura SignerInfo	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir crear la estructura SignerInfo a través de claves privadas y certificados.	

3.2 Descripción del diseño

La solución como se muestra en la Figura 11 está estructurada en tres paquetes para agrupar las clases que se relacionan entre sí o tratan de un determinado tema en común permitiendo esto cierto grado de independencia entre las clases.

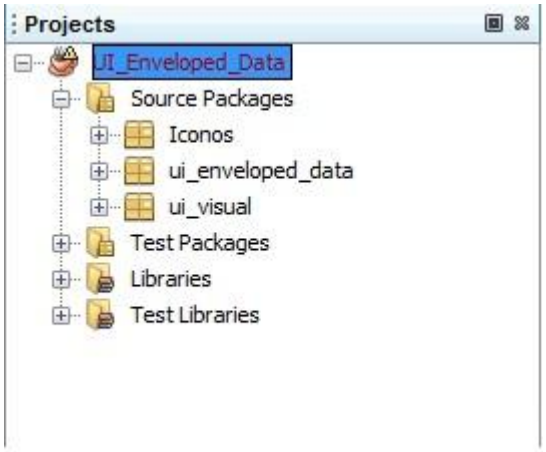


Figura 11 : Paquete de la aplicación

A continuación, una breve descripción de cada paquete del proyecto:

ui_enveloped_data: agrupadas las clases de la capa lógica del negocio del proyecto, es donde están implementadas las funcionalidades principales que dan solución al problema definido.

ui_visual: agrupadas las clases de la capa de presentación que permiten definir e implementar la interfaz gráfica de usuario para un entorno más fácil de usar y amigable.

Iconos: contiene todas las imágenes utilizadas en el desarrollo de la interfaz gráfica de usuario del sistema.

3.3 Descripción del sistema

El sistema permite importar cualquier tipo de ficheros, los cuales son procesados por la clase File_CMS, para el mejor trabajo de las clases, Signed_Data la cual realiza el proceso de firmar y comprobación de esta a los ficheros y Enveloped_Data la cual realiza el cifrado y descifrado de estos. Además permite generar una lista de control de acceso mediante las clases ACL_L la cual posee una lista de Destinatario en la que se encuentran los permisos de cada uno de estos.

3.4 Interfaz Usuario

La interfaz de usuario según (Shneiderman, 1998) es el medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo; normalmente proveen la información necesaria de manera fácil de entender y fácil de accionar.

En la Figura 12 se ofrece la posibilidad de agregar los archivos a encriptar a través de la opción Añadir archivos, aquí puede también eliminar estos.

Luego de cargar los archivos, se procede a importar los certificados desde un almacén de llaves a través de su alias y crear la lista de control de acceso como se muestra en la Figura 13.

La interfaz que muestra en la Figura 14 contiene todos los destinatarios para los cuales se cifrará el mensaje, se le asignará permisos de escritura y se cifrarán finalmente.

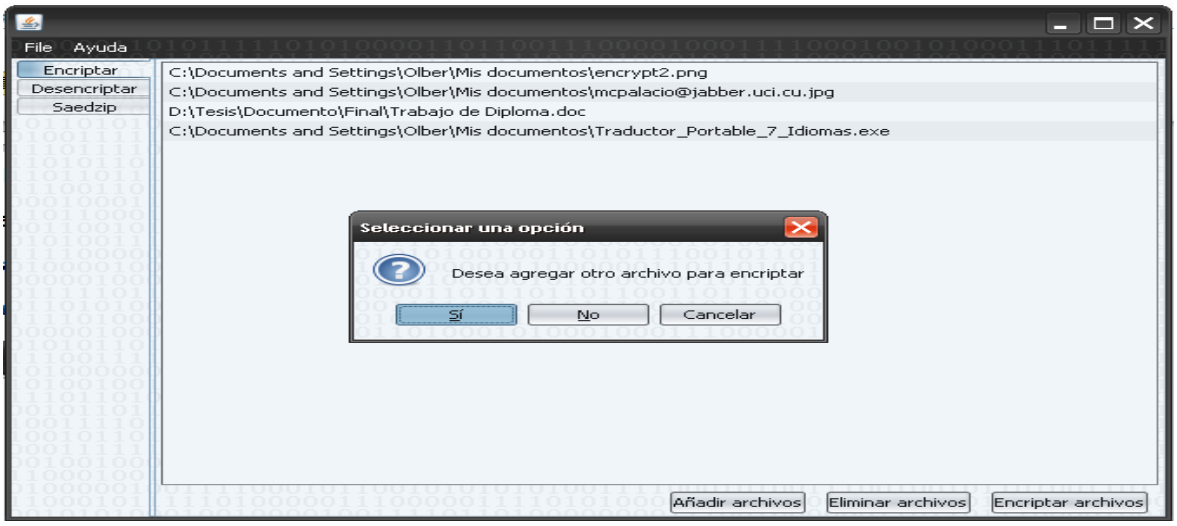


Figura 12: Interfaz para agregar archivos a encriptar

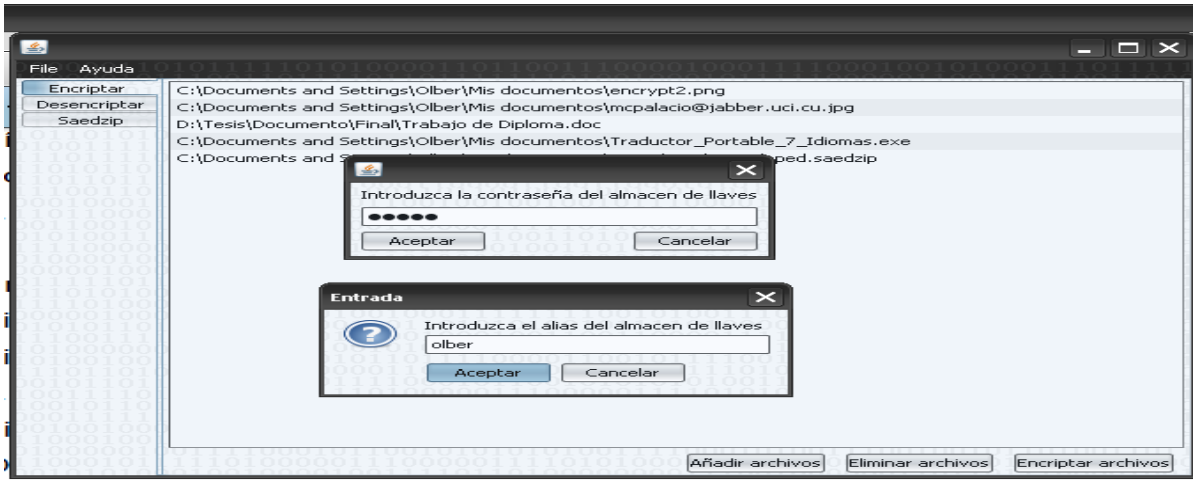


Figura 13: Interfaz para agregar certificados desde un almacén de llaves

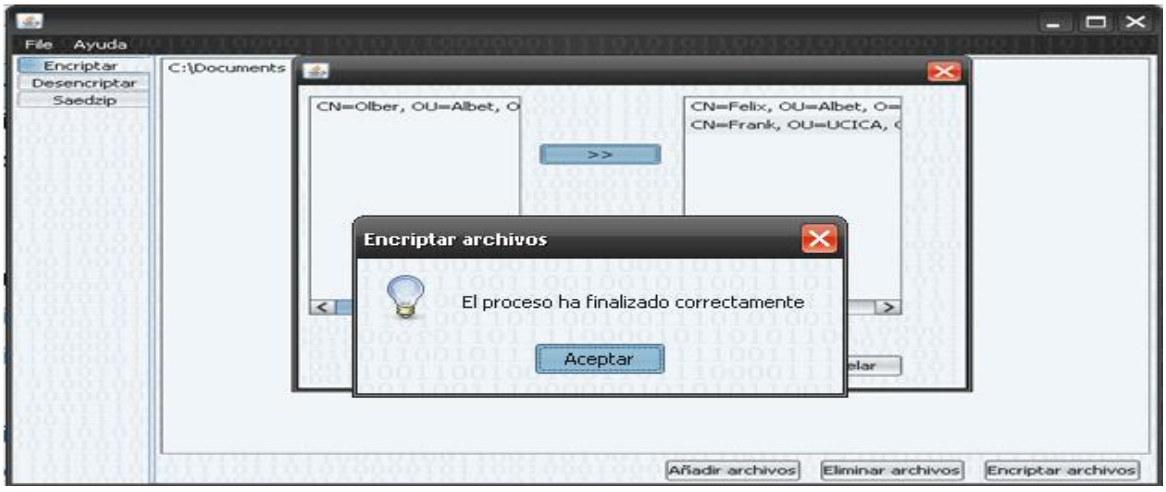


Figura 14: Interfaz para generar la ACL los permisos de los destinatarios

3.6 Estándar de Codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Los estándares de codificación deben ser implementados de forma legible y entendible tendiendo siempre a lo práctico, para los programadores que componen el equipo de desarrollo. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de *software*. La realización de buenas prácticas de programación y técnicas solidas de codificación de alta calidad, son de gran importancia para obtener un *software* de alto rendimiento (Microsoft, 2013). A continuación el estándar de codificación escogido para la realización de la implementación de esta investigación.

Estilo de Codificación

El estilo de codificación que adoptará esta investigación para el desarrollo del *software* es Camel en una de sus variantes *LowerCamelCase,* que consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas menos la primera palabra y el resto en minúscula. Además todos los nombres de las diferentes estructuras de código deberán ser descriptivos.

Nombres de Estructuras

- Nombres de Clases

Los nombres de las clases adoptarán la notación UpperCamelCase con un guión de separación entre cada palabra. Ejemplo:

```
public final class Enveloped_Data {
}
```

- Nombres de Atributos

Los nombres de los atributos adoptarán la notación LowerCamelCase con guión de separación entre cada palabra. Ejemplo:

```
public final class Enveloped_Data {

private CMSEnvelopedData enveloped_Data=new CMSEnvelopedData ();
```

}

- Nombres de Propiedades

Los nombres de las propiedades utilizarán la misma notación que los nombre de los atributos. Ejemplo:

public CMSEnvelopedData **getEnveloped_Data** (){
return enveloped_Data;
}

- Nombre de las Funciones

Los nombres de las funciones utilizarán la misma notación que los nombres de las clases. Ejemplo:

public void **Encrypted_Data** () {
// .....
}

Comentarios de Código

- Comentario de documentación del código

Se utilizarán los comentarios de java, (conocidos como "*doc comments*") estos existen solo en este lenguaje de programación y se limitan por /**...*/. Los comentarios de documentación son para describir la especificación del código. Ejemplo:

/*
***** La clase Enveloped_Data ofrece...
/
public class Enveloped_Data {
}

- Comentarios de implementación

Se utilizarán los comentarios que se encuentran también en C++ y C# delimitados por /*…*/ y //. Los comentarios de implementación se utilizarán para comentar el código o para comentarios acerca de una implementación particular. Ejemplo:

/* return the enveloped_Data ***/***

3.7 Pruebas

El desarrollo de las pruebas permite ver cual es la calidad de este sistema, en esta fase se definen los criterios de calidad para cada historia de usuario. Estos criterios son necesarios para los programadores en la etapa de estimar la duración de cada historia.

Según Lisa Crispin y Tip House (Crispin; House, 2002) la realización de pruebas es la única disciplina que requiere experiencia y entrenamiento, además de habilidad para resolver problemas. También involucra destreza en la comunicación para generar una relación productiva con clientes y programadores.

Pruebas Unitarias

Una prueba unitaria es la verificación de un módulo (unidad de código) determinado dentro de un sistema. El concepto de "módulo" varía de acuerdo con el lenguaje de programación que se esté utilizando; por ejemplo, en Java sería una clase. Las pruebas unitarias nos aseguran que un determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema.

En XP los programadores deben escribir estas pruebas unitarias para cada módulo antes de escribir el código. No es necesario escribir casos de prueba para todos los módulos, solo para aquellos en que exista la posibilidad de que puedan fallar. Luego de escribir el código, los programadores ejecutan las pruebas, las cuales deben resultar 100 % efectivas para que el código pueda integrarse al sistema. En caso contrario hay que solucionar los errores y ejecutar nuevamente los casos de prueba hasta lograr que ninguno de ellos contenga errores (Beck; Gamma, 1998).

El *framework JUnit* desarrollado por Erich Gamma y Kent Beck, permite realizar de manera controlada este tipo de pruebas a aplicaciones desarrolladas en Java. Se utilizó para poder evaluar si el funcionamiento de cada uno de los métodos de las clases se comporta como debería esperarse (Gamma Erich, 1995). Para esta investigación se utilizará este *framework*.

La ausencia de las pruebas unitarias lleva a tener que invertir una gran cantidad de horas en sesiones de *debugging* al momento de integrar el código con el sistema existente.

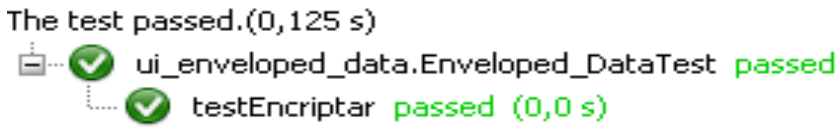
Las pruebas unitarias brindan una inmediata retroalimentación a la realización de este trabajo, permitiendo a los programadores saber si una determinada funcionalidad se puede

agregar al sistema existente sin alterar el funcionamiento actual del mismo. También permiten la aplicación de otras prácticas como *refactoring* y diseño simple al estar respaldados por efectivos casos de prueba.

Resultados de las pruebas unitarias

A continuación se muestra la Tabla 27 con un ejemplo de caso de prueba unitaria las demás están en el Anexo 3.

Tabla 27: Caso de prueba unitaria testEncriptar

Prueba de Unitaria		
Nombre Prueba: testEncriptar		
Estado: Satisfactoria.	**Tipo:** Caja Blanca.	**Última Ejecución:** 13/05/2013
Ejecutado por: Frank Geiler Vega Duverger		**Verificado por:** Félix A. Prieto
Descripción: Para que la prueba se ejecute satisfactoriamente y pueda cifrar los datos se debe proporcionar una lista de archivos y un certificado.		
Entrada: x509v3Certificate certificado , LinkedList<File> lista_Archivos		
Criterio de aceptación: Datos cifrados		
Resultado: 		

Pruebas de Aceptación

Las pruebas de aceptación son pruebas de caja negra definidas por el cliente para cada historia de usuario y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que él espera de ellas.

“Las pruebas de aceptación permiten al cliente saber cuando el sistema funciona, y que los programadores conozcan que es lo que resta por hacer.” (Jeffries, 2000).

A continuación se muestra la tabla con el caso de prueba Generar estructura RecipientInfo las demás pruebas se encuentran en el Anexo 4.

Tabla 28: Prueba de aceptación para generar la estructura RecipientInfo

Caso de Prueba de Aceptación	
Código:HU1_P1	**HU:** Generar estructura RecipientInfo
Nombre: Generar estructura RecipientInfo	
Descripción: Prueba de funcionalidad para la creación de la estructura RecipientInfo	
Condiciones de ejecución: Se debe tener al menos un certificado importado en la aplicación	
Entrada / Pasos de ejecución: • El usuario importa todos los certificados de los destinatarios que se quiere crear una estructura RecipientInfo.	
Resultado esperado: • La creación de una estructura RecipientInfo para cada uno de los destinatarios con una clave simétrica cifrada.	
No conformidades:	
Evaluación de la prueba: Prueba satisfactoria	

Resultados de las pruebas de aceptación

Fueron aplicados 9 casos de pruebas de aceptación a la aplicación que arrojaron varias no conformidades, a continuación se muestra la Tabla 29 y una representación gráfica con la cantidad de no conformidades por iteraciones.

En la primera iteración se le aplicaron las pruebas de aceptación a 5 historias de usuario las cuales arrojaron 15 no conformidades, en la segunda iteración se resolvieron estas no conformidades y se le aplicaron las pruebas a 4 historias de usuario y estas suministraron 7 no conformidades. La tercera iteración es confeccionada para dar solución a las no conformidades encontradas en la segunda iteración, en la que se resolvieron 7.

Tabla 29: No conformidades de los casos de aceptación por iteración

Iteraciones	HU	No conformidades detectadas	No conformidades resueltas
Iteración 1	5	15	0
Iteración 2	4	7	15
Iteración 3	4	0	7

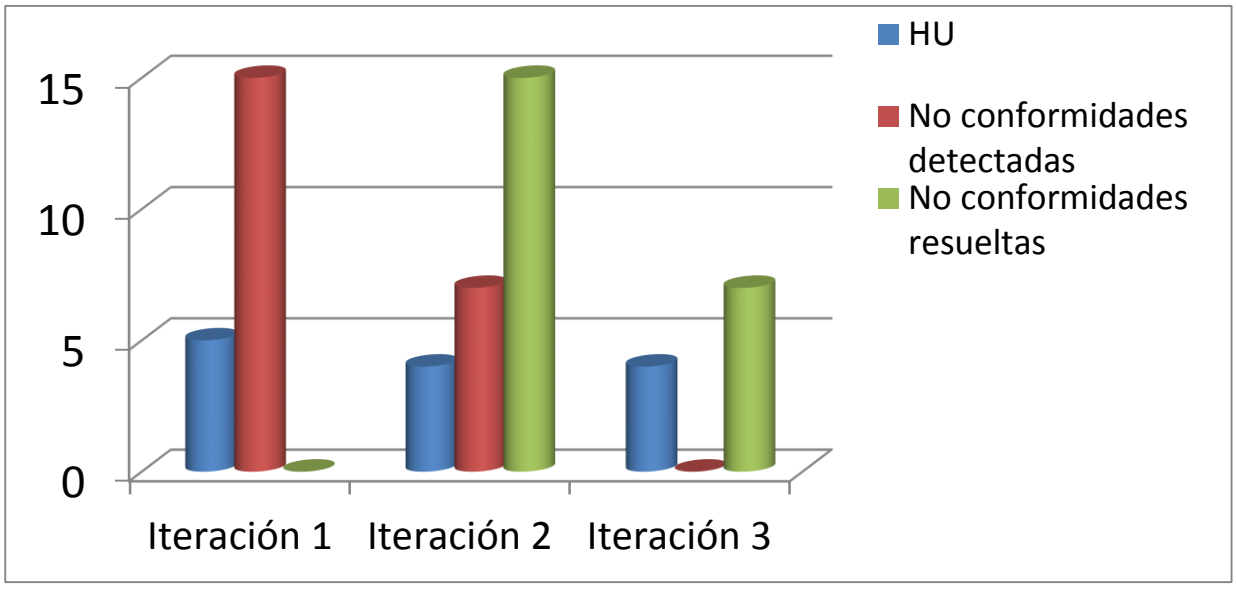


Figura 15: Representación gráfica de los resultados de las pruebas de aceptación

3.8 Conclusiones

1. Mediante el estilo de codificación determinado se concibe un código organizado y fluido logrando así de manera eficaz la implementación de las funcionalidades descritas en las historias de usuario.
2. Con la definición de las iteraciones y de las tareas en cada una de ellas se logró una planificación eficaz de la implementación del sistema en el tiempo especificado para cada una de las iteraciones.
3. Mediante las pruebas realizadas se identificaron los errores del sistema lo que contribuyó a dar solución a los mismos para satisfacer el comportamiento esperado por el usuario final de las funcionalidades del *software* implementado.

Conclusiones

Al finalizar el desarrollo de este trabajo de diploma se ha arribado a las siguientes conclusiones:

1. Mediante el análisis de los algoritmos y estándares criptográficos, las herramientas, tecnologías y las aplicaciones similares se logró un mejor entendimiento del proceso de desarrollo del producto y permitió definir como estándar criptográfico a utilizar debido a sus estructuras de datos al PKCS #7.
2. Mediante la propuesta de solución se elaboró un esquema permitiendo solventar las limitaciones definidas en la situación problemática.
3. Haciendo uso de la librería *BouncyCastle* y las estructuras de datos del estándar PKCS #7 se logró la implementación de las funcionalidades para el cifrado, descifrado, firma digital y control de acceso a los archivos.
4. Con las pruebas realizadas se identificaron los errores del sistema lo que contribuyó a dar solución a los mismos para satisfacer el comportamiento esperado por el usuario final de las funcionalidades del *software* implementado.

Recomendaciones

1. Añadir su propio almacén de llaves.
2. Incluir firma reconocida o completa.

Bibliografía

Philip Zimmermann . [En línea] http://www.philzimmermann.com.

INTECO . 2009. National Institute of Communication Technologies. *National Institute of Communication Technologies.* [En línea] 16 de 12 de 2009. http://www.inteco.es/Seguridad/DNI_Electronico/Firma_Electronica_de_Documentos/Que_es_la_Firma_electronica/que_es_la_firma_27.

Alexander, Christopher. 1979. *The Timeless Way of Building.* s.l. : Oxford University Press, 1979.

Baeza, José T. Hardware Criptográfico en las PKI. *Hardware Criptográfico en las PKI.* [En línea] www.revista-ays.com..

Beck, Kent y Gamma, Erich. 1998. *Test Infected: Programmers Love Writing Tests.* 1998. Vol. 3.

Braicovich, Gustavo Ariel. 1999. *Firma Digital.* 1999.

Calo, Karla Mendes, Estevez, Elsa y Fillottrani, Pablo. 2010. *Evaluación de Metodologías Ágiles para Desarrollo de Software.* Argentina : Dpto. de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, 2010.

Craig, Larman. 1995. *Applying UML and Patterns.* 1995.

—. 1999. *UML y Patrones .* Naucalpan de Juhez : s.n., 1999.

Crispin, Lisa y House, Tip. 2002. *Testing Extreme Programming.* 2002.

Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John. 1995. *Design Patterns: Elements of Reusable Object Oriented Software.* s.l. : Addison Wesley, 1995.

Gamma, Erich y Beck, Kent. JUnit Cookbook. [En línea] http://junit.sourceforge.net/doc/cookbook/cookbook.htm.

Goss, Gerhard, Hartmanis, Juris y Leeuwen, Jan Van. 1994. *Advances in Cryptology Eurocrypt'94.* perugia : s.n., 1994.

Highsmith J., Orr K. 2000. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* s.l. : Dorset House, 2000.

Highsmith, J. 2002. *Agile Software Development Ecosystems.* s.l. : Addison-Wesley Professional, 2002.

Jeffries, Ronald. 2000. *Extreme Testing.* 2000.

Jones, Jennifer y Johnston, Margret. 200. *Digital signature bill enables e-commerce.* 200.

Kent, Beck. 2000. *Una explicación de la programación extrema.Aceptar el cambio.* s.l. : Addison Wesley, 2000.

Koch, Werner. 2002. GnuPG. [En línea] 16 de 07 de 2002. http://www.gnupg.org/.

Laboratorio de Redes y Seguridad. UNAM - Facultad de Ingeniería, División de Ingeniería Eléctrica. 2012. Universidad Nacional Autónoma de México. *Facultad de Ingeniería.* [En línea] 2012. http://redyseguridad.fi-p.unam.mx/proyectos/criptografia/criptografia/index.php/2-tecnicas-clasicas-de-cifrado/23-numero-de-claves.

Letelier, Patricio. 2006. Departamento de Sistemas Informáticos y Computación (DSIC). *Universidad Politécnica de Valencia (UPV).* [En línea] 2006. [Citado el: 5 de Febrero de 2013.] www.cyta.com.ar/ta0502/v5n2a1.htm.

Lezcano, Elisa Zoraida Mattos. 2003. *Seguridad en el Comercio Electronico.* 2003.

—. 2003. *Seguridad en el Comercio Electronico.* 2003.

López, Manuel J. Lucena. 2004. *Criptografía y Seguridad.* 2004.

Llorente, Cesar de la Torre, y otros. 2010. *Guía de patrones N-capas orientado al dominio con .NET.* 2010.

Marquina, Ernesto . 2008. *Guía de Patrones, Prácticas y Arquitectura .NET.* 2008.

Microsoft. 2013. MSDN. *MSDN.* [En línea] 06 de 05 de 2013. http://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

Penalver, Juan Lopez. 1998. HispaSec. *HispaSec.* [En línea] 03 de 11 de 1998. [Citado el: 23 de 05 de 2013.] http://unaaldia.hispasec.com/2012/06/la-creacion-del-certificado-falso-usado_11.html.

Perramon, Xavier. 2004. *Aplicaciones Seguras.* 2004.

Ramos, Gianny Miranda. 2010. *Componente de software para la firma digital de documentos jurídicos tratados en formato electrónico en el proyecto Tribunales.* 2010.

Rivest, R.L., Shamir, A. y Adleman, L. 1978. *A Method for Obtaining Digital Signature and Public-Key Cryptosystems.* 1978.

Roberth G. Figueroa, Camilo J. Solís, Armando A. Cabrera. 2011. [En línea] 2011. http://www.docstoc.com/docs/102328746/articulo-metodologia-de-sw-formato.

ROBINSON, MATTHEW y VOROBIEV, PAVEL. 2005. *Java Swing 2nd edition.* 2005. Vol. 2.

Shneiderman, Ben. 1998. *Designing the user interface: Strategies for effective Human-computer interaction.* s.l. : Addison-Wesley, 1998.

Tau Ceti Co-operative Ltd. 2000. The Legion of the Bouncy Castle. *The Legion of the Bouncy Castle.* [En línea] 13 de 04 de 2000. [Citado el: 22 de 5 de 2013.] http://www.bouncycastle.org/.

Winters, Travis. 2000. *The Legion of the Bouncy Castle.* [En línea] 2000. www.bouncycastle.org.

Anexos

Anexo 1: Tareas de la Ingeniería Detalladas Iteración 1

Tabla 30: HU1_T2

Tarea	
Número de la Tarea: HU1_T2	**HU:** Generar estructura RecipientInfo
Nombre: Importar certificado	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir importar certificados.	

Tabla 31: HU3_T1

Tarea	
Número de la Tarea: HU3_T1	**HU:** Generar mensaje cifrado.
Nombre: Diseñar una interfaz de usuario para la gestión de archivos a cifrar	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: La interfaz debe permitir agregar, eliminar y cifrar archivos.	

Tabla 32: HU3_T2

Tarea	
Número de la Tarea: HU3_T2	**HU:** Generar mensaje cifrado
Nombre: Importar claves desde un almacén de llaves	

Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: El sistema debe permitir importar las claves, con las que se va a cifrar el mensaje desde un almacén de llaves.	

Tabla 33: HU3_T3

Tarea	
Número de la Tarea: HU3_T3	**HU:** Generar mensaje cifrado
Nombre: Cifrar un mensaje	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: El sistema debe permitir cifrar archivos para varios destinatarios a través de certificados.	

Tabla 34: HU6_T1

Tarea	
Número de la Tarea: HU6_T1	**HU:** Descifrar contenido de la estructura RecipientInfo
Nombre: Importar clave desde un almacén de llave	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir importar claves desde un almacén de llaves.	

Tabla 35: HU6_T2

Tarea

Número de la Tarea: HU6_T2	**HU:** Descifrar contenido de la estructura RecipientInfo
Nombre: Descifrar estructura RecipientInfo	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir descifrar cada recipiente de la estructura RecipientInfo a través de una clave guardada en un almacén de llaves.	

Tabla 36: HU6_T2

Tarea	
Número de la Tarea: HU6_T2	**HU:** Descifrar contenido del mensaje
Nombre: Diseñar una interfaz de usuario para la gestión de archivos a descifrar	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: La interfaz debe permitir agregar, eliminar, y descifrar archivos.	

Tabla 37: HU7_T2

Tarea	
Número de la Tarea: HU7_T2	**HU:** Descifrar contenido del mensaje
Nombre: Importar clave desde un almacén de llaves	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: El sistema debe permitir importar claves desde un almacén de llaves para descifrar los archivos cifrados.	

Tabla 38: HU7_T3

Tarea	
Número de la Tarea: HU7_T3	**HU:** Descifrar contenido del mensaje
Nombre: Descifrar un mensaje	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: El sistema debe permitir agregar archivos para descifrar a través de claves guardadas en un almacén de llaves.	

Anexo 2: Tareas de la Ingeniería Detalladas Iteración 2

Tabla 39: HU2_T2

Tarea	
Número de la Tarea: HU2_T2	**HU:** Generar estructura SignerInfo
Nombre: Importar clave privada desde un almacén de llaves	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir importar claves almacenadas en almacenes de llaves.	

Tabla 40: HU4_T1

Tarea	
Número de la Tarea: HU4_T1	**HU:** Generar estructura SignedAndEnvelopedData
Nombre: Crear estructura SignedAndEnvelopedData	
Tipo: Desarrollo	**Puntos Estimados:**1

Responsable: Frank Geiler Vega Duverger
Descripción: El sistema debe permitir empaquetar la estructura SignerInfo y los archivos cifrados en una estructura SignedAndEnvelopedData.

Tabla 41: HU5_T1

Tarea	
Número de la Tarea: HU5_T1	**HU:** Gestionar contenido de la estructura SignedAndEnvelopedData
Nombre: Gestionar contenido de la estructura SignedAndEnvelopedData	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Frank Geiler Vega Duverger	
Descripción: El sistema debe permitir gestionar la estructura SignerInfo y los archivos cifrados que están en una estructura SignedAndEnvelopedData.	

Tabla 42: HU5_T1

Tarea	
Número de la Tarea: HU5_T1	**HU:** Descifrar contenido de la estructura SignerInfo
Nombre: Descifrar contenido de la estructura SignerInfo	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir descifrar la estructura SignerInfo a través de certificados.	

Tabla 43: HU5_T2

Tarea	
Número de la Tarea: HU5_T2	**HU:** Descifrar contenido de la estructura SignerInfo
Nombre: Importar certificados	
Tipo: Desarrollo	**Puntos Estimados:**1
Responsable: Olber Rios Cutiño	
Descripción: El sistema debe permitir importar certificados de diferentes extensiones.	

Anexo 3: Casos de pruebas unitarias

Tabla 44: Caso de prueba unitaria Desencriptar

Prueba de Unitaria		
Nombre Prueba: testDesencriptar		
Estado: Satisfactoria.	**Tipo:** Caja Blanca.	**Última Ejecución:** 13/05/2013
Ejecutado por: Frank Geiler Vega Duverger		**Verificado por:** Félix A. Prieto
Descripción: Para que la prueba se ejecute satisfactoriamente y pueda cifrar los datos se debe proporcionar una lista de archivos y un certificado.		
Entrada: PrivateKey private_Key, LinkedList<File> lista_Archivos		
Criterio de aceptación: Datos cifrados		
Resultado: 		

Tabla 45: Caso de prueba unitaria Firmar_Archivos

Prueba de Unitaria		
Nombre Prueba: testFirmar_Archivos		
Estado: Satisfactoria.	**Tipo:** Caja Blanca.	**Última Ejecución:** 13/05/2013
Ejecutado por: Olber Rios Cutiño		**Verificado por:** Félix A. Prieto
Descripción: Para que la prueba se ejecute satisfactoriamente y pueda firmar los datos se debe proporcionar una lista de archivos y una clave privada.		
Entrada: PrivateKey private_Key, LinkedList<File> lista_Archivos		

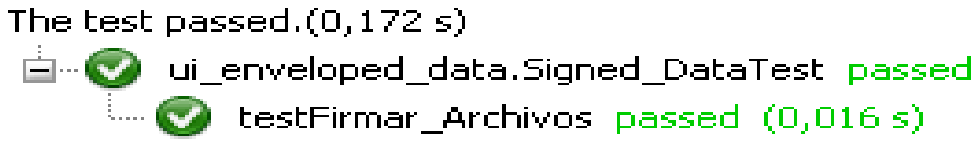
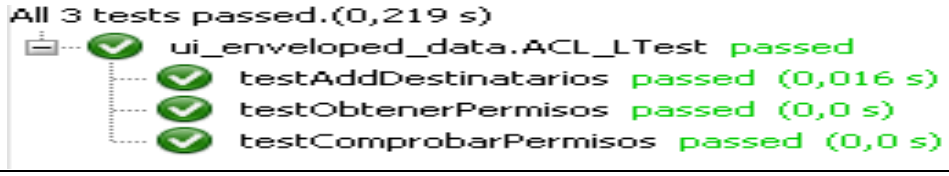
Criterio de aceptación: Datos firmados
Resultado: 

Tabla 46: Caso de prueba unitaria ACL

Prueba de Unitaria		
Nombre Prueba: testACL		
Estado: Satisfactoria.	**Tipo:** Caja Blanca.	**Última Ejecución:** 13/05/2013
Ejecutado por: Olber Rios Cutiño		**Verificado por:** Félix A. Prieto
Descripción: Para que la prueba se ejecute satisfactoriamente y pueda crear la ACL debe proporcionar una lista de certificados de los destinatarios con acceso al mensaje.		
Entrada: LinkedList< x509v3Certificate > certificados		
Criterio de aceptación: Creación de una lista de control de acceso		
Resultado: 		

Anexo 4: Casos de pruebas de aceptación

Tabla 47: Prueba de aceptación para generar mensaje cifrado con la ACL

Caso de Prueba de Aceptación	
Código:HU3_P2	**HU:** Generar mensaje cifrado con la ACL
Nombre: Generar mensaje cifrado con la ACL	
Descripción: Prueba de funcionalidad para la generación de un mensaje cifrado con una ACL	
Condiciones de ejecución: Se debe importar al menos un certificado para crear la ACL y un archivo	
Entrada / Pasos de ejecución: • El usuario importa todos los archivos a cifrar • El usuario importa todos lo certificados para crear la ACL	

Resultado esperado: • La creación de un mensaje cifrado con una lista de control de acceso
No conformidades:
Evaluación de la prueba: Prueba satisfactoria

Tabla 48: Prueba de aceptación para descifrar contenido de la estructura RecipientInfo

Caso de Prueba de Aceptación	
Código:HU6_P3	**HU:** Descifrar contenido de la estructura RecipientInfo
Nombre: Descifrar contenido de la estructura RecipientInfo	
Descripción: Prueba de funcionalidad para descifrar de la estructura RecipientInfo	
Condiciones de ejecución: Se debe importar una clave privada asocia a la estructura RecipientInfo	
Entrada / Pasos de ejecución: • El usuario debe importar una clave privada	
Resultado esperado: • La estructura RecipientInfo asociada a la clave privada descifrada	
No conformidades:	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 49: Prueba de aceptación para descifrar contenido del mensaje

Caso de Prueba de Aceptación	
Código:HU7_P4	**HU:** Descifrar contenido del mensaje
Nombre: Descifrar contenido del mensaje	
Descripción: Prueba de funcionalidad para descifrar el contenido del mensaje	
Condiciones de ejecución: Se debe importar un certificado para seleccionar la estructura RecipientInfo para obtener la clave de sesión y una clave privada para esa estructura	
Entrada / Pasos de ejecución: • El usuario debe importar un certificado y una clave privada	

Resultado esperado: • El mensaje descifrado con la ACL
No conformidades:
Evaluación de la prueba: Prueba satisfactoria

Tabla 50: Prueba de aceptación para gestionar certificados

Caso de Prueba de Aceptación	
Código:HU9_P5	**HU:** Gestionar certificados
Nombre: Gestionar certificados	
Descripción: Prueba de funcionalidad para importar certificados	
Condiciones de ejecución: Se deben importar certificados	
Entrada / Pasos de ejecución: • El usuario debe importar al menos un certificado	
Resultado esperado: • Los certificados cargados correctamente	
No conformidades:	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 51: Prueba de aceptación para generar estructura SignerInfo

Caso de Prueba de Aceptación	
Código:HU2_P6	**HU:** Generar estructura SignerInfo
Nombre: Generar estructura SignerInfo	
Descripción: Prueba de funcionalidad para crear estructura SignerInfo	
Condiciones de ejecución: Se debe importar una clave privada	
Entrada / Pasos de ejecución: • El usuario debe importar clave privada • Realizar resumen al mensaje • Cifrar el resumen con la clave privada	

Resultado esperado: • Obtención de una firma digital
No conformidades:
Evaluación de la prueba: Prueba satisfactoria

Tabla 52: Prueba de aceptación para generar estructura SignedAndEnvelopedData

Caso de Prueba de Aceptación	
Código:HU4_P7	**HU:** Generar estructura SignedAndEnvelopedData
Nombre: Generar estructura SignedAndEnvelopedData	
Descripción: Prueba de funcionalidad para crear estructura SignedAndEnvelopedData	
Condiciones de ejecución: Se debe crear las estructuras SignerInfo, RecipientInfo y el mensaje cifrado con la ACL	
Entrada / Pasos de ejecución: • El usuario debe crear todas la estructuras mencionadas en las condiciones de ejecución para encapsularlas en SignedAndEnvelopedData • Exportar la estructura en un formato definido (.saedzip)	
Resultado esperado: • Obtención de la estructura SignedAndEnvelopedData con todas sus estructuras cifradas	
No conformidades:	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 53: Prueba de aceptación para Gestionar contenido de la estructura SignedAndEnvelopedData

Caso de Prueba de Aceptación	
Código:HU5_P8	**HU:** Gestionar contenido de la estructura SignedAndEnvelopedData.
Nombre: Gestionar contenido de la estructura SignedAndEnvelopedData.	
Descripción: Prueba de funcionalidad para gestionar estructura SignedAndEnvelopedData	
Condiciones de ejecución: Se debe importar un estructura SignedAndEnvelopedData con extensión .saedzip	

Entrada / Pasos de ejecución: • Importar estructura SignedAndEnvelopedData • Descifrar la estructura SignedAndEnvelopedData
Resultado esperado: • Las estructuras SignerInfo, RecipientInfo y el mensaje con la ACL
No conformidades:
Evaluación de la prueba: Prueba satisfactoria

Tabla 54: Prueba de aceptación para Descifrar contenido de la estructura SignerInfo

Caso de Prueba de Aceptación	
Código:HU8_P9	**HU:** Descifrar contenido de la estructura SignerInfo
Nombre: Descifrar contenido de la estructura SignerInfo	
Descripción: Prueba de funcionalidad para descifrar contenido de la estructura SignerInfo	
Condiciones de ejecución: Se debe importar los certificados	
Entrada / Pasos de ejecución: • Importar certificados para descifrar la estructura SignerInfo • Comprobar firma digital	
Resultado esperado: • La verificación de la firma sea correcta	
No conformidades:	
Evaluación de la prueba: Prueba satisfactoria	