Proyecto Fin de Carrera:

Implementación de un esquema criptográfico para gestionar de forma segura las historias médicas de los pacientes a través de una red de comunicaciones.

Fran Cáncer Giner.
Ingeniería en Informática

Jordi Castellà Roca
Consultor

Fecha de entrega
7 de enero de 2007

Agradecimientos y Dedicatoria.

Finalizando este proyecto no solo termino una asignatura más sino que doy por acabado un viaje que ha durado varios años y que me ha proporcionado muchos conocimientos y satisfacciones.

Es esta una de las raras oportunidades en las que se te pide que hables de ti y de lo que este proyecto ha significado en lugar de que hables de los conocimientos que has adquirido y es por eso que quiero aprovechar este pequeño y restringido foro para expresar mi agradecimiento a mi mujer, Mª Angeles Guallarte Orta. Ella me ha apoyado y animado desde el principio demostrando tener fe en mí y una gran visión de futuro. Gracias cariño.

Espero que en los años venideros consiga sacar fruto al esfuerzo realizado y sin ninguna duda poder continuar con el proceso de formación que seguramente se prolongará durante muchos años más.

Dedico este proyecto a mi mujer.

Todas las personas tarde o temprano han de asistir al médico para consultar alguna dolencia o enfermedad. La centralización de los datos y su fácil acceso permite facilitar el diagnóstico a los especialistas. Cada vez son menos los centros de salud que guardan las historias clínicas en papel. La naturaleza de los datos clínicos, radiografías, pruebas de sangre y orina, diagnósticos, prescripciones son informaciones que se han de tener bajo una buena custodia ya que con estos datos se podría llegar a hacer un uso fraudulento.
Hay muchos problemas que hay que solucionar, como por ejemplo: ¿Cómo podemos asegurar la autenticidad y la integridad de los datos?, ¿Cómo podemos estar seguros que quien consulta los datos es quien dice ser?

En este proyecto de final de carrera se ha diseñado, implementado y probado un sistema que utilizando criptografía de clave pública, garantiza la correcta utilización de los datos en un proceso de consulta y cumplimentación de una historia clínica informatizada consultada a distancia.

En un proceso de informatización de una historia clínica hay que garantizar como mínimo las siguientes propiedades: La confidencialidad y autenticidad de los datos, la integridad de la información guardada, la imposibilidad del repudio.

Garantizar estas propiedades utilizando un sistema informático no es una tarea trivial, y a demás de los elementos de la red de comunicaciones, hace falta la utilización de medidas concretas a nivel de aplicación. Por este motivo se ha diseñado un protocolo criptográfico para cada uno de los procesos. El conjunto de estos protocolos forman el esquema criptográfico que ha sido implementado en este sistema para gestionar de forma segura las historias médicas de los pacientes a través de una red de comunicaciones.

El sistema presentado permitiría que actualmente se pudieran gestionar datos médicos en un entorno controlado. El hecho de que los datos estén en formato digital facilita mucho la gestión de los mismos. El paciente puede consultarlos desde cualquier punto conectado a la red, al igual que los médicos.

Índice

Índice de figuras.

.

1. Introducción.

1.1 Justificación del Proyecto Final de Carrera.

En los tiempos en los que vivimos la informática se ha extendido a todos los aspectos de la vida diaria. Ya es muy difícil desarrollar una actividad profesional sin la presencia de sistemas informáticos.
El negocio de la sanidad es un negocio de carácter prioritario. Cualquier usuario de un hospital, clínica o ambulatorio quiere ser atendido de manera rápida y precisa sin tener que aguantar largas esperas y mucho menos fallos administrativos que le puedan acarrear problemas para su salud.

La propagación de las tecnologías de la información y principalmente de Internet, han incrementado notablemente las posibilidades que se les ofrecen a los usuarios permitiendo en algunos casos incluso la asignación de visitas o consulta de resultados desde la casa del cliente.

La cantidad de información que puede acumular un paciente a lo largo de un tratamiento por una enfermedad es muy grande y además se ha de tener toda en cuenta para futuros diagnósticos o futuras dolencias. Uno de los grandes retos que se han planteado los centros de salud es la eliminación parcial o completa del papel en las historias clínicas. Otro reto es la centralización de la información de forma que cualquier profesional pueda consultar la totalidad de la información sin tener primero que solicitar que le traigan la historia. Como ejemplo podemos plantear el caso de un servicio de urgencias el cual recibe un paciente que ha tenido un accidente de coche. No pueden permitirse perder demasiado tiempo en conocer datos vitales que les permitirán realizar un diagnóstico más ajustado.

Dentro de esta problemática surge la necesidad de implantar un sistema que permita todo lo anterior sin dejar de lado la seguridad. Los datos médicos son muy susceptibles de ser usados malintencionadamente. Se podrían vender a un laboratorio médico para que pudiera realizar una campaña de marketing de un producto a personas que saben que tienen cierta enfermedad.
Por ahora en España hay una ley sobre la privacidad de la información LOPD que hay que cumplir.

1.2 Contexto en el que se desarrolla.

Hace relativamente poco tiempo todo lo que este proyecto propone habría estado un poco fuera de lugar, pero hoy se nos presenta un marco en el que un paciente puede consultar sus datos clínicos desde su propia casa y en estos casos un sistema como este es necesario.

Las principales razones del porqué hace poco no se habría podido hacer son el casi inexistente acceso desde las casa a la red, que no se popularizó hasta finales de los 90. La capacidad de las máquinas para realizar operaciones complejas en un tiempo razonable también era un problema añadido, ya que se necesita una capacidad de proceso importante a la hora de realizar los cálculos criptográficos y de comunicaciones por la red.

Hoy en día la mayoría de los hogares de nuestro país tienen ordenador y a la vez disponen de una conexión, con más o menos ancho de banda a Internet, pero suficiente para cubrir las necesidades que este sistema necesita. Estas características hacen que la implantación del sistema sea relativamente fácil.

1.3 Objetivos del Proyecto Final de Carrera.

El objetivo del proyecto final de carrera es la implementación de un sistema para la incorporación y consulta de datos en una historia clínica informatizada cumpliendo en todo momento las siguientes propiedades:

Características que el sistema criptográfico debería de cumplir:

- Confidencialidad:
  Se tiene que preservar la confidencialidad de los datos del historial médico de los pacientes.

- Autenticidad:
  La información que se guarda en el sistema tiene que disponer de una prueba de su autenticidad.

- Integridad:
  Una vez la información ha sido generada se tiene que garantizar en todo momento su integridad.

- No repudio:
  Si un usuario del sistema hace una cierta acción más tarde no puede negar que la haya hecho.

Estas características se deben de mantener durante todos los procesos que se van a implementar:

- Consulta de los datos de un paciente.
- Consulta de los pacientes de un médico.
- Inserción de una nueva visita.

En los siguientes capítulos de esta memoria se tratarán cada uno de estos procesos más ampliamente.

A demás de estos objetivos iniciales, se quiere que el sistema ejecute un aplicativo de manera remota y que conecte a un servidor central. Existirá un aplicativo para el médico y uno para el paciente. Es necesario que los aplicativos se autentifiquen en el momento que quieren realizar alguna operación con el servidor central.

Este servidor central tendrá acceso a una base de datos que contendrá todos los datos de todos los pacientes.

Así pues, a los objetivos iniciales se añaden los siguientes:

- La implementación de un sistema para representar los datos de manera que las partes se entiendan: XML [10].

- La comunicación entre el aplicativo utilizado por el médico y el sistema central y entre el paciente y el sistema central se realizará utilizando RMI[5] de Java[4]. La utilización de RMI[5] reducirá en gran medida el tiempo de desarrollo del proyecto.

- Una implementación para llevar a cabo la gestión de la base de datos. Esta gestión incluye, la entrada de las personas en la base de datos que se realizará desde el gestor de historias clínicas. La entrada de datos por parte del médico. Finalmente todas las consultas relacionadas con estas acciones.

- Implementación de un protocolo de autenticación que permitirá tener la certeza de quién pide realizar una operación y si está autorizado.

Se ha realizado un esfuerzo importante a la hora de diseñar las clases que implementarán los objetivos enumerados, de manera que , si en el futuro es necesario modificar alguna funcionalidad del aplicativo, ya sea para añadir funcionalidades o para modificar las características actuales, solo sea necesario modificar el mínimo de clases posibles. De aquí la importancia de realizar un diseño y una implementación incremental, como se muestra en la planificación del proyecto.

1.4 Enfoque y método seguido.

Este sistema necesita la interacción de los siguientes módulos:

- El esquema criptográfico es la base del proyecto. Las clases se han diseñado de manera que puedan ser reutilizadas en cualquiera de los aplicativos.

- Los usuarios del sistema se han de comunicar y por este motivo se necesita una plataforma de comunicaciones que permita la transmisión de los datos en formato XML[10] entre el servidor central y las otras partes del sistema, ya sea el médico o el paciente.

- La información que recibe el gestor de historias clínicas se guarda en una base de datos.
- Finalmente, cada usuario del sistema necesita una interfaz gráfica para llevar a cabo las funcionalidades del sistema.

Así pues resumiendo, el sistema contará con los siguientes módulos:

- Esquema Criptográfico.
- Representación de los datos en XML[10].
- Comunicación de los componentes utilizando RMI[5].
- Protocolo de autenticación
- Base de datos.
- Interfaz gráfica.

El método que se ha seguido para implementar el sistema ha sido el de ir construyendo cada uno de los módulos sucesivamente uno detrás del otro efectuando pruebas unitarias con cada uno de ellos. Cada módulo se integrará en el siguiente y se realizarán pruebas unitarias de nuevo. Así el que se ha hecho paso a paso es:

- Definición del protocolo criptográfico.
- Creación de la infraestructura de clave pública.
- Implementación del protocolo de las historias clínicas en un entorno local, esto incluye todo el esquema criptográfico.
- Integración del módulo en XML[10].
- Implementación de las comunicaciones de los componentes. En este paso se envían telemáticamente los documentos XML[10] entre los entornos locales (aplicativo del médico o del paciente) y el servidor central utilizando la tecnología RMI[5] de Java[4].
- Implantación de la base de datos. Los datos recibidos vía RMI[5] se guardan en la base de datos MySQL[9].
- Implementación del sistema de autenticación de los médicos y de los pacientes contra el servidor central. Se añade este módulo a las partes que ya estaban implementadas hasta el momento.
- Cuando todo este sistema funciona, se crea una interfaz para el médico y otra para el paciente para poder utilizar las diferentes funciones de una manera intuitiva y cómoda.

Es importante destacar que la parte de la interfaz ha sido uno de los objetos menos prioritarios del sistema. Se ha priorizado el correcto funcionamiento del sistema de acuerdo con los objetivos fijados. La interfaz podría hacer más o menos cosas, pero eso ya podría formar parte de un proyecto en si mismo, aumentando la usabilidad del entorno gráfico, sobretodo para personas no familiarizadas en informática.

Durante la implementación del proyecto, se ha intentando en todo momento utilizar software de libre distribución. De esta manera la implementación se ha realizado utilizando código Java[4] sobre el entrono de desarrollo Eclipse[7]. La base de datos utilizada es MySQL[9]. La única parte del proyecto que no es de libre distribución es la librería criptográfica IAIK[15], aunque para fines educativos si que lo es. Si se decidiera poner el sistema en producción se tendría que adquirir una licencia de la librería en cuestión o escoger otra.

1.5 Planificación del proyecto.

El proyecto empezó al inicio del cuatrimestre de otoño del año 2007. La planificación que hice se detalla a continuación.

Del 24 al 30 de septiembre:

- Instalación y pruebas de la librería criptográfica IAIK. [15]
- Creación de los certificados genéricos para la Autoridad de Certificación, el médico, el paciente y el Administrador. (ver sección 2. PKI).

Del 1 al 21 de octubre:

- Diseño, implementación, test y documentación del esquema criptográfico:
  - Consulta de un historial médico.
  - Consulta de los pacientes asignados a un médico.
  - Inserción de datos en el historial médico de un paciente.

Del 22 de octubre al 4 de noviembre:

- Diseño, implementación, test y documentación del esquema XML[10]:

Del 5 al 18 de noviembre:

- Diseño, implementación, test y documentación de la base de comunicación RMI [5]
  - Parte del paciente.
  - Parte del médico.

Del 19 de noviembre al 2 de diciembre:

- Instalación de la base de datos.
- Creación del modelo.
- Pruebas de integración con el sistema actual.

Del 3 al 30 de diciembre:

- Creación de la interfaz gráfica.
- Pruebas de integración.

Del 31 de diciembre al 6 de enero:

- Documentación.

A continuación se incluye un documento MSProject que muestra de forma gráfica la planificación del proyecto. Este documento también se presenta adjunto a la memoria.

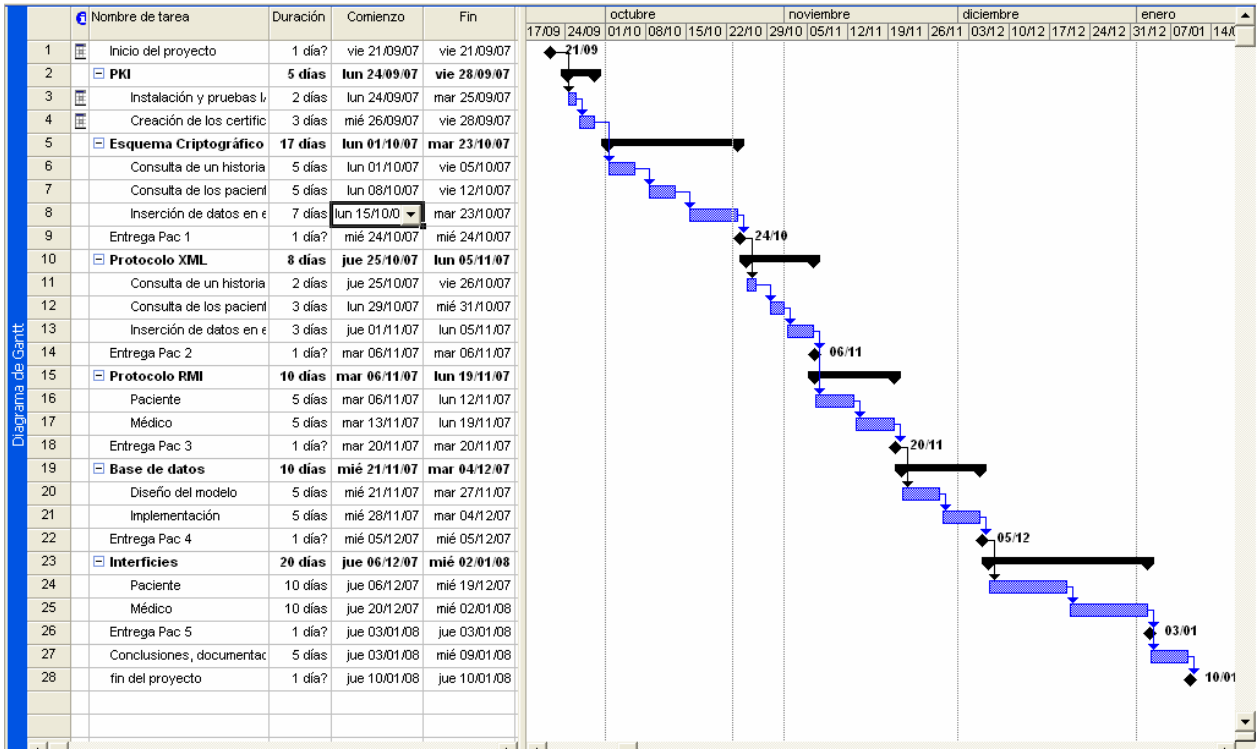


Imagen ***1****: Planificación del proyecto – Microsoft Project*

Como se puede ver en la planificación, la parte que me ha llevado más tiempo ha sido el estudio e implementación del protocolo criptográfico.

Hasta el momento de la entrega del proyecto se han hecho revisiones del código para asegurar que sea óptimo y libre de errores. Cabe destacar que la documentación se ha ido completando en cada fase dando lugar al documento actual.

1.6 Esquema general del proyecto.

A continuación se incluye el esquema general del funcionamiento del proyecto, esta imagen podría servir de resumen de lo que se ha venido explicando en esta introducción.

Es importante ver que hay una separación clara en los elementos del proyecto. Se trabaja siempre en un entorno cliente-servidor. Las aplicaciones no tienen acceso directo a la base de datos si no es mediante el servidor de historias clínicas. De la misma manera, el protocolo criptográfico lo ejecutan los clientes en sus máquinas locales y únicamente reciben los datos obtenidos vía RMI[5]. Siguiendo este modelo las claves privadas no viajan nunca por la red, y están siempre en la máquina del cliente.

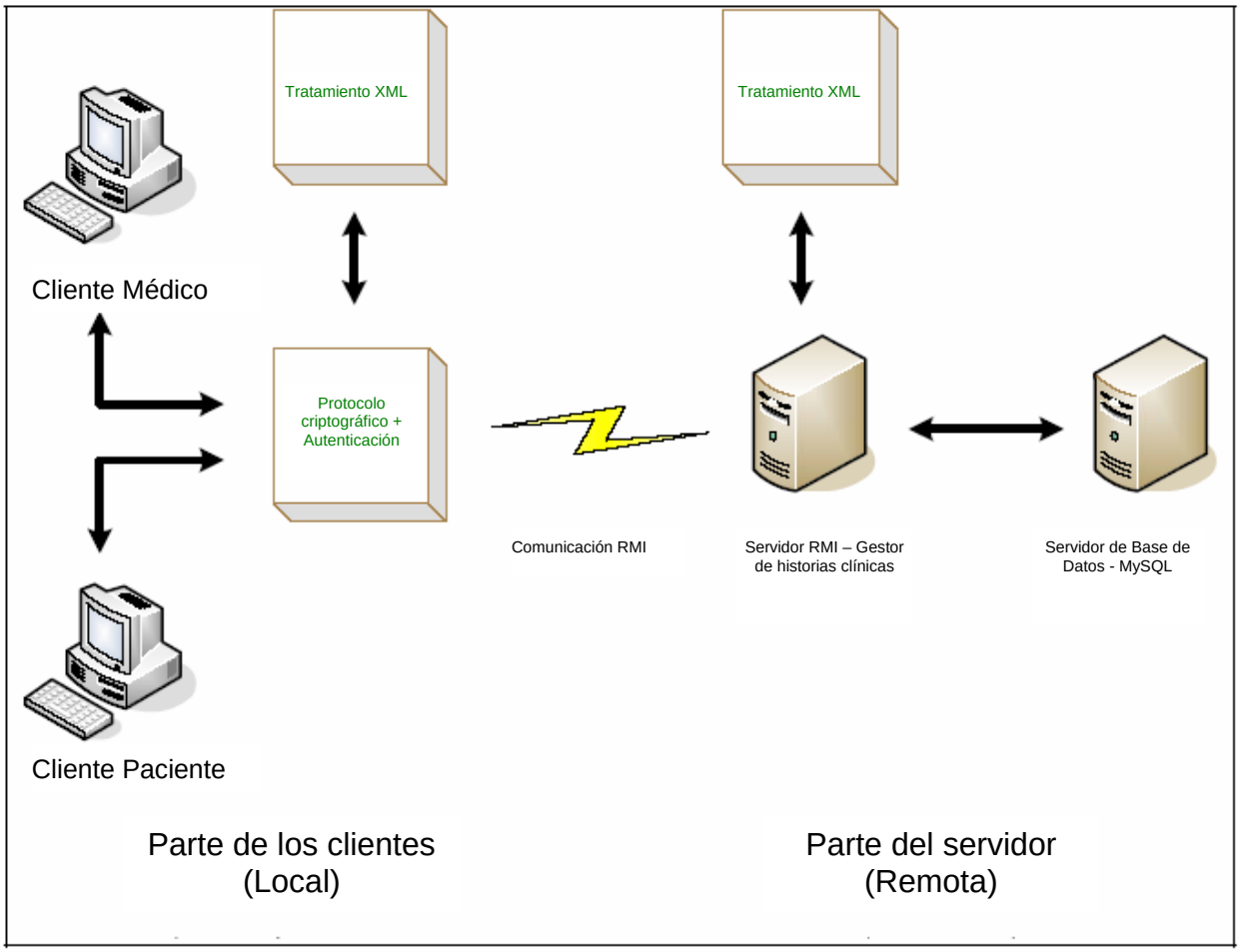


Imagen ***2****: Esquema general del proyecto*

1.7 Productos obtenidos.

Una vez todo el sistema ha sido implementado se ha obtenido un producto que está formado por diferentes componentes. El sistema completo lo integran 3 partes diferenciadas:

- Sistema gestor de historias clínicas. Incluye el servidor RMI[5], que permite que los clientes ejecuten código remoto, y también incluye el sistema gestor de bases de datos. A la base de datos solo tiene acceso el gestor de historias clínicas. No hay una imagen del servidor en funcionamiento porque este se ejecuta como un servicio interno de la máquina host del servidor.

- Aplicativo del médico. El médico utiliza este aplicativo para introducir los datos del paciente o para solicitar los datos del paciente.

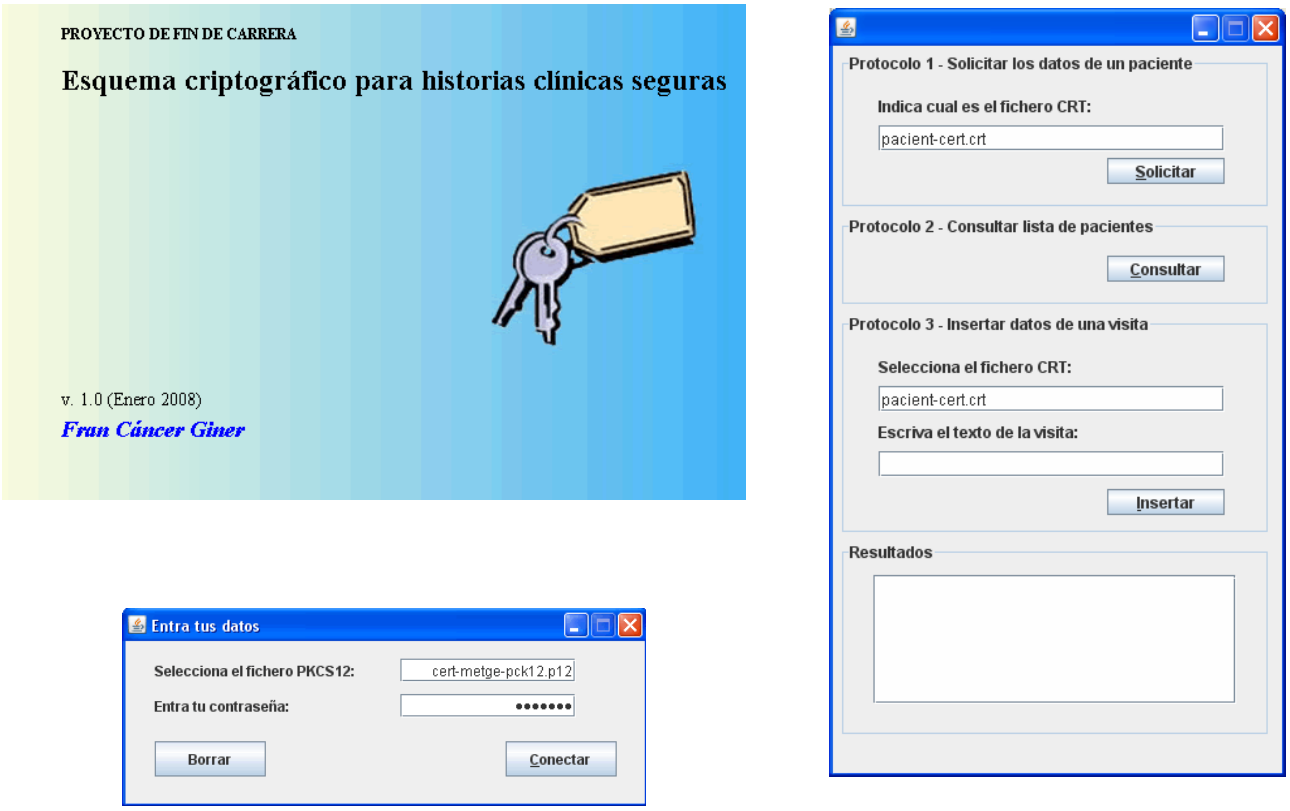


Imagen **3***: Aplicativo del médico ejecutado en un Windows XP.*

- Aplicativo del paciente. El paciente utiliza esta aplicativo para consultar los datos de su historia clínica.

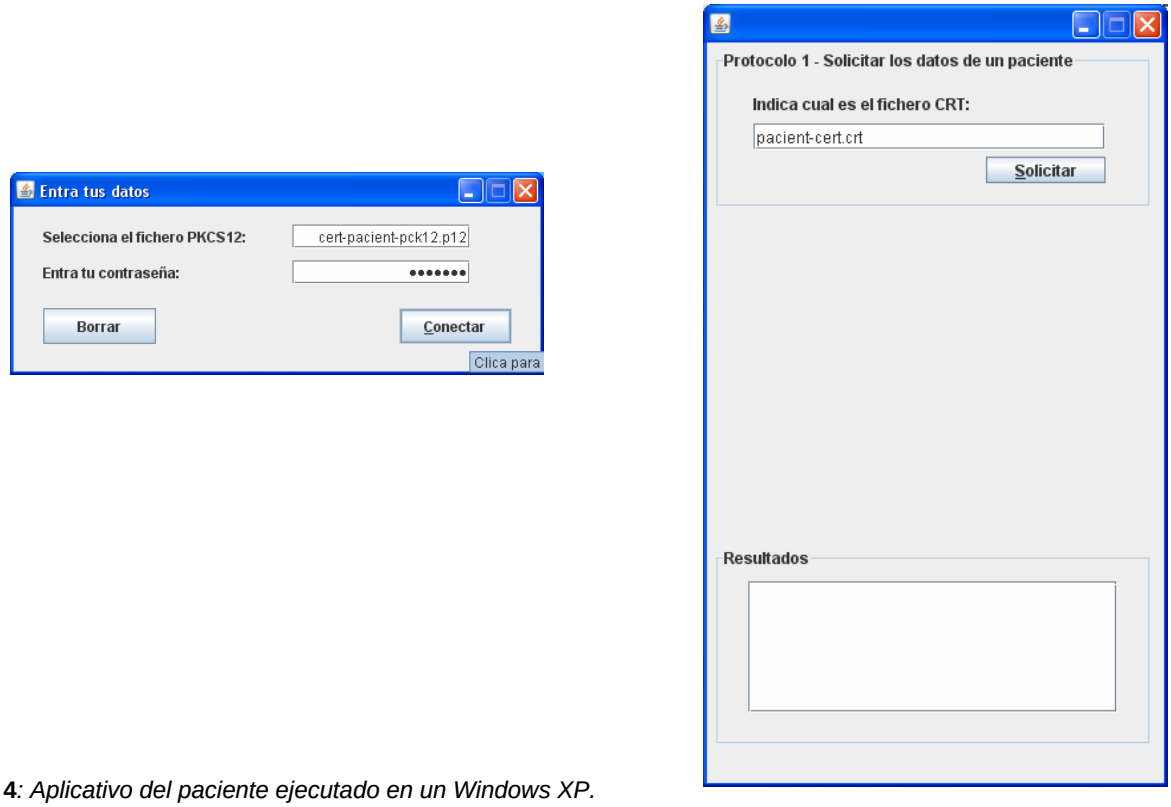


Imagen **4***: Aplicativo del paciente ejecutado en un Windows XP.*

La explicación detallada de estas interfaz y de su funcionamiento concreto se puede encontrar en la sección 8. Interfaz gráfica.

1.8 Descripción de los siguientes capítulos de la memoria.

Los siguientes capítulos de la memoria hacen referencia a las decisiones de diseño y a la implementación de los módulos necesarios del proyecto. La estructura de los capítulos sigue el diseño incremental comentado previamente.

- PKI
  En el resumen del proyecto se ha comentado que el esquema se basa en criptografía de clave pública. Por lo tanto, cada usuario del sistema ha de disponer de una pareja de claves. La infraestructura de clave pública necesaria para emitir y gestionar las parejas de claves de un grupo de usuarios con sus respectivos certificados se llama PKI. En este primer capítulo se explica qué es una infraestructura de clave pública y como se tiene que implementar en este caso.

- Esquema Criptográfico
  En este capítulo se explican los diferentes procesos que forman parte del mantenimiento de una historia clínica y el protocolo criptográfico de cada uno de los procesos elegidos para desarrollar en este proyecto. El conjunto de los protocolos es el que se llama esquema criptográfico.

- Representación de los datos: XML[10].
  La representación de los datos de manera que puedan ser gestionados y interpretados fácilmente es un punto importante. En el sistema presentado los datos intercambiados están en documentos XML[10]. En cada fase del protocolo solo se pasan entre las partes los datos de la historia que sean necesarios. Con el fin de llevar a cabo esta tarea se utiliza la librería JDOM[8].

- Comunicación de los componentes.
  Los programas del médico, el paciente y el gestor de historias clínicas se ejecutan en diferentes dispositivos. La comunicación de estos programas o componentes del sistema es una parte importante. Para reducir el tiempo de desarrollo se pensó en la utilización del RMI[5] de Java[4]. Este capítulo explica como se ha realizado la comunicación entre los clientes y el servidor utilizando RMI[5].

- Base de datos.
  La base de datos es necesaria para guardar el contenido de las historias clínicas. Se espera que el volumen de datos sea importante, de manera que el uso de una base de datos se hace indispensable. El sistema gestor de bases de datos (SGBD) que se ha utilizado es MySQL[9], porque es de libre distribución y cubre perfectamente las necesidades del proyecto.

- Protocolo de autenticación.
  Es necesario identificar y autentica a los usuarios. El sistema gestor necesita asegurarse que los clientes que están haciendo las peticiones son quienes dicen ser y que tienen derechos para poder acceder a los servicio que ofrece.

- Interfaz gráfica.
  Este capítulo explica como se ha diseñado la interfaz gráfica. Este era uno de los requerimientos menos prioritarios del sistema. De todas formas, cabe destacar que al mimo tiempo es fácil de utilizar y cumple todos los requerimientos de funcionalidad del sistema.

- Juego de pruebas.
  En este capítulo se explica con un ejemplo el funcionamiento de todo el proceso; desde el momento de crear los certificados, pasando por todo el ciclo de vida de una historia clínica. También se comenta la configuración de la máquina virtual Java[4], RMI[5], la base de datos, etc.

- Diagramas.
  Este capítulo Incluye el diagrama de clases completo.

- Trabajo futuro.
  Como en todo sistema complejo siempre hay cosas que mejorar, o nueva funcionalidades que añadir. En este capítulo se explican las mejoras que no se han acabado de implementar porque no estaban en la planificación inicial, pero que a lo largo del proyecto fui viendo que serían interesantes en el futuro para ampliar las funcionalidades y características del sistema.

- Conclusiones.
  En este capítulo se expone los objetivos conseguidos y como se han conseguido. También se destacan las consideraciones más importantes experimentadas durante la implementación.

- Bibliografía y anexos.
  En la bibliografía están las referencias a todos los documentos consultados, además de las direcciones donde se pueden conseguir los programas y librerías necesarias para poner en explotación el sistema. En los anexos se incluyen archivos de configuración, el glosario, etc.

2. PKI

2.1 Introducción

El esquema criptográfico implementado en este proyecto necesita que las diferentes partes (pacientes, médicos, administradores) dispongan de una pareja de claves y sus respectivos certificados.

Con tal de gestionar los certificados (emisión, revocación, etc) de un grupo de usuarios se utiliza una infraestructura de clave pública. Típicamente para hacer referencia a esta infraestructura se utilizan las siglas PKI, que corresponden al término en inglés Public Key Infraestructura.

Una PKI consta de una autoridad de certificación notada como CA, estas siglas corresponden al término en inglés Certification Authority. Otro componente de la PKI son las autoridades de registro, notadas con las siglas RA (Registry Autorty). Cuando un usuario quiere obtener un certificado normalmente realiza los pasos siguientes. En un primer paso crea una pareja de claves y realiza una petición de certificado mediante una RA. La RA valida la identidad del usuario que ha pedido el certificado y envía la petición a la CA. La CA recibe las peticiones de las RA y emite los certificados. La clave privada de la CA es una pieza de información muy sensible y por eso está en un entorno con un alto nivel de seguridad.

Si un usuario ve la clave privada correspondiente a su certificado a sido comprometida lo ha de comunicar a la CA. La CA revoca este certificado incluyéndolo en la lista de certificados revocados. La lista de certificados revocados la notaremos con las siglas CRL en inglés Certificado Revocation List.

La verificación de la validez de un certificado es un paso muy importante. No se puede aceptar un certificado como válido sin realizar los pasos siguientes.

Una primera comprobación es saber que CA lo ha emitido. Si la CA es de nuestra confianza, por ejemplo porque es una entidad de reconocido prestigio, seguimos con el proceso de verificación. El segundo paso es verificar que el certificado no está revocado. Para hacer esta comprobación podemos preguntarle directamente a la CA utilizando un protocolo diseñado para hacer esta consulta como por ejemplo OCSP[23], o descargándonos la CRL de la CA y buscando si está el certificado.

Para profundizar en el tema se recomienda ver las siguientes referencias [1]

En el proyecto se ha utilizado la librería de libre distribución Openssl[12] para construir de forma rápida y sencilla una pequeña PKI. Openssl está disponible con todas las versiones de Unix y existe una versión compilada para plataformas Windows[13].
En la implementación he utilizado la versión para Windows Win32OpenSSL-0_9_8e y las instrucciones que se muestran a continuación.

Los certificados emitidos siguen el estándar X.509[22]. La clave privada y el correspondiente certificado se han almacenado en un fichero siguiendo el formato estándar PKCS12[20].

2.2 Pasos que hay que seguir para generar los archivos necesarios.

El esquema implementado necesita que cada uno de los usuarios, incluidos el administrador tenga una pareja de claves con un fichero en formato PKCS12[20]. Este archivo es un contenedor con los siguientes elementos:

- Pareja de claves, pública y privada, del usuario.
- Certificado del usuario emitido por una autoridad de certificación (CA).
- Certificado de la autoridad de certificación.

El primer paso es la obtención del certificado de la autoridad de certificación. Si este sistema se quisiera implantar en un organismo que ya dispone de una infraestructura de clave pública con una autoridad de certificación propia no supondría ningún cambio substancial. En este caso no disponemos de esta autoridad y por eso el primer paso es crear una.

Los pasos para generar el certificado de la CA son:

- Generar la pareja de claves de la CA con una longitud de clave de 2048 bits. Esta acción se realiza utiliza ejecutando una serie de comandos desde el interprete de comandos de windows "Cmd". El fichero de salida que contiene la pareja de claves tiene el nombre de CA.key.

- Generar un certificado autofirmado con la pareja de claves de la CA. Este es el certificado de la CA. Para emitir este certificado se utiliza el fichero CA.key y una serie de comando. El fichero resultante con el certificado se llama CA.crt.

Ahora ya puedo pasar a crear los ficheros de los usuarios del sistema. Primero voy a crear el del médico.
.

- Generamos una pareja de claves para el médico. La longitud de las claves serán de 1024 bits.
- Emitimos una petición de certificado contra la CA que ya tenemos disponible.
- La CA emite el certificado. Se tiene que utilizar el fichero de configuración "openssl.cnf". Este fichero se puede encontrar en el anexo del proyecto.
- Generar el fichero PKCS12[20]. Este archivo contiene la pareja de claves del paciente, su certificado y el certificado de la CA.

Este mismo proceso se debe de hacer para el paciente y para el administrador. Los pasos son los mismos y de esta manera se pueden crear tantos usuarios como sea necesario.

Al final de este proceso tendremos los siguientes ficheros:

- CA.crt
- Cert-metge-pck12.p12
- Cert-pacient-pck12.p12
- Cert-administrador-pck12.p12

Los tres últimos son los que utilizará el aplicativo. Adjunto a la memoria se incluyen los archivos que se van utilizando durante el desarrollo.

2.3 Comandos de generación de los certificados.

Lo primero será crear mi CA (Autoridad Certificadora) que es la que nos valida y confirma que nuestro certificado es válido.

- Creación de una CA.

```
OPENSSL genrsa -des3 -out CA.key 2048

OPENSSL req -x509 -new -key CA.key -out CA.crt -days 360

    Country Name = ES
    State Name = BARCELONA
    Locality Name = BARCELONA
    Organization Name = UOC
    Organizacional Unit Name = CA
    Common Name = Fran Cáncer Giner
    Email Ardes = fcancerginer@gmail.com
```

Con este comando creamos una CA para certificados x509 con algoritmo de encriptación rsa triple DES de 2048 bytes.

Ahora voy a crear los certificados para los clientes. Estos certificados los usarán los distintos usuarios que quieran acceder a la aplicación.

- Creamos una clave privada para cada usuario:

Pacient:
openssl genrsa –des3 –passout pass:pfc2007 –out pacient-priv.key 1024

Metge:
openssl genrsa –des3 –passout pass:pfc2007 –out metge-priv.key 1024

Administrador:
openssl genrsa –des3 –passout pass:pfc2007 –out administrador-priv.key 1024

- Generar petición del certificado

Pacient:
Openssl req -new -sha1 -key pacient-priv.key -passin pass:pfc2007 -out petic-cert-pacient.csr

Pais: ES
Provincia: BARCELONA
Ciudad: BARCELONA
Organización: UOC
Unidad organizativa: Pacients
Nombre : Fran Cancer Giner
Correo : fcancerginer@gmail.com
contraseña: pfc2007
nombre de la compañia opcional: UOC

Metge:
Openssl req -new -sha1 -key metge-priv.key -passin pass:pfc2007 -out petic-cert-metge.csr

Pais: ES
Provincia: BARCELONA
Ciudad: BARCELONA
Organización: UOC
Unidad organizativa: Metges
Nombre : Fran Cancer Giner
Correo : fcancerginer@gmail.com
contraseña: pfc2007
nombre de la compañia opcional: UOC

Administrador:
Openssl req -new -sha1 -key administrador-priv.key -passin pass:pfc2007 -out petic-cert-administrador.csr

Pais: ES
Provincia: BARCELONA
Ciudad: BARCELONA
Organización: UOC
Unidad organizativa: Administradors
Nombre : Fran Cancer Giner
Correo : fcancerginer@gmail.com
contraseña: pfc2007
nombre de la compañia opcional: UOC

- Emitimos el certificado

Antes de emitir el certificado hay que tocar 2 líneas del fichero de configuración openssl.cnf
basicConstraints = critical,CA:FALSE
extendedKeyUsage = clientAuth
Este fichero completo se puede consultar en el anexo de la memoria.

Pacient:

Openssl x509 -req -in petic-cert-pacient.csr -days 180 -CA CA.crt -CAkey CA.key -extensions usr_cert -out pacient-cert.crt -CAcreateserial

Metge:

Openssl x509 -req -in petic-cert-metge.csr -days 180 -CA CA.crt -CAkey CA.key -extensions usr_cert -out metge-cert.crt

Administrador:

Openssl x509 -req -in petic-cert-administrador.csr -days 180 -CA CA.crt -CAkey CA.key -extensions usr_cert -out administrador-cert.crt

- Generamos el fichero comprimido con formato pkcs12

Usaré como passphrase: pfc2007

Pacient:
openssl pkcs12 -export -in pacient-cert.crt -inkey pacient-priv.key -name cert-pacient-pck12 -chain -CAfile CA.crt -out cert-pacient-pck12.p12

Metge:
openssl pkcs12 -export -in metge-cert.crt -inkey metge-priv.key -name cert-metge-pck12 -chain -CAfile CA.crt -out cert-metge-pck12.p12

Administrador:
openssl pkcs12 -export -in administrador-cert.crt -inkey administrador-priv.key -name cert-administrador-pck12 -chain -CAfile CA.crt -out cert-administrador-pck12.p12

Una vez la parte de la PKI está completada paso a describir el esquema criptográfico implementado en el proyecto.

3. Esquema criptográfico

3.1 Introducción

La historia clínica de un paciente es un conjunto de datos que nos explican todos los pormenores de la salud de un paciente como por ejemplo, las vacunas que tiene puestas, si tiene alguna alergia conocida, las visitas que se le han hecho al paciente, las pruebas médicas que se le han realizado, etc.
Vamos a dividir el mantenimiento de una historia clínica en varios procesos. Diseñaré un protocolo criptográfico para cada una de los procesos que me interese desarrollar. El conjunto de estos protocolos recibe el nombre de esquema criptográfico.

En este capítulo se describen algunos procesos del mantenimiento de una historia clínica y cuales son sus protocolos criptográficos asociados. Se hace énfasis en el diseño, el funcionamiento y las decisiones de diseño que se han tomadas durante la implementación.

Para una buena comprensión de este capítulo se recomienda disponer de unos conocimientos previos de criptografía, o utilizar [1] como referencia.

3.2 Mantenimiento de una historia clínica.

He utilizado el término “mantenimiento de una historia clínica” para agrupar los distintos procesos que permiten tener unos datos médicos almacenados y disponibles para ser consultados en cualquier momento.
En este proyecto nos centraremos en 3 procesos que servirán de ejemplo y que cumplirán los requisitos de seguridad que nos hemos propuesto en este proyecto. Abordar todos los posibles procesos que se derivan de una gestión completa de una historia clínica sería demasiado complejo y queda como propuesta para futuras versiones.

- Consulta de un historial médico: Puede ser utilizado por el médico o por el paciente. El paciente solo puede consultar sus datos. El médico puede consultar solo sus historias.

- Consulta de los pacientes asignados a un médico: Este proceso permite consultar la lista de todos los pacientes que tiene cierto médico asignado.

- Inserción de datos en el historial médico: En este proceso se solicita una nueva visita al paciente.

Cada uno de estos procesos corresponde a una parte del protocolo del esquema criptográfico que se ha comentado anteriormente.

3.3 Notación utilizada en el protocolo.

En la descripción de los protocolos es necesario la utilización de una cierta notación criptográfica, que se muestra a continuación.

- K: Clave de un sistema criptográfico.

- EK(M): cifraje simétrico de un mensaje ***M*** con la clave ***K***.

- DK(M): Descifraje simétrico de un mensaje ***M*** con la clave ***K***.

- (PEntidad, SEntidad): Pareja de claves asimétricas propiedad de Entidad, donde ***P*** corresponde a la clave pública y ***S*** a la clave privada.

- SEntidad (M): Firma de un mensaje ***M*** con la clave privada ***S*** de Entidad.

- PEntidad(M): Cifraje del mensaje ***M*** con la clave asimétrica pública ***P*** de Entidad.
- H(M): salida de una función resumen criptográfica del mensaje ***M***, estas funciones reciben el nombre de funciones de hash.

3.4 Actores del sistema

El sistema contará con tres actores, dos son personas físicas y uno es el sistema.

- Paciente: Usuario del sistema que puede consultar sus datos.
- Médico: Usuario del sistema que se introduce y gestiona los datos de sus pacientes.
- Gestor: Usuario del sistema que verifica que los procesos se desarrollen correctamente. Puede consultar información de los usuarios en la base de datos.

3.5 Consulta de un historial

Esta es la parte del protocolo correspondiente a la consulta de un historial médico. Los actores que intervienen en este proceso son el actor que hace la consulta y el gestor de historiales.

Protocolo 1

1. U realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 1 con la clave pública PU, y obtener PG[Ni, Id_usuarioU];
   (b) Enviar PG[Ni, Id_usuariooU] a G;

2. G realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 2 con PG[Ni, Id_usuarioU], y obtener PU[Ni,NG, Id_usuarioG];
   (b) Enviar PU[Ni,NG, Id_usuarioG] a U;

3. U realiza les operaciones siguientes:
   (a) Descifrar PU[Ni,NG, Id_usuarioG] con la clave privada SU, y obtener NG, N y ‘ y Id_usuarioG;
   (b) Si Ni’ = Ni hacer:
      i. Cifrar NG, Consulta, Id_usuario con la clave pública PG de G, PG[NG,Consulta, Id_usuario]. Consulta indica que se quiere consultar la historia del usuario identificado con Id_usuario;
      ii. Enviar PG[NG,Consulta, Id_usuario] a G;
   (c) Sino Devolver error;

4. G realiza las operaciones siguientes:
   (a) Descifrar PG[NG, Consulta, Id_usuario] con la clave privada SG, y obtener NG’, Consulta, Id_usuario;
   (b) Recuperar NG de la BD. En el paso 4 del Procedure 4 NG y Ni se han guardado en la BD;
   (c) Si NG’ ?=NG hacer:
      i. Si (Id_usuarioU = Id_usuario) o (Id_usuarioU es el médico y Id_usuario es un paciente de Id_usuarioU) hacer:
         A. Ejecutar el Procedure 3 con el Id_usuario y PU, y obtener PU[H];
         B. Enviar PU[H] a U.
      ii. Sino Devolver error;
   (d) Sino devolver error;
   (e) Borrar NG y Ni de la BD;

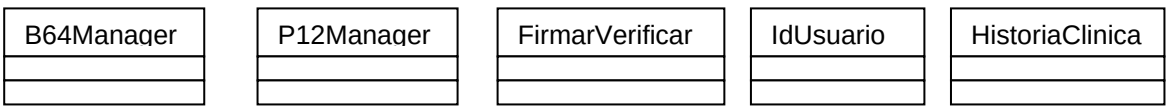
5. U realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 4 con Pu[H] y obtener H
   (b) Mostrar H.

Clases necesarias para la obtención de la historia clínica de un paciente.

Para realizar estos pasos se ha propuesto tener una clase para firmar las historias y para comprobar las firmas, este clase se llamará FirmarVerificar. Esta clase tiene otra clase que se encarga de gestionar los archivos PKCS12[20] con el fin de obtener las claves privadas y públicas. Se recuerda que la obtención de estos archivos se ha explicado en la sección 2. PKI.

Se tiene que tener en cuenta otro detalle. Los datos que se generan después de realizar la firma de un texto no se pueden transportar ni tratar como si fuera una cadena de caracteres por problemas de codificación. Lo que se ha de hacer si se quiere almacenar como cadena es pasar los datos a base64. Como un objetivo es guardar los datos en forma de cadena en la base de datos, hace falta una clase que se encargue de de llevar a cabo esta conversión a base64 y a binario de nuevo. Esta clase se llamará B64Manager.

Se muestra a continuación una definición simple en UML[14] de las clases nombradas hasta el momento:



Durante la evolución del proyecto se podrá ver como estas clases forman parte del esqueleto del sistema, proveyendo las nuevas necesidades de otras clases como en el caso de la base de datos o la comunicación por RMI[5], por ejemplo.

Como se puede comprobar en la definición de los casos de uso que se muestran a continuación, se hace referencia a guardar los datos en una base de datos. Dado que en este punto del desarrollo el sistema aun no dispone de este servicio, se entenderá que estos datos se almacenan de una manera lógica en la clase HistoriaClinica (esta clase desaparecerá).

Si el lector no está familiarizado con los diagramas UML[14] se recomienda ver[2] como referencia.

Diagrama de casos de uso de este protocolo:

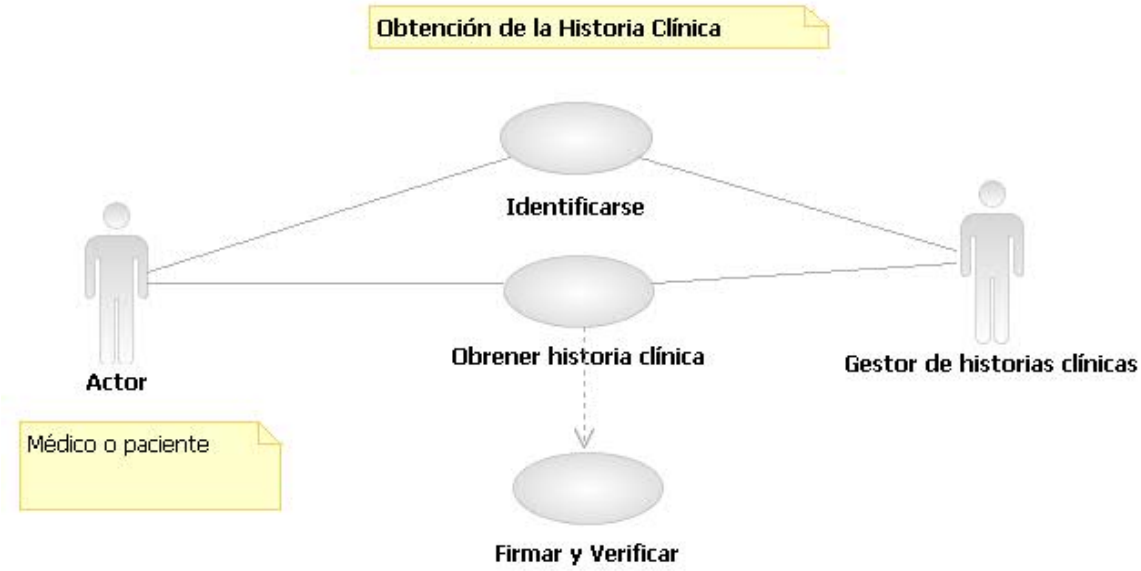


Imagen **5***: Diagrama de casos de uso protocolo 1*

Descripción de los casos de uso de esta opción:

Obtener la historia clínica

Caso de uso: Obtener la historia clínica.
Actores: Médico o paciente y Gestor.
Propósito: Pedir al gestor que le devuelva por pantalla los datos de la historia clínica de un identificador id_usuario
Tipo: Primario y esencial.

Curso típico de los acontecimientos:

Acciones de los actores	**Acciones del sistema**
1. El caso de uso empieza cuando el actor quiere obtener la historia clínica de un paciente definido por un identificador.	
2. El actor envía el identificador al servidor indicando que quiere consultarlo.	
	3. El servidor recibe los datos, los procesa y accede a la base de datos buscando el id del paciente.
	4. Envía la historia clínica al actor.
5. El actor verifica los datos recibidos.	
6. El actor ve la historia clínica por pantalla	

Cursos alternativos:

Punto 3: El servidor recibe el identificador de un paciente que no tiene historia clínica asociada en la base de datos. En ese caso se devuelve un error.

Punto 5: La verificación de la firma es incorrecta, se devuelve un error.

Identificarse

Caso de uso: Identificarse
Actores: Médico o Paciente y gestor
Propósito: Reconocer al actor como usuario válido del sistema y saber su nivel de confidencialidad. Identificar al gestor.
Tipo: Secundario y esencial.

Curso típico de los acontecimientos:

Acciones de los actores	**Acciones del sistema**
1. El caso de uso empieza cuando el actor realiza una acción en el sistema y se le pide que se identifique.	
2. El actor prepara los datos que enviará al servidor y los envía.	
	3. El servidor recibe los datos, los procesa y autentifica al actor.
	4. Envía confirmación y datos auxiliares al cliente.
5. El cliente recibe la confirmación y los datos auxiliare con los que identifica al gestor.	

Cursos alternativos:

Punto 3: El servidor recibe datos que son incorrectos y no autentifica el cliente. En ese caso se devuelve un error.

Firmar y verificar

Caso de uso: Firmar y verificar
Actores: Médico
Propósito: Firmar la historia clínica y verificar los datos firmados.
Tipo: Primario y esencial.

Curso típico de los acontecimientos:

Acciones de los actores	Acciones del sistema
1. El actor pide firmar o verificar la firma.	2. El sistema devuelve los datos firmados o verificados.

Cursos alternativos:

Punto 2: El sistema retorna un error al verificar una firma.

Procedures necesarias.

El Procedure 1 contiene una parte de la autenticación del protocolo de Needham-Schroeder. Estos pasos los utilizaremos en otros protocolos. Estos procedures pueden ser utilizados por los médicos , los pacientes o por el gestor.

Procedure 1 (PU)
1. Obtener un valor de forma aleatoria, Ni;
2. Cifrar Ni y Id_usuarioU con la clave pública de G, PG[Ni, Id_usuarioU];
3. Enviar PG[Ni, Id_usuarioU] a G.

El Procedure 2 contiene otra parte de l'autenticación del protocolo de Needham-Schroeder. Esta parte será ejecutada por el Gestor.

Procedure 2 (PG(Ni, Id_usuarioU))
1. Descifrar PG[Ni, Id_usuarioU] con SG, y obtener; Ni y Id_usuariU;
2. Obtener el certificado de U a partir de Id_usuarioU. Suponemos que el sistema dispone de una Base de Datos (BD) donde por cada Id_usuario encontramos su certificado correspondiente. A partir del certificado se puede obtener la clave pública PU;
3. Obtener un valor de forma aleatoria, NG;
4. Guardar a la BD los valores Ni y NG asociados con U;
5. Cifrar Ni,NG, Id_usuarioG, con la clave pública PU de U, PU[Ni,NG, Id_usuarioG];
6. DevolverPU[Ni,NG, Id_usuarioG].

El gestor G utiliza el Procedure 3 para encontrar el historial que se le ha pedido y cifrarlo con la clave del usuario que lo quiere consultar.

Procedure 3 (Id_usuario,PU)
1. Buscar el historial H correspondiente a Id_usuario;
2. Descifrar la parte de H que está cifrada utilizando la clave privada SG de G;
3. Cifrar H con la clave pública PU, PU[H];
4. Devolver PU[H].

Un usuario utiliza el Procedure 4 para descifrar un historial enviado por el gestor G y verificar que el historial es correcto.

Procedure 4

1. Descifrar PU[H] con la clave privada SU de U, SU[PU[H]];
2. Para cada entrada del historial H que está firmada hacer:
    (a) Verificar la firma digital de M;
    (b) Verificar la firma digital de G;
    (c) Verificar la secuencia;
3. Devolver H.

3.6 Consulta de los pacientes asignados a un médico

Esta es la parte del protocolo correspondiente a la búsqueda que hace un médico de la lista de sus pacientes. Los actores que intervienen en este proceso son el médico y el gestor de historiales.

Protocolo 2

1. U Realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 1 con la clave pública PU, y obtener PG[Ni, Id_usuarioU];
   (b) Enviar PG[Ni, Id_usuarioU] a G;

2. G Realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 2 con PG[Ni, Id_usuarioU], y obtener PU[Ni,NG, Id_usuarioG];
   (b) Enviar PU[Ni,NG, Id_usuarioG] a U;

3. U Realiza las operaciones siguientes:
   (a) Descifrar PU[Ni,NG, Id_usuarioG] con la clave privada SU, y obtener NG, Ni y Id_usuarioG;
   (b) Si Ni' = Ni hacer:
      i. Cifrar NG y Lista de pacientes con la clave pública PG de G, PG[NG, Lista pacientes]. Lista pacientes indica que se quiere un listado de los identificadores de usuario correspondiente a los pacientes del médico identificado como Id_usuarioU
      ii. Enviar PG[NG, Lista pacientes] a G;
   (c) Sino devolver error;

4. G Realiza las operaciones siguientes:
   (a) Descifrar PG[NG, Lista pacientes] con la clave privada SG, y obtener NG' y Lista pacientes;
   (b) Recuperar NG de la BD. En el paso 4 del Procedure 4 NG y Ni han sido guardados en la BD;
   (c) Si NG' = NG hacer:
      i. Si Id_usuarioU es medico hacer:
         A. Ejecutar el Procedure 5 con Id_usuarioU y PU, y obtener PU[{Id_usuario1, . . . , Id_usuarion}, SG[{Id_usuario1, . . . , Id_usuarion}]];
         B. Enviar a U PU[{Id_usuario1, . . . , Id_usuarion}, SG[{Id_usuario1, . . . , Id_usuarion}]];
      ii. Sino devolver error;
   (d) Sino devolver error;
   (e) Borrar NG y Ni de la BD;

5. U Realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 6 con PU[{Id_usuario1, . . . , Id_usuarion}, SG[{Id_usuario1, . . . , Id_usuarion}]], y obtener {Id_usuario1, . . . , Id_usuarion};
   (b) Mostrar {Id_usuario1, . . . , Id_usuarion}. Amb el Procedure 5 el gestor G obtiene el listado de los pacientes assignados al medico Id_usuario.

Procedures necesarias.

Procedure 5 (Id_usuario, PU)
1. Buscar en la BD els pacientes asignados al medico Id_usuario, obteniendo {Id_usuario1, . . . , Id_usuarion};

2. Firmar {Id_usuario1, . . . , Id_usuarion} con la clave privada SG de G, SG[{Id_usuario1, . . . , Id_usuarion}];
3. Cifrar {Id_usuario1, . . . , Id_usuarion} y SG[{Id_usuario1, . . . , Id_usuarion}] con la clave públicade Id_usuario, PU, PU[{Id_usuario1, . . . , Id_usuarion}, SG[{Id_usuario1, . . . , Id_usuarion}]];

4. Devolver PU[{Id_usuario1, . . . , Id_usuarion}, SG[{Id_usuario1, . . . , Id_usuarion}]].
El Medico utiliza el Procedure 6 para obtener la Lista de sus pacientes y verificar que ha sido generada por el Gestor G.

Procedure 6 (PU({Id_usuario1, . . . , Id_usuarion}, SG({Id_usuario1, . . . , Id_usuarion})))
1. Descifrar PU[SG[{Id_usuario1, . . . , Id_usuarion}]] con la clave privada SU de U, SU[PU[SG[{Id_usuario1, . . . , Id_usuarion}]]] y obtener SG[{Id_usuario1, . . . , Id_usuarion}] ;

2. Verificar la firma digital SG[{Id_usuario1, . . . , Id_usuarion}] con la clave pública PG de G;
3. Si la verificación anterior es correcta devolver {Id_usuario1, . . . , Id_usuarion}.

Clases necesarias para la obtención de la historia clínica de un paciente.

No es necesaria modificar la actual estructura.

Diagrama de casos de uso de este protocolo:

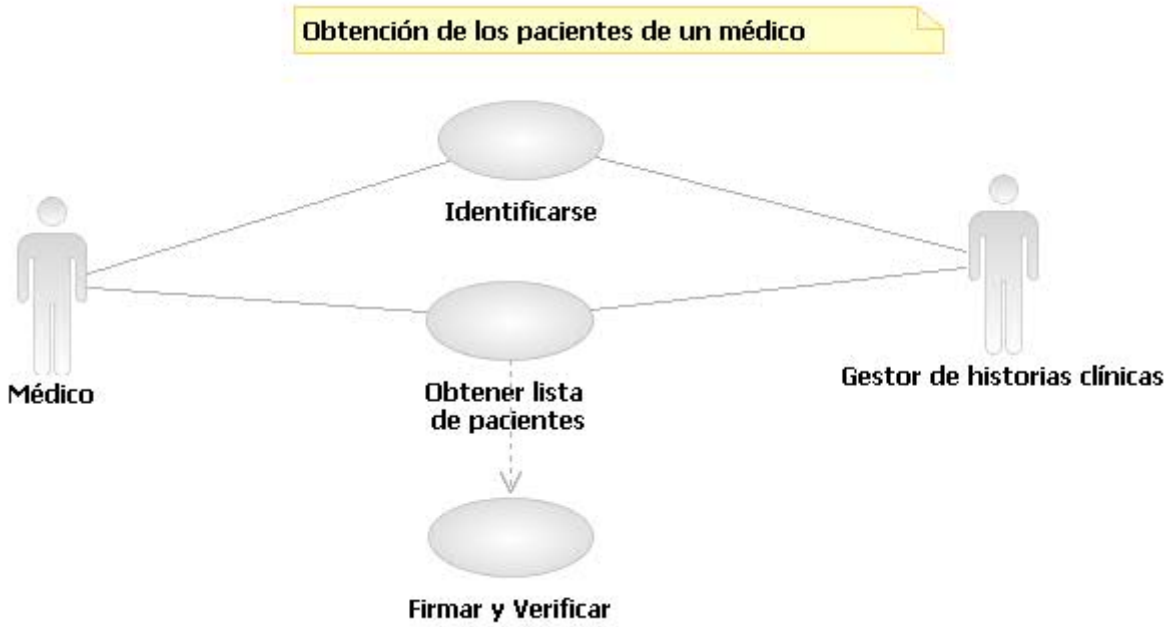


Imagen **6***: Diagrama de casos de uso protocolo 2*

Descripción de los casos de uso de esta opción:

Obtención del identificador del paciente

Caso de uso: Obtener lista de pacientes.
Actores: Médico.
Propósito: Pedir al gestor que le devuelva por pantalla los identificadores de sus pacientes.
Tipo: Primario y esencial.

Curso típico de los acontecimientos:

Acciones de los actores	**Acciones del sistema**
1. El caso de uso empieza cuando el médico quiere obtener los identificadores de sus pacientes. 2. El médico envía su identificador al servidor	
	3. El servidor recibe los datos, los procesa y accede a la base de datos buscando la información solicitada. 4. Envía la lista de id _paciente al médico.
5. El médico verifica los datos recibidos. 6.El médico ve la lista de id_paciente por pantalla	

Cursos alternativos:

Punto 3: No se encuentran datos en la base de datos. En ese caso se devuelve un error.

3.7 Inserción de visitas en la historia médica.

En este protocolo se supone previamente a la inserción de los datos que el médico ha consultado la historia clínica del paciente y por lo tanto conoce el Id del paciente. Este protocolo está pensado únicamente para añadir una nueva visita a la historia clínica.
La aplicación del médico realiza los siguientes pasos a la hora de introducir los datos de un paciente en el sistema gestor de pacientes.

Protocol 3

1. M Realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 1 con la clave pública PM, y obtener PG[Ni, Id_usuarioM];
   (b) Enviar PG[Ni, Id_usuarioM] a G;

2. G Realiza las operaciones siguientes:
   (a) Ejecutar el Procedure 2 con PG[Ni, Id_usuarioM], y obtener PU[Ni,NG, Id_usuarioG];
   (b) Enviar PM[Ni,NG, Id_usuarioG] a M;

3. M Realiza las operaciones siguientes:
   (a) Descifrar PM[Ni,NG, Id_usuarioG] con la clave privada SM, y obtener NG, N0 y i Id_usuarioG;
   (b) Si N0 y ?= Ni hacer:
      i. Obtener los datos de la visita V. La visita debería de incluir como mínimo Id_usuarioP;
      ii. Firmar V con la clave privada SM de M, SM[V];
      iii. Cifrar NG, V y SM[V] con la clave pública PG de G, PG[NG, Insertar visita, V, SM[V]]. Insertar visita indica que se quiere añadir V al historial del paciente P;
      iv. Enviar PG[NG, Inserir visita, V, SM[V]] a G;
   (c) Sino devolver error.

4. G Realiza las operaciones siguientes:
   (a) Descifrar PG[NG, Inserir visita, V, SM[V]] con la clave privada SG, y obtener N0G, Inserir visita, V y SM[V];
   (b) Recuperar NG de la BD. En el paso 4 del Procedure 4 NG y han sido guardados a la BD.
   (c) Si NG' = NG hacer:
      i. Obtener Id_usuarioP a partir de V;
      ii. Verificar que Id_usuarioM es médico3;
      iii. Verificar que Id_usuarioP es un paciente asignado a Id_usuarioM 4;
      iv. Si les verificaciones anteriores son correctas hacer:
         A. Verificar la firma digital SM[V] con la clave pública PM;
         B. Obtener el instante de tiempo actual T;
         C. Obtener el número de série X de la última visita del historial H del paciente Id_usuarioP;
         D. Incrementar en una unidad X, X + 1;
         E. Firmar V, SM[V], T, X + 1, con la clave privada SG de
         G, SG[V, SM[V], T, X + 1];
         F. Cifrar V y SM[V] con la clave pública SG de G, PG[V, SM[V]];
         G. Firmar Id_usuarioP y X+1 con la clave privada SG de G, SG[X + 1, Id_usuarioP];
         H. Guardar a la BD PG[V, SM[V]], X + 1, T, SG[V, SM[V], T,X + 1] y SG[X + 1, Id_usuarioP] ;
      v. Sino devolver error.
   (d) Sino devolver error.

Clases necesarias para la inserción de visitas en la historia clínica de un paciente.

Es necesaria una clase para guardar las visitas en la base de datos, pero eso se detalla en el capítulo correspondiente en el momento de la implantación de la base de datos. En este punto todavía no existe el acceso a la base de datos, por lo tanto se guardan los datos en una clase vacía llamada Visita.

En este punto del protocolo se añade una nueva funcionalidad que hasta ahora no había sido necesaria. El gestor de historias clínicas cifra la visita y la guarda en formato PKCS7[21], es por eso que se añade

una nueva clase que se llama Cifrador. El formato PKCS7[21] es un contenedor de datos criptográficos, como puede ser una firma digital, una CRL, etc.

Se muestra a continuación una definición simple en UML[14] de las clases nombradas hasta el momento:

Visita

Cifrador

Para hacer el juego de pruebas se ha implementado una clase vacía para guardar las visitas “visita”. Esta clase se irá modificando hasta el momento en que se implemente la parte correspondiente a XML[10].

Diagrama de casos de uso de este protocolo:

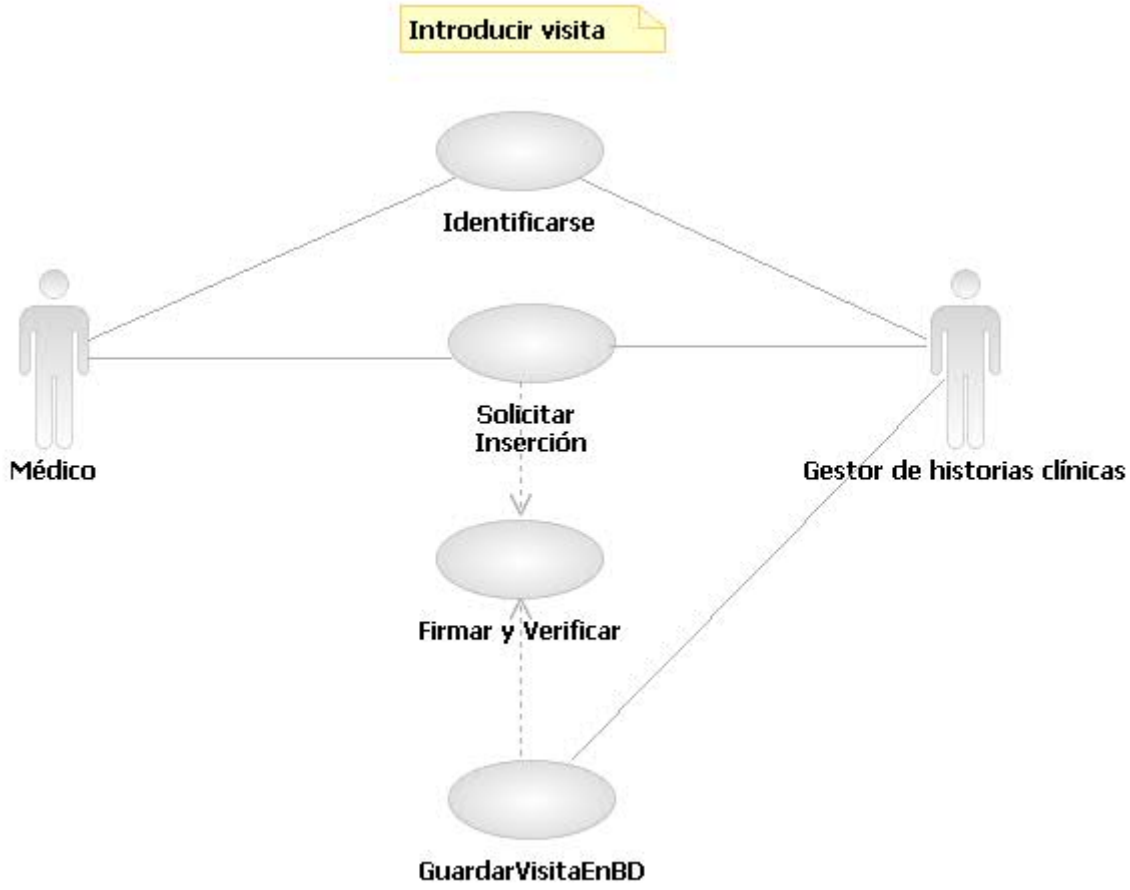


Imagen **7***: Diagrama de casos de uso protocolo 3*

Descripción de los casos de uso de esta opción:

Solicitar inserción

Caso de uso: solicitar Inserción
Actores: Médico.
Propósito: solicitar al gestor la inserción de una nueva visita para el paciente.
Tipo: Primario y esencial.

Curso típico de los acontecimientos:

Acciones de los actores	**Acciones del sistema**
1. El caso de uso empieza cuando el médico	

quiere introducir una visita de un paciente del cual ya conoce su id_paciente. 2. El médico envía la petición al servidor	
	3. El servidor recibe los datos y los procesa.

Guardar Visita en la Base de datos.

Caso de uso: Guardar Visita en la base de datos.
Actores: Gestor de historias clínicas.
Propósito: El gestor guarda la información cifrada en la base de datos.
Tipo: Primario y esencial.

Curso típico de los acontecimientos:

Acciones de los actores	**Acciones del sistema**
1. El caso de uso empieza cuando recibe una petición de un médico para introducir una visita en la base de datos.	
	2. El sistema guarda los datos encriptados en la base de datos.

3.8 Diagrama de clases del esquema criptográfico.

Como se ha visto al final de cada punto del protocolo se iban comentando las clases que se utilizaban. A continuación se muestra el diagrama de clases completo del esquema criptográfico. Se incluye a demás una clase para cada uno de los actores.

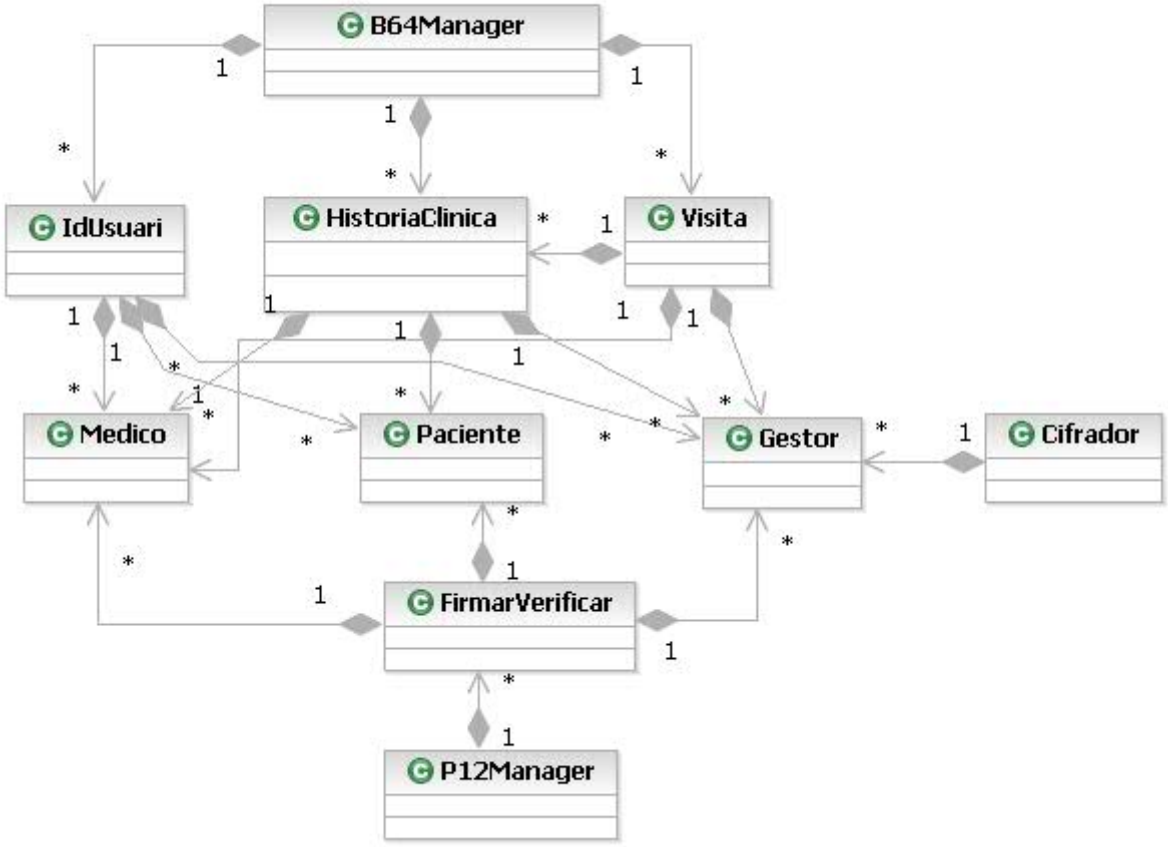


Imagen **8***: Diagrama de clases preliminar del esquema criptográfico*

Comentarios de este diagrama:

- Cada clase de usuario utilizará solo la clase “firmar y verificar” que se utilizará durante toda la ejecución del aplicativo.
- La única clase que tiene acceso al Cifrador es el gestor.

3.9 Pruebas realizadas.

Con el fin de tener un juego de pruebas sobre el protocolo, se ha utilizado una clase de pruebas que contiene un ejemplo de los procesos que se han detallado. El código de esta clase no incluye como era de esperar en este momento del diseño incremental del proyecto, ni acceso a métodos remotos utilizando RMI[5], ni acceso a base de datos, ni tampoco la implementación del método para autenticar los diferentes clientes contra el gestor de historias clínicas. Este código se puede ver en un anexo de esta memoria.

4. Representación de los datos: XML.

4.1 Introducción.

XML[10] es el acrónimo de eXtensible Markup Language. Des de que apareció esta forma de representar los datos se ha impuesto como una de las formas más eficientes para intercambiar y almacenar datos entre aplicaciones y protocolos.

En el proyecto se ha utilizado XML[10] para hacer las transferencias de los datos en las consultas de las historias clínicas. De la misma manera también se utiliza XML[10] para hacer la transferencia de datos durante el protocolo de autentificación entre el cliente y el servidor.

La idea general es que se creará un documento XML[10] con la estructura necesaria vacía que sirva de soporte para realizar todas las consultas. Dispondremos de una clase que nos permitirá acceder a todos los parámetros y métodos necesarios. El documento XML[10] dispondrá de un apartado para cada una de las diferentes transacciones que se realizan entre cliente y servidor. Una vez rellenados los parámetros de cada transacción la sección utilizada se encriptará y se pasará a base 64.

Para realizar un trabajo de conversión a XML[10] y viceversa se ha utilizado la librería pública JDOM[8]. Si en un futuro se decidiera a integrar esta aplicación a otra, el hecho de tener los datos en XML[10] facilitaría el proceso.

4.2 Estructura del documento XML

La estructura del documento XML[10] utilizada por la aplicación para transferir información entre el cliente y el servidor es la siguiente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <Peticion xmlns:cp="http://www.uoc.edu/tfc">
        - <Orden id="ID1723292983">
                <Date>Mon Oct 29 15:21:29 CET 2007</Date>
                 <Protocolo>1</Protocolo>
                - <DetalleOrden>
                        - <OrdenIdentificacion>
                                <AleatorioNi />
                                <IdUsuarioU />
                          </OrdenIdentificacion>
                        - <OrdenValidacion>
                                <AleatorioNip />
                                <AleatorioNg />
                                <IdUsuarioG />
                          </OrdenValidacion>
                        - <OrdenConsulta>
                                <AleatorioNgp />
                                <Consulta />
                                <IdUsuario />
                          </OrdenConsulta>
                        - <OrdenResultado>
                                <Resultado />
                          </OrdenResultado>
                  </DetalleOrden>
        </Orden>
 </Peticion>
```

La explicación de cada uno de los campos del XML[10] y de su contenido es como sigue:

- <Peticion> es la raiz del documento, aquí se guarda toda la información de la petición.

- <Orden> Estructura que guarda la información referente a la orden en sí. Tiene 3 atributos

    - <id> Un identificador de la orden
    - <Date> Fecha y hora de la creación de la orden
    - <Protocolo> Indica bajo que protocolo se hace la petición.

- <DetalleOrden> Estructura que guarda los distintos momentos de la orden.

- <OrdenIdentificacion> Contiene los atributos necesarios para solicitar una identificación al gestor. El origen puede ser el médico o el paciente.
    - <AleatorioNi> Número aleatorio del usuario que hace la petición.
    - <IdUsuarioU> Identificador del usuario que hace la petición.

- <OrdenValidacion> Contiene los atributos necesarios para que el usuario verifique al gestor y pueda hacer la petición.
    - <AleatorioNip /> Número aleatorio del usuario que hace la petición
    - <AleatorioNg /> Número aleatorio del gestor
    - <IdUsuarioG /> Id de usuario del gestor.

- <OrdenConsulta> contiene los atributos necesarios para que el gestor verifique al usuario y sepa cual es la consulta que quiere hacer.
    - <AleatorioNgp /> Número aleatorio del gestor
    - <Consulta /> Indica que se solicita una consulta.
    - <IdUsuario /> Id del usuario del cual se solicita la consulta.

- <OrdenResultado> Contiene el resultado de la consulta
    - <Resultado /> Contiene el resultado de la consulta.

Variaciones para el protocolo 2

```
<OrdenResultado>
        <Resultado/>
        <P7/>
</OrdenResultado>
```

- <OrdenResultado> Contiene el resultado de la consulta
    - <Resultado /> Contiene el resultado de la consulta.
    - <P7/> Contiene un sobre con la firma del valor que hay en resultado.

Variaciones para el protocolo 3

```
<OrdenConsultaTemporal>
        <IdUsuarioV/>
        <NombreV/>
</OrdenConsultaTemporal>

<OrdenConsulta>
        <AleatorioNgp/>
        <Consulta/>
        <DatosVisita/>
        <P7DatosVisita/>
</OrdenConsulta>
```

- <OrdenTemporal> contiene los atributos que se han de guardar en disco
    - <idUsuarioV /> Id del usuario al cual se le añade una visita
    - <NombreV /> Nombre del usuario al que se le añade una visita.

- <OrdenConsulta> contiene los atributos necesarios para que el gestor verifique al usuario y sepa cual es la consulta que quiere hacer.
    - <AleatorioNgp /> Número aleatorio del gestor
    - <Consulta /> Indica que se solicita una consulta.
    - <DatosVisita /> son los datos que aparecen en el elemento “Orden Temporal”
    - <P7DatosVisita /> un sobre P7 con los datos de la visita.

4.3 Funcionamiento de la representación de datos mediante XML.

En el diseño se ha considerado el hecho que posteriormente se utilizará una base de datos con tal de guardar los datos que llevan estos documentos de manera que el gestor de historias clínicas ha de poder acceder de manera sencilla y rápida.
Cuando las aplicaciones piden información, esta se busca en las bases de datos y se envía utilizando las estructuras de los documentos XML[10]. Tendremos un único documento XML[10] para cada proceso que se irá enviando y completando a cada fase del protocolo. La parte de la aplicación que lo recibe se encargar de analizarlo de extraer o incorporar la nueva información y volver a remitirlo.

Utilizando este protocolo de transmisión se asegura una transferencia mínima de datos en las comunicaciones de las dos partes, evitando la carga de las líneas de comunicación.

Decisiones tomadas para la implementación:

- La clase XMLManager es la encargada de pasar los datos a base64 y al revés. Este proceso se realiza de una manera transparente para el resto de clases. Solo se han de pasar los datos en binario y esta clase se encargará de guardarlos en base64 y de revolverlos en binario.

- A menudo es necesario que se devuelvan junto con los datos las cabeceras XML[10], con el fin de poder identificar correctamente la información. Esta tarea también la realiza la case XMLManager.

- Esta clase es la encargada de llevar a cabo el paso de una cadena tipo String a documento XML[10].

- A partir de este punto la clase HistoríaClinica deja de tener sentido. No pasa lo mismo con la clase IdUsuario que aun se utiliza para disponer de métodos útiles con el fin de concatenar los datos de un identificador y devolver estructuras fácilmente tratables utilizables con XML[10].

- Dispondré de una clase auxiliar llamada XMLtemporal en la cual despliego las estructuras xml desencriptadas (en claro) y que me permite coger los valores de sus atributos. Esta clase una vez utilizada se borra y siempre creará un documento XML[10] de forma lógica.

4.4 Diagrama de clases de la representación de datos mediante XML.

En el diagrama de clases del capítulo anterior hemos sustituido la case Historia clínica por la clase XMLManager:

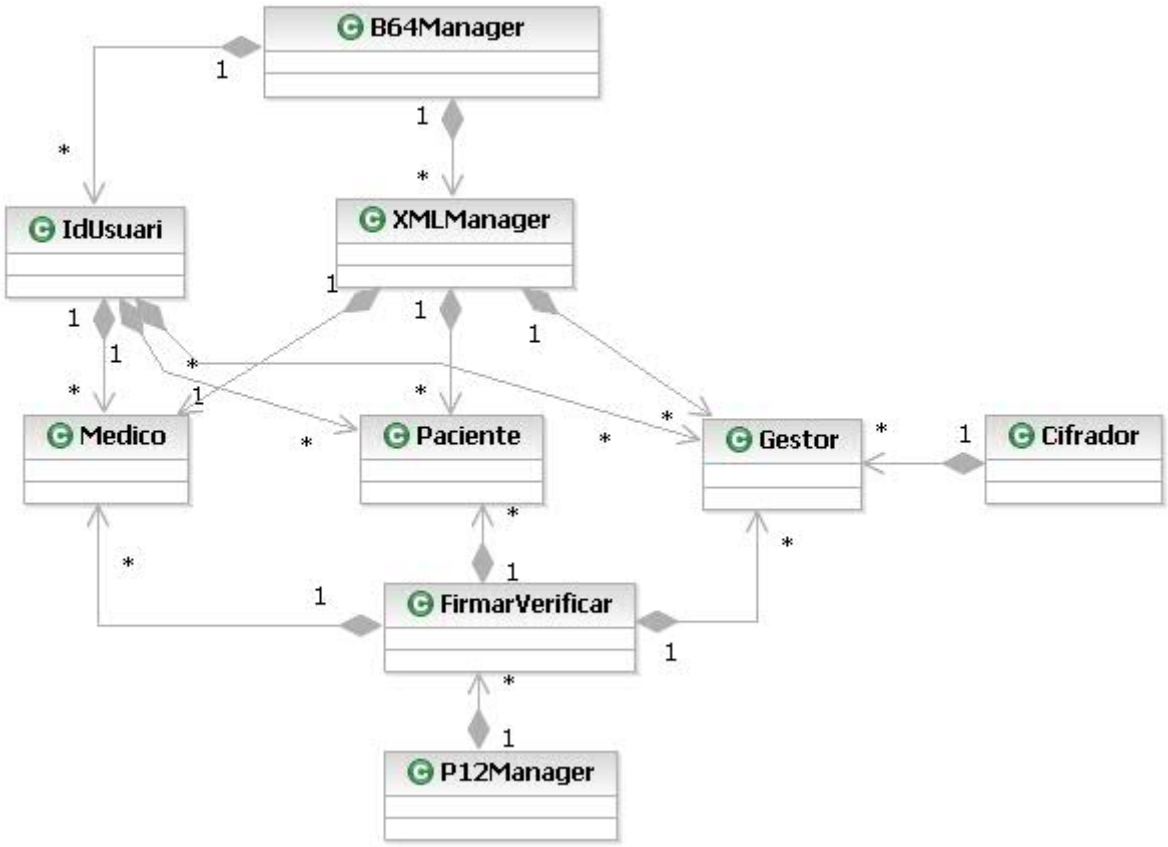


Imagen **9***: Diagrama de clases de la representación de datos XML*

Esta clase es especialmente extensa porque contiene los atributos para cada una de los elementos que componen el documento XML[10] y un método asociado a cada atributo que retorna el contenido XML[10]. En el contenido XML[10] también está la cabecera, y los datos están codificados en base64 para ser tratados posteriormente en la base de datos.

Cuando se quieren pasar datos firmados, primero se sustituyen los atributos vacíos por los atributos correctos y luego se serializa la estructura a partir del Parent. Este String resultante de la serialización se codifica o se firma según el caso y luego se pasa a bas64. Después se sustituye el Parent entero por su equivalente manipulado en base64.

En temas del diseño se puede añadir que esta clase será utilizada por los tres actores. Más adelante se verá que las clases que implementan los protocolos de cada uno de estos actores, consiguiendo así la abstracción de las vistas, serán los únicos que acceden a esta clase.

4.5 Pruebas realizadas.

Igual que en el capítulo anterior, con el fin de probar que los datos se convierten correctamente al formato XML[10] y al revés, se ha modificado el archivo de test. El código modificado y que ya utiliza la nueva clase se puede ver en el anexo G: Código del juego de pruebas de la representación de los datos en XML de esta memoria.

5. Comunicación de los componentes.

5.1 Introducción.

La comunicación de los diferentes componentes es una parte esencial del proyecto. El médico ha de poder enviar y consultar los datos del paciente de forma remota. Los pacientes han de poder consultar sus datos de forma remota. La comunicación de los diferentes componentes del sistema tradicionalmente supondría el diseño de un protocolo o mecanismo de comunicación. Para evitar la sobrecarga de trabajo, y dado que la parte esencial es el esquema criptográfico he optado por la utilización de la tecnología RMI[5].

RMI[5] son las siglas de Remote Method Invocation. Java[4] incorpora esta tecnología a el API estándar. RMI[5] consta de un servidor donde se ejecutan diferentes instancias de las clases servidoras que se necesitan.

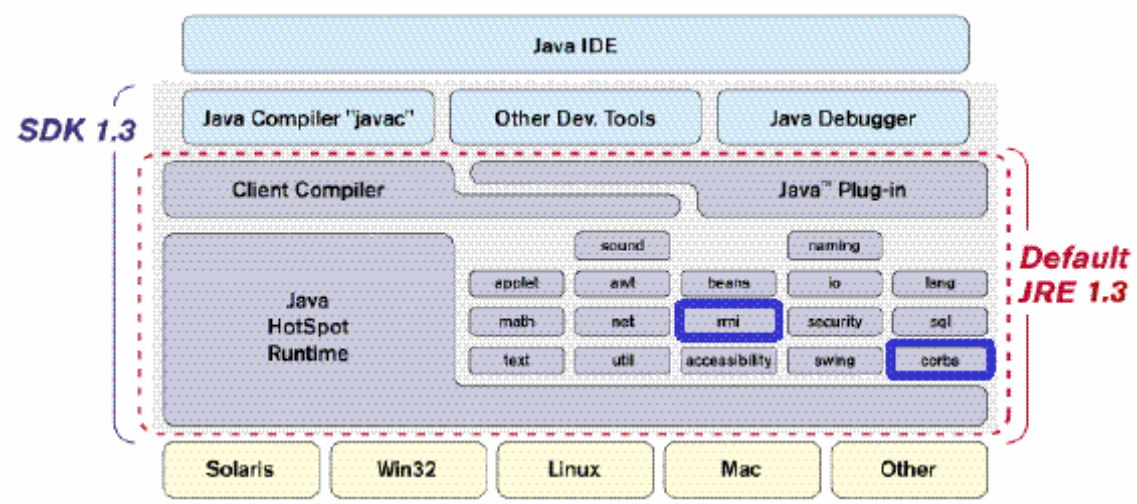


Imagen **10***: La arquitectura de "Java 2 SDK, Standard Edition v.1.3" .*

Las aplicaciones que quieren utilizar los métodos remotos únicamente necesitan saber la interfaz del servidor, es decir, los métodos que ofrece la clase que está en el servidor. La implementación de esta interfaz está oculta y el cliente no llega nunca a saber que es lo que está ejecutando.

En el proyecto se ha utilizado RMI[5] para comunicar los aplicativos del médico y del paciente con el gestor de historias clínicas. Toda la información que se envía entre las partes se hace utilizando RMI[5].

RMI [5] ha reducido notablemente el tiempo de desarrollo.

5.2 Funcionamiento de la comunicación con RMI

El funcionamiento de la tecnología RMI[5] es el siguiente. El servidor RMI[5] establece un contrato con las aplicaciones remotas. Esto quiere decir que el servidor informa de las clases y métodos que estarán disponibles. Esta parte se hace creando una interfaz que será conocida por todos. A la vez el servidor RMI[5] tiene la implementación de la interfaz.

Cuando la clase está implementada se ha de proceder a crear los puntos de conexión que utilizarán las aplicaciones remotas. El resultado de este procedimiento crea dos clases que los clientes han de conocer. Estas se conocen como Skeleton y Stub. Una vez preparadas, las aplicaciones remotas ya pueden acceder a hacer las instanciaciones de los objetos remotos. Se ha de tener que los objetos remotos son persistentes.

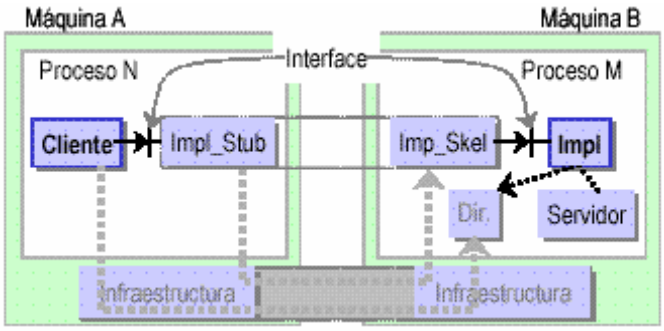


Imagen **11***: Funcionamiento de la comunicación con RMI*

5.3 Implantación de RMI en el Sistema

Utilizando RMI[5] podemos llevar a cabo la gestión de los archivos XML[10] y acceso a la base de datos de una forma centralizada, de manera que los usuarios remotos solo conozcan el nombre del método que invocan, pero realmente no saben lo que está pasando en el otro extremo de la comunicación.

Con tal de llevar a cabo esta tarea hace falta la creación de nuevas clases y de una interfaz remota. La interfaz se llama **IMetodos** e incluye la descripción de los métodos de los que se pueden disponer de terceros a través del servidor. Como toda interfaz es necesario tener una clase que implemente los métodos descritos. Esta clase se llama **IMetodosImpl**. Esta contiene la implementación de los métodos que básicamente lo que hace es gestionar las conexiones a la base de datos y la gestión de los documentos XML[10]. Finalmente se necesita una clase que instancia un servidor y lo haga público en la red. Esta clase se llama **Servidor.**

Las clases remotas que acceden a este servidor son dos, una para el médico y una para el paciente.. Cada una de estas clases crea un vínculo de comunicaciones con el servidor que permite hacer las llamadas a todos los métodos públicos. Hasta ahora, en las pruebas, estas llamadas se hacían de manera local.

Decisiones tomadas en la implementación:

- Utilizar RMI[5] porque reduce el tiempo de desarrollo de las aplicaciones remotas. Los test han verificado la portabilidad, tanto del código Java[4], como la ejecución remota en diferentes sistemas operativos como Win32, Linux y Mac OS X.

5.4 Diagrama de clases de la comunicación de los componentes.

Poco a poco se va viendo como se van relegando las tareas hacia el servidor. En la parte de la base de datos se verá como se introducen cada uno de los protocolos por los actores por separado con una abstracción más general.

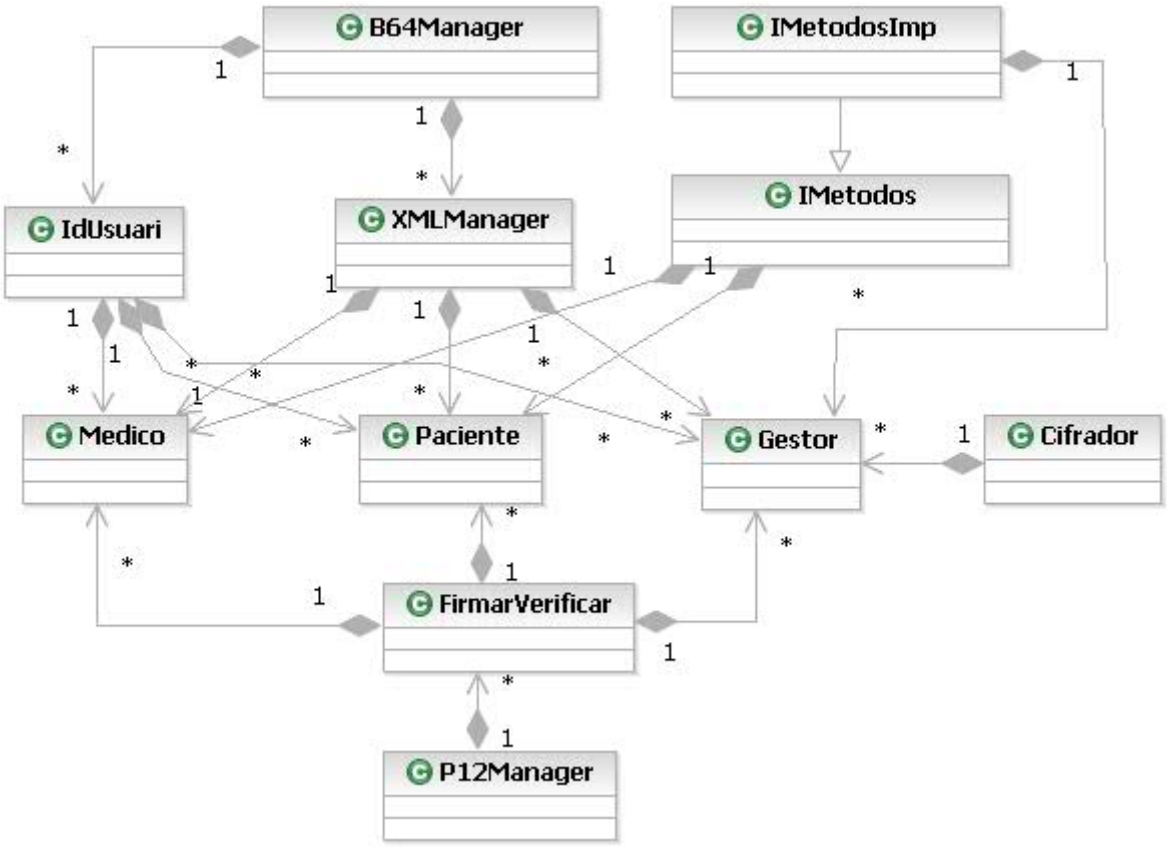


Imagen **12***: Diagrama de clases de la comunicación de los componentes*

La clase **Servidor** solo sirven para hacer públicos a través de la red los métodos descritos en la interfaz **IMetodos**, por eso no sale en el diagrama de clases.

5.5 Pruebas realizadas.

Con el fin de probar el sistema RMI[5] lo que se ha hecho es crear dos clases, una para el médico y otra para el paciente. El código que contiene cada una de ellas es la misma que ya se ha comentado en el capítulo anterior con la única diferencia que ahora ya se hacen las llamadas necesarias a los métodos remotos.

Así por ejemplo, cuando el médico quiere enviar los datos al gestor de historias clínicas se utiliza el siguiente código:

```
 cM = (IMetodos)Naming.lookup("rmi://localhost:1099/ServicioServidor");
 utils.logManager( " - Pruebas : Instancio el servidor remoto" );
...
 // enviamos el resultado Pg[Ng, insertar visita, V, Sm[V]] al Gestor --->
try {
    resultado = cM.Proceso4_p3Execute(resultado);
    utils.logManager(" - Pruebas :                " + resultado);
}catch(Exception e){
         e.printStackTrace();
         System.exit(0);
}
```

El resultado que retorna esta llamada va en función de la ejecución que se realiza en el servidor. Lo que el servidor realiza en cada momento depende de la fase en la que se encuentra la ejecución, pero en general el trabajo en este test se limita a verificar las firmas de los datos enviados, ya que el sistema aun no dispone de una base de datos. En este código se puede ver el método anterior.

```
// Constructor para declarar que puede ocurrir "RemoteException"
public IMetodosImpl() throws java.rmi.RemoteException {
       super();
 }

/**
 * Método que ejecuta el proceso 4 del protocolo3 criptográfico
 * @param fichero, es el XML parseado.
 * @throws Exception
 */
 public String Proceso4_p3Execute(String fichero) throws Exception{
        return SVGestor.Proceso4_p3Execute(fichero);
 }
```

Lo mismo se realiza en el resto de procesos. No se incluye el resto del código ya que es un poco extenso y no aporta más información útil.

5.6 Evolución del Protocolo.

A continuación se explica cual es el resultado final de la implementación RMI[5] del proyecto. Una vez la base de datos y la interfaz han sido implementadas es necesario tener una clase tanto para el médico como para el paciente que contengan el protocolo criptográfico por procesos y permitan la abstracción de los métodos de la interfaz.

Estas dos clases más la que implementa el protocolo del servidor son las encargadas en últma instancia de acceder, tal y como se ha comentado en el capítulo anterior, a la clase **XMLManager**.

Durante la implementación de este capítulo, a demás se ha utilizado un nuevo método que se llama **LogManager** y se ubica en la case **Utils**. El objetivo de este método es el de ir anotando en un archivo todas las incidencias de ejecución de la aplicación. La utilización de este método se hizo necesario una vez se vio que el contenido de log mostrado por pantalla era demasiado extenso. Como la clase **Utils** solo es una clase auxiliar se comenta con más detalle en uno de los anexos. Tampoco no se mostrará en los diagramas de clases ya que casi todas las clases acceden y dificultaría la lectura.

6. Base de datos.

6.1 Introducción.

Los test que he realizado hasta este punto necesitaban simular los datos que se recuperaban de las historias clínicas desde programa, así que cada vez que se iniciaba el programa de test, los datos se creaban de nuevo estando incluidos en el código.

La utilización de una base de datos permite almacenar los datos de las historias clínicas de forma persistente, que es el objetivo principal de este proyecto.

El sistema Gestor de Bases de Datos utilizado es MySQL[9]. Este, como la gran mayoría de programas utilizados es de libre distribución. A demás, se puede encontrar una implementación tanto para plataformas MS Win32, Linux o MacOSX, cosa que facilita la instalación en todo tipo de plataformas.

6.2 Utilidad de la base de datos.

En la base de datos se guardan todos los datos referentes al sistema, desde los datos de los certificados, la identidad de los actores, así como los datos de las historias clínicas y finalmente los datos referentes a los clientes autenticados.

El encargado de acceder a la base de datos es exclusivamente el gestor de historias clínicas, que recibirá las peticiones de los clientes. Este era uno de los objetivos iniciales del proyecto, tal y como se puede ver en el esquema de la introducción.

6.3 Modelo de la base de datos.

A continuación se muestra un esquema del modelo de la base de datos:

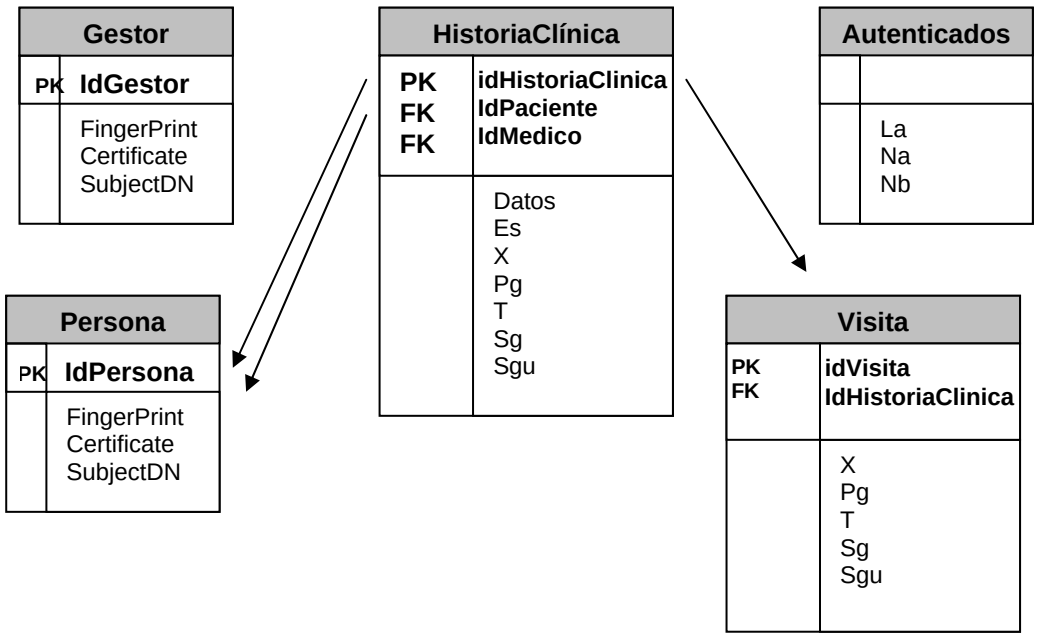


Imagen **13***: Modelo de la base de datos*

Las relaciones son las siguientes:

- Persona -> Historia clínica: Relación de 1 a 1. Relaciona a las personas paciente de la tabla Persona con las historias clínicas. Un paciente solo puede tener una única historia clínica.

- Persona -> Historia clínica: Relación de 1 a 0,,*. Relaciona a las personas médico de la tabla Persona con las historias clínicas. Un médico puede ser médico de 0 o muchas historias clínicas.

- Visita -> Historia clínica: Relación de 0..* a 1. Relaciona las visitas de la tabla Visita con las historias clínicas. Una visita solo puede pertenecer a una historia clínica y una historia clínica puede tener muchas visitas.

Como se puede ver la tabla Persona contiene tanto a los médicos como a los pacientes. Se ha optado por esta solución por ser la más sencilla y la que a priori da menos problemas. Si se cree que por razones de seguridad es mejor tener a los médicos y a los pacientes en tablas separadas, los cambios que se han de realizar son mínimos.

6.4 Descripción de las tablas de la base de datos.

Tabla Autenticados:

Esta tabla contiene los datos referentes a los clientes autenticados durante el proceso de autenticación para acceder a las opciones del sistema, y tiene los campos siguientes:

- La: función de Hash sobre el certificado del cliente en base64 (fingerprint).

- Na: número aleatorio generado por el cliente.

- Nb: número aleatorio generado por el servidor. Este campo sirve para comprobar que el cliente es quien dice ser y que tiene permisos para realizar las acciones solicitadas.

Tabla Persona:

Esta tabla almacena todos los datos, tanto del médico como de los pacientes. Realmente lo que incluye esta tabla son los certificados y los fingerprints de los mismos. A parte se añaden los datos relativos al certificado que puede servir en el futuro. Los campos que contiene son:

- IdPersona: número secuencial.

- Fingerprint: función de Hash sobre el certificado de la persona en base64.

- Certificate: el certificado X509 de la persona.

- SubjectDN: datos auxiliares sobre el certificado de las personas.

Tabla Gestor:

Esta tabla es idéntica a la anterior, pero por cuestiones de seguridad almacena a los gestores de las historias clínicas en un espacio a parte. Los campos que contiene son:

- IdGestor: número secuencial.

- Fingerprint: función de Hash sobre el certificado de la persona en base64.

- Certificate: el certificado X509 del Gestor.

- SubjectDN: datos auxiliares sobre el certificado del los gestores.

Tabla HistoriaClínica:

La tabla historiaClinica se utilza para guardar los datos de las historias clínicas de los pacientes. Contiene el identificador del médico responsable de la misma y un identificador que sirve para relacionarlo con las visitas. La descripción de los campos que contiene son los siguientes:

- IdPaciente: fingerprint del paciente, para futuras comprobaciones.
- IdHistoriaClinica: identificador de la historia clínica que sirve para relacionarlo con las visitas.
- IdMedico: fingerprint del médico, para futuras comprobaciones.
- Datos : estructura XML[10] de los datos del paciente en base64. (nombre, apellido1, apellido2) estos datos están cifrados con la clave privada del gestor.
- Es: firma digital de los datos del paciente, Esta en formato XML[10] y en base64.
- X: número de serie incremental.
- Pg: datos cifrados con la clave pública del gestor
- T : instante de tiempo.
- Sg: Firma de los datos, instante de tiempo T y número de serie con la clave privada del gestor.
- Sgu: Firma de el número de serie y el id_usuario con la clave privada del gestor.

Tabla Visita:

La tabla visita se utiliza para guardar los datos de las visitas que han tenido los pacientes. Contiene el identificador de la historia clínica del paciente. La descripción de los campos que contiene son los siguientes:

- IdHistoriaClinica: identificador de la historia clínica que sirve para relacionarlo con las visitas.
- X: número de serie incremental de la última visita del paciente.
- Pg: datos de la visita cifrados con la clave pública del gestor
- T : instante de tiempo.
- Sg: Firma de los datos de la visita, instante de tiempo T y número de serie con la clave privada del gestor.
- Sgu: Firma de el número de serie y el id_usuariop con la clave privada del gestor.

6.5 Clases responsable del acceso a la base de datos.

La clase responsable de acceder a la base de datos es DBManager. Su trabajo incluye las operaciones, tanto de inserción de personas, como las consultas de las historias clínicas y las visitas, a la vez que gestiona la entra y salida de clientes autorizados.

En el diagrama de clases de la aplicación, esta solo es usada por la clase definida en el capítulo anterior llamada Gestor.

6.6 Diagrama de clases de la parte de la base de datos.

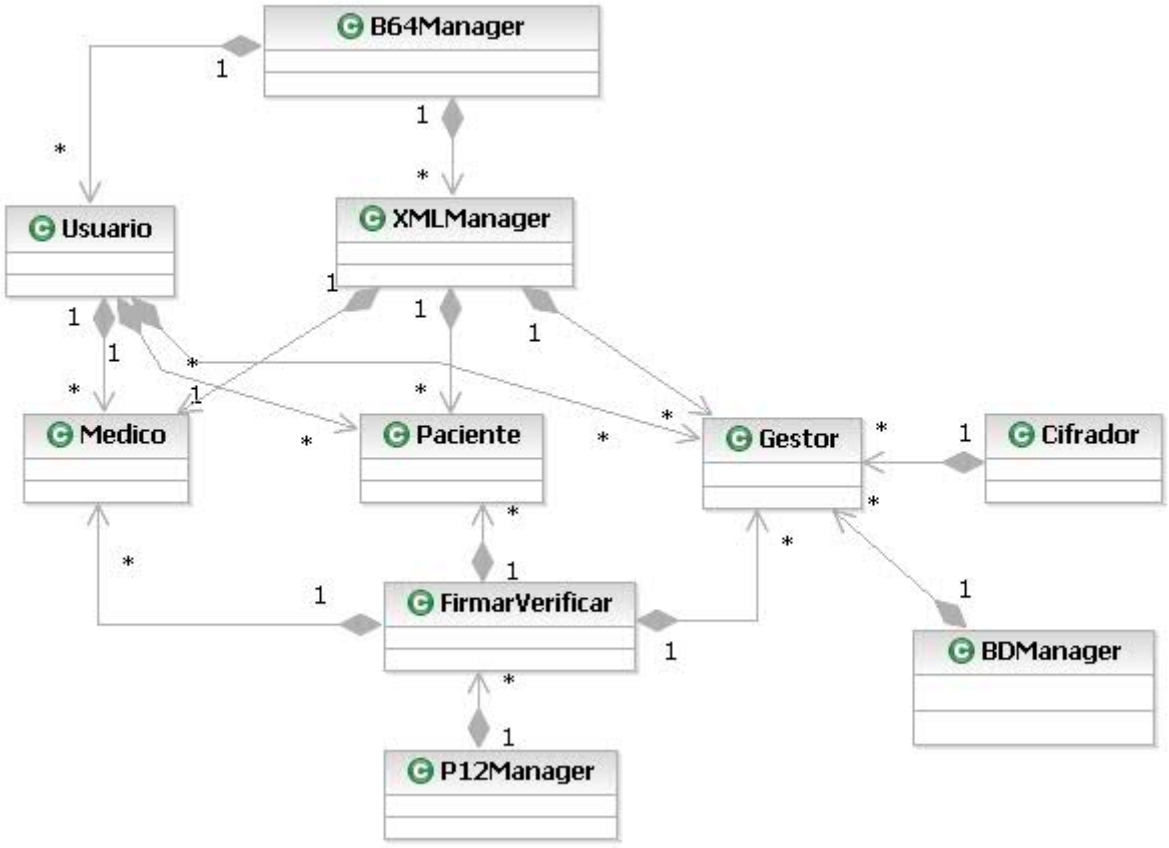


Imagen **14***: Diagrama de clases de la parte de las bases de datos*

6.7 Parametrización del acceso a la base de datos.

Para facilitar el acceso a la base de datos desde cualquier localización en la que se instale la aplicación, se ha optado por tener un archivo de configuración que permita entrar los datos básicos sobre la situación de la BD y la cuenta de acceso a la misma. El archivo se encuentra situado en la raiz de la aplicación y se llama "**host.txt**". Este que se encuentra a continuación es un ejemplo de configuración con los siguientes campos:

- Nombre de la base de datos a conectar (pfc2007 por defecto)
- Dirección IP del host de la base de datos y el puerto.
- Nombre del superusuario en la base de datos.
- Contraseña del superusuario.

Ejemplo de "hosts.txt":

```
pfc2007
localhost:3306
USUARI
pfc2007
```

7. Protocolo de autenticación.

7.1 Introducción.

Hasta este punto se ha descrito el esquema criptográfico, la representación de los datos, la comunicación de los componentes y la base de datos. Ara bien, un punto esencial es poder identificar y autenticar a los usuarios del sistema para que según quien sea cada usuario pueda realizar unas u otras operaciones. Este paso es uno de los primeros que se realiza cuando se solicitan las distintas opciones que ofrece la interfaz gráfica.

El protocolo de autenticación que se utiliza en el proyecto está basado en el protocolo modificado según Needham – Schroeder. [17].

7.2 Funcionamiento del protocolo.

El protocolo de Needham – Schroede en si es muy sencillo de implementar. En este proyecto se realiza una validación de cada usuario cuando accede a la interfaz utilizando su certificado y el conocimiento que tiene el usuario de su clave del mismo. Luego para cada uno de los protocolos se ha implementado una autenticación individual.
El protocolo establece la comunicación entre dos partes, que en esta descripción recibirán el nombre de cliente y servidor para hacer más comprensible la explicación.

La notación que se utiliza es la misma que se ha utilizado en la descripción del esquema criptográfico añadiendo las siguientes expresiones:

- Nx: número aleatorio, tendríamos $N_{cliente}$ y $N_{servidor}$.
- Lx: fingerprint o Hash del certificaado de la parte x, habrá dos, una para el cliente y otra para el servidor.

Los pasos a seguir son los siguientes.

1. Pi Realiza las operaciones siguientes:
   (a) obtener un valor de forma aleatoria, Ni.
   (b) Cifrar Ni y Pi con la clave pública de G, EG(Ni, IPi) Pi es el identificador de Pi;
   (c) enviar EG(Ni, Pi) a G;

2. G Realiza las operaciones siguientes:
   (a) Descifrar EG(Ni, IPi) con SG, y obtener; Ni y Pi ;
   (b) obtener el certificado de Pi con Pi . A partir del certificado obtendrá PPi ;
   (c) obtener un valor de forma aleatoria, NG.
   (d) Cifrar Ni,NG, IG, con la clave pública PPi de Pi, EPi(Ni,NG, IG);
   (e) enviar EPi(Ni,NG, IG) a Pi;

3. Pi Realiza las operaciones siguientes:
   (a) Descifrar EPi(Ni,NG, IG) con la clave privada SPi , y obtener NG, Ni y IG;
   (b) Cifrar NG con la clave pública PG de G, EG(NG);
   (c) enviar EG(NG) a G;

4. G Realiza las operaciones siguientes:
   (a) Descifrar EG(NG) con la clave privada SG, y obtener N0G ;
   (b) si NG' =NG,
      G y Pi están autenticados bilateralmente.

7.3 Implementación del protocolo en el Proyecto.

Para implantar el protocolo explicado en el proyecto, y tal como ya he comentado, se ha añadido una clase con dos métodos, uno local y otro remoto para simular cada una de las partes. Siguiendo las decisiones de diseño del proyecto, los datos que Intercambian el cliente y el servidor están en formato XML[10].

La integración del protocolo anterior varia un poco de la implementación que se puede encontrar en el proyecto. Con el fin de ahorrar una transferencia de datos se hizo que una vez el cliente ha realizado el primer paso y envía los datos al servidor, este crea una entrada en la base de datos conforme el cliente ha iniciado el proceso de autenticación. Con los datos que envía, el servidor puede estar seguro de su autenticidad.

Una vez los datos están introducidos en la base de datos, el servidor envía la respuesta al cliente, de manera que únicamente el cliente legítimo puede obtener el valor $N_{servidor}$ que le permitirá autentificarse más tarde contra el servidor.

Para hacer esto, en el momento en que el cliente ha de realizar alguna acción que requiere autenticación envía una petición al servidor y adjunta el valor $N_{servidor}$ que había obtenido anteriormente. Lo que hace el servidor es comprobar si en la base de datos existe algún cliente previamente donde el valor $N_{servidor}$ coincida con el que acaba de recibir. De alguna manera podríamos decir que estamos haciendo los dos últimos pasos del protocolo de autenticación pero en el momento de realizar la petición y no antes. En el caso que el servidor encuentre el valor en la base de datos procede a dar paso al cliente para realizar las acciones pertinentes a la vez que borra de la base de datos el registro del cliente autenticado. De esta manera si el cliente requiere otra acción, ha de volver a realizar todo el proceso nuevamente. Así evitamos los ataques de reutilización de valores (reply attacks).

Los datos que se pasan entre el cliente y el servidor durante esta fase están almacenados dentro de una clase que se ha declarado serializable para que pueda ser pasada a través de RMI[5] sin problemas.

8. Interfaz gráfica.

8.1 Introducción.

La interfaz gráfica de una aplicación acostumbra a ser una de las partes más críticas de implementar. El usuario final pasará muchas horas delante de la aplicación y una interfaz mal diseñada o complicada de utilizar puede condenar a la aplicación al fracaso.

En el proyecto, la interfaz era uno de los requerimientos menos importantes, se trataba de realizar una que fuera lo suficientemente simple de utilizar y que a la vez diera todas las posibilidades de realizar todas las opciones del sistema. Se ha optado por la más simple, una única pantalla para el médico como para el paciente.

8.2 Librería utilizada: SWING

SWING es un Standard Widget Toolkit, y permite diseñar interfaces de una manera sencilla y efectiva. El paquete SWING es de libre distribución y viene con JAVA. No es necesario ni instalar ni configurar nada en especial.

8.3 Aplicativo del médico.

Como se apuntaba en la introducción de la memoria, se dispone de dos pantallas, la primera de ellas es la del aplicativo del médico.

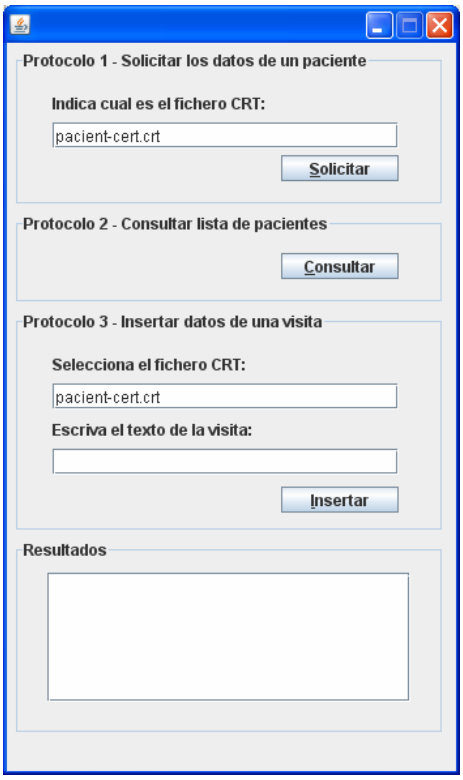


Imagen **15***: Aplicativo del médico en un Windows XP*

Esta pantalla está dividida en 4 partes. Las 3 primeras corresponden a los 3 protocolos que se han implementado en esta práctica. Se puede apreciar la simplicidad de la pantalla tal y como se había comentado.

- Protocolo 1. Consta de un campo donde se puede identificar el fichero que contiene el certificado del paciente que el médico quiere consultar. Luego hay un botón que permite ejecutar la consulta

- Protocolo 2. Este apartado solo contiene un botón para solicitar la lista de los pacientes que tiene asignados el médico que realiza la consulta.

- Protocolo 3. Dispone de un campo donde se puede identificar el certificado del paciente al cual se le quiere añadir una visita y también hay un campo donde se pueden poner los datos de la visita en concreto. Como solo es un ejemplo del funcionamiento, solo he puesto un único campo. Hay un botón que permite insertar los datos en la base de datos.

- Resultados. En este apartado van apareciendo los distintos mensajes relacionados con los protocolos que se van ejecutando. En esta ventana aparecerán los datos que se recuperan de la base de datos y otros comentarios útiles para seguir el progreso del programa.

8.4 Aplicativo del paciente.

De la misma manera, el paciente dispone de su propia interfaz gráfica. El paciente dispone de una interfaz que da pie a menos opciones ya que solo tendrá opciones de consulta sobre los datos de su historia clínica.

La interfaz del paciente es tal y como se muestra a continuación:

Imagen **16***: Aplicativo del paciente en un Windows XP*

Esta pantalla está dividida en 2 zonas. A diferencia del médico, el paciente solo tiene permisos para ejecutar el protocolo 1 ya que un paciente no puede consultar la lista de pacientes que tiene un médico asignado y tampoco puede introducir visitas en la base de datos. Un paciente solo puede consultar sus datos.

- Protocolo 1. Consta de un campo donde se puede identificar el fichero que contiene el certificado del paciente que el paciente quiere consultar. En este caso ha de ser el certificado propio. Luego hay un botón que permite ejecutar la consulta

- Resultados. En este apartado van apareciendo los distintos mensajes relacionados con los protocolos que se van ejecutando. En esta ventana aparecerán los datos que se recuperan de la base de datos y otros comentarios útiles para seguir el progreso del programa.

9. Juego de pruebas.

9.1 Introducción.

El objetivo del juego de pruebas es tener un ejemplo de la configuración y ejecución completa del sistema presentado. En este capítulo se muestra como crear los certificados de los usuarios, la configuración de la base de datos, la ejecución del servidor RMI[5] y un ejemplo completo de la ejecución de los procesos utilizando los aplicativos correspondientes.

9.2 Generación de los certificados.

Lo primer que hace falta es preparar los archivos necesarios para los usuarios de la aplicación. Lo primero que se ha de preparar es el certificado auto firmado de la entidad certificadora. Para llevar a término esta tarea se utilizan los comandos que se han comentado en el capítulo 2. Siempre que se ha tenido que poner un contraseña he utilizado “pfc2007” así será más sencillo acordarse de ella y para llevar a cabo este ejemplo es suficiente.

Adjunto a la memoria se ha subministrado un directorio llamado PKI donde se pueden encontrar los certificados creados para este proyecto. Se recomienda situarse en el directorio donde esté el openssl instalado para ejecutar más cómodamente los comandos.

Primero he creado las claves de la CA con una longitud de 2048 bits y después el certificado auto firmado.

```
OPENSSL genrsa -des3 -out CA.key 2048
```

El resultado es un archivo llamado CA.key que contiene la pareja de claves de la CA. A continuación se ha de generar un certificado auto firmado por la CA. Se ha hecho de la siguiente manera.

```
OPENSSL req -x509 -new -key CA.key -out CA.crt -days 360

    Country Name = ES
    State Name = BARCELONA
    Locality Name = BARCELONA
    Organization Name = UOC
    Organizacional Unit Name = CA
    Common Name = Fran Cáncer Giner
    Email Ardes = fcancerginer@gmail.com
```

El parámetro 360 se refiere a los días que el certificado será válido. Durante la ejecución de esta última comanda se pide que se introduzcan los datos para identificar a la CA.

El resultado de las acciones se ven a continuación.

```
C:\OpenSSL\bin>OPENSSL genrsa -des3 -out CA.key 2048
Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
...+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for CA.key:
Verifying - Enter pass phrase for CA.key:

C:\OpenSSL\bin>OPENSSL req -x509 -new -key CA.key -out CA.crt -days 360 -subj "/
C=ES/ST=BARCELONA/L=BARCELONA/O=UOC/OU=CA/CN=Fran                    Cancer
Giner/emailAddress=fcan
cerginer@gmail.com"
Enter pass phrase for CA.key:
```

El resultado es el archivo CA.crt que será necesario para hacer el resto.

Ahora hace falta la generación de los archivos de los usuarios. El primero que se hace es crear la pareja de claves del paciente. En este caso la longitud de las claves de de 1024 bits.

```
openssl genrsa –des3 –passout pass:pfc2007 –out pacient-priv.key 1024
```

Seguidamente se ha de generar la petición de certificado para la CA que se ha creado en el apartado anterior. Utilizamos el siguiente comando:

```
Openssl req -new -sha1 -key pacient-priv.key -passin pass:pfc2007 -out petic-cert-pacient.csr -subj "/C=ES / ST=BARCELONA / L=BARCELONA / O=UOC / OU=Pacients / CN=Fran Cancer Giner/emailAddress=fcancerginer@gmail.com/dnQualifier=1234p”
```

Durante la ejecución de esta última instrucción se han introducido todos los datos que identificarán al paciente.

Al trabajar en un sistema windows, no es necesario tener los archivos producidos en una ubicación concreta. Los dejaremos todos en el directorio raíz del proyecto.

Ahora se tiene que hacer que la CA emita el certificado.

```
Openssl x509 -req -in petic-cert-pacient.csr -days 180 -CA CA.crt -CAkey CA.key -
extensions usr_cert  -out pacient-cert.crt -CAcreateserial
```

Finalmente se tiene que generar el fichero .p12 que es el que la aplicación utilizará.

```
openssl pkcs12 -export -in pacient-cert.crt -inkey pacient-priv.key -name cert-pacient-pck12 -chain -CAfile CA.crt -out cert-pacient-pck12.p12
```

El resultado de las acciones se pueden ver a continuación:

```
C:\OpenSSL\bin>openssl genrsa -des3 -passout pass:pfc2007 -out pacient-priv.key
1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.....++++++
....++++++
e is 65537 (0x10001)

C:\OpenSSL\bin>Openssl req -new -sha1 -key pacient-priv.key -passin pass:pfc2007
 -out petic-cert-pacient.csr -subj "/ C=ES / ST=BARCELONA / L=BARCELONA / O=UOC /
OU=Pacients / CN=Fran Cancer Giner / emailAddress=fcancerginer@gmail.com /
dnQualifier=1234p"

C:\OpenSSL\bin>Openssl x509 -req -in petic-cert-pacient.csr -days 180 -CA CA.crt
 -CAkey CA.key -extensions usr_cert  -out pacient-cert.crt -CAcreateserial
Loading 'screen' into random state - done
Signature ok
subject=/C=ES/ST=BARCELONA/L=BARCELONA/O=UOC/OU=Pacients/CN=Fran  Cancer
Giner/em
ailAddress=fcancerginer@gmail.com/dnQualifier=1234p
Getting CA Private Key
Enter pass phrase for CA.key:

C:\OpenSSL\bin>openssl pkcs12 -export -in pacient-cert.crt -inkey pacient-priv.k
ey -name cert-pacient-pck12 -chain -CAfile CA.crt -out cert-pacient-pck12.p12
Loading 'screen' into random state - done
Enter pass phrase for pacient-priv.key:
Enter Export Password:
Verifying - Enter Export Password:
```

El resultado es el archivo cert-pacient-pck12.p12

Ahora es necesario hacer lo mismo para el medico y para el administrador que utilizaremos en el ejemplo. La ejecución es la misma que para el paciente. Se trata de hacer lo mismo que se ha hecho teniendo en cuenta:

- En todos los sitios que se ha puesto pacient, se tiene que sustituir por Metge o por administrador.
- En todos los certificados se tienen que poner los datos:
  - Organizational Unit Name: Metges o Administradors

Ejecutando todo esto se tendrían que tener tres archivos .p12

- Cert-metge-pck12.p12
- Cert-pacient-pck12.p12
- Cert-administrador-pck12.p12

Guardamos estos tres archivos en la raíz del proyecto.
Esta fase está completada. Ahora explicaré como preparar la base de datos.

9.3 Preparación de la base de datos.

Se parte de la base de que disponemos de una máquina dedicada a alojar el servidor de base de datos, en la que se ha instalado previamente MySQL[9]. Para la gestión de la base de datos se ha utilizado los programas que vienen incorporados como el MySQL Administrador y el MySQL Query Browser.
Lo primero que se ha de ejecutar es el fichero que se puede encontrar en el anexo D (creación de las tablas y carga de los datos). Un fichero crea la base de datos pfc2007 y las tablas necesarias y el otro fichero carga unos datos básicos necesarios para poder probar la aplicación.
Seguidamente es indispensable dar permisos a todos los usuarios que se conectarán desde el aplicativo de gestor de historias clínicas en la base de datos que se ha creado. Se ha de crear un usuario llamado "USUARI" con password "pfc2007", hay que dar permisos al usuario USUARI con la dirección localhost de la máquina que ejecuta el servidor gestor de historias clínicas.
Para hacer el acceso a la base de datos de manera sencilla se ha habilitado la posibilidad de tener un fichero llamado "**host.txt**", que se encuentra en la raíz de la aplicación del gestor que contiene los datos del usuario que se conecta a la base de datos.

Ejemplo de "hosts.txt":

```
pfc2007
localhost:3306
USUARI
pfc2007
```

En este ejemplo se estaría diciendo que la base de datos se llama pfc2007, que está localizada en el servidor con dirección localhost:3306 y que el usuario se llama USUARI con contraseña pfc2007. Por lo tanto, según la configuración del sistema en concreto hay que dar permisos al usuario USUARI con la dirección localhost de la máquina que ejecuta el servidor gestor de historias clínicas.

Una vez hecho esto se puede dar por configurada la base de datos que ya está preparada para aceptar las peticiones que el gestor de historias clínicas realice. La primera de estas peticiones es la de introducir los datos de los certificados que se han generado en el primero paso utilizando la clase **CertManager**.

El siguiente paso para probar la aplicación será ejecutar el script "cargaDeDatos.sql" que añade unas cuantas historias clínicas con datos cifrados y alguna visita. Estos datos son los que consultarán y recuperarán las opciones que se han desarrollado en este proyecto.

9.4 Inserción de datos en la base de datos.

Este paso es crítico y a veces fácil de olvidar. Si la base de datos no cuenta con los datos de los certificados no puede comprobar que los datos que los clientes le están enviando en el momento de la autenticación son correctos.
Para hacer esta inserción se dispone de una aplicación realizada en Java[4]. Esta aplicación se llama **CertManager**.java y necesita de las clases que el gestor dispone en su directorio, concretamente utiliza BDManager, B64Manager y P12Manager. La ejecución se realiza de la siguiente forma:

```
Java CertManager filename.p12 password –g|-p
```

El último parámetro indica si el usuario que se entra es una persona que utilizará la aplicación o bien es un gestor. El trato para los dos es diferente. La ejecución para la introducción del gestor sería:

```
Java CertManager filename.p12 password –g
```

La ejecución para los otros usuarios, médico y pacientes sería:

```
Java CertManager filename.p12 password –p
```

Una vez realizadas estas ejecuciones, los datos de los actores del sistema están correctamente entrados en la base de datos.

En este punto el servidor está casi preparado. Solo Faltara arrancar el servidor RMI[5] que lo dejará en disposición de recibir peticiones de los clientes remotos, antes, se ha de configurar la máquina virtual Java[4] para que reconozca las librerías externas que se han utilizado.

9.5 Configuración para la ejecución en Java.

Para que todo se pueda ejecutar sin problemas en un entorno Java[4], es necesario modificar algunos archivos de la instalación base de Java[4] y a la vez se han de añadir las librerías propias que se tienen que utilizar.

La principal librería es la de IAIK[15] que se ha copiado en %JAVA_HOME%\jre\lib|ext

- Iaik_jec_full.jar -> contiene los algoritmos ESDH, IDEA, RC5 y RC6 firmados y sin firmar. Bajada la última versión: ver. 3.142 de https://jce.iaik.tugraz.at/crm/freeDownload.php

La instalación de la anterior librería conlleva la modificación de dos archivos de seguridad que se encuentran en % JAVA_HOME %\jre\lib|security.

- local_policy.jar y US_export_policy.jar -> Políticas de seguridad que permiten utilizar cualquier longitud de clave.
  Bajada la última versión: Unlimited Strength Jurisdiction Policy Files 6.0 RC de http://java.sun.com/javase/downloads/index.jsp

También son necesarias las librerias de JDOM[8] para tratar documentos XML[10] y la librería de acceso a base de datos MySQL[9].

- jdom.jar
- mysql-connector-java-3.0.15-ga-bin.jar

Los dos archivos se han copiado en la misma ubicación que la otra librería. %JAVA_HOME%\jre\lib|ext La instalación de las librerias gráficas se comentan en el apartado del cliente.

9.6 Ejecución del servidor RMI

El servidor RMI[5] depende de que Java[4] este correctamente instalado, cosa que no depende de este proyecto y a la vez de que no haya un firewall que bloquee las conexiones de red. El puerto que utiliza RMI[5] en general es el 1099. Una vez está asegurado que este puerto esta abierto podemos pasar al siguiente paso. Por una parte es necesario compilar el código fuente y después es necesario preparar las estructuras que permitirán a RMI[5] aceptar conexiones.

Suponiendo que ya se tiene todo le código compilado se deberían ejecutar los siguientes comandos.

```
rmic -d . uoc.pfc.IMetodosImpl
start rmiregistry
start java uoc.pfc.Servidor
```

El primer comando prepara las estructuras que ya se han comentado en el capítulo de RMI[5], en concreto creará un archivos que es indispensable

- IMetodosImpl_Stub.class

Los aplicativos del médico han de tener acceso a estos archivos ya se vía las variables de entorno o por que se encuentran en la misma carpeta del aplicativo.

El segundo comando ejecuta como servicio el rmiregistry, que abre el puerto necesario y se pone a escuchar. Finalmente, la última comanda ejecuta el servidor.
La parte del servidor está lista.

9.7 Ejecución de la interfaz gráfica del médico.

Como ya se ha comentado en su capítulo, para las interfaces gráficas se ha utilizado la librería que vienen con las distribuciones de JAVA, llamadas SWING.

La clase con la interfaz del médico se llama SWGMedico y para ejecutarla se puede usar el siguiente comando:

```
Java SWGMedico
```

Aparecerá una pantalla de bienvenida como la siguiente:

Imagen **17***: Pantalla de bienvenida.*

Una vez cargado el programa se pide primero y una única vez que el usuario se identifique. Para hacer esto se ha de colocar en el disco el archivo cert-metge-pck12.p12 que se ha creado en el primer punto (creación de los certificados) y entrar la contraseña (“pfc2007” en este caso). Eso ser ve así:

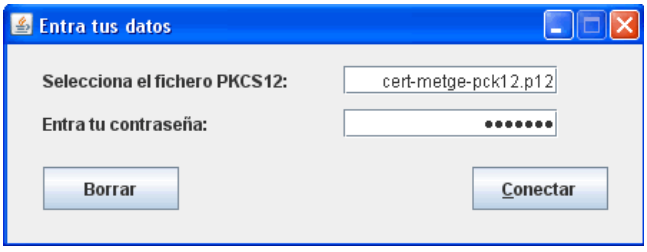


Imagen 18*: Pantalla de autenticación.*

Si la identificación no provoca ningún error, entonces se presenta la pantalla principal del médico.

9.8 Ejecución de la interfaz gráfica del paciente.

Los pasos para ejecutar la interfaz del paciente son los mismos con la diferencia que en este caso la clase se llama SWTPaciente y se ejecuta con:

```
Java SWTPaciente
```

Aparecerá una pantalla de bienvenida idéntica a la anterior y se pedirá igualmente que el usuario se identifique. En este caso se ha de localizar el archivo cert-pacient-pck12.p12 que se ha creado en el primer punto (creación de los certificados) y entrar la contraseña “pfc2007”.

9.9 Consulta de los datos de un paciente

Esta opción puede ser solicitada por el médico y por el paciente utilizando las mismas pantallas.
La pantalla principal del médico o del paciente es la que ya se ha mostrado anteriormente:

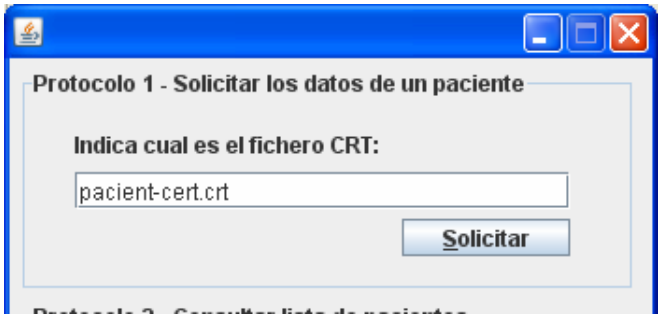


Imagen **19***: Aplicativo del médico en un Windows XP- Protocolo 1.*

Como se puede ver en la imagen se han rellenado los campos de texto de la petición. Una vez los campos están rellenados correctamente se puede apretar el botón “Solicitar”

Si todo ha ido bien, se mostrará el siguiente mensaje:

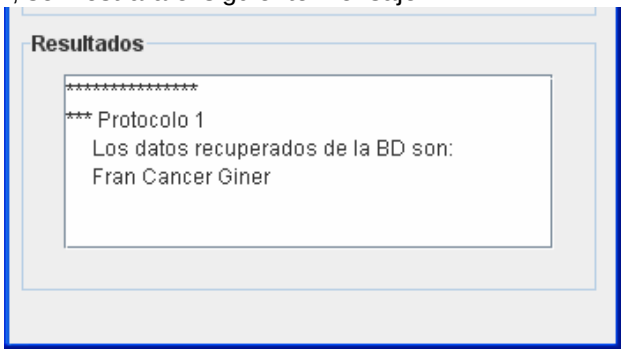


Imagen **20***: Datos recuperados del protocolo 1*

Si lo ha solicitado un médico, los datos que recupera son los del paciente cuyo certificado puso en el campo CRT. Si lo ha solicitado un paciente, los datos que recupera son los suyos propios y el certificado que ha indicado es el suyo.

Con esto se da por completada el proceso de consulta de una historia clínica.

9.10 Consulta de los pacientes asignados a un médico

La pantalla principal del médico es la que ya se ha mostrado anteriormente:

Imagen **21***: Aplicativo del médico en un Windows XP- Protocolo 2.*

Cuando se desee se puede apretar el botón “Consultar”

Si todo ha ido bien, se mostrará el siguiente mensaje:

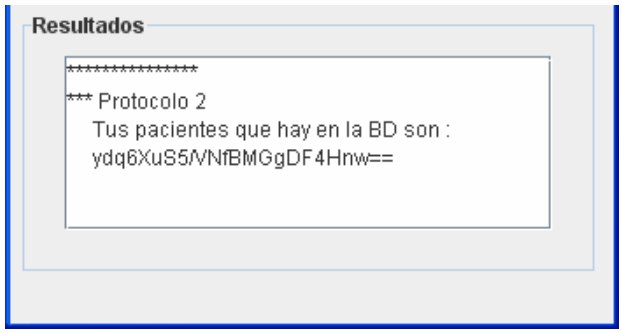


Imagen **22***: Datos recuperados del protocolo 2*

Con esto se da por completada el proceso de consulta de los pacientes asignados a un médico..

9.11 Inserción de una visita en la BD.

La pantalla principal del médico es la que ya se ha mostrado anteriormente:

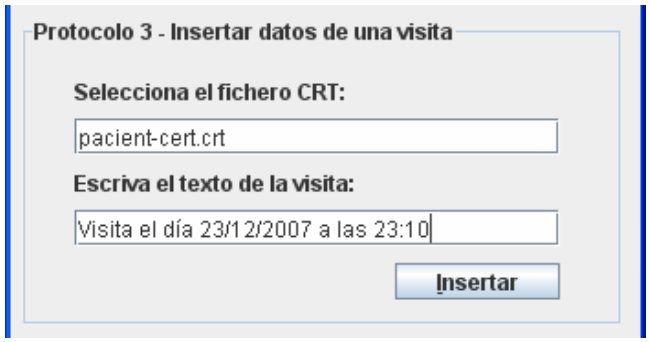


Imagen **23***: Aplicativo del médico en un Windows XP- Protocolo 3.*

Como se puede ver en la imagen se han rellenado los campos de texto de la petición. Una vez los campos están rellenados correctamente se puede apretar el botón “Insertar”

Si todo ha ido bien, se mostrará el siguiente mensaje:

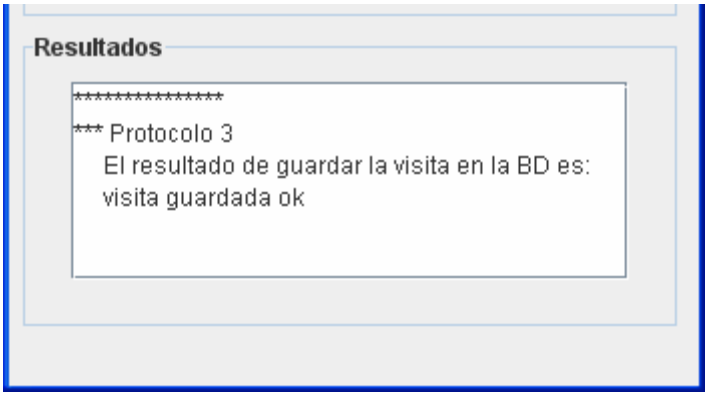


*Imagen **24**: Datos recuperados del protocolo 3*

Con esto se da por completada el proceso de inserción de una visita en la BD.

9.12 Apagar el sistema.

Para salir de los programa del Médico y del paciente en cualquier momento se puede apretar el botón Salir (X en la esquina superior derecha) y este terminará. El médico o el paciente podrán arrancar de nuevo la aplicación y podrá recuperar los datos que estén almacenados en la base de datos.
Respecto a la parada del servidor, como aun no se dispone de una interfaz para el gestor de historias clínicas, esta se ha de hacer manualmente. Es decir, se ha de ejecutar el visualizador de procesos del sistema operativo en cuestión y acabar las dos operaciones que se habían arrancado al principio.

En primer lugar se ha de parar el proceso “rmiregistry” y posteriormente se ha de parar el proceso “java Servidor nombrefichero.p12 password”.

Con estos últimos pasos el juego de pruebas se da por finalizado.

10. Diagramas.

10.1 Introducción.

Durante la explicación de los capítulos anteriores se ha ido añadiendo poco a poco las clases que forman el sistema completo, de manera que se iba viendo como la estructura del diagrama de clases se parecía al esquema general de la aplicación.

En este capítulo se muestra el diagrama de clases completo.

10.2 Diagrama de clases.

El diagrama de clases[2] que se presenta a continuación incluye la gran parte de las clases que se han utilizado en el proyecto. Las que no aparecen es porque son clases auxiliares que no modifican la ejecución del sistema. Como ejemplo de estas clases auxiliares podemos nombrar la clase **LogManager**, descrita en los anexos o la clase **Servidor** que solo sirve para hacer pública la clase **IMetodos**.

Otras clases aunque son importantes se ha preferido dejar fuera del diagrama para no complicarlo. Por ejemplo, hay una clase por cada procedure, un total de 6 procedures, estas 6 procedures son usadas por la clase médico, paciente y gestor. Se han representado como una única clase de forma genérica ya que representar todas las relaciones de las clases con estas procedures complicaría mucho el diagrama y sería difícil de interpretar.

Diagrama de clases

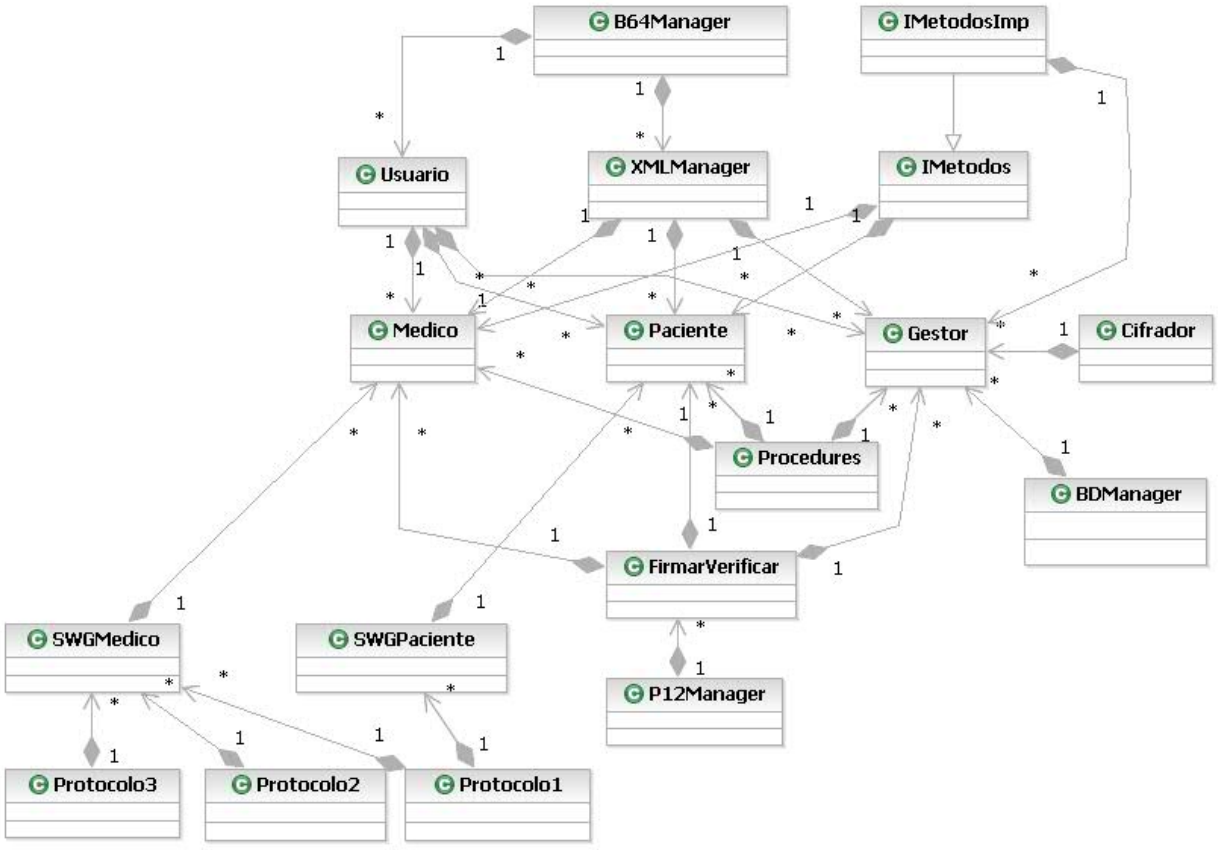


Imagen **25***: Diagrama de clases completo*

11. Trabajo futuro.

11.1 Introducción.

Un proyecto de software raras veces se puede dar por finalizado, siempre hay algún aspecto que no ha sido cubierto o alguna mejora que se podría añadir. Este capítulo está dedicado a las posibles ampliaciones que se podrían implementar en el sistema.

11.2 Mejoras a implementar.

Lo que se ha implementado en este proyecto debería de servir de base para desarrollar la gestión de una historia clínica completa utilizando sistemas criptográficos.

Como puntos importantes sobre el trabajo a realizar en el futuro destacamos:

- Mejora de la interfaz gráfica adaptándola a las necesidades de cada centro médico.

- Añadir una interfaz para el gestor de historias clínicas, ya que ahora mismo los datos de los usuarios se han de introducir manualmente en la base de datos. De la misma manera, esta interfaz debería de dar acceso a la base de datos para gestionar las historias clínicas y el resto de los datos como súper usuario. Por ejemplo, ahora mismo no hay manera de eliminar un registro de la base de datos si no es utilizando el SGBD.

- Actualmente los campos que se guardan de la visita y los que se guardan del paciente son mínimos y sirven para probar la aplicación. Se debería hacer un estudio para determinar cual es la verdadera necesidad de información que se quiere guardar ampliando los actuales campos.

- Se debería hacer un estudio de que campos puede ver el paciente y que campos no y crear un sistema de seguridad que permitiera asignar a cada usuario un rol de forma que añadiendo roles a un paciente pudiera ver más o menos campos. Esto no está implementado ya que para probar la aplicación solo se han utilizado unos pocos campos.

- Permitir la comprobación de los certificados con el certificado de la CA. Esto quería decir incluir un repositorio de CA's de confianza. Esta tarea no se ha acabado de implementar en este proyecto.

- Estudiar la posibilidad de utilizar Smart Cards para realizar la entrada y la autenticación al sistema. De esta manera el usuario dispone de una tarjeta con la que no ha de ir arriba y abajo con sus archivos PKCS12[20]. Realmente sería cómodo de utilizar. Cabe destacar que la estructuración en módulos diferenciados, remitiría incluir esta posibilidad con sencillas modificaciones.

- Actualmente se ha utilizado PKI de libre distribución Openssl. En un sistema de producción sería necesario la utilización de una PKI para la gestión de los certificados de los usuarios del sistema.

- Relacionado con el punto anterior, y para mejorar el procedimiento de la creación de los PKCS12[20] se podría implementar, en la misma aplicación, un sistema para crear los archivos de manera automática y cómoda. Se podría mirar de sincronizar con el servidor de personas de la empresa o institución.

- De la misma manera que en muchas aplicaciones, donde existe la posibilidad de utilizar varios idiomas, se podría estudiar la posibilidad de traducir la aplicación a diversos idiomas y que dependiendo de la máquina host de cada usuario se utilizar un idioma u otro.

- De la misma manera que se ha hecho con la base de datos, donde se ha permitido la utilización de un fichero de configuración para localizar el servidor MySQL[9], se podría facilitar el acceso al servidor con un fichero de configuración para el servidor RMI[5]. Así en el caso de cambio de máquina no se debería de recompilar el código.

12. Conclusiones.

Una vez llegados a este punto es el momento de hacer una valoración del trabajo requerido y del trabajo presentado. En este capítulo se demostrará como los requisitos que se pidieron en el enunciado del proyecto han estado satisfactoriamente conseguidos.

De entrada y como punto más importante, se dispone de un esquema criptográfico que nos asegura el principal objetivo de este proyecto: la disponibilidad de realizar consultas e introducción de datos de forma remota, protegida y asegurando en todo momento la identidad de los usuarios.

Si se repasan las líneas principales del enunciado se puede ver como poco a poco todos los puntos críticos se han ido consiguiendo.

Punto 1: Creación de un esquema criptográfico.

Como ya se ha comentado, y tal como se citaba en los objetivos iniciales de la memoria se ha conseguido que el esquema criptográfico cumpliera lo siguiente:

- Confidencialidad.
- Autenticidad.
- Integridad.
- No repudio.

De la misma manera se puede comprobar como se ha conseguido que el sistema garantice coherencia con los procesos que se apuntaban al inicio.

- Consulta de la historia clínica de un paciente.
- Consulta de los pacientes de un médico.
- Inserción de una nueva visita a un paciente.

Punto 2: Representación de los datos XML.

El sistema dispone de un sistema de representación de los datos que nos ha permitido guardar los datos en documentos XML[10] para hacer más sencilla la portabilidad entre los diferentes aplicativos del proyecto. Este objetivo ha sido alcanzado por completo tanto por la información de consulta de las historias clínicas como por la información de añadir nuevas visitas a los pacientes.

Punto 3: Comunicación de los componentes.

El sistema funciona con una base para la comunicación entre las diferentes partes montada sobre la API RMI[5] que Java[4] facilita. Este era uno de los objetivos primordiales, ya que de no funcionar así, se abría optado por un sistema web donde la gran parte del trabajo la haría el servidor en lugar de estar repartida como está ahora.

Punto 4: Base de datos.

El servidor cuenta con un sistema gestor de bases de datos en el que puede almacenar toda la información de las historias clínicas y las visitas, a demás de la información relativa a las personas que pueden acceder al sistema. La base de datos se ejecuta utilizando MySQL[9].

Punto 5. Protocolo de autenticación.

Finalmente, antes de pasar a desarrollar la interfaz gráfica se ha conseguido implementar el protocolo de autenticación que permite a los usuarios realizar las acciones que soliciten si y solo si el servidor comprueba que son quienes dicen ser.

Punto 6: Interfaz gráfica.

Se ha intentado hacer la interfaz gráfica lo más simple e intuitiva posible facilitando la interacción con el sistema. Cabe destacar, que a primer golpe de vista al ver la interfaz puede parecer que el programa en si es muy simple y que se buscan los datos solicitados en la base de datos y ya está. No obstante no es así, la calidad del trabajo interno que se realiza para cada una de las opciones que la interfaz permite es muy importante.

Opinión personal.

Mi opinión respecto del proyecto no podría ser más positiva. Ha sido muy gratificante poder unir todas las herramienta y formas de trabajar que había utilizado por separado en tantas asignaturas durante la licenciatura. La verdad es que no ha sido un simple proyecto en el que solo he tenido que aplicar todo lo que sabía, sino que para cada punto he tenido que profundizar en la materia para descubrir el verdadero funcionamiento y la mejor manera de aplicarlo sin que el resto del proyecto se viera perjudicado, si no al contrario, beneficiado.

Lo que más me ha gustado ha sido trabajar utilizando un diseño y una implementación incremental, cosa que permite que una vez que una parte funciona bien se puede casi olvidar para centrarse en la siguiente que funcionará por encima. La verdad es que normalmente me habría imaginado que la parte dada por buena se habría de ir modificando una y otra vez para adaptarla a las nuevas funcionalidades y la verdad es que no ha sido así, cosa que me ha sorprendido y encantado.

Si ahora volviera a comenzar el proyecto desde cero encontraría muchos detalles que funcionarían de otra manera y probablemente el resultado sería un poco mejor. Esto pasa siempre y es debido a que hay muchas cosas que hasta que no te pones no piensas que sucederán. Supongo que esta situación se hace cada vez menos usual a medida que pasan los años y la experiencia te ayuda a planificar mejor los proyectos.

Bibliografia

[1] Apuntes de Criptografía de la UOC. Versión de l'any 2006.

[2] Apuntes d'Enginyeria del Software III de la UOC Versión de l'any 2006.

[3] Jordi Herrera-Joancomartí, Josep Prieto-Blázquez, Jordi Castellà-Roca: A Secure Electronic Examination Protocol using Wireless Networks. ITCC (2) 2004: 263-268.

[4] The J2SE Development Kit (JDK), java.sun.com/downloads

[5] Java Remote Method Invocation (Java RMI), http://java.sun.com/products/jdk/rmi/.

[6] Java 2 Standard Edition Development Kit 5.0, Installation Notes, http://java.sun.com/j2se/1.5.0/install.html.

[7] Eclipse universal tool platform, www.eclipse.org (download www.eclipse.org/downloads/index.php).

[8] The JDOM XML API, www.jdom.org/docs/apidocs/index.html (web page www.jdom.org).

[9] The MySQL database server, www.mysql.com/documentation/index.html, www.mysql.com/doc/en/index.html, (download www.mysql.com/downloads/index.html).

[10] Extensible Markup Language (XML), www.w3.org/XML

[11] XML Tutorial www.w3schools.com/xml, www.w3schools.com/dtd

[12] Openssl: The open source toolkit for SSL/TLS, www.openssl.org, www.openssl.org/support/faq.html

[13] The Win32 OpenSSL Installation Project, www.slproweb.com/products/Win32OpenSSL.html

[14] The Unified Modeling Language: UML, www.uml.org

[15] The "Institute for Applied Information Processing and Communication", (download http://jce.iaik.tugraz.at/sic/download).

[16] A graphical tool to manage your MySQL database anywhere in the world: SQLyog, www.webyog.com/sqlyog/index.php, (download www.webyog.com/sqlyog/download2.html).

[17] Protocol d'autenticació Needham-Schroeder, dimacs.rutgers.edu/Workshops/ Security/program2/boyd/node14.html

[18] Menezes A.J, van Oorschot P.C, Vanstone S.A., Handbook of applied cryptography, CRC Press, ISBN 0-8493-8523-7.

[19] Anderson R., Security Engineering – A guide to building dependable distributed systems, John Wiley & Sons, Inc, ISBN 0-471-38922-6.

[20] PKCS #12: Personal Information Exchange Syntax Standard, http://www.rsa.com/node.aspx?id=1155

[21] PKCS #7: Cryptographic Message Syntax Standard, http://www.rsasecurity.com/rsalabs/node.asp?id=2129

[22] RFC 2510 - Internet X.509 Public Key Infrastructure Certificate Management Protocols, http://www.faqs.org/rfcs/rfc2510.html

[23] RFC 2560 - X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP, http://www.faqs.org/rfcs/rfc2560.html

Anexos

Anexo A: Glosario de términos.

API: Siglas de Application Programming Interface. Interfaz a través de la cual un programa accede a los servicios del sistema operativo y otros. Una API da un nivel de abstracción entre la aplicación y el kernel con tal de asegurar la portabilidad del código.

Aplicativo: En nuestro caso, los aplicativos son el conjunto de clases que conforman Cada una de las herramientas que los actores del sistema utilizan.

Arquitectura cliente – servidor: Sistema a través del cual, un usuario hace peticiones que son transmitidas al servidor, que las trata y las reenvía a el usuario. En general, las máquinas del cliente y del servidor son diferentes, pero no se descarta que pueda funcionar sobre la misma.
.

Autoridad de Certificación: Entidad que emite certificados digitales a los usuarios o compañías, de manera que estas se puedan identificar delante de un tercero. Es de vital importancia que la autoridad de Certificación compruebe que la parte que pide un certificado es realmente quien dice ser.

Base 64: Codificación que utiliza solo 6 bits por carácter. De esta manera tenemos 64 posibles valores. En nuestro caso permite transmitir cadenas que contienen las firmas como cadena de caracteres sin estropear el contenido de las mismas.
Base64 también se utiliza mucho en comunicaciones con datos binarios provenientes de texto como por ejemplo del correo electrónico.

Base de Datos: Un o mas conjuntos estructurados de datos persistentes, normalmente asociados a programario que las consulten y actualicen. Una base de datos es un componente de un Sistema Gestor de base de Datos.

Caso de uso: Diagrama perteneciente a las especificaciones UML que permite ver Gráficamente y para cada actor del sistema, las acciones que pueden llevar a cabo y las relaciones de estas acciones con el sistema.

Certificado: Archivo que contiene los datos que dan fe de la autenticidad de la persona o entidad que la presenta.

Clave: Pieza de información que se utiliza en criptografía simétrica para cifrar y descifrar un mensaje.. En criptografía asimétrica la clave puede ser pública o privada. La clave pública se utiliza para cifrar mensajes o verificar una firma. La clave privada se utiliza para descifrar o pera firmar unos datos. La longitud en bits de la clave suele dar una idea de la robusteza del sistema que utilizan.

Document Type Definition: Definición de un documento XML o SGML. Consiste en unas reglas para interpretar los documentos y para establecer las regles de construcción de los mismos.

DTD: Ver Document Type Definition.

Entorno de desarrollo: Conjunto de herramientas que permiten el desarrollo de un programario de manera integrada, des de la redacción del código, hasta la posterior compilación y la ejecución a pasos (debugging) para verificar los errores.

Fichero de configuración: Archivo accesible para el usuario con opción a ser modificado y que permite que el sistema se adapte a otro entorno de ejecución sin necesidad de recompilar el código fuente. En este proyecto los ficheros de configuración sirven para configurar el acceso a la base de datos.

Fingerprint: Función de hash que se aplica sobre un certificado de un usuario para obtener un identificador único y mas cómodo de utilizar. En el proyecto se usan fingerprintos para identificar de manera única a los actores del sistema a partir de sus certificados.

Función de Hash: Resumen con pérdida que da lugar a una secuencia de longitud fija a partir de unos datos sin importar la longitud de estos. Sus propiedades permiten su utilización a la vez de verificar la integridad de las datos. las funciones de hash son utilizadas para asignar posiciones en temes de búsqueda y ordenación.

IAIK: Siglas de "Institute for Applied Information Procesing and Communication". Estos son los desarrolladores de la librería criptográfica con el mismo nombre.

Interfaz: Punto de interacción y/o comunicación entre un ordenador y otra entidad ya sea persona u otro equipo.

Java: Lenguaje de programación multi-plataforma, robusto, interpretado, distribuido, orientado a objetos, portable, desarrollado por Sun Microsystems a mediados de los 90.

JDOM: Siglas de Java Document Object Model. Solución completa basada en Java para acceder y modificar documentos XML des de código Java.

Libre distribución: Se dice así un programario que se ofrece libremente sin la necesidad de que el usuario final abone una cantidad de dinero para su utilización. Normalmente, con la distribución se incluye código fuente al que el usuario tiene acceso para modificarlo y redistribuirlo si así lo desea.

Log: En informática, se conoce el log, como la zona del sistema donde se anotan las incidencias que van ocurriendo. Sirve como información y a la vez como guía para detectar y solucionar problemas de las aplicaciones y sistemas.

Longitud de clave: En termes de criptografía indica el nombre de bits de la clave que usan para cifra y descifrar los datos. Las claves simétricas y privadas se han de mantener en lugar seguros.

Máquina Virtual: Máquina abstracta para la que existe un intérprete. En general se utiliza en sistemas operativos para a asegurar la portabilidad de las aplicaciones entre ellos mismos. Existen máquinas virtuales para muchos lenguajes de programación siendo Java la más popular hoy en día.

MySQL: Sistema Gestor de base de Datos de libre distribución.

Número aleatorio: Número generado sin que el usuario tenga contacto con el proceso.
Generalmente en nuestro caso, y en criptografía, se utiliza para a poder demostrar que entre sesiones, un usuario es quien dice ser ya que solo las dos partes implicadas conocen el número generado.

PKCS: Siglas de Public-Key Cryptography Standards. Son un conjunto de estándares definidos por los laboratorios RESA que especifican los estándares de clave pública.

PKI: Siglas de Public Key Infrastructure corresponent a infraestructura de clave pública.

Puerto: Camino lógico extremo de un canal en un sistema de comunicaciones. Los protocolos TCP y UDP utilizados en Ethernet utilizan los puertos para desmultiplexar canales lógicos de la misma interfaz de red de una computadora.

Protocolo de autenticación: Conjunto de operaciones que llevan a término dos o más partes de manera que al final al menos una de las partes queda autenticada delante del resto.

Pruebas de integración: Conjunto de test que se llevan a cabo sobre una aplicación en fase de desarrollo con tal de asegurar su correcto funcionamiento como los de otros aplicativos o sistemas con los que interactúa.

RMI: Siglas de Remote Method Invocation. API propietaria de Java que permite a las aplicaciones locales ejecutar código que se encuentra alojado en otra máquina remota. Esta última pone a disposición unos métodos públicos que serán accesibles a través de una interfaz.

Signatura: Documento generado a partir de un mensaje y la clave privada de un usuario. Al cifrar los datos del mensaje con la clave privada se genera un mensaje cifrado que una tercera persona puede descifrar con la clave pública y comprobar que es lo mismo que los datos originales. De esta manera se asegura que el origen de las datos no ha estado modificado y que la persona que lo envía es quien dice ser, ya que solo ella tiene acceso a su clave privada.

Smart Cards: Tarjeta que dispone de un chip electrónico seguro contra manipulaciones.

Sobre digital: Método utilizado en la criptografía que utiliza los dos tipos de Criptosistemas. Por una parte el mensaje a transmitir se cifra utilizando una cifra de clave simétrica. A continuación, la clave que se utiliza para llevar a cabo este cifraje se cifra utilizando un sistema de cifraje asimétrico. El resultado de las dos operaciones es el documento que se envía a la otra parte. La ventaja de utilizar este sistema es que se reduce drásticamente el tiempo utilizado en cifrar el documento original ya que al cifrado simétrico es mucho más rápido que el asimétrico.

Script: Conjunto de operaciones que se agrupan en un archivo que las ejecuta una detrás de la otra para a facilitar la ejecución de tareas repetitivas. En nuestro caso usamos scripts para la generación de los certificados. Durante el desarrollo también los he usado para arrancar el servidor.

Sistema Gestor de base de Datos: Conjunto de aplicaciones que normalmente llevan control de un conjunto de datos persistentes, ofreciendo a la vez facilidad de acceso y consulta a los usuarios finales.

SGBD: Ver Sistema Gestor de base de Datos.

SWT: Siglas de Standard Widget Toolkit. Librería para a la creación de interfaces gráficas desarrollada por el proyecto Eclipse, y que es de libre distribución.

UML: Siglas de Unified Model Language. Lenguaje no propietario de especificación. Es un método utilizado para especificar, visualizar, construir y documentar un sistema orientado a objetos en fase de desarrollo.

www: Red de computadores que contienen lugares de Internet que ofrecen texto y imágenes, sonido, animaciones a través del protocolo de red HTTP (HyperText Transfer Protocol)

Anexo B: Fichero de configuración para PKI

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################
[ ca ]
default_ca     = CA_default          # The default ca section

####################################################################
[ CA_default ]

dir             = ./CAPFC             # Where everything is kept
certs           = $dir/certs          # Where the issued certs are kept
crl_dir         = $dir/crl            # Where the issued crl are kept
database        = $dir/index.txt      # database index file.
new_certs_dir   = $dir/newcerts               # default place for new certs.

certificate     = $dir/CA.crt # The CA certificate
serial          = $dir/serial         # The current serial number
crl             = $dir/crl.pem                # The current CRL
private_key     = $dir/private/CA.key # The private key
RANDFILE        = $dir/private/.rand  # private random number file

x509_extensions       = usr_cert            # The extentions to add to the cert

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days   = 365                  # how long to certify for
default_crl_days= 30                  # how long before next CRL
default_md      = sha1                    # which md to use.
preserve       = no                   # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy         = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = optional
emailAddress            = optional
```

```
####################################################################
[ req ]
default_bits           = 1024
default_keyfile        = privkey.pem
distinguished_name     = req_distinguished_name
attributes             = req_attributes
x509_extensions        = v3_ca # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or UTF8Strings
# so use this option with caution!
string_mask = nombstr

# req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName                    = Country Name (2 letter code)
countryName_default              = ES
countryName_min                       = 2
countryName_max                       = 2

stateOrProvinceName            = State or Province Name (full name)
stateOrProvinceName_default    = Catalunya

localityName                   = Locality Name (eg, city)
localityName_default           = Barcelona

0.organizationName             = Organization Name (eg, company)
0.organizationName_default     = Universitat Oberta de Catalunya

# we can do this but it is not needed normally :-)
#1.organizationName            = Second Organization Name (eg, company)
#1.organizationName_default    = World Wide Web Pty Ltd

organizationalUnitName         = Organizational Unit Name (eg, section)
organizationalUnitName_default         = Consultors

commonName                     = Common Name (eg, YOUR name)
commonName_max                 = 64

emailAddress                   = Email Address
emailAddress_max               = 40

# SET-ex3                      = SET extension number 3

[ req_attributes ]
challengePassword              = A challenge password
challengePassword_min          = 4
challengePassword_max          = 20

unstructuredName               = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.
  basicConstraints=CA:FALSE
# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                   = server
```

```
# For an object signing certificate this would be used.
# nsCertType = objsign
# For normal client use this is typical
nsCertType = client, email
# and for everything including object signing:
# nsCertType = client, email, objsign
# This is typical in keyUsage for a client certificate.
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
# This will be displayed in Netscape's comment listbox.
nsComment                       = "Seguretat en Xarxes de Computadors"
# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
subjectAltName=email:copy

# Copy subject details
issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]
# Extensions for a typical CA

# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign
# Some might want this also
# nsCertType = sslCA, emailCA
# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy
# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
```

Anexo C: El método LogManager.

A medida que el proyecto ha ido creciendo seguir el rastro de comentarios mostrados en la consola es un trabajo cada vez más complicado. He optado por crear un método específico para escribir todos estos comentarios en un fichero físico en el disco.

Los datos que se almacenan en cada escritura son:

- El momento en el tiempo.
- El método que escribe al log.
- El mensaje reportado.

Solo se utiliza el método para escribir.

En general todas las clases utilizan la clase Utils así que no aparece en los diagramas de clases para no complicar dicho diagrama.

Un ejemplo de una parte de los datos contenidos en el fichero de log podría ser el siguiente. Contiene el log de ejecución de la primera parte del esquema criptográfico.

```
Thu Nov 08 18:00:45 CET 2007  - Pruebas :            *
Thu Nov 08 18:00:45 CET 2007  - Pruebas :            ****************** INICIO PROTOCOLO 1 **********************
Thu Nov 08 18:00:45 CET 2007  - Pruebas :            *
Thu Nov 08 18:00:45 CET 2007  - Pruebas :            Médico pide servicio al el gestor. Primera fase identificación
Thu Nov 08 18:00:45 CET 2007  - Procedure1 :         Ni Metge =1830
Thu Nov 08 18:00:45 CET 2007  - Usuario :            cert-administrador-pck12.p12 pfc2007
Thu Nov 08 18:00:46 CET 2007  - P12Manager :         Certificado cert-administrador-pck12.p12 validado
Thu Nov 08 18:00:46 CET 2007  - P12Manager :         ID: cert-administrador-pck12 -/- cert-administrador-pck12
Thu Nov 08 18:00:46 CET 2007  - FirmarVerificar :    Signature OK from signer: dnQualifier=1234m,
EMAIL=fcancerginer@gmail.com, CN=Fran Cancer Giner, OU=Metges, O=UOC,L=BARCELONA,ST=BARCELONA,C=ES
Thu Nov 08 18:00:46 CET 2007  - Procedure2 :         Gestor Recupera Ni: 1830
Thu Nov 08 18:00:46 CET 2007  - Procedure2 :         Gestor IdUsuarioU : 1234m
Thu Nov 08 18:00:46 CET 2007  - Usuario :            cert-metge-pck12.p12 pfc2007
Thu Nov 08 18:00:46 CET 2007  - P12Manager :         Certificado cert-metge-pck12.p12 validado
Thu Nov 08 18:00:46 CET 2007  - P12Manager :         ID: cert-metge-pck12 -/- cert-metge-pck12
Thu Nov 08 18:00:46 CET 2007  - Procedure2 :         Ng Metge =2059
Thu Nov 08 18:00:46 CET 2007  - FirmarVerificar :    Signature OK from signer: dnQualifier=1234a,
EMAIL=fcancerginer@gmail.com, CN=Fran Cancer Giner, OU=Administradors, O=UOC
Thu Nov 08 18:00:46 CET 2007  - Médico :             Médico Recupera Ni': 1830
Thu Nov 08 18:00:46 CET 2007  - Médico :             Médico Recupera Ng': 2059
Thu Nov 08 18:00:46 CET 2007  - Médico :             Médico recupera IdUsuarioG : 1234a
```

Anexo D: Fichero de configuración de la base de datos.

Script de creación de las tablas.

```
/*
SQL
Host - localhost : Database – pfc2007
************************************************************
*/
create database if not exists pfc2007;
use pfc2007;

/*
Table structure for autenticados
*/
drop table if exists autenticados;

CREATE TABLE autenticados (
la text NOT NULL,
na text NOT NULL,
nb text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*
Table structure for visita
*/
drop table if exists visita;

CREATE TABLE visita (
idVisita int(11) NOT NULL auto_increment,
idHistoriaClinica int(11) NOT NULL,
X int(2),
Pg text,
T text,
Sg text,
Sgu text,
PRIMARY KEY (idVisita)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*
Table structure for gestor
*/
drop table if exists gestor;

CREATE TABLE gestor (
idGestor int(11) NOT NULL auto_increment,
FingerPrint text NOT NULL,
Certificate text NOT NULL,
SubjectDN text NOT NULL,
PRIMARY KEY (idGestor)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
/*
Table structure for persona
*/
drop table if exists persona;

CREATE TABLE persona (
idPersona int(11) NOT NULL auto_increment,
FingerPrint text NOT NULL,
Certificate text NOT NULL,
SubjectDN text NOT NULL,
PRIMARY KEY (idPersona)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


/*
Table structure for historia Clinica
*/
drop table if exists historiaClinica;

CREATE TABLE  historiaClinica  (
idHistoriaClinica int(11) NOT NULL auto_increment,
idPaciente text NOT NULL,
idMedico text NOT NULL,
Datos text NOT NULL,
Es text,
X text,
Pg text,
T text,
Sg text,
Sgu text,
PRIMARY KEY (idHistoriaClinica)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Este script se puede encontrar en los archivos anexados al proyecto. \doc\scripts

Script para la carga de datos auxiliares.

```
/*
creamos un registro para que se pueda encontrar la historia clínica del paciente de pruebas
 el idPaciente = fingerPrint del certificado del paciente.
 el idMedico = fingerPrint del certificado del médico.
 el campo datos tiene el nombre "Fran Cáncer" cifrado con la clave privada del administrador.
*/

insert into historiaclinica (idPaciente,idMedico, datos, es, x, pg, t, sg, sgu) values
('ydq6XuS5/VNfBMGgDF4Hnw==','aIc3cDw8jRWarRVq4fu0yg==','MIIDPwIBADGCAvswggE5AgEAMI
GhMIGTMQswCQYDVQQGEwJFUzESMBAGA1UECBMJQkFSQ0VMT05BMRIwEAYDVQQHEwlCQ
VJDRUxPTkExDDAKBgNVBAoTA1VPQzELMAkGA1UECxMCQ0ExGjAYBgNVBAMTEUZyYW4gQ2F
uY2VyIEdpbmVyMSUwIwYJKoZIhvcNAQkBFhZmY2FuY2VyZ2luZXJAZ21haWwuY29tAgkAqL7C8hE
o1oYwDQYJKoZIhvcNAQEBBQAEgYAqukqTe3QWwZjIigvYU8xn3HUnnjViUyOpSW6+qf1UyhpH7OK
BS5vTU/vXacGyvjd2zDoDO47xQm4vie5pVveSmK+4aWFSQVQsVCSfYa9F7bOI3vb14RWAGLjyBSB
6rxh34LAKW7QeAMtOtok+jJ6CpN8kkHw1DAZc4NW6uDL+RTCCAboCAQAwgaEwgZMxCzAJBgNV
BAYTAkVTMRIwEAYDVQQIEwlCQVJDRUxPTkExEjAQBgNVBAcTCUJBUkNFTE9OQTEMMAoGA1
UEChMDVU9DMQswCQYDVQQLEwJDQTEaMBgGA1UEAxMRRnJhbiBDYW5jZXIgR2luZXIxJTAjBg
kqhkiG9w0BCQEWFmZjYW5jZXJnaW5lckBnbWFpbC5jb20CCQDAajAinZ2OITANBgkqhkiG9w0BAQ
EFAASCAQAaR1dKFPKFCtFCK9ZIbQ+i9vTc+/gP1bKY1K+CM354HSCmnKJ5foh+NmeSwXHMbT7k
pOfxq98MxIrWf1PJ62Xfu1AknrfdhKQWrDMqP3IxtDLuXkah+mSGn1y1YmAorRKjdM9XSs8wue72cd6
Rz+Pmh05RQqDN3Q88JxbVNY0yII5EZC8s5q7QZ6RPDprBHdY5Banto6hsDo28eJzWP+SC66Z+qO
SUiNO4AtREnQDnm7iPTJGmSraZ31Ld/HdXlMwq8gI00DWkUAiYKfDh+wrC5Y/QjhO2hZqBRGnIrBL7
xKdRGVe67IkQJvgW3Qn/DYSigYtHTsnfXh060LsrDXWFMDsGCSqGSIb3DQEHATAUBggqhkiG9w0
DBwQIvTUIjWwQfdyAGJMPzOUfKh0bPiklLWuA1WXVXzyFDSoAdw==', 'es', 'x', 'pg', 't', 'sg','sgu');

/*
insertamos una visita al paciente cuya historia es "1" para tener el contador x inicializado
*/

insert into visita (idHistoriaclinica,   x,pg, t, sg,sgu) values (1,  1,'pg','t','sg','sgu')
```

Este script se puede encontrar en los archivos anexados al proyecto. \doc\scripts

Anexo E: Contenido de los archivos adjuntos a la memoria.

Juntamente con esta memoria se adjuntan unos archivos que contienen tanto el código de la aplicación como el código compilado, etc.

Carpeta	Contenido	Descripción
/pki	CA.crt Cert-metge-pck12.p12 Cert-pacient-pck12.p12 Cert-administrador-pck12.p12 Pacient-cert.crt Metge-cert.crt Administrador-cert.crt	Archivos PKCS12[20] con los certificados de los actores del sistema. Contiene a demás la estructura PKI para crear más usuarios. También se añaden los certificados.
/src	Todos los archivos .java que se utilizan en la aplicación CompilarTodo.bat arrancarServidor.bat host.txt Como Ejecutar la aplicación.txt	Código fuente del sistema .bat que compila la aplicación. .bat para arrancar el servidor RMI. Fichero con parámetros para la base de datos. Fichero de ayuda.
/bin	Todos los archivos .class arrancarServidor.bat host.txt Como Ejecutar la aplicación.txt	Código compilado y archivos necesarios para la configuración y ejecución. .bat para arrancar el servidor RMI. Fichero con parámetros para la base de datos. Fichero de ayuda.
/lib	/dll	Carpeta que contiene las librerías que se tienen que instalar según se explica en el capítulo Juego de Pruebas.
/doc	Pfc.mpp Pfc.ppt Pfc.doc	Archivo MSProject con la planificación del proyecto y esta documentación. Archivo PowerPoint con la presentación del proyecto. Archivo Word con este documento.
/doc/Scripts	crearBaseDatos.sql cargarBaseDatos.sql arrancarServidor.bat CompilarTodo.bat	Te permiten crear y cargar las bases de datos que usa la aplicación. .bat para arrancar el servidor RMI. .bat que compila la aplicación.
/doc/XML	XMLProtocolo1.xml XMLProtocolo2.xml XMLProtocolo3.xml	Estructuras XML vacías de los 3 protocolos. Se usan de manera interna por el programa. Las pongo a modo de ejemplo.
/doc/logs	logManager.txt	Ejemplo de un fichero logManager.txt después de ejecutar los 3 protocolos del médico.
/doc/javadoc	Index.html	Documentación de las clases. Javadoc.
/project	IDE eclipse	Todos los archivos del proyecto tal y como están utilizando eclipse.

Anexo F: Código del juego de pruebas del esquema criptográfico.

A continuación se adjuntan algunos fragmentos de código de la clase que implementa el juego de pruebas de la parte del esquema criptográfico.

Proceso 1:

```
private void protocolo1() throws FileNotFoundException,    IOException,    CertificateException {

        // El usuario realiza las siguientes operaciones:
        // ejecuta la procedure1 con su clave pública de Pu y optiene Pg[Ni,IdUsuario]
        System.out.println("Médico pide servicio al el gestor. Primera fase identificación");
        byte[] resultado1 = SVMedico.Procedure1Execute();
        // enviamos el resultado1 al Gestor G --->
        // G realiza las siguientes operaciones:
        // Ejecuta el procedure 2 con el resultado1 y optiene Pu[Ni,Ng,IdUsuario]
        byte[] resultado2 = SVGestor.Procedure2Execute(resultado1);
        // enviamos el resultado 2 a U --->
        // U realiza las siguiente operaciones:
        byte[] resultado3 = SVMedico.Proceso3Execute(resultado2);
```

```
public class Medico extends Usuario {
        // nombre del fichero del certificado del gestor
         private String _fileNameGestor;
         // certificado del gestor
         private X509Certificate[] _x509;
         // utils
         Utils utils;
        /**
    * Método que ejecuta el procedure 1 del protocolo criptográfico
         *
         * @return byte[] con el resultado esperado
         */
         public byte[] Procedure1Execute() {
                        Procedure1 procedure1 = new Procedure1();
                        procedure1.setIdUsuarioU(this.getIdUsuario());
                        procedure1.setCertificateGestor(this.getCertificateGestor());
                        procedure1.setCertificateFirma(this.getX509Certificate());
                        procedure1.setPrivateKeyFirma(this.getPrivateKey());
                        byte[] resultado1 = procedure1.execute();
                        return(resultado1);
         }
```

```
public class Procedure1 {
		// IdUsuario que se envía en la petición. puede ser el médico o el paciente.
		private String _idUsuarioU;
		// clave privada para firmar
		private PrivateKey _privateKey;
		// certificado para firmar
		private X509Certificate[] _CertificateFirma;
		// certificado del gestor
		private X509Certificate[] _CertificateGestor;
		// instancia de la librería de utilidades
		private Utils utils;
		// Número aleatorio que identifica al usuario que hace la petición
		private int _Ni;

		/**
		 *  Método execute(), ejecuta el funcionamiento del Procedure1
		 *  @return byte[] con Pg[Ni, IdUsuariU] encriptado y firmado por el usuario que hace la petición
		 */

		public byte[] execute(){

		  // Obtenim un valor de forma aleatoria Ni
		    SecureRandom Ieb = SecRandom.getDefault();
			int Ie = Ieb.nextInt();
			String _Ni = (Ie+"").substring(1,5);
			System.out.println("Ni Metge ="+ _Ni);

			FirmarVerificar firmarVerificar = new FirmarVerificar();
			byte[] data = utils.concatena(_Ni.getBytes(),_idUsuarioU.getBytes());

			PrivateKey[] privateKey = new PrivateKey[1];
			privateKey[0]=this._privateKey;
			firmarVerificar.setSignersPrivateKey(privateKey);

			X509Certificate[] x509T = _CertificateGestor;
			X509Certificate[] x509F = _CertificateFirma;
			// pongo el certificado que va a firmar
			firmarVerificar.setSignersCertificateChain(x509F);
			firmarVerificar.setReceiversCertificateChain(x509T);
			// encriptamos
			firmarVerificar.signEncrypt(data);

			// mensaje encriptado
			byte[] criptogram = firmarVerificar.getSignedEncryptedData();

			// paso el mensaje a base 64 para poder meterlo en un xml
			char[] x = B64Manager.encode(criptogram);
	byte[] y = utils.convertByte(x);
	return y;
	}
```

```
public class Procedure2 {
		// instancia de la librería de utilidades
		private Utils utils;
		// texto encriptado
		private byte[] _encryptedData;
		// certificado del gestor
		private X509Certificate[] _CertificateGestor;
		// privateKey del gestor
		private PrivateKey _privateKey;

		/**
		 *  Método execute(), ejecuta el funcionamiento del Procedure2
		 *  @return byte[] con Pu[Ni,Ng,IdUsuariG] encriptado y firmado por el médico
		 */
		public byte[] execute(){

		FirmarVerificar firmarVerificar = new FirmarVerificar();
		PrivateKey[] privateKey = new PrivateKey[1];
		privateKey[0]=_privateKey;
		firmarVerificar.setSignersPrivateKey(privateKey);
		   // antes de desencriptar lo convertimos porque está en base64.
		char[] y = utils.convertChar(_encryptedData);
		byte[] x = B64Manager.decode(y);
		firmarVerificar.verifyDecrypt(x);
		byte[] clearMessageByteB64 = firmarVerificar.getDecryptedData();
		byte[] Ni64 = utils.parte(clearMessageByteB64,0,clearMessageByteB64.length-5);
		char[] Ni64Char = utils.convertChar(Ni64);
	String Ni = new String(Ni64Char);
		System.out.println("Gestor Recupera Ni: "+Ni);
		byte[] idUsuarioUByteb64 = utils.parte(clearMessageByteB64,clearMessageByteB64.length-5,5);
		char[] idUsuarioUChar = utils.convertChar(idUsuarioUByteb64);
	String idUsuarioU = new String(idUsuarioUChar);
		System.out.println("Gestor IdUsuarioU : "+idUsuarioU);
		Medico SVMedico = new Medico();
		SVMedico.setFileP12("cert-metge-pck12.p12","pfc2007");

  // Obtenim un valor de forma aleatoria Ng
		SecureRandom Ieb = SecRandom.getDefault();
		int Ie = Ieb.nextInt();
		String _Ng = (Ie+"").substring(1,5);
		System.out.println("Ng Metge ="+_Ng);
		Name name = (Name)_CertificateGestor[0].getSubjectDN();
		String idUsuarioGestor = name.getRDN(ObjectID.dnQualifier);

		FirmarVerificar firmarVerificarEnvio = new FirmarVerificar();
		byte[] data =
			utils.concatena(utils.concatena((Ni+"").getBytes(),_Ng.getBytes()), idUsuarioGestor.getBytes());

		privateKey = new PrivateKey[1];
		// en este caso es la clave privada del gestor la que uso para firmar y ya he usado para descifrar.
		privateKey[0]=this._privateKey;
		firmarVerificarEnvio.setSignersPrivateKey(privateKey);

		// encripto con la pública del médico
		X509Certificate[] x509T = SVMedico.getX509Certificate();
		// firmo con el certificado del gestor
		X509Certificate[] x509F = _CertificateGestor;
		// pongo el certificado que va a firmar
		firmarVerificarEnvio.setSignersCertificateChain(x509F);
		firmarVerificarEnvio.setReceiversCertificateChain(x509T);
		// encriptamos
		firmarVerificarEnvio.signEncrypt(data);

		 // mensaje encriptado
		 byte[] criptogram = firmarVerificarEnvio.getSignedEncryptedData();

		 // paso el mensaje a base 64 para poder meterlo en un xml
	char[] x1 = B64Manager.encode(criptogram);
	byte[] y1 = utils.convertByte(x1);
	return y1;
```

Anexo G: Código del juego de pruebas de la representación en XML.

Se incluye a continuación una parte del código para ejemplificar parte del protocolo 1 y el uso de la clase XMLManager.

```
public String Procedure1Execute(String protocolo) {
                // creamos un documento XML con la petición de identificación
                XMLManager xmlManager = new XMLManager(null);
                xmlManager.setProtocolo(protocolo);
                _ficheroXML = xmlManager.createPeticion();

                Procedure1 procedure1 = new Procedure1();
                procedure1.setIdUsuarioU(this.getIdUsuario());
                procedure1.setCertificateGestor(this.getCertificateGestor());
                procedure1.setCertificateFirma(this.getX509Certificate());
                procedure1.setPrivateKeyFirma(this.getPrivateKey());
                _ficheroXML = procedure1.execute(_ficheroXML);
                _Ni=procedure1.getNi();
                return utils.serializeElement(_ficheroXML.getDocumentElement());     }
```

```
/**
 *  Método constructor
 *  implementa el funcionamiento de un Procedure1
 */
public  Procedure1(){
            utils = new Utils();
}

/**
 *  Método execute(), ejecuta el funcionamiento del Procedure1
 *  @param documento xml con la estructura a tratar
 *  @return byte[] con Pg[Ni, IdUsuariU] encriptado y firmado por el usuario que hace la petición
 */
public Document execute(Document ficheroXML){
            // cargamos el fichero XML
            _ficheroXML = ficheroXML;
            XMLManager xmlManager = new XMLManager(_ficheroXML);

            // Obtenim un valor de forma aleatoria Ni
            SecureRandom Ieb = SecRandom.getDefault();
            int Ie = Ieb.nextInt();
            _Ni = (Ie+"").substring(1,5);
            utils.logManager(" - Procedure1 :            " + "Ni Metge o Pacient ="+ _Ni);
            FirmarVerificar firmarVerificar = new FirmarVerificar();

            // creo los parámetros de la petición
            xmlManager.setValorElemento("AleatorioNi",_Ni.getBytes());
            xmlManager.setValorElementoB64("IdUsuarioU",_idUsuarioU);
            // serializo los parámetros para encriptarlos
            String ordenItemS = utils.serializeElement(xmlManager.getElement("OrdenIdentificacion"));

            byte[] data = ordenItemS.getBytes();
            // de este certificado se obtendrá la clave pública para encriptar los datos
            PrivateKey[] privateKey = new PrivateKey[1];
            privateKey[0]=this._privateKey;
            firmarVerificar.setSignersPrivateKey(privateKey);

            X509Certificate[] x509T = _CertificateGestor;
            X509Certificate[] x509F = _CertificateFirma;
            // pongo el certificado que va a firmar
            firmarVerificar.setSignersCertificateChain(x509F);
            firmarVerificar.setReceiversCertificateChain(x509T);
            // encriptamos
            firmarVerificar.signEncrypt(data);

            // mensaje encriptado
            byte[] criptogram = firmarVerificar.getSignedEncryptedData();
            xmlManager.setEncryptedData(criptogram, "OrdenIdentificacion", "OrdenIdentificacionEncriptado");
            return xmlManager.getDocument();
}
```

```
/**
 * Método que ejecuta el procedure 2 del protocolo criptográfico
* @param String fichero XML de entrada
* @return String fichero XML de salida
* @throws Exception
*/
public String Procedure2_Execute(String fichero) throws Exception {
            this.setFileP12("cert-administrador-pck12.p12","pfc2007");
            _ficheroXML = utils.readFileString(fichero);
            Procedure2 procedure2 = new Procedure2();
            procedure2.setCertificateGestor(this.getX509Certificate());
            procedure2.setPrivateKeyGestor(this.getPrivateKey());
            _ficheroXML = procedure2.execute(_ficheroXML);
            _Ng=procedure2.getNg();
             return utils.serializeElement(_ficheroXML.getDocumentElement()); }
```

```
/**
 *  Método constructor
 *  implementa el funcionamiento de un Procedure2
 */
public Procedure2(){
              utils = new Utils(); }
/**
 *  Método execute(), ejecuta el funcionamiento del Procedure2
 *  @param documento xml con la estructura a tratar
 *  @return byte[] con Pu[Ni,Ng,IdUsuariG] encriptado y firmado por el médico
 * @throws Exception
 */
public Document execute(Document ficheroXML) throws Exception{
             _ficheroXML = ficheroXML;
             // creamos un documento XML con la petición de identificación
            XMLManager xmlManager = new XMLManager(_ficheroXML);
            byte[] x = xmlManager.getValorElement("OrdenIdentificacionEncriptado");
            FirmarVerificar firmarVerificar = new FirmarVerificar();
            // uso la clave privada del gestor para desencriptar.
            PrivateKey[] privateKey = new PrivateKey[1];
            privateKey[0]=_privateKey;
            firmarVerificar.setSignersPrivateKey(privateKey);
           // paso el certificado del gestor
            firmarVerificar.verifyDecrypt(x);
            byte[] clearMessageByteB64 = firmarVerificar.getDecryptedData();
            char[] z1 = utils.convertChar(clearMessageByteB64);
            String t = new String(z1);
            // almacenamos el xml serializado en un xml temporal
            // cuando extraiga los datos se borrará y es lógico
            XMLTemporal xmlTemporal = new XMLTemporal(t);
            String Ni = xmlTemporal.getValorElemento("AleatorioNi","N");
            utils.logManager(" - Procedure2 :             " + "Gestor Recupera Ni: "+Ni);
            String idUsuarioU = xmlTemporal.getValorElementoB64("IdUsuarioU","N");
            utils.logManager(" - Procedure2 :             " + "Gestor IdUsuarioU : "+idUsuarioU);
            // liberamos el xml temporal
            xmlTemporal = null;
            // como tengo idUsuarioU puedo buscar en la base de datos su certificado y así su clave pública Pu
            BDManager bdManager = new BDManager();
            // abrimos la conexión con la base de datos
            bdManager.ConnectDataBase();
            // buscamos el certificado del usuario en la base de datos.
            bdManager.SqlExecutePersona(idUsuarioU);
            // certificado de la persona que solicita el servicio que tengo en la base de datos.
            X509Certificate x509 = new X509Certificate(bdManager.getCertificate());
            // Obtenim un valor de forma aleatoria Ng
            SecureRandom Ieb = SecRandom.getDefault();
            int Ie = Ieb.nextInt();
            _Ng = (Ie+"").substring(1,5);
            utils.logManager(" - Procedure2 :             " + "Ng Gestor ="+_Ng);
            // guardamos en la base de datos Ni y Ng y idUsuarioU, luego buscaré por Ng.
            byte[] fingerPrint = _CertificateGestor[0].getFingerprint();
            // String idUsuarioGestor = b64Manager.encode(,null);
            boolean res = bdManager.SqlExecuteInsertAutenticados(idUsuarioU, Ni, _Ng);
            if (res) {
              utils.logManager(" - Procedure2 :             " + "Error 8 - Error al grabar autenticados");
            } else {
                utils.logManager(" - Procedure2 :             " + "Registro grabado en autenticados correctamente");
            }
```

```
            bdManager = null;
            //Cifrador cifrador = new Cifrador();
            FirmarVerificar firmarVerificarEnvio = new FirmarVerificar();
            // creo los parámetros de la petición
            xmlManager.setValorElemento("AleatorioNip",Ni.getBytes());
            xmlManager.setValorElemento("AleatorioNg",_Ng.getBytes());
            xmlManager.setValorElemento("IdUsuarioG",fingerPrint);
            // serializo los parámetros para encriptarlos
            String ordenItemS = utils.serializeElement(xmlManager.getElement("OrdenValidacion"));
            byte[] data = ordenItemS.getBytes();
            // de este certificado se obtendrá la clave pública para encriptar los datos
            privateKey = new PrivateKey[1];
            // en este caso es la clave privada del gestor la que uso para firmar y ya he usado para descifrar.
            privateKey[0]=this._privateKey;
            firmarVerificarEnvio.setSignersPrivateKey(privateKey);
            // encripto con la pública del médico
            X509Certificate[] x509T = new X509Certificate[1];
            x509T[0]=x509;
            // firmo con el certificado del gestor
            X509Certificate[] x509F = _CertificateGestor;
            // pongo el certificado que va a firmar
            firmarVerificarEnvio.setSignersCertificateChain(x509F);
            firmarVerificarEnvio.setReceiversCertificateChain(x509T);
            // encriptamos
            firmarVerificarEnvio.signEncrypt(data);
            // mensaje encriptado
            byte[] criptogram = firmarVerificarEnvio.getSignedEncryptedData();
            // sustituyo los datos en claro por los encriptados
            xmlManager.setEncryptedData(criptogram, "OrdenValidacion", "OrdenValidacionEncriptado");
        return xmlManager.getDocument();
}
```

Se puede comprobar que los datos se almacenan directamente en la instancia de la clase XMLManager llamada xmlManager. Cabe destacar que los documentos XML[10] solo se escriben físicamente al disco al finalizar los distintos proceso por la clase usuario o por la clase gestor.

Anexo H: Mensajes de error de los aplicativos.

#1: Error 1 - Signature ERROR from signer:

#2: Error 2 - Error al borrar autenticados

#3: Error 3 - este paciente no pertenece a este médico

#4: Error 4 - el usuario no es médico

#5: Error 5 - el Ni no coincide

#6: Error 6 - Error al grabar la visita

#7: Error 7 - Certificado xxxx NO validado

#8: Error 8 - Error al grabar autenticados