Departamento de Tecnología Electrónica

PROYECTO FIN DE CARRERA

SISTEMA SEGURO DE GESTIÓN DE IDENTIDAD MEDIANTE TARJETAS INTELIGENTES

Autor: Eugenio Hernández Martínez

Tutor: Raúl Sánchez Reíllo

Director: Raúl Alonso Moreno

Leganés, septiembre de 2010

Título: Sistema seguro de gestión de identidad mediante tarjetas inteligentes

Autor: Eugenio Hernández Martínez

Director: Raúl Alonso Moreno

Tutor: Raúl Sánchez Reíllo

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día ___ de _____ de 20___ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

A mis padres
A mi hermana
A mis amigos

One card to rule them all, one card to find them,
One card to bring them all and in the darkness bind them.

Resumen

La necesidad de una mayor seguridad y privacidad se ve incrementada a medida que los sistemas de gestión de la identificación electrónicos van sustituyendo a métodos basados en papeles y presencia física. El auge de Internet y la expansión de las redes corporativas para permitir el acceso a los recursos desde fuera de un entorno seguro han aumentado la demanda de soluciones basadas en la tecnología de clave pública.

Algunos ejemplos del tipo de servicios que nos ofrece la tecnología de clave pública son: el establecimiento de canales de comunicación seguros a través de redes inseguras (Internet), la firma digital que asegura la integridad y confidencialidad de los datos, la autenticación de un cliente frente a un servidor y viceversa y el uso de tarjetas inteligentes para una autenticación segura.

Abstract

The need for greater privacy and security is increased as the systems of electronic identification are replacing paper-based methods and physical presence. The Internet boom and the expansion of corporate networks to allow the access to resources from an insecure environment have increased the demand for solutions based on public key technology.

Some examples of the services that the public key technology offers are: the establishment of secure communication channels over insecure networks (Internet), digital signature ensures the integrity and confidentiality of data, authentication between a client and a server and vice versa and the use of smart cards for secure authentication.

ÍNDICE

Índice de Figuras

Índice de Tablas

Listado de Acrónimos

AC	Autoridad de Certificación
ACV	Access Condition Value
AES	Advanced Encryption Standard
API	Application Programming Interface
AR	Autoridad de Registro
CLA	CLAss byte.
CPU	Central Processing Unit
CSP	Cryptographic Service Provider
DES	Data Encryption Standard
DF	Directory File
DNIe	Documento Nacional de Identidad Electrónico.
EEPROM	Electrically-Erasable Programmable Read-Only Memory
EF	Elementary File
FAR	False Accept Rate
FD	Firma Digital
FRR	False Rejection Rate
G&D	Giesecke & Devrient
INS	INStruction byte
IPF	Internal Public File
ISF	Internal Secret File
ISO	International Standards for Organizations
ITU-UIT	Unión Internacional de las Telecomunicaciones
MF	Master File
P1,P2,P3	Bits de parámetros en comandos.
PC	Personal Computer
PC/CTI	Personal Computer/Card Terminal Interface
PC/SC	Personal Computer/Smart Card
PIN	Personal Identifier Number
PKCS	Public Key Cryptography Standards
PKI	Public Key Infraestructure – Infraestructura de clave Pública.
PSC	Proveedor de Servicios de Certificación
RAM	Random Access Memory
ROM	Read Only Memory.
SDK	Software Development Kit
SM	Secure Messaging.
SO	Sistema Operativo
SOTI	Sistema Operativo de las Tarjetas Inteligentes

SSL	Secure Socket Layer
STARCOS SPK	SmarT cARd Chip Operating System/Standard with Public Key extension
TLV	Tag Length Value.
USB	Universal Serial Bus.

1. Introducción

Este documento recoge la información y la descripción del trabajo realizado para el proyecto de fin de carrera titulado "*Sistema seguro de gestión de identidad mediante tarjetas inteligentes*", el cual ha sido realizado por Eugenio Hernández Martínez, bajo la tutela del profesor Raúl Sánchez Reíllo y la dirección de Raúl Alonso Moreno. El proyecto ha sido desarrollado en el Grupo Universitario de Tecnologías de Identificación (GUTI) perteneciente al Departamento de Tecnología Electrónica de la Universidad Carlos III de Madrid.

1.1 Descripción del proyecto. Objetivos

El objetivo de este proyecto es el de proveer de un sistema de gestión de identidad mediante tarjetas inteligentes. El proyecto permitirá la generación de una firma digital y validación de la misma y así poder identificar de forma unívoca al remitente; establecer comunicaciones seguras dentro de la red gracias al cifrado asimétrico proporcionado por la tarjeta inteligente. La firma se realizará aprovechando una infraestructura de clave pública (PKI) utilizando para ello una TI.

Haciendo uso de los certificados, previamente almacenados en la tarjeta, y de su módulo criptográfico, se puede conseguir un sistema de firma capaz de proveer una identificación no repudiable del firmante.

Los únicos requisitos del sistema son, por lo tanto, que el usuario esté en posesión de una tarjeta con capacidades criptográficas (junto a un lector de tarjetas apropiado) y

disponer de la aplicación desarrollada en el proyecto. El receptor, por su parte, necesitará disponer del certificado digital del emisor para poder realizar la validación de la firma recibida.

La idea del proyecto es desarrollar una aplicación que se comunique, por medio del lector, con la tarjeta criptográfica y que se encargue de enviar y recibir los comandos precisos para la generación de firmas y el cifrado de los datos que se van a transmitir. Igualmente, será preciso crear, almacenar y transmitir los certificados correspondientes para la infraestructura PKI implementada.

1.2 Estructura de la Memoria

A continuación se expone una breve descripción de la estructura de la memoria y el contenido a tratar en cada capítulo:

La memoria se va a dividir en ocho bloques principales:

- En el primer bloque se realiza una breve introducción al proyecto y se marcan brevemente los objetivos mínimos que el sistema a desarrollar tiene que cumplir.

- En el segundo bloque se introducen los componentes principales en los cuales se sustenta el desarrollo del proyecto, como son las tarjetas inteligentes, la criptografía, los certificados digitales de usuario y la identificación biométrica.

- En el tercer punto se define el alcance del problema y los servicios que la aplicación debe proporcionar. Se analizarán los componentes que forman parte de la infraestructura desarrollada y los medios técnicos utilizados para poder llevar a cabo de manera satisfactoria el desarrollo del sistema.

- En el cuarto bloque se procede a presentar las pruebas y resultados obtenidos en la ejecución de los programas desarrollados. Se mostrará a su vez, de modo complementario, la capacidad de proceso del sistema.

- En el quinto bloque se mostrará un presupuesto estimado para la realización del proyecto.

- En el sexto bloque se exponen las conclusiones y resultados obtenidos, comparándolos con los objetivos fijados al inicio del desarrollo. También se introducirán posibles mejoras y líneas futuras del sistema.

- El séptimo bloque se compone de la bibliografía utilizada para desarrollar el sistema.

- El último bloque recogerá los anexos necesarios.

2. Estado Del Arte

En esta sección se ofrece una descripción de las tecnologías existentes estudiadas y analizadas para poder desarrollar este proyecto.

2.1 Tarjetas Inteligentes

2.1.1 Introducción

Una tarjeta inteligente (TI), es una tarjeta de plástico de dimensiones normalizadas, que contiene un microprocesador y una memoria (la cual nos permite guardar y procesar información en la tarjeta), cumpliendo el estándar ISO 7816 [31].

Aunque las TI pueden ser clasificadas en base a distintos parámetros, se realiza una clasificación de las TI en base de los componentes, la interfaz y finalmente el sistema operativo (SO) que soporta. Esta división se puede observar más fácilmente en la Figura 1.

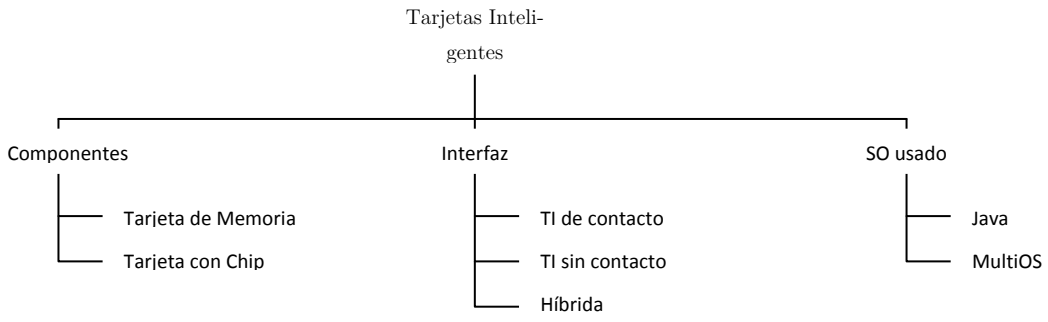

FIGURA 1. CLASIFICACIÓN TI.[1]

2.1.2 Bloques de una Tarjeta Inteligente

Fijándose en la primera clasificación de la Figura 1, la principal diferencia entre las TI y el resto de las tarjetas, es que el circuito integrado es un microcontrolador, es decir, un microprocesador el cual está normalmente asociado a una memoria y distintos periféricos. El esquema de los diversos bloques que componen una TI se puede ver a continuación:

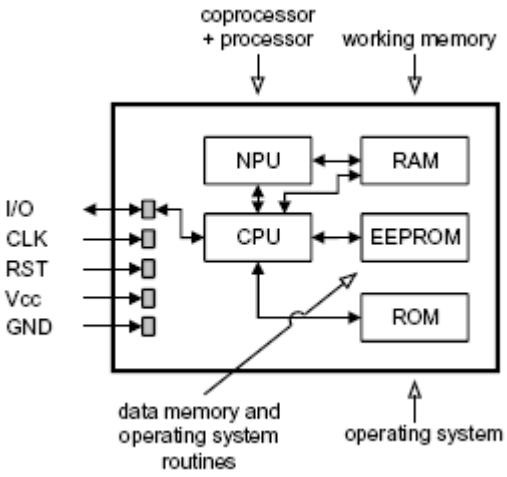

FIGURA 2. DISTRIBUCIÓN GENÉRICA DE UNA TI.[1]

UNIDAD CENTRAL DE PROCESO (CPU)

La CPU es el elemento principal de una tarjeta inteligente, ya que es el elemento que permite la integración con un SO para poder controlar las operaciones que se llevan a

cabo dentro de la tarjeta inteligente. La CPU, es un microprocesador de 8,16 o 32 bits, que se encarga de ejecutar las operaciones de bajo nivel tomando las decisiones necesarias en cada momento.

La CPU por sí sola no tiene mucha utilidad para el usuario final. Lo que realmente utiliza el usuario es el Sistema Operativo de Tarjeta Inteligente (SOTI) que está programado sobre la CPU y es el que toma las decisiones en ejecución.

MEMORIAS

Después del procesador, el componente más importante del microcontrolador son los distintos tipos de memorias. Pueden ser utilizadas para guardar el propio SO de la tarjeta (SOTI), los códigos de ejecución de los programas de la TI y los datos. En la Figura 2, se aprecia la división de la memoria en tres distintos tipos. La división final depende de la aplicación a la que la tarjeta está destinada.

- ROM: este tipo de memoria no puede ser escrito. No es necesario suministrar voltaje de forma continua para que los datos no se borren. La ROM de una TI contiene la mayoría de las rutinas del SOTI, así como distintos test y funciones de diagnóstico. Esas rutinas están grabadas por el fabricante. La memoria es transparente para todo aquel que no sea el programador del SOTI.

- RAM: es la memoria usada para mantener la información utilizada durante el manejo de la TI. El número de accesos a esta memoria es ilimitada, pero a diferencia de la ROM, es necesario alimentar la memoria para poder trabajar con ella. La memoria es transparente tanto para el usuario final de la tarjeta como para el programador de la aplicación.

- EEPROM: este tipo de memorias son usadas en las TI para todos aquellos programas o datos que pueden ser modificados o borrados en algún momento. Es similar al disco duro de un PC. Es la única memoria que va a ver el usuario final y el programador de la aplicación. Esta memoria, puede estar estructurada en ficheros y directorios, permitiendo al SOTI proteger diversas partes de la misma. El tamaño de la memoria nos va a limitar la cantidad de datos y aplicaciones que se quieran grabar en la TI.

Bloque de entrada y salida

El bloque de entrada y salida permite la comunicación de la tarjeta inteligente con el exterior. Para realizar este proceso, el sistema operativo de la tarjeta debe disponer de unas rutinas de control muy robustas, de modo que la comunicación se realice según los protocolos específicos de las tarjetas. Aunque nos encontramos con la limitación de la velocidad de transmisión de datos, impuesta principalmente por la velocidad del microprocesador de la tarjeta inteligente.

Coprocesador para Aplicaciones Criptográficas

Para los cálculos en el ámbito de las aplicaciones criptográficas, como puede ser la generación de claves RSA y su utilización en cifrado/descifrado o firma digital, se han desarrollado unidades especialmente diseñadas para este aspecto con los componentes habituales de un microcontrolador. Este tipo de unidades solo pueden llevar a cabo esta clase de algoritmos. La velocidad de estos componentes, optimizados para realizar operaciones aritméticas (exponenciación y cálculos modulares), se debe al diseño de su arquitectura. Esta unidad es llamada por el microprocesador, el cual suministra los datos, ya sea directamente o a través de un puntero a memoria y emite una instrucción para comenzar el proceso. Una vez finalizado y con el resultado almacenado en la RAM, el control vuelve al microprocesador. Estos procesadores suelen procesar longitudes de clave hasta 1024 bits para el algoritmo RSA, pudiendo llegar a longitudes 2048 bits.

2.1.3 Sistema Operativo de Tarjeta Inteligente

El SOTI provee un interfaz de alto nivel entre el hardware y el usuario de la tarjeta inteligente. Una vez cargado en la tarjeta, ésta se convierte en un dispositivo muy seguro, ya que no permite en ningún momento la ejecución de instrucciones de bajo nivel o que puedan poner en riesgo la seguridad.

- Bloquea la ejecución de instrucciones hasta que no se efectúe un reset. Cuando se reciba el reset, inicializa los parámetros de la tarjeta inteligente, ofreciendo una respuesta y se pondrá a la espera de comandos del exterior.

- Cuando la tarjeta reciba una instrucción, comprueba que ésta es válida siendo sus parámetros introducidos de manera correcta. Verificará el estado de seguridad en el que se encuentra la TI, para permitir el acceso a determinadas secciones de memoria protegidas. Si todas las verificaciones han sido correctas; el SOTI ejecuta la instrucción y devuelve la respuesta al usuario. Si en cambio, alguna verificación resultara fallida, se envía un mensaje el error al usuario.

- Gestión de memoria transparente, el usuario no ve direcciones de memoria, sino que ve una estructura a la que se encuentra familiarizado, con directorios y ficheros, similar a la ofrecida por un PC.

El SOTI hace que cada modelo de tarjeta inteligente sea distinto a los demás del mercado. Enmascara los bloques expuestos anteriormente al usuario. Por lo tanto, para el programador, la tarjeta inteligente viene caracterizada por:

- La forma de guardar los datos en la tarjeta
- Capacidad de la memoria EEPROM
- Arquitectura de Seguridad.
- Lista de Comandos que soporta la tarjeta.

El SOTI es grabado en la ROM en la fase de fabricación de la tarjeta y no puede ser ni actualizado ni cambiado, siendo por este motivo y por su deber de proteger datos confidenciales muy fiable y robusto.

Se puede hacer una subdivisión de los distintos SOTIs según su aplicación final:

- De propósito general
- Dedicados
- Abiertos

2.1.4 Sistema de ficheros

Las tarjetas actuales, incluyen auténticos sistemas de administración de ficheros que siguen una estructura jerárquica. El SOTI está orientado a trabajar con objetos, es decir, que todos los datos referentes a un fichero están contenidos en él mismo. Los ficheros están divididos en dos secciones distintas como se puede ver en la Figura 3, la primera es la cabecera que contiene datos referentes a estructura y condiciones de acceso y la segunda parte es la que contiene los datos útiles.

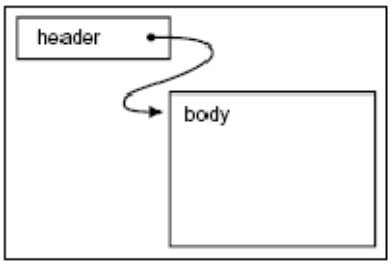

FIGURA 3. FORMATO DE FICHERO EN UNA TI[1]

La estructura interna de un sistema de ficheros se encuentra especificada en la ISO 7816/4 y tiene una estructura similar a un sistema en WINDOWS o UNIX. Se tiene un directorio raíz y a partir de él cuelgan los demás directorios y/o ficheros (Figura 4).

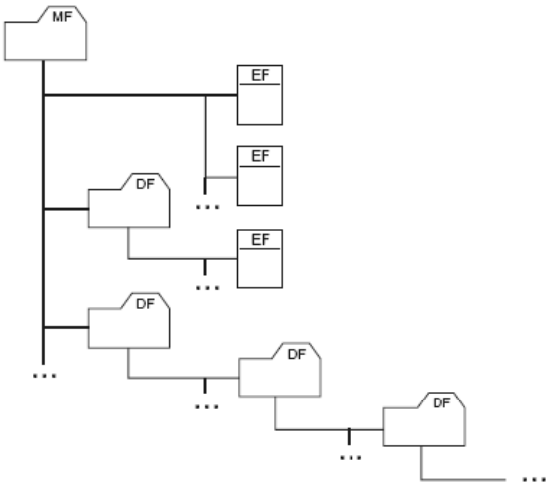

FIGURA 4. DIVERSOS TIPOS DE FICHEROS EN UNA TI.[1]

Se distinguen los siguientes tipos de estructuras en el sistema de ficheros de una tarjeta inteligente:

- Master File (MF): El directorio raíz se denomina "*master file*". Se selecciona automáticamente después de efectuar un reset de la tarjeta. Contiene todos los demás directorios o archivos. Siempre tiene que estar presente en una tarjeta funcional.

- Dedicated File (DF): El DF es un directorio donde otros archivos (DFs y EFs) pueden ser agrupados. En principio no hay limitación del número de niveles de DFs.

- Elementary File (EF): Los datos son guardados en los EF. Para permitir guardar la información con el mínimo gasto en memoria un EF siempre tiene una estructura de archivo interna. Ésta es la principal diferencia entre los EFs y los archivos en el PC, donde la estructura está determinada por la aplicación y no por el SO. Se puede clasificar los EFs en dos tipos, “EFs internos” donde se pueden almacenar claves privadas que no podrán ser leídas y “EFs de trabajo” que son los típicos archivos, que el usuario puede leer y modificar con la aplicación.

2.2 Criptografía

2.2.1 Introducción

Según el Diccionario de la Real Academia, la palabra Criptografía proviene del griego *κρύπτος*, que significa oculto, y, *γραφο* que significa escritura, y su definición es: *"Arte de escribir con clave secreta o de un modo enigmático"*. Obviamente la Criptografía hace años que dejó de ser un arte para convertirse en una técnica, o más bien un conglomerado de técnicas, que tratan sobre la protección -ocultamiento frente a observadores no autorizados- de la información [6].

Los principales problemas de seguridad que resuelve la criptografía son: la privacidad, la integridad, la autenticación y el no rechazo.

- La privacidad, se refiere a que la información sólo pueda ser leída por personas autorizadas.
- La integridad, garantiza que la información no pueda ser alterada en el transcurso del envío.
- La autenticidad, confirma que el mensaje recibido ha sido mandado por quien dice que lo mandó o que el mensaje recibido es el que se esperaba.
- El no rechazo, o no repudio, evita que el sujeto pueda negar la autoría de un mensaje enviado.

La criptografía se divide en dos grandes ramas, la criptografía de clave privada o simétrica y la criptografía de clave pública o asimétrica, podríamos destacar los algoritmos DES [33] o AES [13] como pertenecientes al primer grupo y RSA al segundo [14].

2.2.2 Criptografía de clave privada

La criptografía de clave privada o simétrica agrupa el conjunto de algoritmos que permiten establecer una comunicación entre dos partes, utilizando tanto para el cifrado como para el descifrado una clave idéntica [4].

En la criptografía simétrica, la clave debe ser conocida por ambas partes, de hecho, la clave es el secreto compartido que prueba la identidad de las partes. La mayor dificultad en este aspecto es la distribución de las claves de una manera segura a través de canales inseguros.

Se puede ofrecer una clasificación de este tipo de criptografía, la criptografía simétrica de bloques (*block ciphers*) y la criptografía simétrica de flujo (*stream cipher*). En la criptografía del tipo *stream cipher*, se opera con un único bit en cada instante de tiempo y se puede implementar un algoritmo para que la clave vaya cambiando dinámicamente. En cambio, en la criptografía del tipo *block cipher*, se cifra un tamaño de datos especificados por la longitud del bloque, utilizando la misma clave para todo el bloque. En general, si se utiliza *block cipher*, el mismo texto de entrada producirá el mismo texto cifrado, siempre que se utilice la misma clave en el proceso mientras que con *stream cipher* no siempre será así.

En la década de los 40, Shannon demostró para el tipo más sencillo de cifrado simétrico (una operación XOR) que el sistema es completamente fiable, siempre y cuando se utilice una clave de la misma longitud que el texto cifrado [35]. Esto aunque pueda parecer una ventaja, no es así, ya que el usuario debe afrontar el problema de intercambiar la clave de manera segura.

2.2.3 Criptografía de clave pública

Los algoritmos de clave pública, o algoritmos asimétricos, han demostrado su interés para ser empleados en redes de comunicación inseguras (p.e. Internet). Introducidos por Whitfield Diffie y Martin Hellman [4][6] a mediados de los años 70, su novedad fundamental con respecto a la criptografía simétrica es que las claves no son únicas, sino que forman pares. Hasta la fecha han aparecido multitud de algoritmos asimétricos, la mayoría de los cuales son inseguros; otros son poco prácticos, bien sea porque el criptograma es considerablemente mayor que el mensaje original, bien sea porque la longitud de la clave es enorme. Se basan en general en plantear al atacante problemas matemáticos difíciles de resolver. En la práctica muy pocos algoritmos son realmente útiles. El más popular por su sencillez es RSA, que ha sobrevivido a multitud de ataques, si bien necesita una longitud de clave considerable. Los algoritmos asimétricos emplean generalmente longitudes de clave mucho mayores que los simétricos

Los algoritmos asimétricos poseen dos claves diferentes en lugar de una, K_p y K_P, denominadas clave privada y clave pública. Una de ellas se emplea para cifrar, mientras que la otra se usa para descifrar. Dependiendo de la aplicación que le demos al algoritmo, la clave pública será la de cifrado o viceversa.

Una de las aplicaciones inmediatas de los algoritmos asimétricos es el cifrado de la información sin tener que transmitir la clave de descifrado, lo cual permite su uso en canales inseguros.

La segunda aplicación de los algoritmos asimétricos es la autenticación de mensajes, con ayuda de funciones resumen, que nos permiten obtener una firma digital a partir de un mensaje. Dicha firma es mucho más pequeña que el mensaje original, y se basa en la dificultad de encontrar otro mensaje que dé lugar a la misma. Se trata el tema de firma digital en el siguiente punto.

2.2.3.1 Introducción a RSA

De entre todos los algoritmos asimétricos, quizá RSA [14] sea el más sencillo de comprender e implementar. Como ya se ha dicho, sus claves sirven indistintamente tanto para cifrar como para autenticar. Debe su nombre a sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman, y estuvo bajo patente de los Laboratorios RSA hasta el 20 de septiembre de 2000, por lo que su uso comercial estuvo restringido hasta esa fecha.

Sujeto a múltiples controversias, desde su nacimiento nadie ha conseguido probar o rebatir su seguridad, pero se le tiene como uno de los algoritmos asimétricos más seguros. RSA se basa en la dificultad para factorizar números grandes. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos primos grandes. El atacante se enfrentará, si quiere recuperar un texto claro a partir del criptograma y la llave pública, a un problema de factorización [32].

Por lo tanto, si definimos una clave privada como el par de números $K_p \equiv \{d, n\}$ y una clave pública como $K_P \equiv \{e, n\}$, obtenemos las siguientes relaciones para el cifrado y descifrado de datos con ese par de claves:

El cifrado se lleva a cabo según la expresión

$$c = m^e mod\ n$$

Mientras que el descifrado se hará de la siguiente forma:

$$m = c^d mod\ n$$

Se puede indicar, que en los últimos meses (Marzo 2010), se ha logrado romper una implementación del algoritmo RSA determinada mediante la variación de los niveles de tensión enviados al destinatario para generar cifrados defectuoso. Esto ayudó a recrear

la clave privada mediante la combinación de una serie de fragmentos obtenidos en el proceso. Con este sistema se logró romper una clave de 1024 bits [15][16].

2.3 Certificado Digital

2.3.1 Introducción al Certificado X509

Un certificado digital, es un documento mediante el cual un tercero de confianza (una Autoridad de Certificación) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

Si bien existen varios formatos para certificados digitales, los más usados siguen el estándar X.509-UIT [17].

La primera versión apareció en 1988 y fue publicada como X.509v1, siendo la primera propuesta para una infraestructura de clave pública (PKI) a nivel mundial. Su publicación en ISO/ITU, han hecho del estándar X.509 el PKI más utilizado. En 1993 fue ampliado a la versión 2 [18] incluyendo únicamente dos nuevos campos, identificando de esta forma al emisor y usuario del certificado. La versión 3 [18] del certificado X.509 amplia la funcionalidad del estándar X.509 y es la usada actualmente.

El certificado contiene usualmente el nombre de la entidad certificada, un número de serie, fecha de expiración, una copia de la clave pública del titular del certificado (utilizada para la verificación de su firma digital) y la firma digital de la autoridad emisora del certificado, de forma que el receptor pueda verificar que esta última ha establecido realmente la asociación. La siguiente imagen muestra la estructura de un certificado digital del estándar X.509.

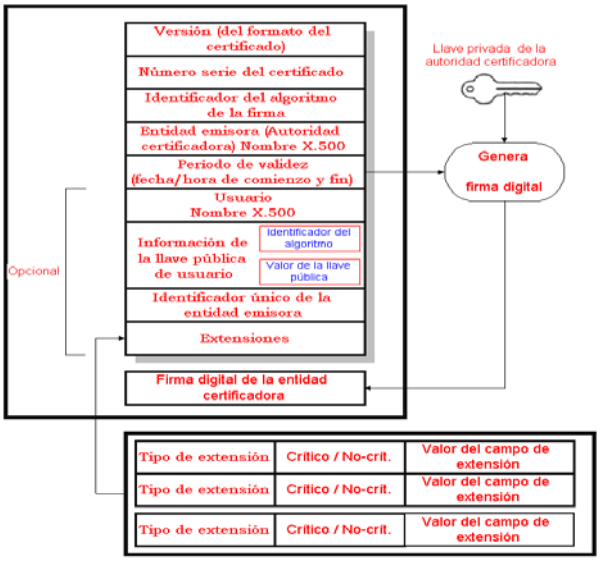

FIGURA 5. ESTRUCTURA CERTIFICADO X509V3

Existen distintos tipos de certificados digitales, en función de la información que contiene cada uno y a nombre de quien se emite [7]:

Certificado personal, identificando al sujeto.

Certificado de pertenencia a empresa, que además de la identidad del titular acredita su vinculación con la entidad para la que trabaja.

Certificado de representante, que además de la pertenencia a empresa ofrece también los poderes de representación que ostenta el titular.

Certificado de persona jurídica, identificando a una empresa o entidad como tal a la hora de realizar trámites ante las distintas administraciones o instituciones.

Certificado de firma de código, garantiza la identidad del autor y la integridad del contenido de una aplicación software y permite la distribución segura del mismo a través de internet.

Certificado de servidor seguro, permite establecer comunicaciones seguras y autenticadas entre servidor y cliente SSL. Además aseguran al usuario la autenticidad del sitio web al que se ha conectado.

Si bien cualquier individuo o institución puede generar un certificado digital, si el emisor no es reconocido como entidad de confianza por quienes interactúan con el propietario del certificado, se puede equiparar a un certificado sin firmar.

Por ello, los emisores de certificados digitales deben acreditarse para ser reconocidos por otras personas, comunidades, empresas o países como autoridad de certificación pública.

No obstante, cualquier empresa o entidad puede crear certificados digitales para uso interno, perfectamente válidos entre sus miembros. En función del ámbito de actuación pueden distinguirse entre PSC públicos y privados. La normativa española vigente distingue, a efectos de validez legal, entre certificados reconocidos y no reconocidos.

Asimismo se encomienda a la Fábrica Nacional de la Moneda y el Timbre (FNMT) la prestación de servicios de certificación que permitan la relación telemática entre ciudadanos y Administraciones Públicas y de Administraciones Públicas entre sí.

En lo que a seguridad respecta, en 2005 A. Lenstra y B. de Wegger demostraron cómo usar colisiones de hash para construir dos certificados X.509 basados en función hash MD5 que contienen firmas idénticas pero difieren en la clave pública. Dicho ataque se realizó basándose en un ataque de colisión en la función de hash MD5. Esta vulnerabi-

lidad es inherente al uso de la función hash MD5. Los certificados X.509 basados en la función hash SHA-1 se presuponen seguros.

2.3.2 Concepto de Firma Digital o Electrónica

La firma digital (FD) nos ayuda a identificarnos de manera segura y puede ser utilizada, tanto en el ámbito de relaciones entre los ciudadanos y la administración y entre los propios ciudadanos y las empresas. La firma digital cumple con los cuatro requisitos que soluciona la criptografía [2.2.1].

Consiste en aplicar cierto algoritmo matemático, denominado función hash o resumen, a una serie de datos y seguidamente cifrar ese resultado (hash) con la clave privada, generando así la firma digital.

Cuando se transmita esa información, el equipo receptor calcula el hash de los datos enviados y lo compara con el que resulte del descifrado de la firma digital recibida y si no coincidiera, se advertirá que el documento base ha sido modificado y que la firma no es correcta [Figura 6].

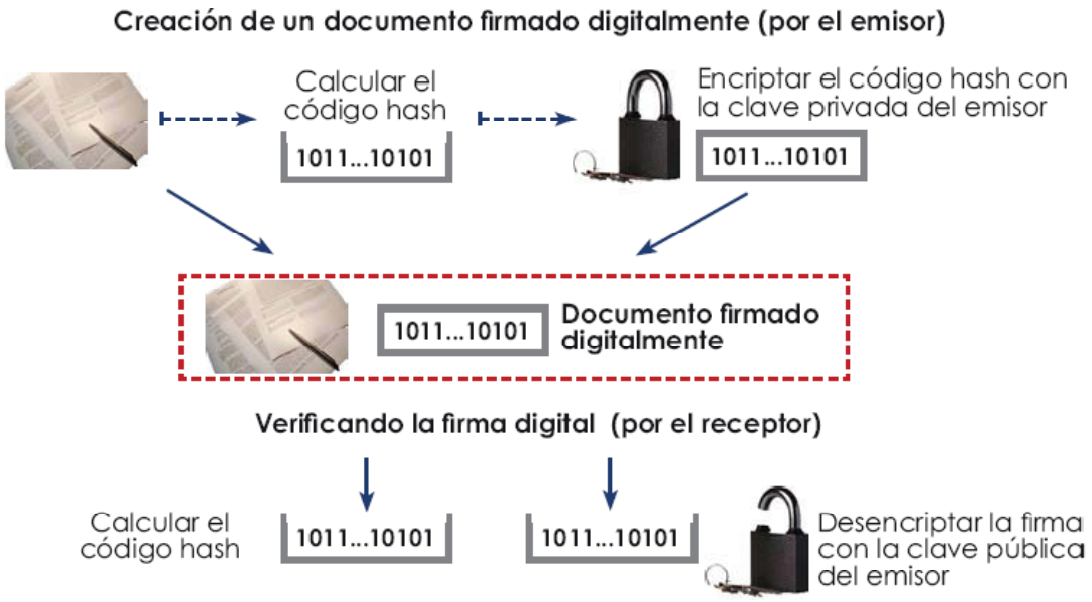

FIGURA 6. PROCESO DE CREACIÓN Y VERIFICACIÓN DE UNA FD.

Los mecanismos de FD permiten a la persona que recibe una información:

- Tener plena certeza de quién es el autor del mensaje/archivo.
- Verificar que el archivo no ha sido modificado desde su creación.

La firma digital que se utiliza actualmente cumple todos estos requisitos de forma mucho más precisa que la manual y es mucho más difícil de falsificar. Podemos identificar distintos tipos de firma digital según distintos criterios:

- Según la ubicación de la información firmada:
    - Explícita o "Detached": La estructura de la firma es independiente al documento firmado, es decir, obtenemos dos archivos distintos, uno con la firma y otro con el documento original.
    - Implícita o "Attached": La firma se incorpora al documento

- Según la jerarquía de la firma electrónica
    - Jerárquica: Cuando la estructura de la FD es secuencial, es decir, antes de firmar un usuario ha de ser firmado por otro previamente. Lo que se firma realmente es la firma anterior.
    - Paralela: Lo que importa realmente es un conjunto de n firmas independientes de un documento.

- Según la Ley 59/2003, de 19 de Diciembre, de Firma Electrónica (en adelante LFE):
    - Simple: Conjunto de datos en forma electrónica consignados junto a otros que pueden ser utilizados como medio de identificación del firmante.
    - Avanzadas: Aquellas que permiten identificar al firmante y detectar cualquier cambio de los datos. Vincula al firmante con los datos de manera única.
    - Reconocidas o Cualificada: Son las únicas válidas legalmente ante terceros. Firmas basadas en un certificado reconocido y generadas mediante un dispositivo seguro de creación de firma.
    - No reconocidas. No tienen validez legal, aunque se pueden considerar fiables.

- Según las normas europeas TS 101 733 [36]/ TS 101 903 [37] como se recogen en la siguiente figura:

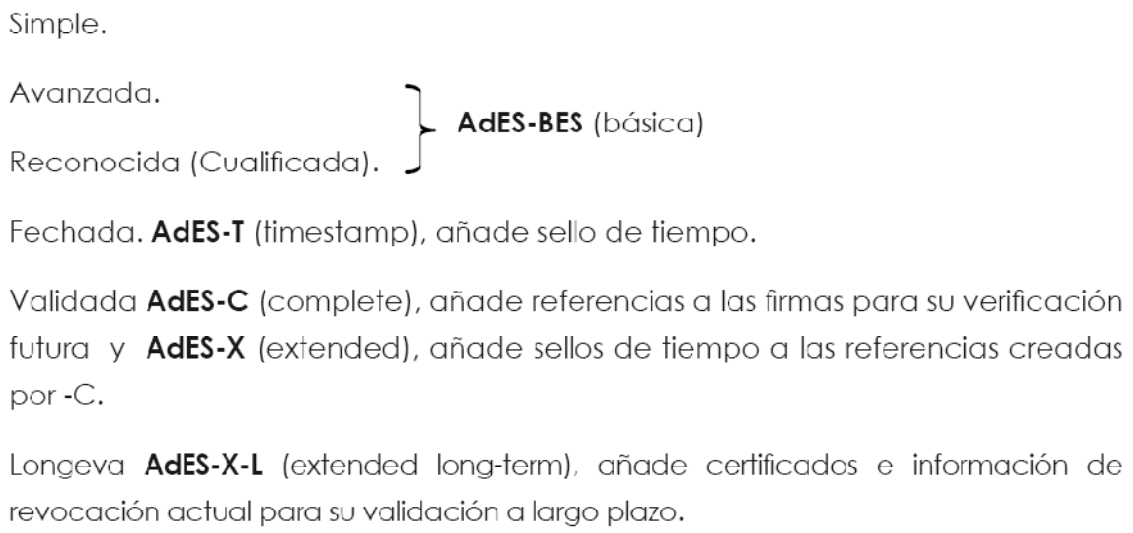

FIGURA 7. TIPOS DE FIRMA[15]

2.3.3 Infraestructuras de Clave Pública (ICP o PKI)

Se ha hecho referencia a que la criptografía asimétrica o de clave pública precisa de una Infraestructura de Clave Pública (ICP o PKI en inglés). Una Infraestructura de Clave Pública es un conjunto de protocolos, servicios y estándares que soportan aplicaciones basadas en criptografía de clave pública. También se puede definir como el conjunto de componentes y políticas necesarias para crear, gestionar y revocar certificados digitales que pueden ser utilizados para autenticar cualquier aplicación, persona, proceso u organización de una red de empresas, extranet o Internet. La base de estas infraestructuras es el modelo de confianza en Terceras Partes Confiables (TPC ó TTP, *Trusted Third Party*). La seguridad ofrecida por la firma digital es garantizada a través de la seguridad de la clave utilizada para su generación. Se presenta en la Figura 8, la relación de los componentes en una ICP.

Las Autoridades de Certificación (AC o CA por sus siglas en inglés) son las entidades encargadas de la generación de las claves y emisión de los certificados, su posterior gestión y revocación. Las Autoridades de Certificación pueden ser tanto públicas como privadas. Las públicas operan para usuarios y empresas a través de Internet.

Las CA emiten su propio certificado firmado por ellas mismas (self signed).Este certificado se denomina *certificado raíz*. La clave pública del certificado raíz se difunde de la misma forma que las claves públicas de los usuarios y es imprescindible para comprobar la validez de un certificado.

Dado que interesa que las claves públicas sean accesibles, es necesario que existan almacenes de certificados. Es muy frecuente el uso de servidores de certificados asociados a las propias Autoridades de Certificación que los emiten.

Cuando el tamaño de la PKI es grande surgen las Autoridades de Registro (AR o RA (*Registry Authority*)). En ellas delegan funciones las Autoridades de Certificación y por tanto, son intermediarias entre los usuarios y las CA.

En la criptografía asimétrica es de vital importancia la privacidad de la clave privada y las políticas de seguridad que se sigan para ello. Ni los dispositivos más seguros ni los algoritmos de cifrado más potentes sirven de nada si, por ejemplo, cualquier compañero puede utilizar libremente nuestra tarjeta criptográfica y por tanto cifrar, descifrar o firmar en nuestro nombre.

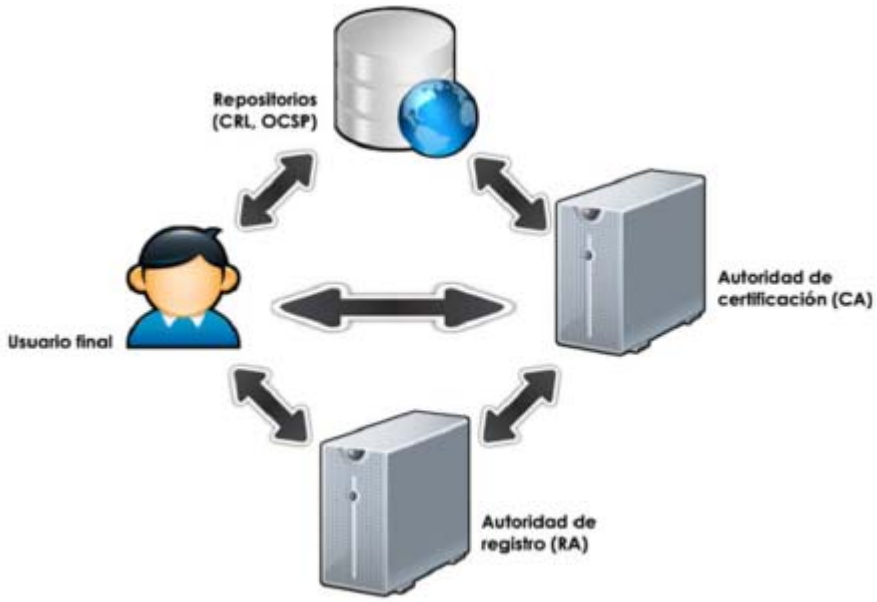

FIGURA 8. RELACIÓN DE LOS COMPONENTE DE UNA PKI

2.4 Introducción a la Biometría

La palabra biometría deriva del griego 'bios' (vida) y 'metron' (medida) y hace referencia a la medida de las características biológicas de un individuo. Según el diccionario de la lengua española el término biometría es el "estudio mensurativo o estadístico de los fenómenos o procesos biológicos", es decir, la aplicación de técnicas matemáticas y estadísticas sobre los rasgos físicos o de conducta de un individuo. Esta definición enmarca una disciplina que se inicia a finales del siglo XIX.

La biometría se puede definir en el campo de certificación, autenticación e identificación de personas, como la ciencia que estudia las características anatómicas o de comportamiento de una persona, con el objeto de que pueda ser reconocida. Las técnicas biométricas se utilizan para medir características corporales o de comportamiento de las personas con objeto de establecer una identidad.

2.4.1 Principios básicos

Los sistemas de seguridad pueden utilizar tres métodos de autenticación:

- Un secreto que conoce el usuario, como puede ser una contraseña, un número PIN, etc.
- Algo que posee el usuario, p.e un token electrónico.
- Una característica del usuario, un rasgo biométrico.

De los tres métodos, la biometría es el más seguro y conveniente. Una contraseña puede ser traspasada, una TI robada, pero un rasgo biométrico no.

Un método de identificación biométrica es aquel que puede identificar de forma inequívoca a una persona, por medio de características biológicas únicas. Aquí se puede hacer una distinción entre las características fisiológicas y de comportamiento [Figura 9]. El método de identificación comprueba una serie de características, si estas se relacionan directamente con el cuerpo de la persona y son totalmente independientes de los patrones de comportamiento consciente, se llaman elementos fisiológicos biométricos. Estas características no pueden ser modificadas por el usuario. Los métodos biométricos basados en características de comportamiento, por el contrario, utilizan ciertas características que pueden ser conscientemente cambiadas dentro de ciertos límites, pero que pueden seguir caracterizando a una persona en concreto. En general, este tipo de identificación permite

un control eficiente y preciso de la identidad de las personas eliminando, en gran parte, los riesgos de suplantación o robo de identidad. Y, como es de prever, sus usos y aplicaciones aumentarán progresivamente, pues existen posibilidades ilimitadas

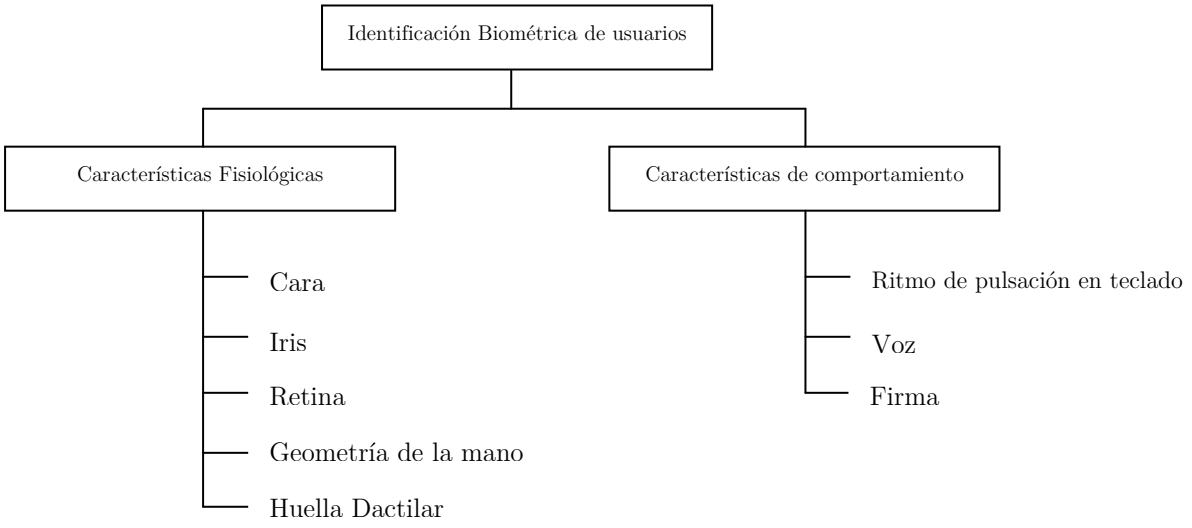

FIGURA 9. CLASIFICACIÓN DE LOS MÉTODOS MÁS IMPORTANTES DE IDENTIFICACIÓN BIOMÉTRICA

Para realizar una autenticación biométrica, primero se debe registrar al usuario en el sistema. Esto se lleva a cabo a través de un dispositivo biométrico que examina el atributo físico o de comportamiento elegido. A través de operaciones matemáticas y estadísticas, se extrae un patrón que representa inequívocamente al usuario.

La autentificación posterior, sucede cuando el usuario presenta el patrón previamente almacenado y lo compara con una nueva muestra capturada para tal efecto. La búsqueda del patrón guardado se puede efectuar de dos maneras:

- Búsqueda uno a muchos (1:N): donde partiendo del rasgo leído el sistema se encarga de efectuar una comparación con todos los que se tienen almacenados. (Identificación)
- Búsqueda uno a uno (1:1): donde el usuario presenta su rasgo y su identidad y el sistema se encarga de buscar su patrón almacenado y compararlo. (Verificación)

Para que la autentificación se lleve de modo correcto, no necesariamente ambos patrones deben coincidir. Como se comenta a continuación, no siempre se va a realizar la misma medida.

Naturalmente no todas las características biológicas son adecuadas para una identificación biométrica. Tienen que cumplir los siguientes requisitos para poder llevar a cabo el proceso:

- Que pueda ser medida eficazmente (en términos del método de medida, tiempo y coste)
- Que se pueda asociar de manera inequívoca a un usuario.
- Que no pueda ser alterada.
- El *patrón* generado debe ser pequeño (100~1000 bytes)

Como sucede al realizar cualquier proceso de medida, el resultado de cada "muestra" no es exactamente igual al anterior, sino que varía de muestra a muestra. Con un método ideal de medida y una cualidad biométrica también ideal, no existe variación en las medidas y la curva de dispersión se ve reducida a una línea vertical. Pero en realidad, esta curva se aproxima más a una curva *gaussiana* como se puede apreciar en la siguiente figura.

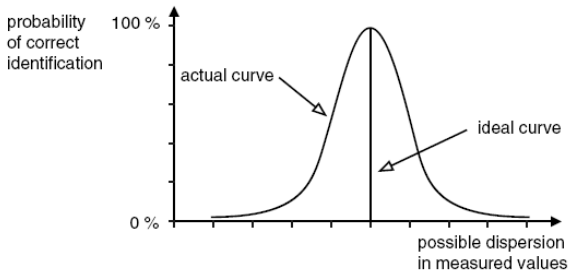

FIGURA 10. DISTRIBUCIÓN DE PROBABILIDAD PARA REPETIDAS MEDIDAS BIOMÉTRICAS

Si se parte de la anterior figura y se añade otra curva similar correspondiente a otro sujeto, se obtiene una representación como la mostrada en la Figura 11. La curva adicional representa una medida que está tan cerca de la original como para afectar en la decisión de identificación. En el punto de intersección existe la misma probabilidad de que una persona sea correctamente identificada o no. Por lo tanto en los sistemas biométricos se define un valor mínimo que debe superar la muestra para poder tomar como correcta la identificación. El valor de la Figura 11 nos muestra cuatro posibles regiones distintas.

Los parámetros básicos para evaluar un sistema biométrico son, la Tasa de Falsa Aceptación (FAR) y la Tasa de Falso Rechazo (FRR). La FAR es la probabilidad de permitir el acceso a personas no autorizadas, mientras que la FRR es la probabilidad de rechazo de personas autorizadas. Lógicamente estas dos probabilidades no pueden ser elegidas libremente, ya que están definidas por el método biométrico. Además, FAR y FRR están ligadas entre sí, una Tasa de Falsa Aceptación baja conlleva una Tasa de Falso Rechazo alta y viceversa.

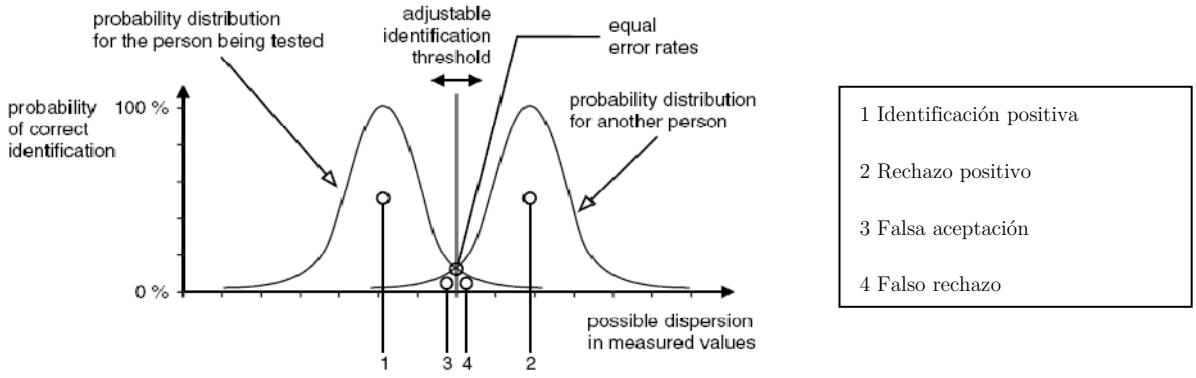

FIGURA 11. FUNCIONES DE PROBABILIDAD DE UNA IDENTIFICACIÓN BIOMÉTRICA

3. Sistema Implementado

En el presente capítulo se describe la arquitectura y diseño del sistema implementado, explicando el funcionamiento de los bloques que forman parte del mismo y argumentando las decisiones de diseño tomadas en el desarrollo global de la aplicación.

3.1 Viabilidad del sistema

En este capítulo se introduce el alcance que va a tener el proyecto, efectuando una comparación con tecnologías existentes similares e introduciendo las funciones globales que va a proporcionar.

3.1.1 Alcance y Descripción

Un Sistema de Identificación seguro, está diseñado para atender un requisito fundamental, que es el de verificar que un individuo realmente es quién dice ser. Todo sistema de gestión de identidad digital se basa en:

- Un dispositivo que contenga la identidad digital.
- Protocolos y mecanismos de autenticación y seguridad que las entidades deben poseer para demostrar que son los propietarios de una determinada identidad.

Cuando se plantea un sistema de gestión de la identidad de forma adecuada, este implementa una cadena de confianza, asegurando a todos y cada uno de los involucrados que el individuo que posee el token, la tarjeta en este caso, es el propietario de las credenciales que están en dicha tarjeta y que dichas credenciales son válidas (el término "credencial" está referido en este documento a la información almacenada en la tarjeta, como puede ser sus certificados digitales propios). Un sistema de identificación seguro, proporciona a los usuarios credenciales que son de entera confianza y que pueden ser usadas para una amplia gama de aplicaciones, desde permitir acceso a propiedades del sistema o redes, hasta proveer autorización para servicios o realizar transacciones en línea.

La TI es un factor crítico dentro del sistema, es utilizada como una representación portátil, confiable y verificable de la identidad del poseedor y de los derechos y privilegios que tiene dentro de la infraestructura creada.

Para que la tarjeta de identificación cumpla con estos requisitos, el sistema de identificación debe asegurar que una autoridad legítima fue la que emitió la tarjeta y que ésta y los datos que están contenidas en la misma, no pueden ser falsificados ni alterados y que la persona que utilice la tarjeta de identificación se corresponda inequívocamente con el usuario a la que fue expedida.

Gracias a la infraestructura de confianza que se crea al tener cada usuario su propia tarjeta que le identifica inequívocamente, se puede establecer una comunicación con esta persona por canales no seguros (p.e Internet) y reconocerle como firmante de un documento.

Ya existen distintas soluciones ampliamente reconocidas y aceptadas en la sociedad, similares a la que se plantea en esta memoria y que han podido servir de inspiración para crear un entorno similar a, por ejemplo, el DNIe, pero de una manera mucho más sencilla y usable, con el cual un usuario español puede, por ejemplo, autenticarse mientras navega por Internet, firmar documentos electrónicamente, etc.

La idea del proyecto es, por lo tanto, obtener un mecanismo de autenticación y firma similar al usado por el DNIe pero añadiéndole nuevas funcionalidades gracias a las posibilidades ofrecidas por las tarjetas criptográficas existentes en el mercado. Otra de las ventajas radica en la creación de un sistema abierto que no dependa de ninguna librería criptográfica como PKCS #11 o CSP, y por lo tanto mostrando mayor flexibilidad, fiabilidad y sencillez.

3.1.2 Componentes y Funciones

La solución desarrollada para la aplicación de identificación de usuarios se trata de una aplicación con una arquitectura basada en cliente-servidor. Una representación del sistema implementado se puede ver en la Figura 12. Como se puede observar el funcionamiento se subdivide en tres bloques independientes entre sí.

FIGURA 12. VISIÓN GLOBAL EJECUCIÓN SISTEMA

Explicación de los tres componentes:

- La tarjeta inteligente criptográfica es una tarjeta STARCOS SPK 2.4. Biometric de G&D con las siguientes características técnicas:

  - Microcontrolador Philips P82E5032.
  - Gestión de memoria:
    - 256 + 2048 bytes RAM
    - 32kbytes ROM (SO).
    - 32kbytes EEPROM (SO, aplicaciones y datos).
  - Formato de datos de 1,4 y 8 bits.
  - Frecuencia de reloj de 1 a 5 MHz
  - Tarjeta de contactos acorde a la norma ISO 7816-2.

Se introduce la TI en este bloque, aunque ésta puede asociarse al bloque hardware, debido a que sobre la TI recae el peso principal de ejecución, ofreciendo al usuario las siguientes funciones:

- Verificación de usuario en tarjeta (*Match-on-card)*
- Capacidad de guardar datos de manera segura en la TI.
- Patrón de huella de 2200 bytes
- 24 Kbyte de EEPROM libre para aplicaciones.
- Generación de clave RSA en la tarjeta.

  - Firma digital y verificación mediante clave RSA con una longitud de clave máxima 1024 bit siguiendo la norma ISO/IEC 7816-8
  - Compatible con PKCS#11.
  - Capacidad de ejecutar los siguientes algoritmos criptográficos DES, DES-3 y SHA-1.

- Comunicación con la tarjeta: este componente del sistema, es el encargado de transportar los comandos a la tarjeta inteligente, y devolver las respuestas de ésta. Para llevar a cabo el intercambio de información con la tarjeta inteligente se puede utilizar la librería compatible con la pila de software Personal Computer/Smart Card (PC/SC) [19].

  La funcionalidad PC/SC se ofrece a las aplicaciones por medio del API de Windows Smart Card (WinSCard) que se implementa en WinSCard.dll y, en menor grado, en scarddlg.dll. El API permite a las aplicaciones host comunicarse con tarjetas inteligentes a través del servicio administrador de recursos de la tarjeta inteligente, conocido por SCardSvr.

  Esta forma de comunicación con la tarjeta es la más básica, global y de bajo nivel que se puede afrontar, enviando los comandos a la tarjeta en hexadecimal y esperando la respuesta de esta.

  Como contrapunto a esta forma de comunicación con la tarjeta, puede utilizarse una librería propietaria de la tarjeta (si bien, como es el caso, suelen cumplir un estándar que los hace similares entre sí), la cual facilita la forma de enviar comandos, transformando ya la propia librería los métodos llamados a su equivalente comando en hexadecimal y permitiendo así al desarrollador una programación en alto nivel de la tarjeta inteligente.

- La aplicación final es una aplicación software, desarrollada con la ayuda de las librerías proporcionadas para la tarjeta inteligente utilizada. Proporciona un interfaz para administrar y utilizar los componentes de la tarjeta que posea el usuario.

3.2 Diseño del sistema

Se trata a continuación de ofrecer una ligera introducción al diseño final desarrollado, detallando los componentes que toman parte del mismo y las funciones básicas que deben proporcionar para cumplimentar el sistema de gestión de identidad. En este punto, se introduce la arquitectura del sistema desarrollado y se propone una arquitectura ampliada de funcionamiento, en la cual explotar al máximo el rendimiento final del sistema.

3.2.1 Arquitectura del sistema

Para la prueba de las funciones implementadas del Sistema de Gestión de la Identidad implementado, se ha diseñado un sencillo sistema correspondiente a la siguiente estructura:

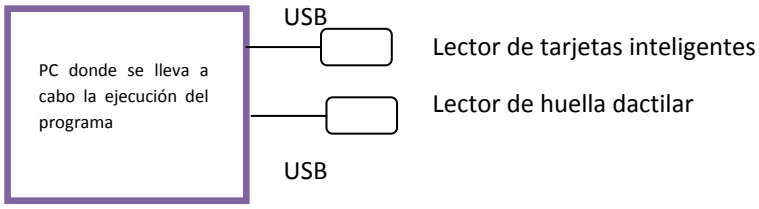

FIGURA 13. ARQUITECTURA DEL SISTEMA DE GESTION IMPLEMENTADO

En este sistema un único equipo actúa como autoridad de certificación, donde obtener los certificados de usuario, y como terminal de usuario, desde donde gestionar la tarjeta inteligente y los métodos implementados. En este equipo se encuentran instaladas las aplicaciones desarrolladas y deberá encontrarse disponible el paquete .NET 3.5 Framework o superior para el correcto funcionamiento del sistema. Se dispone a su vez, conectado a un puerto USB, de un lector de tarjetas inteligentes y de forma opcional el usuario puede disponer de un lector de huella para la identificación biométrica.

A continuación se propone un esquema más general de funcionamiento que el presentado anteriormente. La arquitectura del sistema es la siguiente:

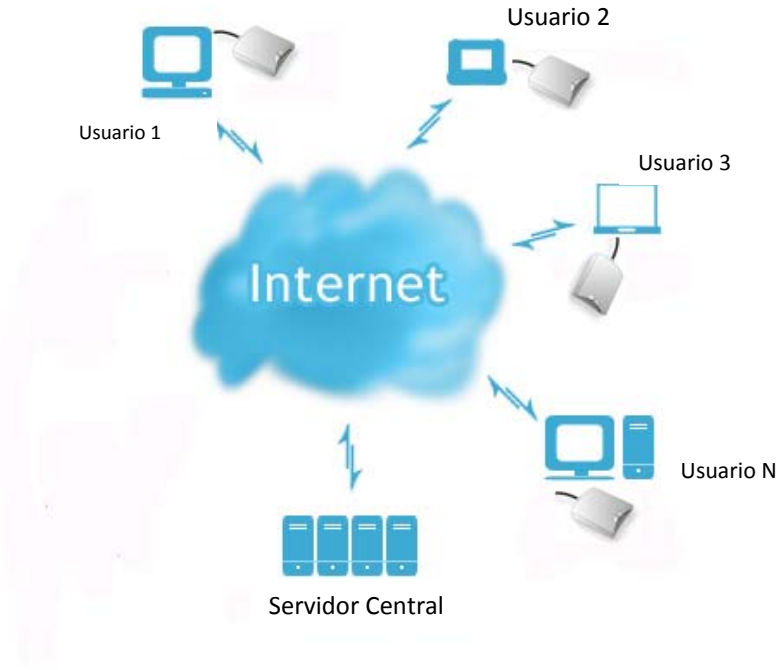

FIGURA 14. ARQUITECTURA GENERAL DE UN SISTEMA DE IDENTIFICACIÓN

El sistema anterior está compuesto de un servidor central, el cual sirve como CA, ejerciendo de tercero de confianza entre los usuarios y será el encargado de suministrar la TI al usuario debidamente inicializada. Como se observa en la Figura 14, el número de usuarios en el sistema es ilimitado. La aplicación general tendrá que ser diseñada para interconectar a todos los usuarios, ofreciendo además la posibilidad de efectuar una conexión en ese servidor central para el manejo de claves y certificados.

3.3 Implementación

En este punto se analiza la implementación que se ha llevado a cabo, justificando la elección del lenguaje de programación adecuado para la misma e indicando las decisiones de diseño que se han tomado y la razón de las mismas.

3.3.1 Equipamiento disponible

Para el desarrollo del presente Proyecto Fin de Carrera se ha contando con los siguientes componentes software y hardware:

- Tarjeta inteligente STARCOS Biometrics SPK 2.4 de tipo Test.

- Un lector de huellas Precise 100SC.

- Un lector de tarjetas inteligentes compatible con la norma ISO 7814.

- Toolkit Biométrico STARCOS, compuesto principalmente por:

    - STARMAG 3.0
    - STARTEST 4.2
    - Librería STARCOS(R) BASIC 4.0
    - Librería STARCOS(R) CRYPT 4.0
    - Librería Biométrica 1.0
    - PC/CTI 4.1
    - OCF Card Terminal API 1.2
    - OCF Card Services 2.0 for STARCOS(R) SPK 2.3 and SPK 2.4
    - Documentación

3.3.2 Lenguaje de programación y Entorno de Desarrollo

Para la elección de un lenguaje de programación se deben valorar los siguientes aspectos:

- Eficiencia y rendimiento: cantidad de recursos (principalmente tiempo de Procesador y espacio de memoria) que generalmente se requiere para ejecutar un programa en un lenguaje de programación.
- Fiabilidad: capacidad del lenguaje de prevenir que errores de propagación o por conversión de datos generen resultados
- Su robustez: capacidad del lenguaje de anticiparse a situaciones de conflicto por incompatibilidad entre tipos de datos y su capacidad para el manejo de excepciones.
- Su portabilidad: rango de arquitecturas y sistemas operativos sobre el cual el código puede ser compilado (o interpretado) y ejecutado.
- Su escalabilidad: como de fácil es incluir nueva funcionalidad o de aumentar la funcionalidad del código existente. Como se realiza la gestión de memoria, si se realiza de forma automática o es el programador quien la gestiona.
- Facilidad de desarrollo de programas.
- Facilidad de corregir errores.
- La adecuación el paradigma al que pertenece al problema que queremos resolver.
- La existencia de herramientas que ayuden al desarrollo.

A continuación se examina los lenguajes utilizados comúnmente en el desarrollo de software junto con sus ventajas e inconvenientes:

C

C es un lenguaje de gran calidad que sigue siendo muy usado frente al auge de los nuevos lenguajes orientados a objetos. Además de tener la ventaja de ser portable a casi cualquier arquitectura y sistema operativo conocido, posee una alta eficiencia y rendimiento frente a los lenguajes más nuevos. Entre las desventajas de usar C está en primer lugar que no tiene manejo de errores ni de excepciones, lo cual puede llevar a una propagación de errores que termine con la parada del programa; por otra parte, y es el mayor problema para los programadores, es que el manejo de memoria en C es manual, el pro-

gramador se encarga de reservar el espacio de memoria para las nuevas estructuras, y debe encargarse de liberarlos cuando ya no se utilicen o de lo contrario se producirán "fugas de memoria", es decir, fallos por falta de memoria libre en el sistema.

C++

Se trata de la evolución del lenguaje C. Es el mismo lenguaje pero con funcionalidad añadida: funciones virtuales, tipos genéricos, la posibilidad de declarar variables en cualquier punto del programa, el uso de clases y la herencia múltiple. Es un lenguaje portable a un gran número de arquitecturas y sistema operativos, un lenguaje eficiente que tiene un alto rendimiento. Además como ventaja frente a C, C++ posee herramientas para la captura y el manejo de excepciones y también se trata de un lenguaje fuertemente tipado lo cual nos ayudar a evitar propagación de errores por fallos en la conversión de tipos. Pero en C++ sigue existiendo el problema de las fugas de memoria ya que el manejo de memoria se lleva a cabo de forma manual.

Java

Java surgió a principios de los 90 como búsqueda de un lenguaje para dispositivos electrónicos (lavadoras, microondas, etc.). Java comparte muchas características con C++: declarar funciones en cualquier punto del programa, uso de clases y herencia, funciones virtuales, etc. Pero Java rompe con algunos elementos de los llamados característicos de C y C++ como son el uso de punteros y las variables globales. Además frente al manejo manual de memoria realizado en C y C++, Java incorpora un manejo automático por lo cual el programador ya no debe encargarse de reservar la memoria y liberarla cuando ya no la necesite. Otra novedad respecto a C y C++ es que los programas Java se ejecutan sobre una máquina virtual llamada JVM por lo que una vez se ha compilado el código java a objeto bytecode, no vuelve a compilarse aunque se cambie el objeto de arquitectura o sistema operativo, lo cual hace que la instalación de programas Java sea más rápida.

C#

C# es otro lenguaje que hereda de C (o mejor dicho C++), pero no extiende de C sino que se basa en él y también en Java. C# fusiona la capacidad de combinar operadores de C++ (sin soportar herencia múltiple) con la orientación a objetos de Java. C# es el lenguaje estrella del entorno .NET lanzado por Microsoft en el año 2000, dicho entorno actúa de máquina virtual, ya que el código en C# es compilado a código CLR (Common Language Runtime), y cuando se ejecuta el programa, la máquina virtual de .NET es la encargada de ejecutar el código CLR. Durante la ejecución del programa algunas partes son compiladas durante el momento de ejecución, y pese a poder dar la impresión de lo

contrario no se ralentiza la misma; sin embargo si es verdad que el entorno .NET consume bastantes recursos. C# es un lenguaje robusto, fuertemente tipado, con una gestión automática de memoria, con herramientas para el manejo y depuración de errores, y su orientación a objetos permite que los programas sean fácilmente escalables y fáciles de desarrollar.

Como punto de partida y analizando las librerías suministradas en el Toolkit Biométrico de la tarjeta inteligente [3.3.1], nos encontramos con una serie de inconvenientes derivados de la naturaleza de los SDKs a utilizar:

- Ambos SDKs, tanto el que nos permite comunicarnos con la tarjeta y acceder a sus funciones, como el que nos permite acceder a los métodos biométricos, están formados por bibliotecas. Incluyen, a su vez, un fichero de librería con extensión "*.lib*" y ficheros de encabezados con la recopilación de todas las estructuras de datos y definición de tipos a utilizar en la tarjeta.

- Las funciones de la biblioteca están programadas en lenguaje C y por lo tanto los parámetros de entrada están definidos en este lenguaje de programación.

Ahora bien, analizando las librerías suministradas por STARCOS, se puede encontrar en ellas programas de ejemplo sobre el uso de los anteriores SDK. En estos ejemplos nos encontramos unas aplicaciones desarrolladas en Visual C# y en C. También podemos encontrar, un API proporcionado por STARCOS para el manejo de la tarjeta por medio de Java, incluyendo una serie de archivos con formato "*.jar"* y "*.java*" que ofrecen total capacidad de desarrollo del sistema, pero sin ningún ejemplo ilustrativo para familiarizarse con el entorno.

En el programa de ejemplo de Visual C# se observa que para la utilización de servicios del sistema se utiliza el paquete .NET. Se trata de una aplicación de Windows Form programada en Visual C# con Microsoft Visual Studio. Después de probar su funcionamiento y familiarizarse con la aplicación se desarrolla la aplicación en este entorno y este lenguaje, ya que proporciona al desarrollador un sistema con múltiples posibilidades de desarrollo gracias a .NET.

.NET Framework se compone de cuatro partes como se muestra en la Figura 15; el entorno común de ejecución, un conjunto de bibliotecas de clases, un grupo de lenguajes de programación y el entorno ASP.NET. Logra aplicaciones Windows mucho más estables, simplificando el desarrollo de aplicaciones y servicios Web [21].

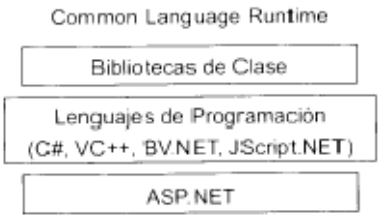

FIGURA 15. COMPONENTES DE .NET FRAMEWORK [21]

Las bibliotecas de clase .NET contienen código para programar hilos, entrada salida de archivos, compatibilidad para bases de datos, etc. Estando estas librerías disponibles para cualquier lenguaje de programación compatible con .NET. Se muestra a continuación un cuadro resumen de las bibliotecas de clase .NET Framework.

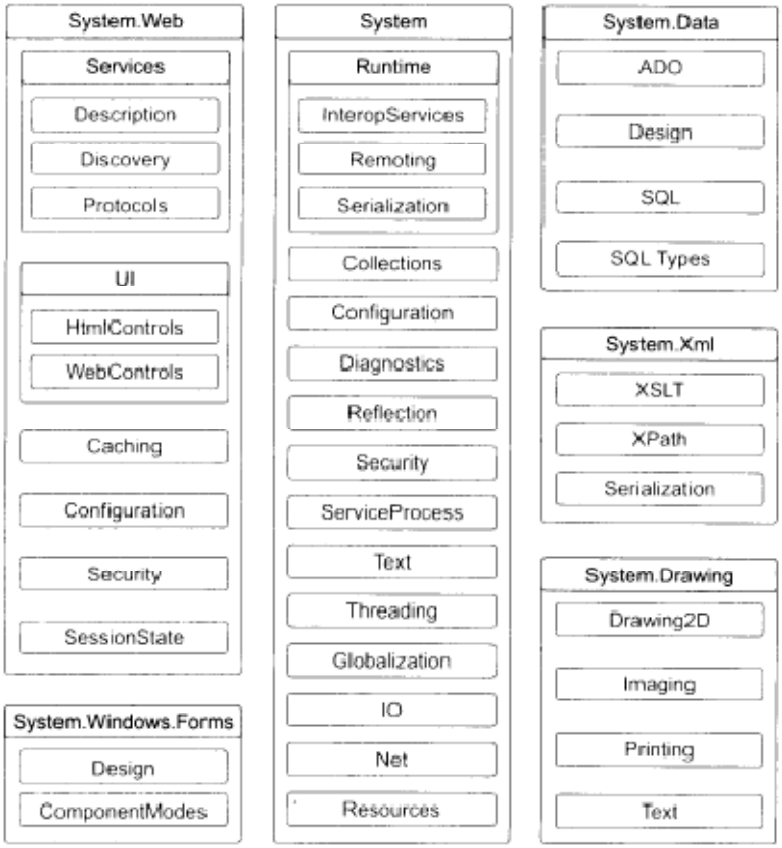

FIGURA 16. BIBLIOTECAS DE CLASE .NET FRAMEWORK [21]

3.3.3 Decisiones de implementación

En este punto se detallan las decisiones de implementación tomadas en el diseño, para ello se plantea las alternativas estudiadas, estableciendo sus puntos positivos y negativos, para finalmente escoger la más adecuada.

3.3.3.1. Tarjeta Criptográfica

La base principal sobre la que se sustenta el desarrollo del presente proyecto fin de carrera, es la tarjeta inteligente criptográfica. La TI escogida es una tarjeta de tipo STARCOS SPK 2.4 Biometric, la cual proporciona todas las características necesarias en la implementación del sistema, como por ejemplo, generación de claves RSA, autenticación, firma digital, hash, biometría, etc. [3.1.2].

Una vez fijada la tarjeta a programar, se puede comprobar que ésta posee la capacidad de soportar PKCS#11 mediante una librería proporcionada por STARCOS, la cual no se encuentra disponible en el SDK suministrado en el ToolKit Biométrico. Por lo que se va a implementar un sistema más sencillo y flexible, sin considerar dichos directorios de almacenamiento para certificados digitales.

3.3.3.2. Estructuración Tarjeta

El primer problema que debemos afrontar, una vez fijada la TI, es como organizar dicha tarjeta para poder guardar los certificados digitales de usuario de una manera segura y poder permitir su intercambio posterior para así poder utilizar las operaciones desarrolladas. Como se puede ver en el punto 3.1.2 no se dispone de una memoria ilimitada en la TI, por lo que el desarrollador debe tener en cuenta el tamaño de los archivos a guardar y la distribución de memoria asignada a cada directorio.

EL SOTI de la tarjeta nos ofrece un sistema de archivos que cumple con la normativa ISO 7814-4, como se mostraba en la Figura 4.

El Sistema Operativo grabado en la tarjeta STARCOS SPK 2.4, proporciona al desarrollador la posibilidad de utilizar otras estructuras de datos para el manejo de las propiedades criptográficas ofrecidas por la TI. Estos archivos se encuentran “colgados” de cada DF o MF creado. La nueva estructura se define con el nombre de ISF *(Internal Se-*

cret File). Este tipo de archivo, está protegido en la TI, el usuario no tendrá acceso ni a la lectura ni escritura de la estructura como en cualquier EF normal (p.e. mediante el comando UPDATE_BINARY). Los ISF son los encargados de almacenar las claves privadas, ya sean códigos PIN, claves RSA privadas, patrones de huellas, claves DES, etc.

La segunda clase de estructura ofrecida por el SOTI para el manejo de claves criptográficas y cuya función es complementaria al ISF se denomina IPF (Internal Public File). La existencia de esta estructura no es automática como en el caso del ISF, no es creada por el sistema operativo de la tarjeta al inicializar un DF o el MF. Esta estructura debe ser creada explícitamente por parte del desarrollador, estableciendo un número de identificación (FileID), como en cualquier otro tipo de archivo EF. El SOTI soporta la existencia de una única estructura de este formato en cada DF o MF. El IPF proporciona una estructura transparente, la cual puede ser seleccionada, leída y modificada mediante comandos (p.e. SELECT , READ_BINARY). El formato de un archivo IPF mantiene la siguiente distribución.

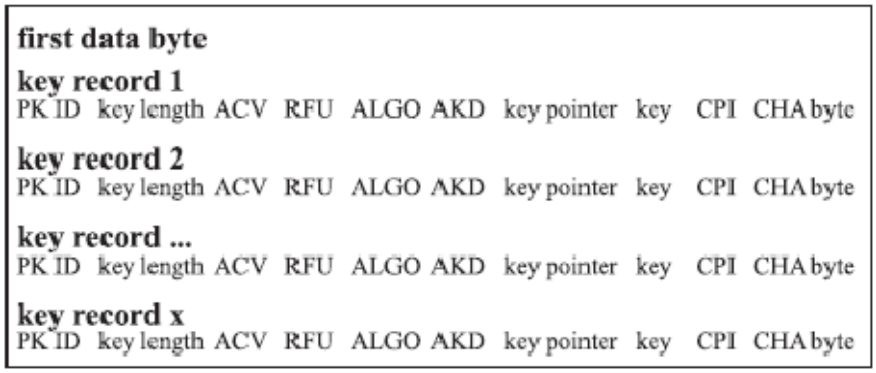

FIGURA 17. ESTRUCTURA IPF

El primer registro del archivo (*first data byte*), corresponde al número de claves almacenadas en el IPF. A este dato, le siguen las claves contenidas en el fichero. Para más información sobre los distintos campos que componen las claves se recomienda consultar el Manual de Usuario de la tarjeta.

Añadiendo estos archivos a la estructura presentada en la Figura 4, se obtiene el diseño general que presentan las tarjetas inteligentes STARCOS SPK 2.4 Biometric.

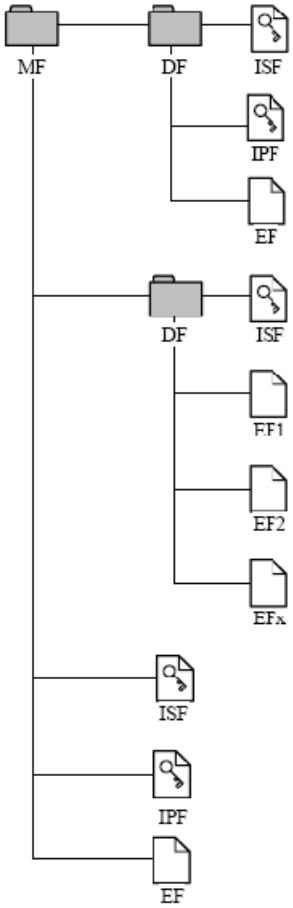

FIGURA 18. ARQUITECTURA TARJETA INTELIGENTE STARCOS SPK 2.4

Introducidas las dos estructuras encargadas de manejar las claves de la tarjeta criptográfica y conociendo la longitud del par de claves a almacenar en la tarjeta, se puede pasar a diseñar una distribución especifica de la tarjeta inteligente adecuada a las funciones que se van a desarrollar en el presente PFC.

A la hora de definir los componentes de la TI se tienen dos posibles vías; colocar las estructuras necesarias "colgando" en el propio MF, evitando así la necesidad de crear varios DF, ISF e IPFs o crear distintos DFs a modo de carpetas. La primera implementación aumenta la velocidad de acceso a la información contenida en la TI, ya que el usuario no tiene que seleccionar los distintos DF donde se encuentren los datos a los que desea acceder. Aunque la anterior ventaja resulta atractiva, el tiempo de ejecución ganado resulta despreciable frente al tiempo total que se va a tener en la aplicación en su ejecución (p.e. proceso de cifrado). Por lo tanto, se opta por estructurar la TI en distintas carpetas. La tarjeta inteligente está formada por tres distintos DFs que tomaran la función de "almacén de certificados". Dentro de estos tres almacenes, se puede distinguir dos tipos distintos; los almacenes de certificados de usuario y el almacén de certificados ajenos. La primera clase, está diseñada para almacenar un par de claves RSA de 1024 bits (longitud máxima de la clave permitida en la TI) y el certificado de usuario (en un archivo EF), ya

sea de firma o de autenticación. El almacén de certificados ajenos aunque disponga de un ISF, no será utilizado, ya que su misión es la de almacenar las claves públicas de otros usuarios del sistema (mediante el IPF creado para tal efecto), para poder así enviarles datos cifrados (recordemos que se cifra con la clave pública) o comprobar la validez de una firma recibida.

Como se ha indicado en el párrafo anterior, se ha desarrollado la estructura gracias al programa STARMAG, facilitándose así la comprobación de errores. Gracias al programa se puede definir también la cantidad de memoria de la EEPROM que se quiere reservar para cada estructura creada (DF, MF, EF, IPF, ISF). STARMAG permite añadir cualquier estructura en la tarjeta menos el MF (creado automáticamente como raíz) y el ISF (creado automáticamente al crear cualquier DF).

Al añadir un DF, se le asigna un identificador de archivo "File ID", el cual no puede estar duplicado en el mismo nivel de la tarjeta. Posteriormente se establece un tamaño determinado al DF, como se observa en la Figura 19. Esta Figura, representa el DF utilizado para almacenar el certificado de autenticación y su correspondiente par de claves RSA. Los certificados de usuario X.509 tienen un tamaño de ~1Kb y el par de claves RSA creadas con sus correspondientes cabeceras necesarias para la correcta utilización, tendrán una longitud en torno a 2 Kbits (Longitud clave >> Longitud cabeceras). En base a las dos premisas anteriores, se ha establecido un diseño de almacén de certificados con una memoria reservada de 4 Kbits (4096 b).

Como recoge la Figura 19, el diseñador tiene la capacidad de establecer la opción "Auto Resize", ajustándose así el tamaño del DF automáticamente a su contenido y efectuándose una distribución más compacta de la memoria (no existirá memoria reservada sin uso). Se ha comprobado en la ejecución, que en el proceso de escritura en la tarjeta de los datos del certificado X.509, ésta lanza un error diciendo que no hay suficiente espacio disponible en el directorio para la creación del archivo. Las demás pestañas, proporcionan al usuario la capacidad de definir las propiedades de seguridad en la creación de archivos. Este aspecto de la implementación es analizado en el punto 3.3.4.3.

El último tipo de almacén por analizar, es el denominado como "almacén para claves ajenas". Al disponer de una memoria limitada en la tarjeta inteligente, se ha decidido que no ofrece ninguna ventaja sustancial guardar los certificados ajenos en la tarjeta. Cuando el usuario tenga que realizar una operación criptográfica con la clave pública, la TI extraerá la clave pública del certificado correspondiente y se almacenará en la TI para futuros usos. Este almacén al disponer únicamente de claves públicas se ha diseñado para almacenar como máximo cinco usuarios distintos, teniendo así de esta manera una capacidad de 6 Kbits (5x1024).

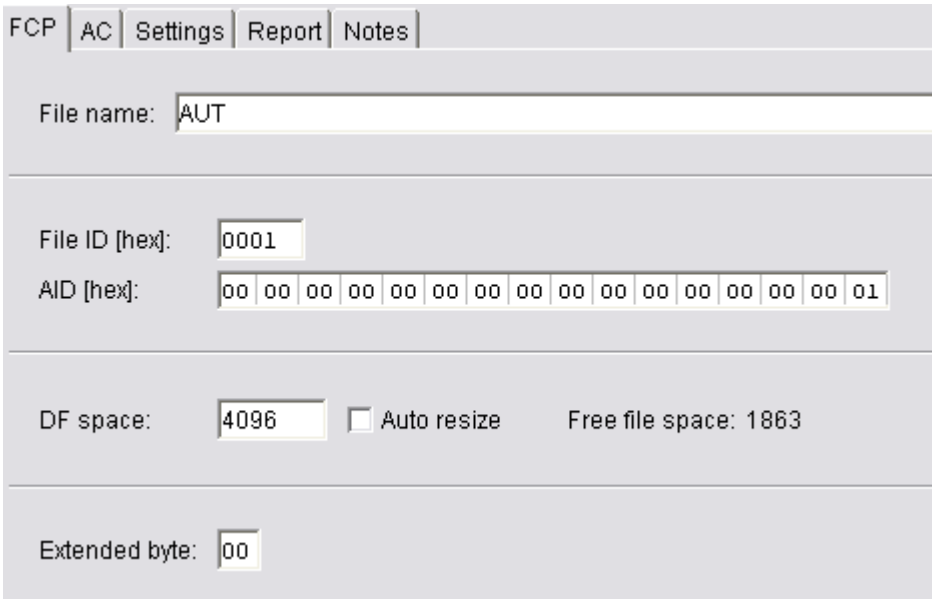

FIGURA 19. PANTALLA STARMAG PERSONALIZACIÓN DF

Agrupando los dos tipos de almacenes anteriores, se parte de la siguiente estructura en la tarjeta, donde cada carpeta contiene los distintos archivos necesarios (ISF, IPF) para satisfacer las necesidades criptográficas.

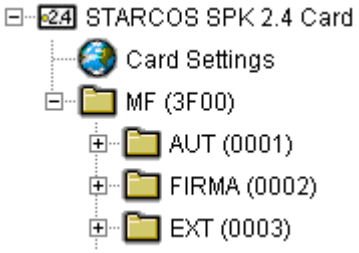

FIGURA 20. RESUMEN ESTRUCTURA TARJETA.

3.3.3.3. SEGURIDAD

Definida la estructura de la tarjeta, se procede a diseñar la arquitectura de seguridad que va a ofrecer la TI.

La tarjeta inteligente STARCOS SPK ofrece un control de seguridad de la TI usando una máquina de estados, permitiendo o prohibiendo diferentes acciones según el estado donde se encuentre. La tarjeta STARCOS proporciona dos tipos de estados diferentes; uno referenciando el MF y otro para cada DF, ofreciendo de esta manera hasta 16 estados de seguridad distintos para cada estructura.

FIGURA 21. TRANSICIÓN DE ESTADOS PARA UN COMANDO DE AUTENTICACIÓN

Para realizar un cambio de estado, se tiene que producir de manera correcta un comando que efectúe una operación de autenticación, como por ejemplo: *VERIFY PIN, VERIFY AND CHANGE, EXTERNAL AUTHENTICATE* o *MUTUAL AUTHENTICATE.*

La transición del estado inicial *(state)*, al estado siguiente *(consecutive state)*, se determina en la definición de la clave [Figura 23].

Existen, en todo momento, dos estados establecidos:

- Nivel de Seguridad Global (dependiente del MF)
- Nivel de Seguridad Local (dependiente del DF en particular)

Al seleccionar un DF, su estado pasa al de mínima seguridad (15).

Los datos guardados en los distintos DFs y EFs están protegidos por las condiciones de acceso impuestas. El usuario será capaz de definir distintas condiciones de acceso según el comando que se quiera utilizar (leer, actualizar, bloquear, etc.). En contraste a las claves almacenadas en la tarjeta, los EFs no necesitan tener un "estado siguiente", ya que si se está accediendo a ellos, no se debe cambiar el estado interno de la tarjeta a otro valor. Las propiedades de acceso al EF son establecidas en su generación. Para que el EF sea creado, se tiene que cumplir las condiciones de acceso impuestas en el DF respectivo.

Introducidas las propiedades de seguridad ofrecidos por la TI STARCOS, se pasa a detallar los elementos que se quieren proteger dentro del sistema.

La tarjeta va a contener un par de claves RSA únicas, pertenecientes al usuario, las cuales se debe proteger para que solo puedan ser utilizadas por el propietario de la TI. De esta manera, se define el primer estado en la tarjeta, donde el usuario accede después de una correcta identificación. En este estado podrá utilizar las propiedades criptográficas asociadas al conjunto de claves RSA (firma y autenticación).

El siguiente elemento a proteger dentro de la TI, son los certificados X509 correspondientes al usuario. Únicamente la "autoridad central" encargada de la creación de los certificados X.509, tiene que tener acceso a guardar los datos de manera segura en la tarjeta y que un tercero no pueda modificar esa información. De este modo se obtiene un nuevo estado de seguridad, en el cual se permite guardar la información contenida en los certificados en la TI. No se ha diseñado un nuevo estado en la tarjeta para permitir la lectura del documento. Como el propietario de la tarjeta es el encargado en la red de enviar su información a los demás usuarios, se asigna como estado de lectura, el estado de seguridad correspondiente a la utilización de las claves criptográficas. El usuario de esta manera, con una única autenticación es capaz de acceder a las propiedades criptográficas y a la lectura de sus certificados de usuario.

Resumiendo, se obtiene el siguiente diagrama de estados, que indica el número del estado en el que se encuentra la tarjeta y las capacidades que tendrá el usuario después de la correcta autenticación mediante el PIN_1 o PIN_2 y el estado al que se accede.

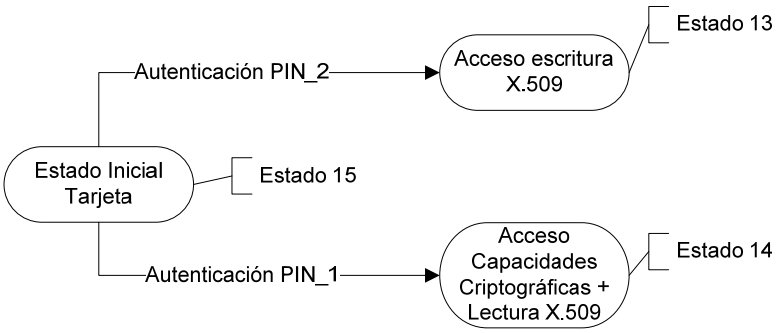

FIGURA 22. DIAGRAMA ESTADOS APLICACION

Para incorporar el sistema anterior de seguridad a la tarjeta, una vez elegido que el proceso de autenticación se efectuará mediante código PIN, fue la de incluir dicho código PIN en cada ISF de la TI. De esta manera, se referencia la seguridad al estado de cada DF. Con esta implementación, se está duplicando el PIN en cada directorio, aumen-

tando la memoria utilizada en la TI. Aunque esta implementación es completamente válida y funcional, a la hora de la ejecución es poco práctica, porque implica que el usuario debe de estar introduciendo su PIN cada vez que accede a un directorio, es decir, como sucede con el DNIe cada vez que se quiere utilizar el módulo criptográfico de la tarjeta inteligente, se solicita al usuario una autenticación (esto sucede por la propiedad de la tarjeta inteligente de llevar el estado al de mínima seguridad cada vez que se selecciona un DF).

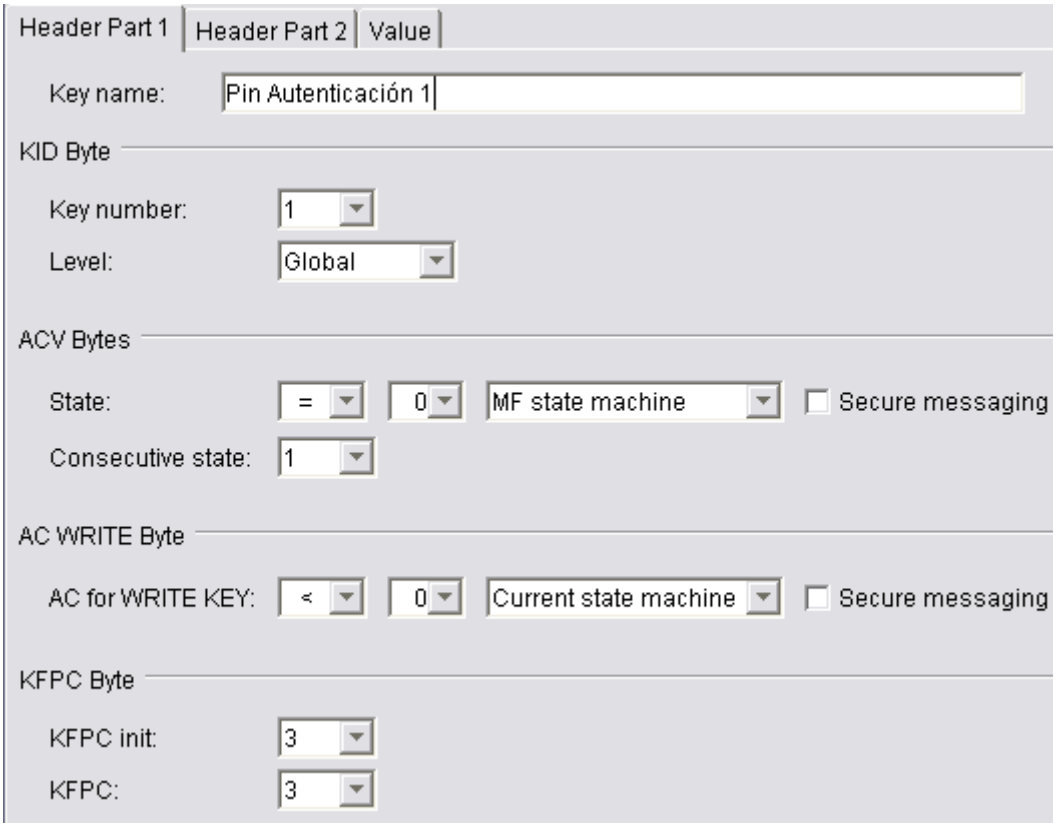

FIGURA 23. INSTALACIÓN DE CÓDIGO PIN EN UN ISF DE UNA TI STARCOS SPK 2.4

Para solventar el anterior problema, se ha incluido el PIN en el ISF del MF definiéndolo así como PIN global. De este modo, las definiciones de seguridad relacionadas con la transición de estados, tienen que hacer referencia al estado del *Master File (MF State Machine)*. Se muestra en la Figura 23 la implementación de un código PIN en la tarjeta gracias al programa STARMAG. Como se puede ver, se asigna un número de identificación único para su posterior elección y se define como global en el campo "*Level*". Para una posterior autenticación correcta, se debe estar en el estado inicial de la tarjeta. La seguridad se establece en el campo ACV (*Access Condition Value).* Como se indicó, el estado inicial (*state*) de la tarjeta es el 15, pero en el programa de desarrollo

está asociado al 0, lo cual nos induce a error y proporciona una tarjeta no acorde a la especificación necesaria. También se establece el estado consecutivo, en este caso el 1 (14), donde pasa la máquina de estados de la tarjeta después de la correcta introducción del PIN. Se inicializa en esta ventana el contador de intentos que tiene el usuario (*KPFC*) a tres. Las otras dos pestañas, se utilizan para inicializar el valor del PIN e indicar que el PIN no puede ser sobrescrito mediante el comando WRITE_KEY, pero si modificado VERYFY_AND_CHANGE.

La implementación del código PIN de la Figura 23, corresponde al código PIN que facilita el acceso a la lectura de los certificados X.509 y utilización de las propiedades criptográficas. El proceso de inicialización del segundo PIN, encargado de permitir la grabación de los datos del certificado X.509 en la TI, se lleva a cabo de igual manera, pero estableciendo como "*Consecutive State*" el estado 13 (o 2 según la nomenclatura de la TI).

3.3.3.4. Generación Claves RSA

Después de definir la longitud del par de claves RSA y los mecanismos de seguridad asociados, existen distintas vías posibles para la generación del par de claves. Las claves criptográficas van a ser el elemento principal de la tarjeta y su seguridad debe ser garantizada en todo momento. Partiendo de la premisa anterior, se analizan los siguientes escenarios de generación de claves.

- Generación de claves mediante las librerías .NET Framework usando la biblioteca criptography.

  Esta biblioteca genera un objeto, que contiene los elementos que componen una clave RSA. Una vez creado, se tiene que extraer las claves para posteriormente poder guardarlas en la tarjeta. Para este proceso se tendría que definir un nuevo estado de seguridad aumentando así la complejidad de la tarjeta y el número de códigos almacenados. Para guardar las claves en la tarjeta, tendríamos dos alternativas, por comandos[1], o utilizar las bibliotecas de la tarjeta[2]. Este proceso de generación de claves con una aplicación externa y posterior grabación en la TI, se puede llevar a cabo con otras herramientas como por ejemplo OpenSSL [22].

- Generación de las claves RSA dentro de la tarjeta. Esto es posible, gracias al SOTI de la TI. La generación se lleva a cabo a partir de la obtención interna

[1] página 87 de [10]
[2] páginas 250-257 de [9]

de números primos, de pasar test de primalidad, etc[3]. La generación de las claves RSA en la TI hace que los tiempos de inicialización sean variables. Con este método, se obtiene un par de claves privadas RSA con las propiedades de seguridad ya establecidas, que nunca van a salir de la tarjeta gracias a las propiedades de los ISF, lo que nos aporta una gran seguridad de la confidencialidad de las claves.

Basando la elección del método de generación en la seguridad y confidencialidad ofrecida por el procedimiento se efectúa la generación de la pareja de claves dentro de la tarjeta inteligente, guardando en la propia TI las respectivas claves pública y privada. En este paso, ya se establece el nivel de seguridad en el campo ACV [Figura 24] y las funciones que se le van a permitir a la clave [Figura 25]. Se asigna a la clave un identificador único, como sucedía en el caso del PIN. Se establece a su vez, la longitud deseada (1024 bits como tamaño máximo). Para cumplir con la condición de uso diseñada, el estado del MF tiene que ser superior o igual al 14 (o 1 según la notación de la Figura 24).

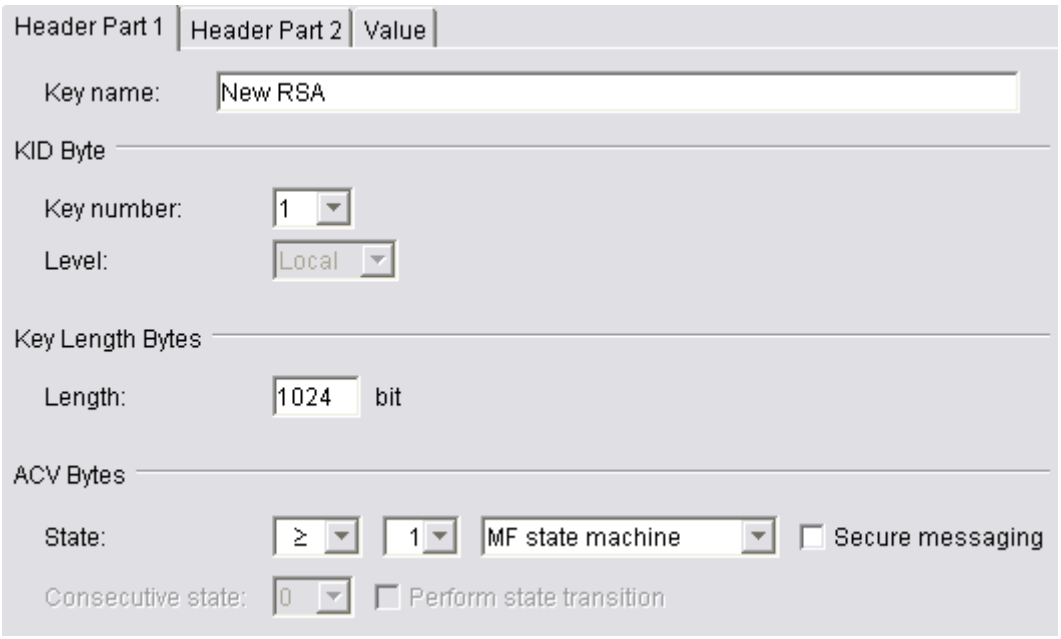

FIGURA 24. CONFIGURACIÓN CLAVE PRIVADA RSA.1

En la segunda parte de la configuración de la clave privada RSA, "*Header Part 2"*, se establecen los usos que el desarrollador quiere que ofrezca la clave RSA. Como se introdujo en el planteamiento del sistema, el usuario dispone de dos claves privadas, don-

[3] Consultar páginas 48-50 de [10]

de en una se tiene que activar la capacidad de descifrar (*Decipher* en la imagen) siendo ésta la clave de autenticación y en la segunda clave se tiene que habilitar la posibilidad de efectuar firmas digitales (*Compute Signature).*

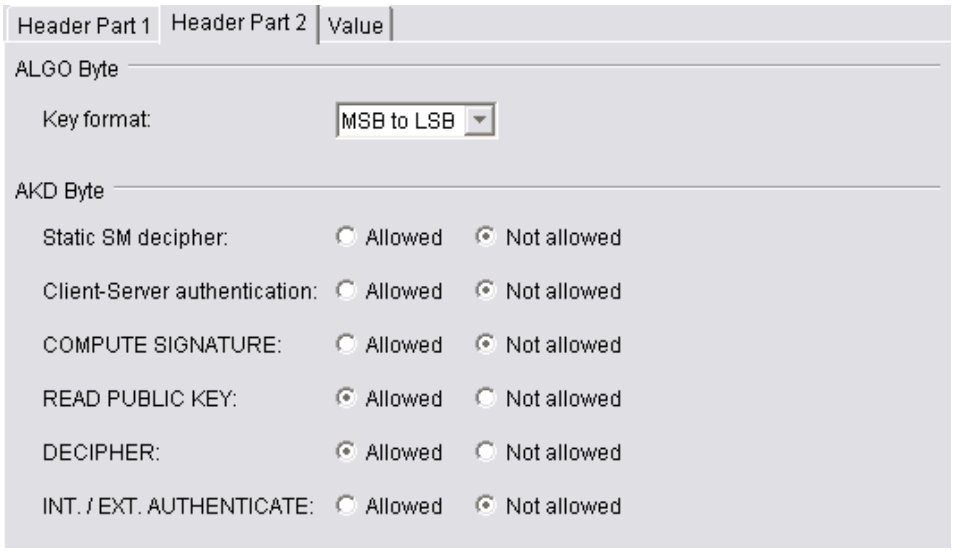

FIGURA 25. CONFIGURACIÓN CLAVE PRIVADA RSA.2

En la pestaña *Value*, se le indica a la tarjeta que debe generar las claves en el momento de la inicialización de la TI y se establece el formato de clave privada que se almacena en la tarjeta.

El proceso de configuración de la clave pública asociada es similar al anterior. El usuario dispondrá de un cuadro de diálogo igual a los de las Figuras 24 y 25, con las particularidades de las claves públicas: cifrado de texto, verificación de firma digital, etc. Para formar el par de claves en la inicialización y que el SOTI lo reconozca como tal, se debe asignar el mismo identificador, *key number,* a la clave contenida en el IPF y en el ISF. La clave pública, $K_P \equiv \{e, n\}$, tendrá como exponente e, siempre el valor $F4 = 2^{16} + 1 = '010001'$.

3.3.3.5. Generación y personalización X.509

Una vez obtenidas las claves RSA del sistema y guardadas en la TI, el siguiente elemento a diseñar es la generación de los certificados X.509 a utilizar en la PKI. Básicamente el certificado de usuario es una serie de datos, como nombre del propietario, periodo de validez junto con la clave pública, firmado todo esto por una CA. Para crear los certificados de usuario se analizan dos vías comerciales; utilizar la herramienta incluida en el paquete de Visual Studio, makecert.exe o utilizar el generador de certificados digitales que incluye en el paquete OpenSSL.

Makecert genera certificados X.509 sólo a efectos de pruebas. Crea un par de claves pública y privada para firmas digitales y las almacena en un archivo de certificado. Asocia el par de claves al nombre especificado de usuario y crea un certificado X.509 que enlaza el nombre de usuario con la parte pública del par de claves. Las opciones básicas que ofrece la herramienta, son las que se utilizan más frecuentemente para crear un certificado. El proceso de generación con Makecert es muy sencillo, llevándose a cabo mediante un comando[4]. El problema a la hora de utilizar Makecert es que no se puede indicar por parámetro una clave pública predefinida a incluir en el certificado.

Por lo anterior y aunque se puede utilizar Makecert para generar un par de claves RSA en un documento "*.pfx*" y utilizar el archivo para crear el certificado correspondiente, se desestima esta opción por la decisión tomada en el punto anterior (se generan las claves en la TI). Para más información sobre las opciones proporcionadas por Makecert y los distintos parámetros de configuración [24].

OpenSSL es un robusto paquete de herramientas de administración y librerías relacionadas con la criptografía, que suministra funciones criptográficas a otros paquetes como OpenSSH y navegadores web. Centrándonos en el funcionamiento de OpenSSL en el ámbito de la generación de certificados X.509, nos encontramos con los mismos problemas que con Makecert ya que no existe la posibilidad de incluir por parámetro la clave pública del usuario en la definición del archivo, por lo tanto no podremos utilizar esta aplicación a la hora de generar los certificados [25].

Al descartar las dos herramientas que ofrecen la posibilidad de crear certificados, nos lleva a desarrollar una aplicación capaz de generar un certificado X.509 partiendo de los datos de usuario y una clave pública. Para ello, se ha reutilizado una librería desarrollada en C# para la gestión de certificados [26]. Gracias a esta librería, se puede crear certificados de usuario más básicos que los que podemos encontrar en las herramientas

[4] makecert -sk XYZ -n "CN=XYZ Company" testXYZ.cer

anteriores, pero completamente útiles para el sistema ya que contiene la clave pública de autenticación o firma, el nombre de usuario, el nombre de la autoridad que firma el documento y un periodo de validez. Se puede añadir opciones complementarias a las anteriores, pero han sido suprimidas por no aportar una mejora sustancial en el funcionamiento del sistema. En el punto 4.2 se muestra la aplicación que genera y controla los certificados X.509.

El certificado creado, seguirá una estructura del siguiente tipo (RFC 3280):

```
/*
 * Certificate  ::=  SEQUENCE  {
 *      tbsCertificate       TBSCertificate,
 *      signatureAlgorithm   AlgorithmIdentifier,
 *      signature            BIT STRING
 * }
 * TBSCertificate  ::=  SEQUENCE  {
 *      version         [0]  Version DEFAULT v1,
 *      serialNumber         CertificateSerialNumber,
 *      signature            AlgorithmIdentifier,
 *      issuer               Name,
 *      validity             Validity,
 *      subject              Name,
 *      subjectPublicKeyInfo SubjectPublicKeyInfo,
 *      issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
 *                           -- If present, version MUST be v2 or v3
 *      subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
 *                           -- If present, version MUST be v2 or v3
 *      extensions      [3]  Extensions OPTIONAL
 *                           -- If present, version MUST be v3 --
 * }
 * Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
 * CertificateSerialNumber  ::=  INTEGER
 * Validity ::= SEQUENCE {
 *      notBefore      Time,
 *      notAfter       Time
 * }
 * Time ::= CHOICE {
 *      utcTime        UTCTime,
 *      generalTime    GeneralizedTime
 * }
 */
```

Una vez creado el documento, se graba en la tarjeta, para ello, como se introdujo en el punto de seguridad se establecen unas restricciones de acceso al archivo en su inicialización.

3.3.3.6. Comunicación con la TI STARCOS SPK 2.4

La comunicación con la tarjeta inteligente se puede afrontar de dos maneras, como se vio en el punto 3.1.2. Mediante comandos recogidos en el manual de la tarjeta STARCOS S2.1 y STARCOS S2.4 [10] o mediante la utilización de los métodos ofrecidos por el SDK facilitado por STARCOS [9].

La primera opción, consiste en la creación por parte del desarrollador del comando en hexadecimal y utilizar únicamente una función de envío a la TI. Estos comandos están formados por un mínimo de 5 bytes (cabecera), la cual está seguida de los datos a enviar (cuerpo) si el comando así lo requiere.

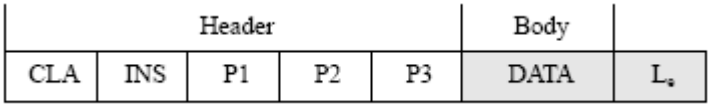

FIGURA 26. FORMATO DE ENVÍO DE COMANDOS [9]

El primer byte, CLA (Class byte), diferencia entre comandos pertenecientes a la ISO (CLA=00) o comandos propietarios del SO de la tarjeta inteligente.

El byte de instrucción (INS), contiene la codificación del comando. Como estándar, se utiliza la codificación ISO, si el comando no estuviera definido en la ISO, se utilizara un código de instrucción recogidas en ETSI/CEN.

Los bytes P1/P2, son bytes que tendrán la función de parámetro para el propio comando

El byte P3, depende del propio comando, de acuerdo con la ISO 7816-4.

La respuesta de la tarjeta consiste como mínimo en 2 bytes de estado y un campo de datos adicional. La respuesta típica de una correcta ejecución del comando es SW = '9000'.

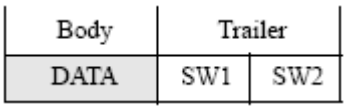

FIGURA 27. FORMATO DE COMANDO ENVIADO POR LA TI [9]

Esta opción conlleva un código menos depurado y más farragoso que la programación mediante comandos de alto nivel, ya que hay que indicar para cada comando su correspondencia en hexadecimal, además de forzar a utilizar un SDK extra para enviar datos, como puede ser el SDK de la propia TI o el SDK PC/SC de Windows [3.1.2]. Por lo tanto y como se justificó en el punto de elección del lenguaje de programación, se van a utilizar las librerías proporcionadas por el fabricante.

Se presentan los archivos que componen las librerías que se van a utilizar, junto con una descripción de su utilidad. Las librerías consisten básicamente en una serie de bibliotecas que contienen la mayoría de las funciones que se pueden solicitar a la tarjeta inteligente. Estas librerías deben estar situadas en el mismo directorio donde se ejecute la aplicación o en su defecto en C:\Windows\System32. El SDK STARCOS se adapta automáticamente a los distintos tipos de sistemas operativos de las tarjetas inteligentes y terminales, por lo tanto el desarrollador puede programar sus aplicaciones, para que puedan ser integradas en distintos SO.

Las bibliotecas que componen el STARCOS SDK 4.0 son:

- Stm4xw32.dll: Controla las funciones de comunicación con la tarjeta inteligente en su función básica. En la función básica y criptográfica permite a su vez la utilización de las funciones criptográficas.

- Stm4xw32.lib: Se trata de una biblioteca de importación que referencia a Stm4xw32.dll. Para su utilización se tiene que añadir el archivo al correspondiente proyecto para su correcta compilación.

- Pcctiw32.dll: Biblioteca dinámica que ofrece funciones PC/CTI al desarrollador permitiendo programar un número amplio de tarjetas inteligentes independientemente de su sistema operativo [27].

- Stm_api.h: En este archivo de encabezado se define una serie de constantes numéricas y estructuras que son utilizadas en los métodos que se encuentran en la librería Stm4xw32.dll.

3.3.3.6.1. Descripción de las funciones contenidas en el SDK v4.00

A continuación, se describen las diferentes funciones incluidas en el manual del programador del SDK de STARCOS. Para una información más precisa de los parámetros que utilizan cada función se recomienda consultar el Manual de la Librería [9]. Conjuntamente con las funciones, se indican los comandos que de la tarjeta inteligente que llama cada función. Todas estas funciones han sido encapsuladas en código C# en el desarrollo de la aplicación. Para llevar a cabo esta tarea, se han tenido que rediseñar los distintos parámetros de entrada y salida de las funciones, evitando de esta manera la utilización de punteros. La utilización de la librería ofrecida por STARCOS no es trivial, ya que ésta no puede ser cargada en el proyecto como un objeto COM, es decir, el desarrollador no puede crear objetos de la librería y ejecutar los métodos proporcionados mediante el objeto instanciado. Para solucionar este inconveniente, se ha desarrollado la aplicación para que se carguen los métodos en la propia ejecución, utilizando para ello el comando *ImportDLL*. La clase desarrollada, encargada del manejo de la librería, proporciona al proyecto una interfaz donde el usuario es capaz de crear objetos que referencien a la TI y de esta manera ser capaz de invocar los distintos métodos para utilizar sus propiedades.

Las funciones contenidas en el manual se pueden clasificar en dos tipos principalmente:

- Funciones básicas. Funciones para la transmisión de datos entre la aplicación y el terminal o tarjeta inteligente.

- STM_Open, STM_Open_Ex

Es la primera función que el desarrollador debe llamar antes de poder realizar cualquier operación con la tarjeta. Esta función abre un canal de comunicación con el lector y devuelve el manejador, que corresponde a una combinación entre el lector y la tarjeta inteligente introducida. Los parámetros de comunicación son establecidos automáticamente por el SOTI. La diferencia principal existente entre las dos funciones es un parámetro. En la primera de ellas, para seleccionar el terminal, únicamente hay que indicar el identificador de puerto donde se encuentra el lector conectado, mientras que la segunda función, tendrá un parámetro más sencillo de utilizar, como es el nombre del propio terminal. Si se desconoce el nombre del terminal, existe en las bibliotecas una función que nos devuelve el nombre de todos los lectores conectados.

El uso del terminal también se establece en esta instrucción, el terminal podrá estar seleccionado de tres modos distintos: exclusivo, compartido o compartido a las demás aplicaciones.

En el primer modo de operación, el terminal solo tendrá un canal de comunicación con la aplicación. En el modo compartido se pueden abrir varios canales de comunicación al terminal, pero todos estos canales serán establecidos únicamente entre una única aplicación y el terminal. En el último modo, el terminal se encuentra disponible para el uso de todas las aplicaciones que lo soliciten.

- STM_Close

Cierra un terminal identificado por un manejador. Después del cierre, ningún recurso de la tarjeta estará disponible. El canal de comunicación es automáticamente cerrado.

Es importante llamar a este método para liberar la tarjeta, porque el borrado del objeto que representa la tarjeta o su liberación, no conlleva la eliminación de los recursos usados por ésta.

- STM_Request

Esta función, efectúa un reset del hardware de la tarjeta. Los parámetros de comunicación y el SOTI de la tarjeta son reconocidos automáticamente por la función. Si este comando se ha llevado a cabo correctamente, el usuario será capaz de establecer una comunicación con la TI. Es posible la existencia de 14 canales simultáneos a la tarjeta. Como sucede con el terminal, la tarjeta inteligente puede ser seleccionada exclusivamente o compartida en la propia aplicación.

Otro canal puede ser abierto mediante la función *STM_OpenChannel()*. Los comandos a continuación son enviados a través de ese nuevo canal lógico creando de esta manera una aplicación multi-hilo. Después de realizar las operaciones oportunas en ese canal, debe de ser cerrado. La librería de STARCOS soporta el concepto de programación multi-hilo, permitiendo una comunicación simultánea con múltiples lectores y TI. En este caso, si existiera alguna región crítica, la librería automáticamente se encargará de sincronizar los hilos mediante el mecanismo de espera activa.

- STM_GetMsg

Esta función ayuda a convertir el código hexadecimal de error a una cadena de texto Unicode, que nos informa del error.

- STM_ReadPublicRSAKey()

La función obtiene la clave pública correspondiente del IPF seleccionado. Devuelve tanto el módulo como el exponente de la clave. Para poder leer la clave se tiene que satisfacer dos condiciones previas a la ejecución del comando; el cumplimiento de la política de seguridad especificada a la clave y una selección previa del IPF. Aunque en C, el método original devuelve los dos parámetros, la versión implementada en C# solo devuelve uno, ya sea modulo o exponente, a elección mediante un parámetro en la cabecera del método.

- STM_WritePublicRSAKey()

Esta función es complementaria a la anterior. Guarda la clave pública suministrada en el IPF. Si no se introducen parámetros, se crea el registro en la TI, pero el SO marcará la clave como incompleta y almacenará un valor de 'FF FF FF FF FF… FF'. Como sucede con el comando *STM_ReadPublicRSAKey()* es necesario seleccionar anteriormente el IPF y cumplir con las políticas de seguridad asociadas a la escritura y acceso del IPF.

- Funciones de seguridad: Módulo que nos ofrecerá funciones para la seguridad de la tarjeta y lector, permitiendo autenticación.

- STM_ChangePIN()

La función *STM_ChangePIN()*, cambia el valor del PIN introducido. Se efectúa primero una autenticación con el PIN antiguo. Si este proceso se realiza de manera correcta, el SOTI de la TI establece el nuevo código PIN y restablece el contador KPFC a su valor máximo. Este modo de operación es el único permitido para realizar el cambio de PIN de la TI dentro del sistema implementado. La TI permite sobrescribir la clave del

ISF, si se cumplen las condiciones de seguridad impuestas en él, mediante el comando *STM_WriteKey()*. Esta función utiliza el comando de la tarjeta inteligente VERIFY AND CHANGE.

- STM_VerifyBiometric()

STM_VerifyBiometric() proporciona un mecanismo de autenticación entre la tarjeta inteligente y el usuario utilizando para ello de datos biométricos. Estos datos enviados a la tarjeta, son comparados con la clave formada por el patrón de minucias almacenado en el ISF. Esta función, permite el cambio de estado en la tarjeta inteligente. La función llama al comando de la tarjeta inteligente VERIFY. Este comando sólo es compatible con tarjetas inteligentes del tipo STARCOS SPK 2.3/2.4 con capacidad biométrica. La función devuelve dos parámetros, un número entero que representa el número de coincidencias entre el patrón de referencia guardado en la tarjeta inteligente y el patrón introducido. El segundo parámetro indica si la verificación ha resultado correcta o incorrecta.

- STM_VerifyPIN()

Ejecuta el proceso de autenticación entre la tarjeta inteligente y el usuario mediante código PIN. Como en el caso anterior, la correcta ejecución de esta función efectúa un cambio en el estado de seguridad actual de la tarjeta inteligente. Esta función llama al comando de la TI, VERIFY. La función es compatible con los sistemas operativos que se rigen por la norma ISO 7816-4 y EMV '96. Si la longitud de la clave es menor que 8 bytes, el SO automáticamente añade relleno, usando el carácter ASCII '20' para las tarjetas del tipo STARCOS.

- STM_SetSE()

La función *STM_SetSE(),* crea referencias u objetos, que contienen las claves, algoritmos, datos de la tarjeta o números aleatorios del entorno de seguridad de la tarjeta. Estos objetos pueden entonces ser accedidos mediante los comandos de la tarjeta inteligente. La función *STM_SetSE(),* llama al comando de la tarjeta inteligente MANAGE

SECURITY ENVIRONMENT. Entre sus argumentos se pueden destacar dos. El primero de ellos, es el patrón de referencia que se establece en la operación siguiente, como por ejemplo:

- STM_TEMPLATE_CT: para cifrado/descifrado de datos.
- STM_TEMPLATE_DST: para firmar o verificar digitalmente.
- STM_TEMPLATE_HT: para cálculo de hash.

El segundo parámetro de importancia, es un puntero a un buffer de datos . Este array está compuesto por objetos TLV (*Tag Length Value*) [10]. Las opciones soportadas por la tarjeta STARCOS SPK 2.4 son:

- '80' establece el algoritmo de referencia de relleno.
- '83' referencia a clave pública. (CHID)
- '84' referencia a clave privada.(Key ID)

A continuación se presentan unos ejemplos ilustrativos de cómo construir el parámetro anterior, indicando un TLV para cifrar y otro para descifrar:

```
byte [] TLV_cifrar = new byte[] { 0x83, 0x01, CHID };

byte [] TLV_descifrar = new byte[] { 0x84, 0x01, Key ID };
```

Donde `byte[0]` es la referencia, `byte[1]` es la longitud de la clave y `byte[2]` el identificador de la clave a utilizar en el algoritmo posterior.

- STM_CardEncipher()

La función *STM_CardEncipher()*, cifra los datos introducidos en la tarjeta inteligente usando una clave asimétrica RSA. En la secuencia de datos introducida, el primer byte debe ser el byte más significativo. Los datos introducidos tienen que tener un formato de texto plano sin relleno. Antes de llamar a esta función se ha de llamar a la función *STM_SetSE()* para establecer un algoritmo de referencia (AR) para el relleno de los datos. Si el AR no es fijado, el texto plano a cifrar tendrá el siguiente formato en hexadecimal, especificada en la ISO 9796 método 2:

"60 00 … 00 01 || número aleatorio (8 bytes) || datos || BC"

Al cifrar mediante una clave RSA, donde el texto plano se divide en fragmentos de igual longitud que el módulo de la clave, se obtiene que la longitud máxima de datos que se pueden cifrar debe ser al menos 11 bytes menor que la longitud del módulo. Esta opción de cifrado proporciona un aumento de seguridad en el texto resultante, ya que dos entradas iguales, nunca ofrecen la misma salida. Si se define matemáticamente la longitud del texto total de salida en el peor de los casos (cuando el resultado de la división no es un número entero), en función del texto de entrada y la longitud de la clave, se obtiene:

$$L_{texto\ cifrado} = \left\{ \left\lfloor \frac{L_{texto\ entrada}}{L_{clave} - 11} \right\rfloor + 1 \right\} \cdot L_{clave}$$

Considerando $\frac{L_{texto\ entrada}}{L_{clave}-11} \gg 1$ y fijando $L_{clave} = 128\ bytes$ la utilización de este modo de cifrado implica un aumento de la longitud del texto a la salida del 9.5% con respecto a la longitud de entrada.

A continuación se presenta una gráfica que muestra la relación entre el texto de entrada y salida según la longitud de la clave elegida. Para minimizar el problema el desarrollador debe escoger la longitud máxima de clave permitida por la TI.

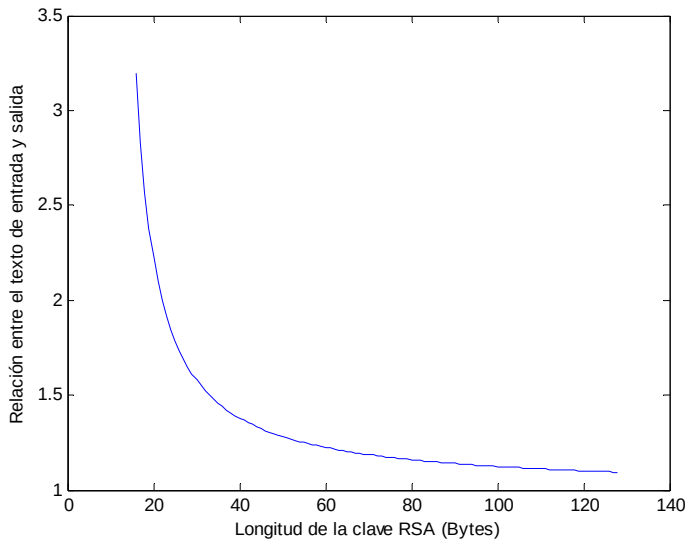

FIGURA 28. RELACIÓN ENTRE TEXTO DE SALIDA Y ENTRADA SEGÚN LA LONGITUD DE LA CLAVE RSA.

- STM_CardDecipher()

La función *STM_CardDecipher,* descifra la información mediante la tarjeta inteligente utilizando para ello la clave RSA privada guardada en el IPF. La clave debe ser seleccionada mediante el comando *STM_SetSE()*. Como sucede con el método de cifrado,

el primer byte del dato es el byte más significativo del dato. Si la longitud del texto es menor al tamaño de clave, el usuario debe realizar relleno siguiendo el siguiente formato:

"00 00 … 00 || Texto cifrado"

El texto descifrado devuelto por la aplicación, no contiene relleno. El tamaño máximo de bloque que nos devolverá la aplicación, como puede derivarse del método anterior será la longitud de la clave menos el relleno que se introduce (11 bytes).

- STM_ComputeHash()

La función *STM_ComputeHash,* calcula el valor del hash en la tarjeta inteligente y devuelve un hash con una longitud establecida por parámetro. El valor del hash es guardado en la propia tarjeta inteligente para el cálculo de firma digital. Esta función utiliza los comandos de la tarjeta inteligente PERFORM SECURITY OPERATION y HASH. En la aplicación se ha calculado el hash mediante el algoritmo SHA-1 estableciendo una longitud de 160 bits. Antes de llamar a este método se tiene que llamar al método *STM_SetSE()* para seleccionar el modo de hash y preparar la tarjeta para la operación.

- STM_ComputeSignature()

La función *STM_ComputeSignature,* calcula la firma digital para un hash calculado en la TI mediante la función *STM_ComputeHash.* Antes de realizar el cálculo de la firma, se efectúa el relleno según el modo seleccionado por el usuario. Esta función llama a los comandos de la tarjeta inteligente PERFORM SECURITY OPERATION y COMPUTE_SIGNATURE. Para la correcta utilización del método de firma, se debe especificar anteriormente el algoritmo y clave a usar mediante la función *STM_SetSE().* El formato de texto a cifrar, mantiene el formato predefinido en la ISO 9796:

"60 00 … 00 01 || número aleatorio (8 bytes) || Hash || BC"

El resultado tiene la longitud de la clave con la que se firma. Aunque se ha desarrollado una aplicación para firmas basadas en RSA, cabe destacar la posibilidad que nos ofrece la tarjeta inteligente de firmar a su vez mediante DSA.

- STM_VerifySignature()

La función *STM_VerifySignature,* complementa la función anterior. Efectúa la verificación de una firma digital para un hash almacenado anteriormente en la TI mediante *STM_ComputeHash.* Como paso previo a la llamada de este método, se tiene que llamar a la función *STM_SetSE()* para indicar la clave a utilizar y el método de relleno utilizado. Para el correcto funcionamiento del método, se debe cumplir con las restricciones de seguridad de las claves, suponiendo a su vez que estas tengan habilitada la funcionalidad de verificar firmas [Figura 25].

Esta función utiliza los comandos PERFORM SECURITY OPERATION con el subcomando VERIFY SIGNATURE.

Por último, se muestran los diagramas de flujo correspondientes a los métodos utilizados en las rutinas de cifrado/descifrado y firma/verificación.

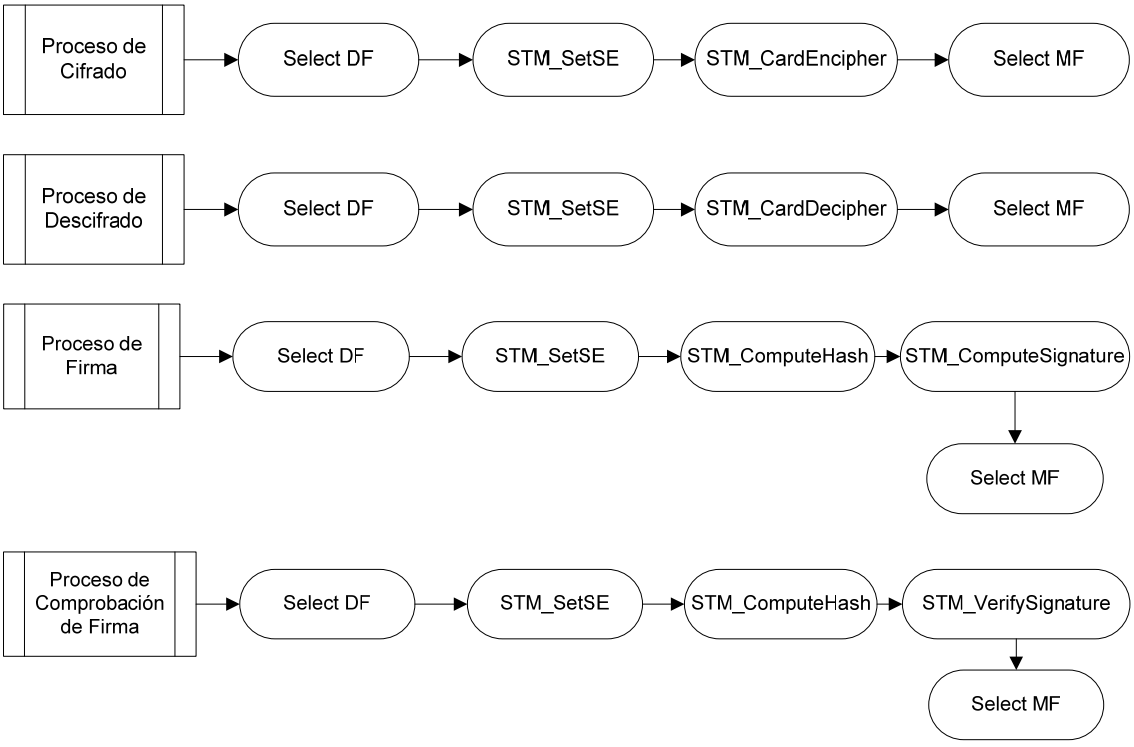

FIGURA 29. DIAGRAMAS DE FLUJO DE LAS DISTINTAS RUTINAS.

Como se puede observar en la Figura 29, para llevar a cabo una operación con el módulo criptográfico de la TI, primeramente se debe seleccionar el "almacén" donde se encuentre la clave a utilizar, mediante el comando *Select DF.* Una vez seleccionado el

DF, se procede a elegir la clave a utilizar en la operación criptográfica y a establecer los distintos parámetros de configuración. El tercer paso es el método que utilizará el modulo criptográfico de la TI. Una vez realizada la operación criptográfica, la aplicación vuelve al directorio principal de la tarjeta.

3.3.3.7. Formato de archivos de Salida

Establecido el tipo de comunicación con la tarjeta, se tiene que definir el formato de los datos que se va a tener en los archivos de entrada y salida. La aplicación permite cifrar y descifrar información y realizar firma y verificación digital. Se permite, por lo tanto, la introducción por parte del usuario de texto plano en formato ASCII a través de un formulario y cargar cualquier archivo para poder utilizar las capacidades criptográficas. Una vez fijado los dos tipos de entrada a la aplicación, se tiene que fijar un formato de datos de salida, que a su vez, servirá como entrada en la parte de verificación de firma digital y de descifrado.

Centrándose en la parte de cifrado de datos, después de realizar el proceso de cifrado con la clave pública, la aplicación obtiene el flujo de bytes correspondiente al texto cifrado. Esta información es almacenada automáticamente en un archivo con la extensión "*.cif*" sin realizar ninguna conversión, obteniendo de esta manera un archivo ilegible. Para el cifrado de textos introducidos mediante el formulario, se ha optado por convertir el flujo de bytes correspondiente el texto cifrado a su equivalente en Base64, formato más legible que el anterior y que el usuario puede copiar de un *textbox* de una manera más sencilla para su posterior descifrado. Esta opción no crea un nuevo archivo con el texto cifrado, pero se proporcionará esta posibilidad al usuario.

En la utilidad de firma digital, siempre se crea un archivo con la firma obtenida. Se ha desarrollado dos posibles formatos de salida; el primero es similar al caso de cifrado, un archivo que contiene la representación del flujo de bytes correspondientes a la firma con extensión "*.starcosFIR*".

La segunda solución desarrollada, está basada en la creación de un archivo "*.pdf*" que contiene los datos del propietario de la tarjeta inteligente, hora de cálculo de la firma digital, tarjeta con la que ha sido realizada la firma y finalmente la firma digital codificada en Base64. Este archivo incluye además el texto que ha sido firmado, cuando se realice la firma del formulario. Por lo tanto, se obtiene un único archivo que contiene el texto y su correspondiente firma de una manera sencilla. Si clasificamos las firmas digitales obtenidas según la clasificación obtenida en el punto 2.3.2, se obtiene una firma del tipo "ex-

plicita" para el caso de firma de archivos y de tipo "implícita" para la opción de firma del formulario.

Como se puede ver en la Figura 16, .NET no proporciona al desarrollador ninguna herramienta para poder crear archivos con un formato especifico, ya sea *".doc", ".pdf", ".xls", etc.* Por lo tanto para llevar a cabo esta labor, después de distintas pruebas se utiliza la librería "*itextsharp.dll*"[28]. Esta librería puede ser añadida al proyecto de una forma sencilla y permite al desarrollador utilizar objetos COM pertenecientes a dicha biblioteca. La creación del documento *".pdf"* estaba fijada en un primer momento para poder incrustar la firma dentro del propio documento y crear de esta manera un archivo doblemente firmado. La documentación sobre la librería en C# indica que para llevar a cabo la firma del documento y posteriormente incrustarla en el archivo, la librería debe ser capaz de acceder a la clave privada del firmante[5], mediante un archivo *".pfx"*. Como se estableció en el diseño de la TI y por las propiedades de los ISF, la clave no puede ser leída en ningún momento, por lo que no se puede incrustar la firma en el archivo. La documentación no indica la posibilidad de incrustar la firma de ninguna otra manera. Por lo tanto, se crea un documento que solo contenga la firma y el texto firmado. Para incrustar los textos se utilizan los métodos que proporciona la librería, para ello previamente se han de inicializar las cadenas de texto que forman las distintas partes del documento. La inclusión de la imagen final (Figura 56) en el documento no es tan sencilla, ya que la librería solamente admite una forma de hacerlo. *ItextSharp* permite cargar únicamente la imagen desde disco, no permitiendo la utilización de una imagen añadida a los recursos del proyecto.

En el proceso de verificación, al tratar con un documento *".pdf"* donde los datos están incrustados en el propio archivo y que no puede ser leído línea a línea como un texto plano, surge el problema de la obtención de la información guardada en el mismo. La primera aproximación para solucionar el problema fue la utilización de la librería utilizada para la generación de documentos. Aunque esta librería permite cargar documentos *pdf*, no proporciona al desarrollador la capacidad de extraer la información contenida en el mismo.

Para realizar el proceso de analizado y extracción de la información se han analizado diversas vías, como *itextsharp*, Adobe PDF Ifilter[6] y finalmente PDFBOX[7].

PDFBOX es una librería para utilizar documentos *pdf* en Java, pero existe una versión en .NET utilizable combinándola conjuntamente con IKVM.NET. Para utilizar esta librería, tanto PDFBOX como IKVM deben estar incluidas en el *classpath* de la

[5] http://itextpdf.sourceforge.net/howtosign.html
[6] http://www.adobe.com/support/downloads/detail.jsp?ftpID=2611
[7] http://www.pdfbox.org/

aplicación. La utilización de esta librería proporciona al usuario un *string* con todo el texto contenido en el archivo. Una vez obtenida dicha cadena, es labor del diseñador tratarla para una posterior utilización, p. e. eliminación de patrones del tipo "\r\n" incluidos en la propia generación de los párrafos y espacios añadidos por la propia librería. Estos espacios ofrecieron al principio problemas en la verificación de la firma, ya que se realizaban comprobaciones erróneas cuando realmente no lo eran. Estos espacios los incluye automáticamente la librería de creación del archivo, en cada línea que compone en el *pdf.*

3.3.3.8 Implementación parte Biométrica.

La tarjeta inteligente STARCOS SPK 2.4, ofrece la capacidad de efectuar una verificación biométrica mediante huella dactilar. Por lo tanto, se propuso añadir esta funcionalidad complementaria a la aplicación y que el usuario pudiera elegir si autenticarse vía código PIN o mediante su huella dactilar. De esta manera se obtiene un nuevo proceso de autenticación, que pasa la máquina de estados de la TI del nivel 15 al 14, quedando así ampliado el diagrama de estados de la Figura 22 en la parte de usuario.

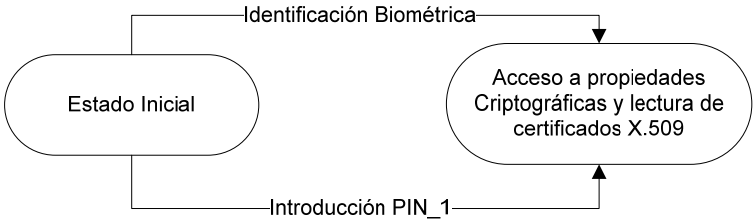

FIGURA 30. ESTADO FINAL ARQUITECTURA ESTADOS USUARIO.

Para efectuar cualquier operación biométrica con la tarjeta STARCOS, G&D ofrece una librería, similar a la ofrecida para el manejo de las TI. Esta librería permite usar diferentes sensores de huella y convertir la imagen a un patrón de huella adecuado para la tarjeta. Esta librería contiene:

- GDSensor.h: Fichero de encabezado, que contiene todas las definiciones de estructuras y tipos de datos que utilizan las librerías.

- GDSensor.dll: Controla las funciones biométricas ofrecidas por G&D para la obtención y posterior comparación del patrón de huella.

- GDSensor.lib: Librería que hace referencia a la anterior "*.dll*", el desarrollador de vincular la librería en el proyecto para una correcta compilación.

Esta librería únicamente soporta TI de tipo STARCOS SPK 2.4 Biometric. Para la utilización de esta librería conjuntamente con los lectores, se tiene que copiar las DLL ofrecidas por los fabricantes al directorio donde se encuentre GDSensor.dll o a C:\Windows\System32. Se introduce a continuación los métodos necesarios para obtener el patrón de huella:

- GDSensor_Open:

Lanza una nueva ventana de captura de huella en la pantalla. El tipo de lector de huella tiene que ser establecido anteriormente mediante *GDSensor_SetGrabberType()*. Solo se puede abrir una única ventana al mismo tiempo. El diseñador tiene que tener cuidado en cerrar la ventana si se ha acabado con su uso, para liberar recursos.

- GDSensor_Close

La función *GDSensor_*Close cierra la ventana de captura. Todos los recursos son devueltos al sistema y entonces es seguro salir de la interfaz de la aplicación. Todos los cambios en la configuración son guardados en *Gd_core.ini*, dependiendo de las propiedades establecidas a través del método *GD_Sensor_SetSaveOptions*.

- GDSensor_OCTGrab

La función *GDSensor_OCTGrab* inicializa una nueva ventana para capturar datos. La imagen es capturada si el botón izquierdo del ratón o la tecla especificada por parámetro es pulsada. Esta función se detiene al pulsar el botón derecho del ratón. Devuelve el patrón de minucias pertenecientes a la huella dactilar.

- GDSensor_GetImage

La función *GDSensor_GetImage,* copia la imagen visualizada en la ventana de captura en el buffer. Se debe llamar antes a la función que abre la ventana de captura. Se le introducirá por parámetro un puntero donde guardar la imagen. Si todo el proceso se lleva correctamente la función devolverá TRUE.

4. Interfaz De Usuario

En este capítulo se analiza con un mayor detalle la implementación del sistema de gestión de identidad desarrollado. La explicación está centrada en describir la interfaz implementada en base a las decisiones tomadas a lo largo de la presente memoria. Se explicará el funcionamiento de las aplicaciones implementadas, haciendo especial hincapié en la labor que debe realizar el usuario para su correcto funcionamiento. Se comentará a su vez los diagramas de flujo correspondientes a la aplicación. Por último en este capítulo, se procederá a analizar el rendimiento del sistema.

Se han programado tres aplicaciones software diferentes para desarrollar el sistema global. La primera de ellas corresponde a la que inicializa la estructura de la tarjeta inteligente. La segunda aplicación es la encargada de crear los certificados de usuario y grabarlos en la tarjeta de un modo seguro y la última aplicación desarrollada, puede considerarse como programa principal, es la encargada de realizar las operaciones criptográficas de la tarjeta y permitirá una gestión de la ICP por parte del usuario, almacenando claves en la tarjeta, etc.

Las aplicaciones, han sido desarrolladas sobre la plataforma .NET Framework. Para ejecutar correctamente los programas, se debe hacer sobre el sistema operativo Windows teniendo instalado .NET Framework 3.5 o superior.

4.1 Programa inicialización tarjeta

La función principal desarrollada en esta aplicación, es la de inicialización de las estructuras necesarias en la TI. La interfaz gráfica, se compone de una barra superior de botones y un cuadro de texto situado en la parte inferior de la ventana, como se puede ver en la Figura 31.

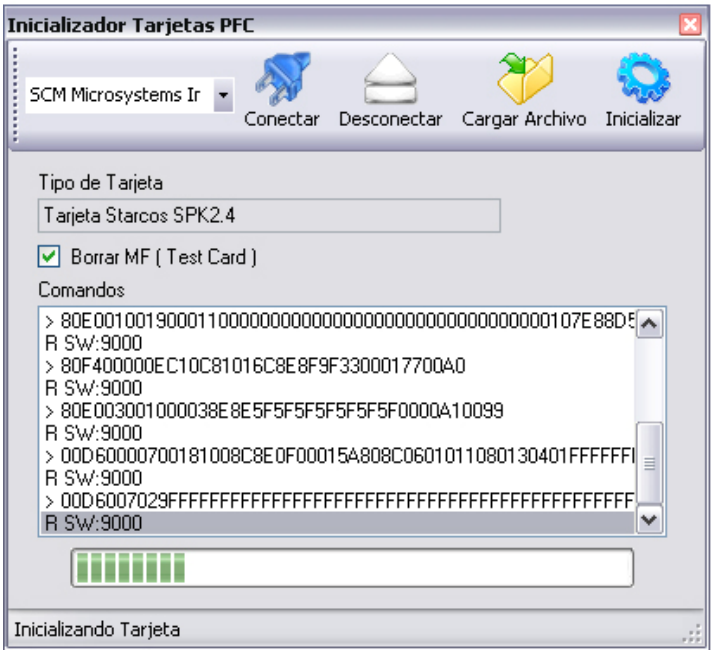

FIGURA 31. PROGRAMA INICIALIZACIÓN TI

Los funciones de los botones de la barra superior son:

- El primer componente de la barra de botones, es un *combobox* que muestra al usuario el nombre de las lectoras disponibles.

- Después de encontrar satisfactoriamente un grupo de lectores de tarjetas inteligentes, se establece una conexión entre la aplicación y el terminal mediante el segundo botón. Si se crea la conexión satisfactoriamente se establece un canal lógico con la tarjeta. Hasta que no se produzca una conexión satisfactoria, no se habilita en la aplicación el uso del resto de los botones.

- El segundo botón, efectúa la desconexión con el lector, permitiendo de esta manera una posterior conexión del lector con otra aplicación.

- El tercer botón permite cargar un archivo de comandos, generado por STARMAG, con el formato siguiente:

```
Command:  80 E0 00 00 13 01 02 03 04 05 06 07 08 00 5C 00 38 9F
9F 9F 9F 00 00 00

Command:  80 F4 00 00 1B C1 0C 01 00 08 0F 0E 5F 33 00 00 07 00
81 C2 0B 01 00 00 30 30 30 30 30 30 30 30
```

La estructura final generada en la tarjeta inteligente, agregando IPF, ISF y archivos auxiliares es la siguiente:

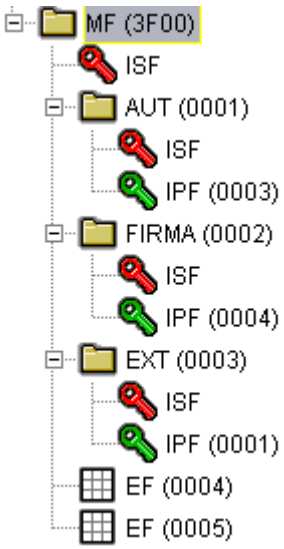

FIGURA 32. ESTRUCTURA FINAL TI (STARMAG)

Cargado satisfactoriamente el archivo, se habilita el último botón que permite inicializar la estructura en la TI, siempre y cuando, el tipo de TI introducido (mostrado en el *textbox*) sea el adecuado.

Mientras la aplicación crea la infraestructura de la de la TI mediante los comandos contenidos en el archivo, se muestran al usuario los comandos enviados y la respectiva respuesta de la TI. A su vez se actualiza el porcentaje de progreso.

Para evitar tener el proceso de actualización y envío de datos dentro del hilo principal (encargado de refrescar el formulario), se ha diseñado una aplicación multi-hilo. De esta manera se evita que el formulario se quede "congelado" hasta que finalice el envío de comandos. Además del hilo de formulario, lanzado por la propia aplicación en la inicialización, se crea un hilo paralelo al anterior con la misión de refrescar la barra de progreso y el cuadro de comandos. Para realizar un sistema multi-hilo en .NET, el desarrollador debe tener en cuenta que los atributos de un objeto solo son accesibles desde el

propio hilo donde fue creado (en este caso, el hilo principal, el cual crea e inicializa los componentes). Para solucionar este inconveniente, el desarrollador puede usar una directiva que permitida .NET[8], extremando la seguridad para evitar las condiciones de carrera [29][30].

Se muestra a continuación, el diagrama de flujo básico de la aplicación, donde Selección Lectora, Cargar Archivo de comandos e Inicializar tarjeta, son las acciones producidas al pulsar cada uno de los botones correspondientes, estando la aplicación parada hasta ese momento.

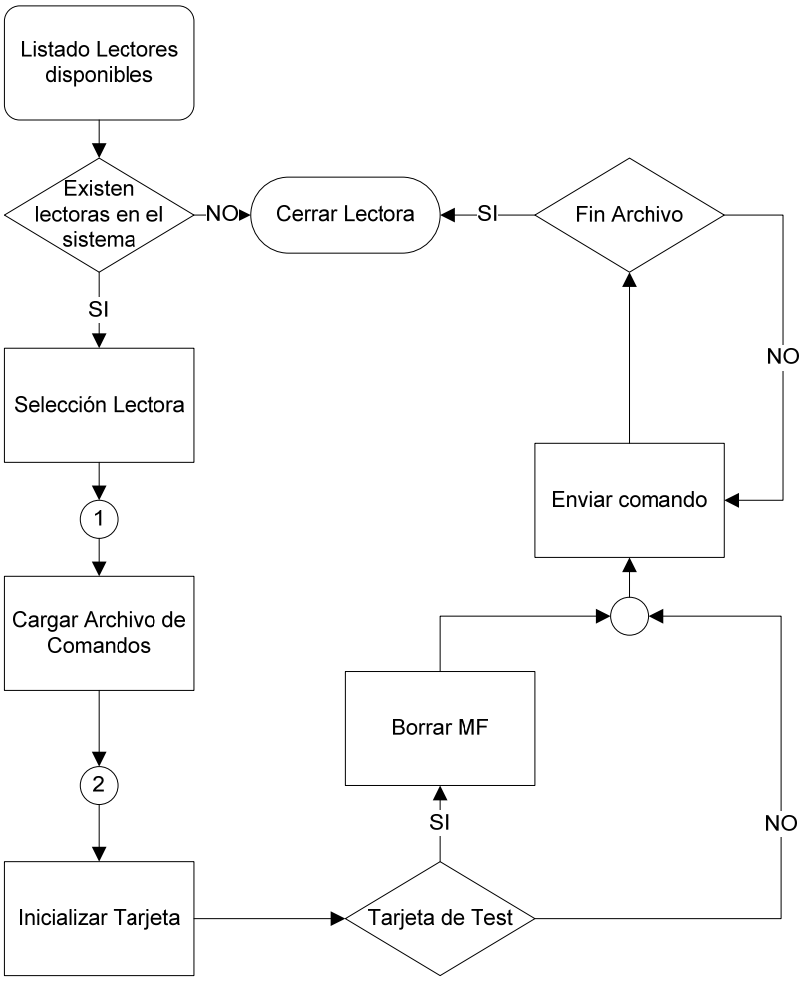

FIGURA 33. DIAGRAMA DE FLUJO INICIALIZADOR TI.

En un sistema global, la inicialización de la TI no tendría que ser realizada por el usuario, ésta aplicación tendrá que ser utilizada por la autoridad de confianza de la PKI.

[8] CheckForIllegalCrossThreadCalls = false

4.2 Programa generación X509

Esta aplicación, es la encargada de la creación y configuración de los archivos incluidos en la TI, como los certificados X.509 y el archivo auxiliar que almacena los datos del poseedor de la TI. La aplicación permite la modificación del PIN de usuario y la inclusión de la clave biométrica del usuario. El programa, se compone de una barra superior con botones y un cuadro con distintas pestañas en la parte inferior, como se puede ver en la Figura 34.

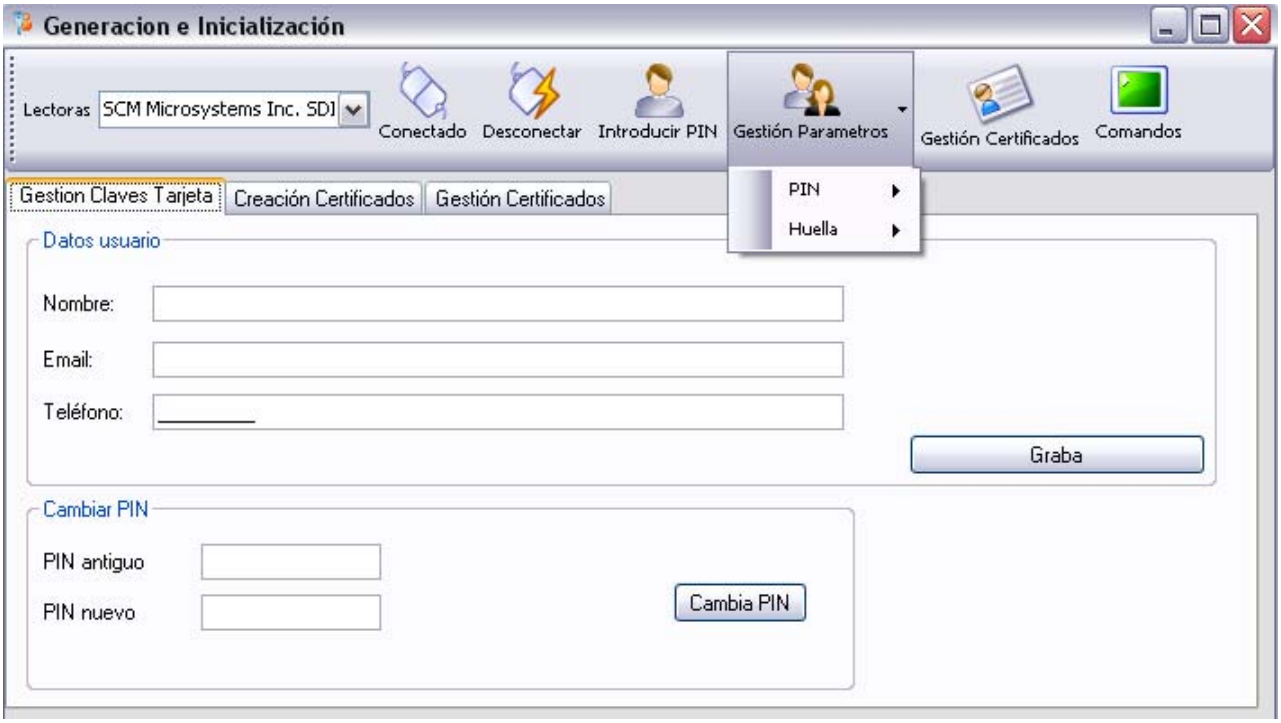

FIGURA 34. PRESENTACION APLICACIÓN 2.

Analizando la barra superior de la figura anterior, el primer botón de la aplicación, sirve para conectarse con el lector de tarjetas inteligentes, seleccionado anteriormente mediante el *combobox*. El segundo botón permite desconectar la TI del terminal seleccionado, dejando así libre el terminal para las demás aplicaciones. Los dos siguientes botones, son los responsables de lanzar la aplicación para realizar una autenticación mediante código PIN [Figura 36]. El botón "*Introducir PIN*" lleva a cabo la autenticación mediante el PIN_1. El botón siguiente, permite dos opciones; llevar a cabo una autenticación mediante del PIN_2 global (proceso a realizar si se quiere guardar los certificados X.509 en la tarjeta, ya que llevará la máquina de estados del MF al estado 13) o guardar la clave biométrica asociada al patrón de minucias del usuario.

El botón Gestión Certificados, habilita la lectura de los datos contenidos en los certificados X.509 guardados en la TI para poder exportarlos fuera de la TI. El último botón, ofrece un modo de depuración. Muestra un cuadro de texto, capaz de monitorizar en tiempo real los comandos enviados y recibidos entre la TI y la aplicación. Esto implica la creación, como en la aplicación anterior, de un sistema multi-hilo. En la ejecución del programa se obtiene tres hilos distintos de ejecución, el primero es el encargado de refrescar el formulario, el segundo tiene la función de monitorizar los comandos intercambiados entre la TI y la aplicación y el último es el encargado del intercambio de comandos entre la TI y el programa.

A continuación se procede a explicar las funciones implementadas en la aplicación más detalladamente, mostrando su correcta ejecución y algunos casos comprobados de fallo.

El primer fallo que se puede encontrar un usuario, es la selección de un lector, pero que la TI no esté introducida. La aplicación muestra entonces el error recogido en la Figura 35. Este error es ofrecido por el SOTI de la TI STARCOS SPK 2.4 Biometric.

FIGURA 35. ERROR TARJETA NO ENCONTRADA.

Antes de poder manejar cualquiera de las utilidades ofrecidas (los botones y las pestañas se encuentran desactivadas), el usuario debe realizar una autenticación satisfactoria. Para llevar a cabo dicha autenticación, puede utilizarse tanto el PIN_1 o PIN_2, habilitando cada autenticación las funciones recogidas en el apartado de seguridad. Cuando el usuario vaya a realizar la autenticación obtiene un cuadro de diálogo como el de la Figura 36. Si la autenticación es errónea la aplicación muestra al usuario un cuadro de error indicando que el PIN es incorrecto además de indicar el número de intentos disponibles.

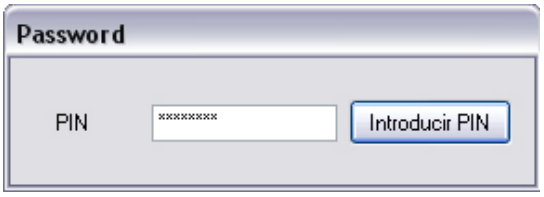

FIGURA 36. CUADRO INTRODUCCIÓN PIN

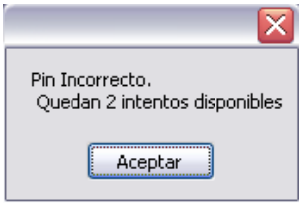

FIGURA 37. INTRODUCCIÓN INCORRECTA CÓDIGO PIN.

Realizada la autenticación de manera correcta, se tiene acceso a la aplicación. Observando la Figura 38, si se observa detenidamente los comandos intercambiados con la TI, se comprueba que se ha realizado dos autenticaciones mediante PIN. La primer realizada en el instante de tiempo 148531 ms, donde se introduce un PIN erróneo con valor igual a 77777777 (en la captura su valor en ASCII). Al ser incorrecto, la tarjeta devuelve un 63 C2, que indica que quedan dos intentos posibles para realizar una autenticación. En cambio, si se observa el instante 249734 ms, se lleva a cabo una autenticación mediante el código 00000000, la tarjeta nos devuelve el comando 90 00, que indica que se ha realizado el proceso correctamente, modificándose de esta manera el estado interno de la tarjeta inteligente.

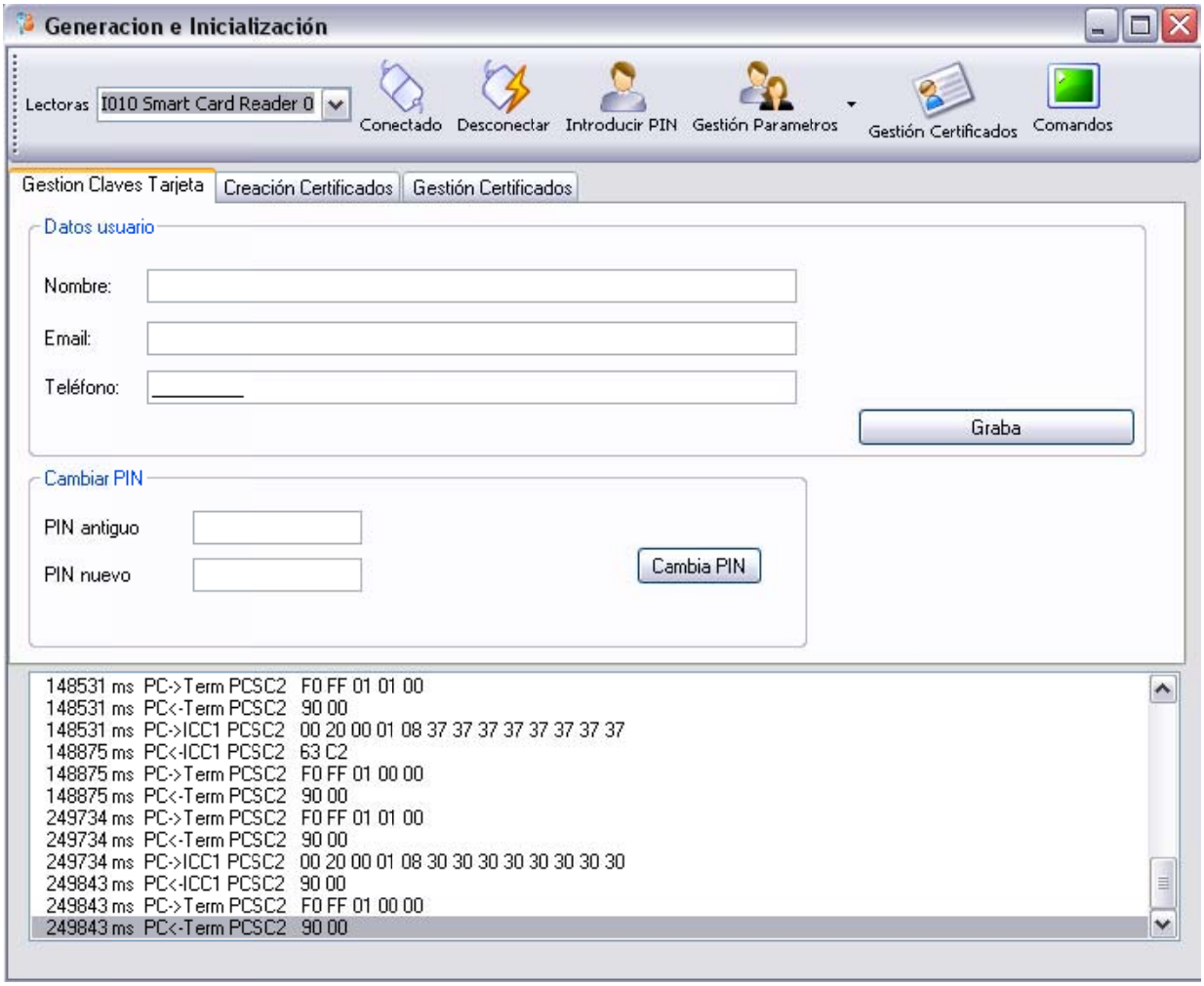

FIGURA 38. PRESENTACIÓN PROGRAMA.

Analizando las funciones que proporciona la pestaña "Gestión Claves Tarjeta", podemos ver una subdivisión de la pestaña mediante dos *GroupBox*. El primer *GroupBox*, recoge los datos del usuario, esta información puede ser ampliada de manera sencilla o incluso ser leída de una base de datos. Mediante la pulsación del botón, se guarda la información en el fichero EF0005 [Figura 32]. De esta manera, la TI contiene los datos de usuario, siendo éstos accesibles y modificables por una autoridad central. El segundo *GroupBox*, permite cambiar el PIN de la TI introducida, acción de extrema seguridad. Si todas las tarjetas se inicializan mediante el mismo archivo de comandos, todas poseerán el mismo PIN, lo cual llevaría a un fácil ataque a la seguridad, p.e ante el robo de un gran número de TI, ya que, conociendo el PIN de una, se conocería el de todas. Esto se puede modificar en la generación de la estructura de la TI, mediante la creación de un script que genere un PIN aleatorio en el comando, para ser posteriormente suministrado al usuario.

La siguiente pestaña, sólo contiene un único *GroupBox* [Figura 39], denominado Creación Certificado X.509. En esta pestaña, el usuario dispone de la posibilidad de inicializar el certificado digital X.509 de firma o de autenticación, introduciendo para ello en el segundo *GroupBox* los datos pertenecientes al usuario y la fecha de caducidad. Si se dispone de un certificado con clave privada[9], existe la posibilidad de generar el certificado digital, mediante este certificado. Es decir, la firma contenida en el certificado expedido al usuario, es realizada mediante la clave privada contenida en ese certificado de CA. Esto nos va a permitir comprobar la validez del certificado, mediante la clave pública de la CA. Para poder realizar la firma con dicho certificado, se deberá conocer el *password* que protege los datos del mismo. En este caso no hará falta introducir el nombre de la autoridad de certificación ya que la aplicación lo obtendrá del archivo.

En la parte derecha de la aplicación, existen tres botones; el primero de ellos denominado "*Previsualización*" ofrece la posibilidad de previsualizar el certificado digital creado con los datos establecidos. El segundo botón, "*Guardar Previsualización*" permite guardar los datos previsualizados en el almacén de certificados seleccionado (en nuestro caso, el almacén corresponde a un DF). El último botón crea y guarda directamente el certificado en la TI sin mostrarlo al usuario. El certificado digital es guardado estableciendo las políticas de seguridad del punto 3.3.3.4. Para la tarjeta STARCOS SPK 2.4 la librería no permite la creación de archivos, por lo que se tiene que hacer mediante comandos.

[9] makecert -n "CN=RootCA" -r -sv RootCA.pvk RootCA.cer

FIGURA 39. PESTAÑA CREACIÓN CERTIFICADOS APLICACION 2.

La última pestaña [Figura 40], proporciona al usuario capacidad de comprobar los datos de ambos certificados, cargarlos con el visor de certificados de Windows y por último exportarlos fuera de la aplicación.

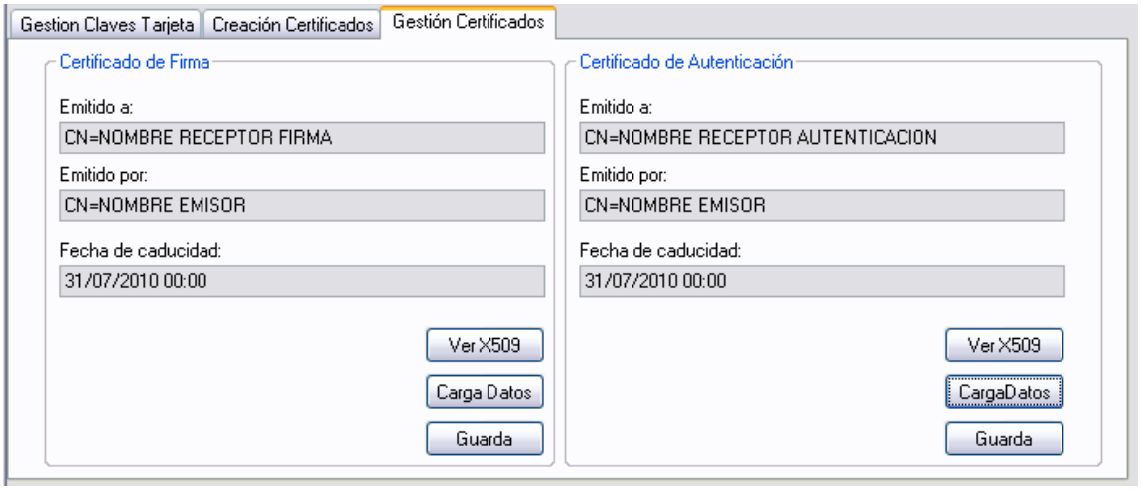

FIGURA 40. PESTAÑA GESTIÓN CERTIFICADOS APLICACIÓN 2.

A continuación se muestra el diagrama de flujo correspondiente a la aplicación

El sistema estará en “1” hasta que se selecciona un lector, pasando al estado “2” que será el estado de reposo, donde se mostrará cualquiera de las pestañas analizadas.

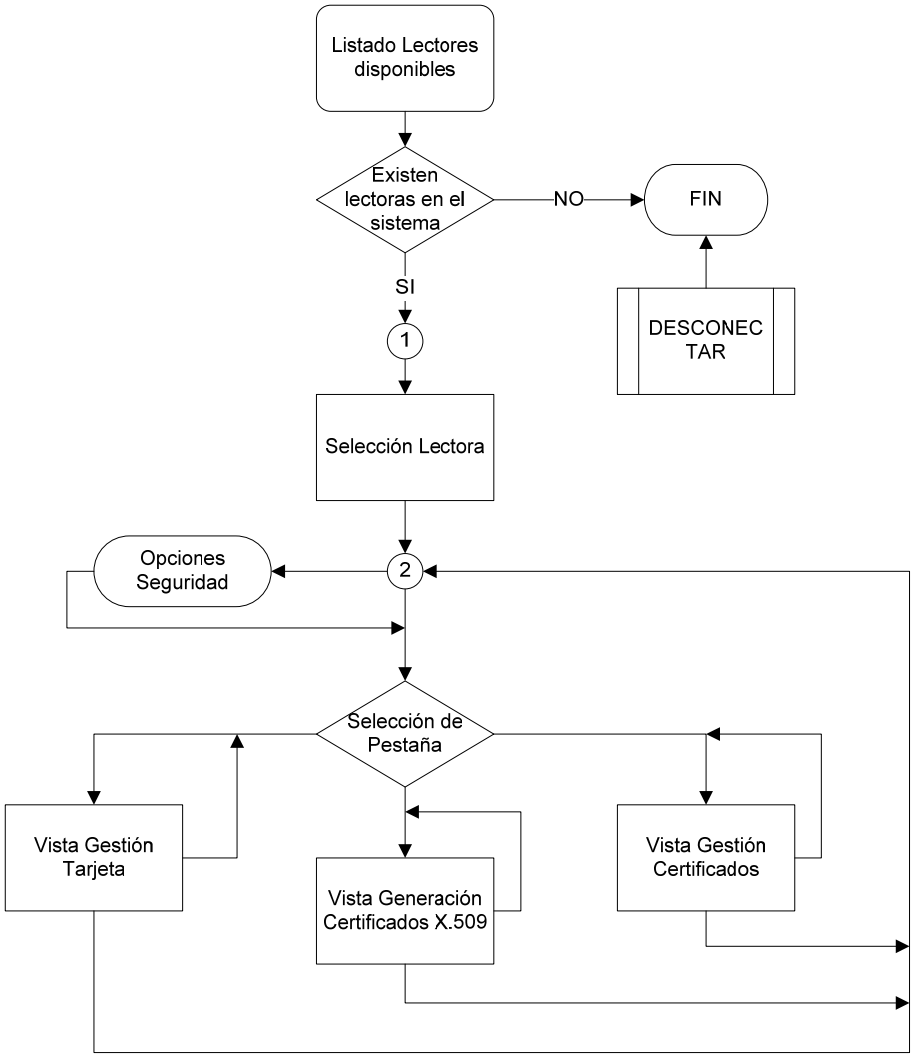

FIGURA 41. DIAGRAMA DE FLUJO APLICACIÓN 2.

4.3 Programa Principal

La última aplicación desarrollada del sistema, es la destinada para el uso de las posibilidades criptográficas del sistema y manipulación de certificados digitales dentro de la PKI. Esta aplicación no se encuentra contenida únicamente en un formulario. Se ha dividido su funcionamiento en dos formularios, apartando las funciones de selección del lector de las funciones de utilización de la tarjeta inteligente.

FIGURA 42. APLICACIÓN PRESENTACIÓN DEL PROGRAMA.

Se ha desarrollado un formulario para la selección del lector [Figura 42]. Este formulario se compone de 2 partes; un *Groupbox* que proporciona información del sistema y un conjunto de botones. El *Groupbox* se subdivide en una *ImageList* que contiene un icono por cada lector de TI disponible en el ordenador. El icono irá cambiando dinámicamente la imagen, según el terminal que representa tenga o no tarjeta y según la validez de la misma. En la parte derecha, se presentan dos cuadros de texto, donde se indica el tipo de tarjeta inteligente introducida e información sobre el posible uso de la tarjeta.

Esta parte del sistema no se puede desarrollar mediante programación con un único hilo ya que además de refrescar el formulario, se tiene que comprobar periódicamente el estado de cada lector del sistema. Para ello se desarrolló una clase, utilizando el API de WinSCard proporcionado por Windows, que monitoriza un lector pasado por

atributo. Cada vez que se produzca una inserción o extracción de tarjeta se provoca la invocación de un método delegado. Dentro de estos métodos se actualizan los campos del *Imagelist* y los respectivos *textbox*.

```
public event CardInsertedEventHandler OnCardInserted = null;

public delegate void CardInsertedEventHandler();

LEC[i].OnCardInserted+=new CardInsertedEventHandler(miCard_OnCardInserted);
```

Como se puede ver en el código anterior, el evento *OnCardInserted*, es inicializado como un delegado, cuyo punto de entrada es el método que actualiza el *groupbox*.

Los delegados en .NET básicamente realizan la misma función que los punteros a funciones en C++. Sin embargo, los delegados son objetos de memoria gestionada y de tipo seguro. Implica que el entorno de ejecución garantiza que un delegado apunta a un método válido. Los métodos delegados funcionan de la siguiente manera: el código cliente hace una invocación a un método, pasándole el método de devolución de la llamada por parámetro. El método invocado lanzará inmediatamente un nuevo hilo de ejecución ejecutándose el método solicitado.

Al poder existir un número ilimitado de hilos en el sistema, uno por cada lector más cada delegado correspondiente, existe la posibilidad de que se produzcan condiciones de carrera. Se han dado casos durante la prueba del sistema, donde en el *ImageList* aparecían un número de lectoras distinto al real, duplicando lectoras existentes. Esto se produce, cuando en mitad del primer hilo de actualización lanzado, el procesador cambiaba al segundo evento, produciéndose de esta manera condiciones de carrera. Para solucionar esto, C# a través de las librerías .NET [Figura 16], ofrece la posibilidad de asegurar la sección critica, mediante monitores, semáforos o cerrojos. En este caso, se utiliza un único cerrojo, permitiendo la entrada de un único hilo a la sección crítica.

Retomando la Figura 42, se puede ver que únicamente se activa el botón de "aceptar" si el usuario ha introducido una tarjeta del tipo STARCOS SPK 2.4, quedando inhabilitado para los demás tipos de TI. Una vez pulsado el botón, este formulario pasa a un segundo plano y lanza el formulario de la Figura 49. Solo se puede volver a este formulario de selección de terminal, cuando se cierre el formulario de la Figura 49. El formulario de presentación le pasa al formulario recién lanzado por parámetro un objeto que contiene el manejador formado por la tarjeta inteligente y terminal seleccionado. Este formulario es el encargado de la eliminación del objeto en su cierre. El segundo botón cierra la aplicación, parando todos los hilos lanzados. El último botón oculta los detalles mostrados.

Analizando la Figura 49 se tiene un formulario similar al de la aplicación de generación de certificados digitales. La interfaz gráfica se compone de una barra superior de botones, un área principal dividida en cinco pestañas y un área inferior para mostrar los comandos en tiempo real.

- El primer botón, permite la autenticación por parte del usuario mediante un código PIN. Sin una autenticación previa con la tarjeta, el usuario no dispondrá de acceso a ninguna parte de la aplicación desarrollada. En este botón también se habilitará cambiar el PIN mediante identificación biométrica.
- El segundo y tercer botón, nos llevan respectivamente a las pestañas de cifrado y descifrado de textos y archivos (Figuras 51 y 52).
- El tercer y cuarto botón, nos permitirá el acceso a las pestañas de firma digital y verificación de firmas, tanto de archivos como de textos.
- El último botón, permite mostrar los comandos intercambiados entre la tarjeta inteligente y la aplicación, lanzando para ello un nuevo hilo encargado de activar la monitorización de comandos, para su posterior lectura y actualización de la interfaz.

La primera pestaña [Figura 50], permite la introducción de datos mediante el formulario y cifrarlos con una clave pública de un usuario del sistema, almacenada previamente en la TI. Se puede realizar la operación de cifrado y descifrado en la misma pestaña mediante los dos botones habilitados para tal efecto. La pestaña se subdivide en texto de entrada y texto de salida mediante dos *Groupbox*.

En el primer *Groupbox*, se diferencia un cuadro de texto donde introducir los datos, un *treeview* que representa la estructura interna de claves contenidas en la TI, además de los botones que lanzan los procesos de cifrado y descifrado. Se analiza el proceso que un usuario tiene que llevar a cabo para cada operación:

- Caso de Cifrado, como en una PKI, para cifrar los datos introducidos en el formulario, que se quiere enviar al usuario USER_1, necesitamos su clave pública, la cual se almacena en la TI leyendo su certificado X.509 mediante la quinta pestaña. Por lo tanto, a la hora de cifrar, el usuario debe seleccionar la clave pública del destinatario y lanzar el método. La tarjeta entra en el almacén donde se encuentra la clave y la selecciona. Una vez seleccionada se llama a la rutina de cifrado y vuelve al directorio inicial de la TI.

- En el caso de descifrado, solo se tiene que introducir el texto a descifrar, ya que dicho texto tendría que haber sido firmado con nuestra clave pública. En caso de producirse algún error en el descifrado se muestra el error recogido en la Figura 43. Aunque el error muestra el mensaje de “signature failed”, éste

indica que la comprobación realizada por la TI del relleno es errónea.(Ver Card_Encipher 3.3.3.6.1).

A modo de ejemplo en las Figuras 50 y 51, se observa que partiendo de un mismo texto a cifrar se obtienen dos textos cifrados distintos, debido al componente aleatorio introducido en el bloque cifrado (Ver Card_Encipher 3.3.3.6.1).

FIGURA 43. ERROR DESCIFRADO DE TEXTOS.

La segunda pestaña [Figura 52], ofrece la misma funcionalidad que la primera, pero orientada a archivos presentes en el sistema. En la parte superior de la pestaña, se encuentra el primer *Groupbox*, que agrupa las rutas donde están almacenados el archivo a cifrar/descifrar y el archivo donde se quiere guardar el resultado de la operación de cifrado/descifrado. El archivo cifrado tendrá una extensión "*.cif*".

En la parte inferior de la interfaz, puede elegir la operación a realizar mediante un *checkbox* (cifrado/descifrado) y la clave pública del usuario al que se desea mandar el mensaje cifrado. Si el usuario no marca ninguna clave, la aplicación lanza un error.

La tercera pestaña [Figura 53], presenta un interfaz similar a la ofrecida de en el caso de cifrado de textos en el formulario. Se ha añadido a esta interfaz de firma, la posibilidad de elegir el formato de firma. En este caso firmamos automáticamente con nuestra clave privada (no se debe introducir el PIN en cada proceso de firma como sucede en el caso del DNIe). La aplicación entra en el directorio correspondiente, seleccionando la clave privada de firma y realizando la operación de firmado. Se mostrará la firma generada en el *textbox* de salida y se guardará automáticamente según el formato elegido. Mientras se firma o se comprueba un texto, se deshabilita la aplicación para que no se pueda modificar los datos de entrada. Como se comentó en el punto 3.3.3.7 se ofrece la opción de generar un archivo con formato "*pdf*". Se muestra un ejemplo del archivo generado en el Anexo 8.2.

En el caso de verificación de firma, si ésta es correcta y el formato elegido de la firma es "*pdf*", la aplicación muestra los datos de usuario guardados en el archivo pdf, además de cambiar el indicador visual de amarillo a verde. En caso de firma incorrecta, cambia a color rojo, mostrando además un mensaje de error.

La cuarta pestaña [Figura 54], permite como sucede en la segunda pestaña trabajar con archivos que están presentes en el sistema. Al igual que sucedía en la pestaña dos, se subdivide en dos distintos *GroupBox*.

- El primero, permite al usuario la selección del archivo que se quiere firmar, y la elección de la ruta donde guardar la firma.

- El segundo *GroupBox*, permite elegir la operación a realizar (firma o validación). En el caso de verificación, el usuario debe seleccionar la clave pública correspondiente al usuario que ha realizado la firma. En esta segunda mitad, también el usuario será capaz de elegir el formato que se va a tener el archivo de firma o de validación en caso de verificación.

  A continuación se muestra el caso de una verificación correcta e incorrecta.

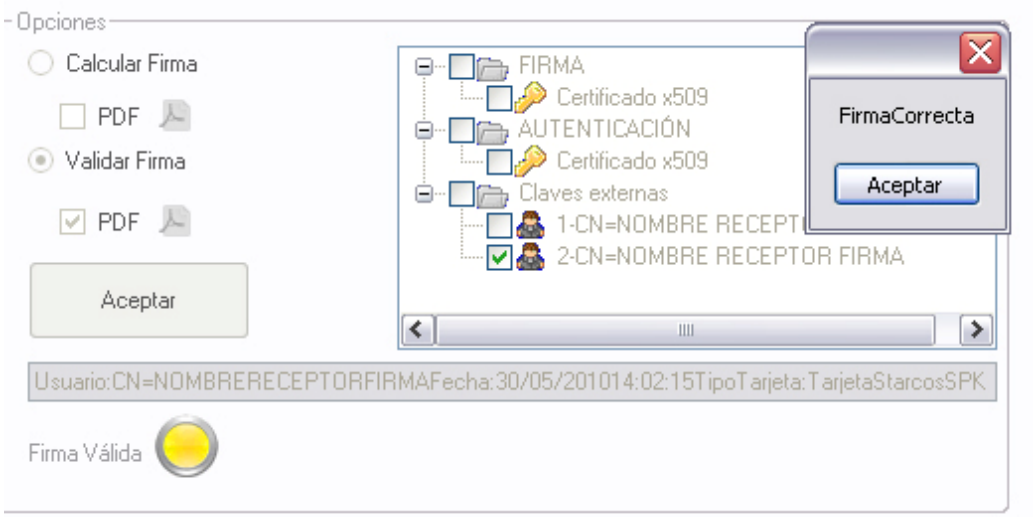

FIGURA 44. VALIDACIÓN FIRMA ARCHIVO CORRECTA.

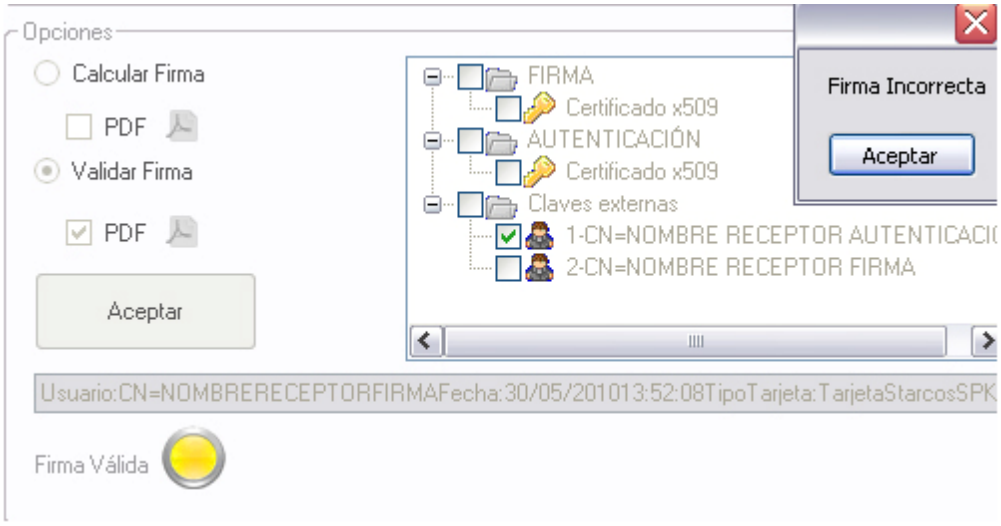

FIGURA 45. VALIDACIÓN FIRMA ARCHIVO INCORRECTA.

Antes de analizar las imágenes anteriores, se expone el caso analizado. La tarjeta contiene en su directorio de claves ajenas dos claves de un mismo usuario, la de autenticación y firma. La firma enviada por el usuario esta codificada en formato PDF. En el primer caso, el usuario que recibe la firma, efectúa una verificación con la clave pública de firma, sucediendo una verificación satisfactoria. En el segundo caso, se elije una clave RSA errónea, lo que lleva a una verificación incorrecta. Se tendría el mismo caso con cualquier otra clave que no fuera la elegida en la Figura 44.

Después de pulsar la ventana emergente, el control vuelve a la aplicación y se cambia la imagen "Firma Válida" a verde en caso correcto o a rojo en caso incorrecto, además de mostrar los datos el firmante.

La última pestaña [Figura 55], se divide en tres *GroupBox*. En el primero de ellos se puede manejar las claves públicas guardadas en la tarjeta mediante su selección en el *treeview*. En la segunda parte el usuario puede exportar sus certificados de usuario contenidos en la tarjeta inteligente. En el tercer GroupBox, el usuario tendrá la capacidad de añadir una nueva clave pública a la TI mediante el correspondiente certificado X.509.

Se presenta a continuación el diagrama de flujo del programa. Los cuatro procesos siguientes a Selección de Archivo son las cuatro pestañas analizadas. La comunicación con la TI se lleva a cabo mediante un nuevo hilo, de igual manera a la aplicación de Creación de Certificados, para que de este modo la interfaz gráfica pueda seguir refrescándose. Por lo tanto, en el peor de los casos se tiene tres hilos en la aplicación, el hilo principal encargado del formulario, el creado para la monitorización de los comandos intercambiados y el que se crea para la comunicación con la TI.

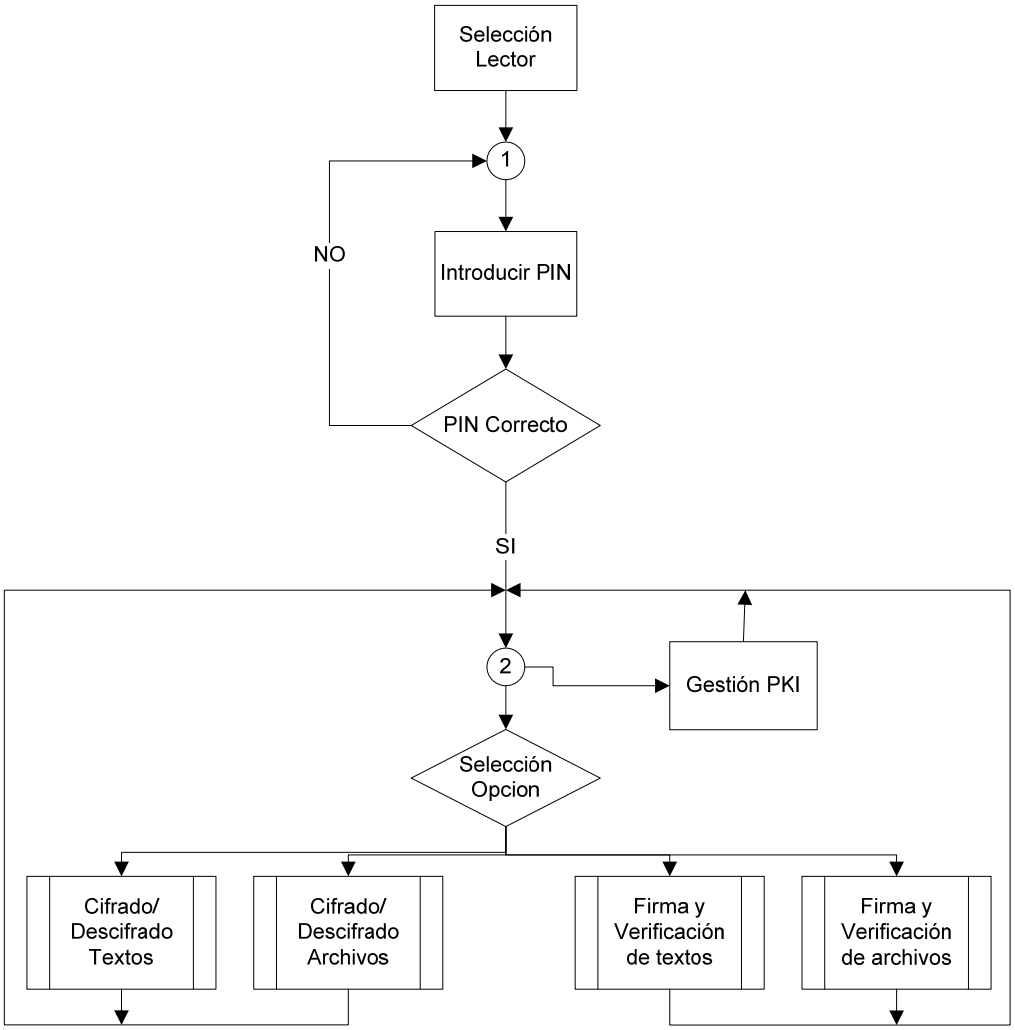

FIGURA 46. DIAGRAMA DE FLUJO PROGRAMA PRINCIPAL.

4.4 Rendimiento Del Sistema Creado.

A las pruebas de ejecución anteriores se añade una prueba de rendimiento del sistema. Se prueba el rendimiento a la hora de cifrar y de calcular la firma digital de un documento. Los casos de verificación y de descifrado, el tiempo es similar, ya que se utilizan el mismo número de comandos. Las medidas realizadas han sido tomadas con la monitorización de intercambio de comandos desactivada.

El primer caso a analizar es el caso de cifrado, cuyos datos son recogidos en la Tabla 1. Para esta prueba se ha ido modificando el tamaño de archivo, desde unos 4 Kb hasta 215 Kb. Para minimizar la varianza, se realizan diez veces cada medida.

Tamaño Archivo (Kb)	Tiempo medio transcurrido en el proceso (s)
3.82	3.19
6.04	4.49
13.3	10.8
49.7	40.3
99.4	80.5
214	173.5

TABLA 1. TIEMPO MEDIO TRANSCURRIDO EN EL PROCESO DE CIFRADO

Ajustando los datos anteriores se obtiene la siguiente relación que representa el sistema de cifrado.

$$t(s) = 0.81\,{}^{s}/_{Kb} \cdot Tamaño(Kb) - 0.09$$

Como podría suponerse la ecuación que representa al sistema es una recta, ya que lo único que hace el sistema si se encuentra con un archivo más grande, es llamar al proceso de cifrado más veces. Para Tamaño = 0, hay que tener en cuenta que se efectúa el cifrado de un tamaño de bloque de 117 bytes.

El segundo caso a analizar, creación de la firma digital, se ha procedido de igual manera a la anterior, variando el archivo de entrada y obteniendo el tiempo de respuesta. De igual manera se ha efectuado el proceso diez veces. Se recoge en la Tabla 2 los datos recogidos

Tamaño Archivo (Kb)	Tiempo medio transcurrido en el proceso (s)
0.5	1.05
6.06	6.4
23.5	23.15
43.2	42.1s
101	98.56
214	206.63

TABLA 2. TIEMPO MEDIO TRANSCURRIDO EN EL PROCESO DE FIRMA DIGITAL

Ajustando los datos anteriores obtenemos la relación que representa el rendimiento del sistema de firma digital implementado.

$$t(s) = 0.951\,{}^{s}/_{Kb} \cdot Tamaño(Kb) + 0.84$$

Como sucede con la operación de cifrado, el tiempo tomado en el cálculo de la firma digital también responde a una recta. Esta utilidad es más lenta que la aplicación de cifrado.

5. Presupuesto

En el presente capítulo se presenta un cálculo aproximado del coste de este proyecto. Para la evaluación de dicho coste es necesario un desglose del proyecto en tareas, a través de la planificación del mismo y la evolución temporal seguida. Cada tarea será cuantificada en función de su duración y los recursos consumidos, proporcionando de manera global un presupuesto aproximado del trabajo desarrollado.

5.1 Descomposición de actividades

A continuación se exponen las cinco tareas llevadas a cabo:

- Fase 1: Documentación para realizar el proyecto y estudio del estado del arte.
- Fase 2: Aprendizaje del lenguaje de programación y uso de la TI específica.
- Fase 3: Implementación de la aplicación.
- Fase 4: Pruebas y depuración del código obtenido.
- Fase 5: Documentación y memoria.

A continuación pasamos a detallar el tiempo tomado en cada tarea mediante Microsoft Project 2003.

	Estudio del Arte y aprendizaje de la platforma C#y VS	60 días
	Análisis de la programación de las TI STARCOS 2.4	14 días
	Estudio de propuestas de implementación	15 días
	Programación de la Aplicación	30 días
	Depuración, pruebas y documentación	15 días
	Escritura y corrección de la memoria	40 días

FIGURA 47. TAREAS DESARROLLADAS EN EL PFC

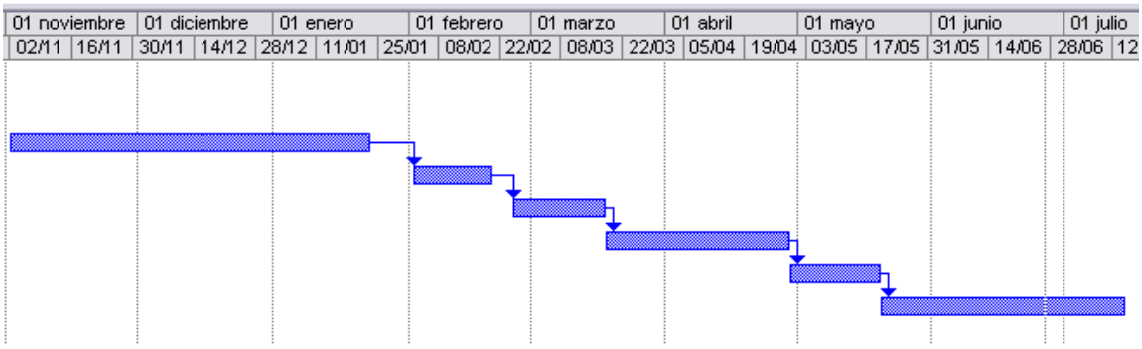

FIGURA 48. GRAFICO DE TIEMPO DE LAS TAREAS REALIZADAS.

Las siguientes tablas recogen el presupuesto ofrecido por el desarrollo del PFC, se destaca que el precio final sería el precio que ofrecería una consultoría, no el precio real gastado en el desarrollo.

PRESUPUESTO DE PROYECTO

1.- Autor: Eugenio Hernández Martínez

2.- Departamento: Departamento de Tecnología Electrónica

3.- Descripción del Proyecto:

Se desarrollará un sistema de gestión de identidad con Tarjetas Inteligentes de distintos fabricantes, utilizando securización y autenticación de todas las comunicaciones. El desarrollo se hará en C#.

- Titulo	**SISTEMA SEGURO DE GESTION DE IDENTIDAD MEDIANTE TARJETAS INTELIGENTES**
- Duración (meses)	**8**
Tasa de costes Indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):

31260 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a título informativo)	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
					0,00
Alonso Moreno, Raúl		Ingeniero Senior	1	4.289,54	4.289.54
Hernández Martínez, Eugenio		Ingeniero	8	2.694,39	21.555.12
					0,00
					0,00
				Total	25.844.66

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Material	600	100	8	60	80,00
Toolkit STARCOS	1500	100	5	60	125,00
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	205,00

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
	Total	0,00

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	25845
Amortización	205
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	5210
Total	31260

El presupuesto total de este proyecto asciende a la cantidad de TREINTA Y UN MIL DOSCIENTOS SESENTA EUROS.

Leganés, Septiembre de 2010

El ingeniero proyectista

Fdo. Eugenio Hernández Martínez

6. Conclusiones y líneas futuras

6.1 Conclusiones y Resultados Obtenidos

El sistema de gestión de identidad implementado se trata de una solución abierta, es decir, se puede utilizar cualquier tarjeta inteligente y lector de tarjetas inteligentes, siempre y cuando las TI tengan capacidades criptográficas y fueran acordes a la ISO-7816.

Es una solución flexible, aunque se ha desarrollado todo el sistema para funcionar en un único PC, la opción futura es tener un servidor donde estén las aplicaciones ejecutándose, y que el usuario pueda acceder a ellas, ya sea mediante un Applet en un navegador, o mediante un cliente implementado en C#. Es también flexible analizando los elementos hardware que lo componen, no se necesita un lector de TI exclusivo. El único punto problemático encontrado ha sido la detección del lector de huella dactilar, aunque este punto no es crítico, ya que el usuario puede realizar la autenticación con la TI mediante PIN.

Por lo tanto se ha obtenido una solución del problema donde un usuario es capaz de identificarse dentro de una red, mediante sus certificados digitales de usuario. Gracias al SDK proporcionado, se ha implementado una librería en C# de fácil uso para la utilización y manejo de cualquier tipo de TI compatible con la utilizada

Analizando los objetivos, se ha implementado y probado la correcta ejecución del sistema. Se ha diseñado a su vez una estructura de la TI muy segura, recordemos que la tarjeta dispone para su seguridad de dos códigos PIN y posibilidad de identificación biométrica. El usuario no podrá modificar los datos (claves RSA y certificados) que contiene la TI ni acceder a las claves privadas sin llevar a cabo una autenticación previa. La seguridad de la TI, también viene ofrecida por la seguridad en la generación de sus claves, ya que la clave privada nunca sale de la TI.

El único punto implementado, pero que no se ha podido probar su ejecución en el sistema global es la autenticación mediante biometría. La librería GDSensor.dll ha sido incapaz de detectar el lector biométrico del sistema "Precise 100SC" aunque éste estaba dentro del catálogo de G&D. A su vez, ninguna de las dos aplicaciones proporcionadas por STARCOS para el desarrollo de aplicaciones ha reconocido el sensor. Además, la documentación sobre el formato de huella guardado en la tarjeta es inexistente, por lo cual es imposible para el desarrollador reproducir ese patrón a través de una imagen leída. Otro punto negativo de la librería proporcionada por G&D es que no existe tampoco un método en la librería que obtenga el patrón, introduciéndole la captura por parámetro. Después de analizar profundamente la librería y ver que no podría reconocer el lector, se proponen varios métodos que podrían utilizarse para llevar a cabo una identificación biométrica con este sistema.

Se creó una aplicación, gracias a las librerías de Precise, capaz de generar el patrón, guardarlo en un servidor y llevar a cabo su posterior verificación, probando de esta manera que el lector funcionaba correctamente. Esto nos lleva a las siguientes propuestas:

- Probar distintos tipos de lectores de huella, hasta llegar a uno compatible.

- Intentar introducir el patrón en la TI directamente, probando los formatos ofrecidos por Precise 100 SC [12] (mediante el comando WRITE_KEY). Esto no funciono correctamente, ya que el patrón generado no pasaba nunca de 800 bytes y la autenticación no se llevaba a cabo.

- Guardar el patrón de huellas en la tarjeta para luego posteriormente hacer la verificación en el servidor: este método es completamente funcional, pero poco seguro, ya que si se quiere proporcionar seguridad al archivo que contiene el patrón para que solo pueda ser accedido por el usuario, éste se vería obligado a introducir el PIN para permitir el acceso a esos datos protegidos, además de la posterior identificación biométrica. Esto proporcionaría aún más seguridad,

ya que para acceder a las propiedades criptográficas, tendrían que sucederse dos autenticaciones, aunque no es lo que se pretendía en la aplicación.

- La última opción que se sugiere y orientada más hacia la arquitectura final del sistema, es la de guardar el patrón de huella en un servidor. Cuando el usuario se conecte al sistema, se haría la identificación en el propio servidor y el servidor envía un comando mediante SM a la TI para cambiar el estado interno de ésta.

Finalmente se optó por guardar el patrón en la tarjeta, pero realizar el proceso de verificación fuera de ella, si bien, como ya se ha dicho, no es la solución más segura, es una solución muy práctica. Dado que el fabricante no proporciona soporte para el lector de huellas ha sido imposible realizar la implementación match-on-card. No obstante con la implementación realizada se puede mostrar el mecanismo de de acceso a cierta información, así como el desbloqueo (o cambio) del PIN. Se puede dejar como trabajo futuro la implementación match-on-card haciendo uso de versiones más recientes del SDK/Lector de Huellas.

6.2 Líneas Futuras

La aplicación directa del sistema es crear una red global de gestión de identificación, para ello se tendría que estandarizar diversos componentes implementados, como el formato de la firma digital, relleno del texto cifrado, creación de certificados X.509. A su vez se tendrían que crear las distintas autoridades de certificación y los repositorios para los certificados digitales.

Introducción en el sistema de las librerías PKCS #11 o CSP, esto nos lleva a tener directorios con certificados y poder llevar a cabo un mejor manejo de estos gracias a estas librerías. Como las librerías son un estándar, cualquier tarjeta que las soporte se podría utilizar en el sistema. Aumentándose de esta manera el número de tipos de tarjetas admitidas.

Proporcionar al usuario un sistema dual, donde un usuario pueda utilizar indistintamente su DNIe (en el caso español) o la tarjeta proporcionada STARCOS, para identificarse en el sistema y poder utilizar la aplicación de autenticación y firma.

Agrupar las operaciones de los programas en un servidor central, que tomará el rol de tercero de confianza [Figura 8], interconectando a los usuarios. Por ejemplo, esta entidad, será la encargada de suministrar los certificados X.509 almacenados y firmados por ella misma del USER1 al USER2.

Implementar la funcionalidad biométrica, como se comentó en el último punto del apartado 5.1. Añadiendo de esta manera más seguridad al proceso de autenticación. Como se comentó, se podría introducir un proceso de autenticación “compuesto”, formado por autenticación en un primer momento por código PIN y en segunda instancia realizar una identificación biométrica.

7. Bibliografía

[1] **Smart Card HandBook.**

Wolgang Rank & Wolfgang Effing. John Wiley &Sons

[2] **La Tecnología de las Tarjetas Inteligentes**

Raúl Sánchez Reíllo, José Carlos Acedo Jiménez, David Cerezo Quesada, Xoan R. Rodríguez Lorenzo

[3] **Certificados digitales y Tarjetas Inteligentes.**

Carlos Verdes Blanco.

[4] **Criptografía Para Principiantes**

José de Jesús Ángel Ángel.

[5] **Seguridad Documental y DNIe en Europa.**

Julián Pérez Muñoz, Pablo F. Pérez Trullós, Pedro J. Latasa López, Santiago Castaño Matilla

[6] **Criptografía y Seguridad en Computadores**

Manuel José Lucena López.

[7] **Criptografía, certificado digital y firma digital. Guía básica de supervivencia**

[8] **Identificación biométrica y su unión con las tarjetas inteligentes.**

Sánchez Reíllo, Raúl.

[9] **Manual Librerías STARCOS 4.0**

[10] **Manual programador STARCOS 2.4**

[11] **Manual programador Toolkit Biométrico STARCOS.**

[12] **Manual programador Precise 100SC.**

[13] **AES estándar**

http://csrc.nist.gov/Fpublications/fips/fips197/fips-197.pdf

[14] **- PKCS #1: RSA Cryptography Standard**

http://www.rsa.com/rsalabs/node.asp?id=2125

[15] **Logran romper cifrado RSA de 1024 bits**

http://bitelia.com/2010/03/logran-romper-encriptacion-rsa-de-1024-bits

[16] **1024 bit RSA Cracked, new Milestone**

http://techie-buzz.com/tech-news/1024-bit-rsa-cracked.html

[17] **Internet X.509 Public Key Infrastructure Certificate and CRL Profile**

http://www.ietf.org/rfc/rfc2459.txt

[18] **ISO X.509v2 v3**

http://www.ietf.org/rfc/rfc2459.txt

[19] **Proteja sus datos con código administrado y las API de tarjetas inteligentes de Windows Vista**

http://msdn.microsoft.com/es-es/magazine/cc163521.aspx

[20] **Using the winscard.dll PC/SC API in Visual Basic 6**

http://www.rolbe.com/2009/03/01/vb6-winscard-api/

[21] **La biblia de C#**

Anaya. Jeff Ferguson Brian Patterson Jason Beres.

[22] **Openssl - Presentation Transcript**

http://www.slideshare.net/danitxu/openssl-283546

[23] **Teorema chino del resto**

http://es.wikipedia.org/wiki/Teorema_chino_del_resto

[24] **Herramienta Creación de certificados (Makecert.exe)**

http://msdn.microsoft.com/es-es/library/bfsktky3%28VS.80%29.aspx

[25] Crear Certificados SSL

http://en.juantxu.net/doku.php/ssl

[26] Project

http://www.koders.com/info.aspx?c=ProjectInfo&pid=AZEWB6HN517P73EAHBMRGM7EZA

[27] PC/CTI STARCOS manual

[28] Manual itextSharp

http://itextpdf.com/

[29] Long-Running Operations

http://codeidol.com/csharp/windows-forms-programming/Multithreaded-User-Interfaces/Long-Running-Operations/

[30] Give Your .NET-based Application a Fast and Responsive UI with Multiple Threads

http://msdn.microsoft.com/en-us/magazine/cc300429.aspx

[31] ISO 7816

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29257

[32] El algoritmo RSA y la factorización de números grandes

http://www.opendomo.com/dlerch/sources/doc/algoritmo_rsa.html

[33] Data Encryption Standard

http://www.itl.nist.gov/fipspubs/fip46-2.htm

[34] RSA Standard

http://www.rsa.com/rsalabs/node.asp?id=2125

[35] Shannon One-Time Pad

http://en.wikipedia.org/wiki/One-time_pad

[36] ETSI TS 101 733 V1.5.1 (2003-12) Electronic Signatures and Infrastructures (ESI); Electronic Signature Formats

http://portal.etsi.org/docbox/ec_files/ec_files/ts_101733v010501p.pdf

[37] *TS 101 903* **- V1.2.2 - XML Advanced Electronic Signatures (XAdES)**

http://uri.etsi.org/01903/v1.2.2/ts_101903v010202p.pdf

8. Anexos

Se acompaña la memoria con dos anexos, que serán referenciados desde la misma. La primera parte son las imágenes de la ejecución del programa y la segunda parte es un ejemplo de firma.

8.1 IMÁGENES

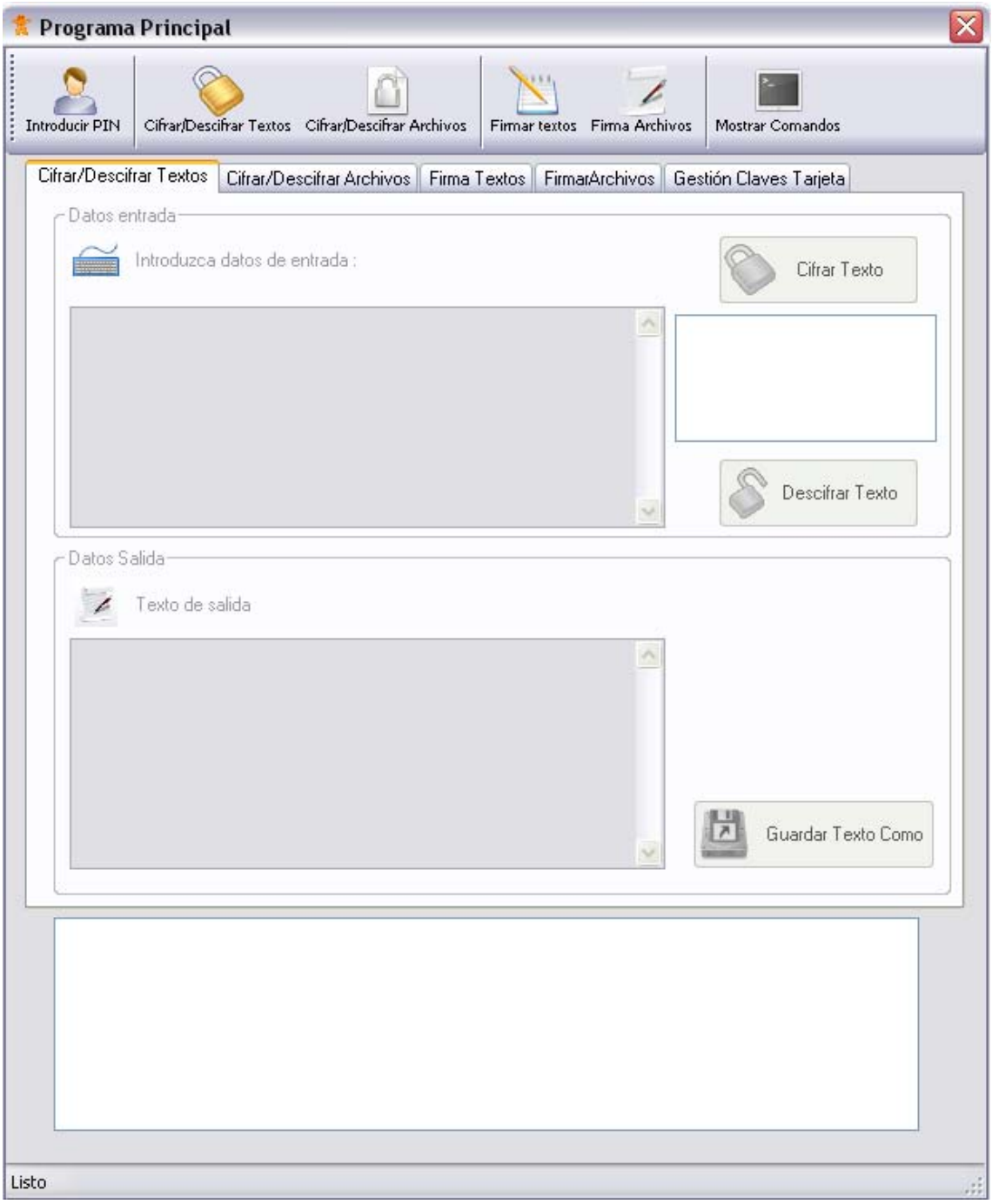

FIGURA 49. VISTA PRINCIPAL EJECUCION PROGRAMA PRINCIPAL.

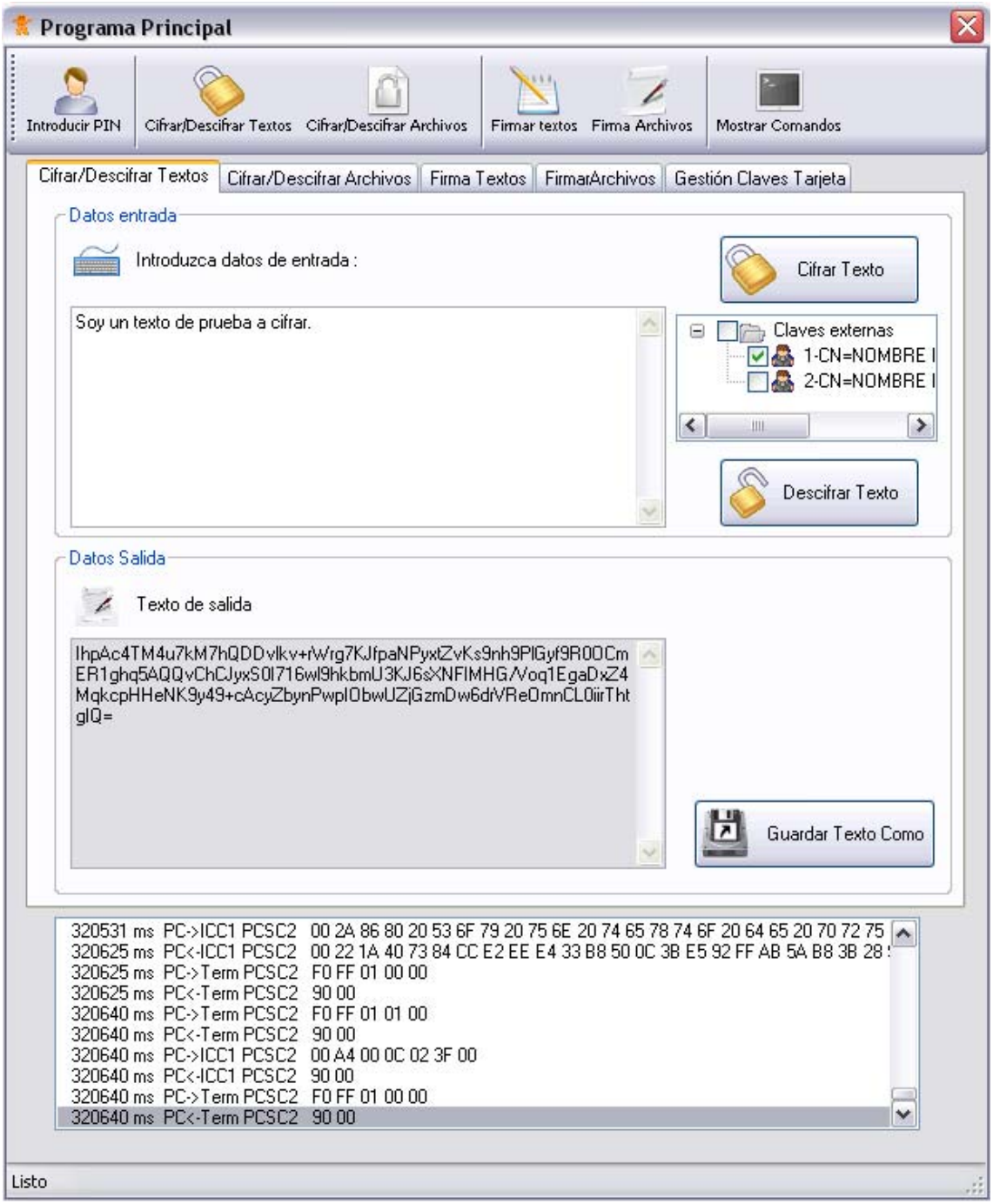

FIGURA 50. PESTAÑA CIFRAR/DESCIFRAR TEXTOS

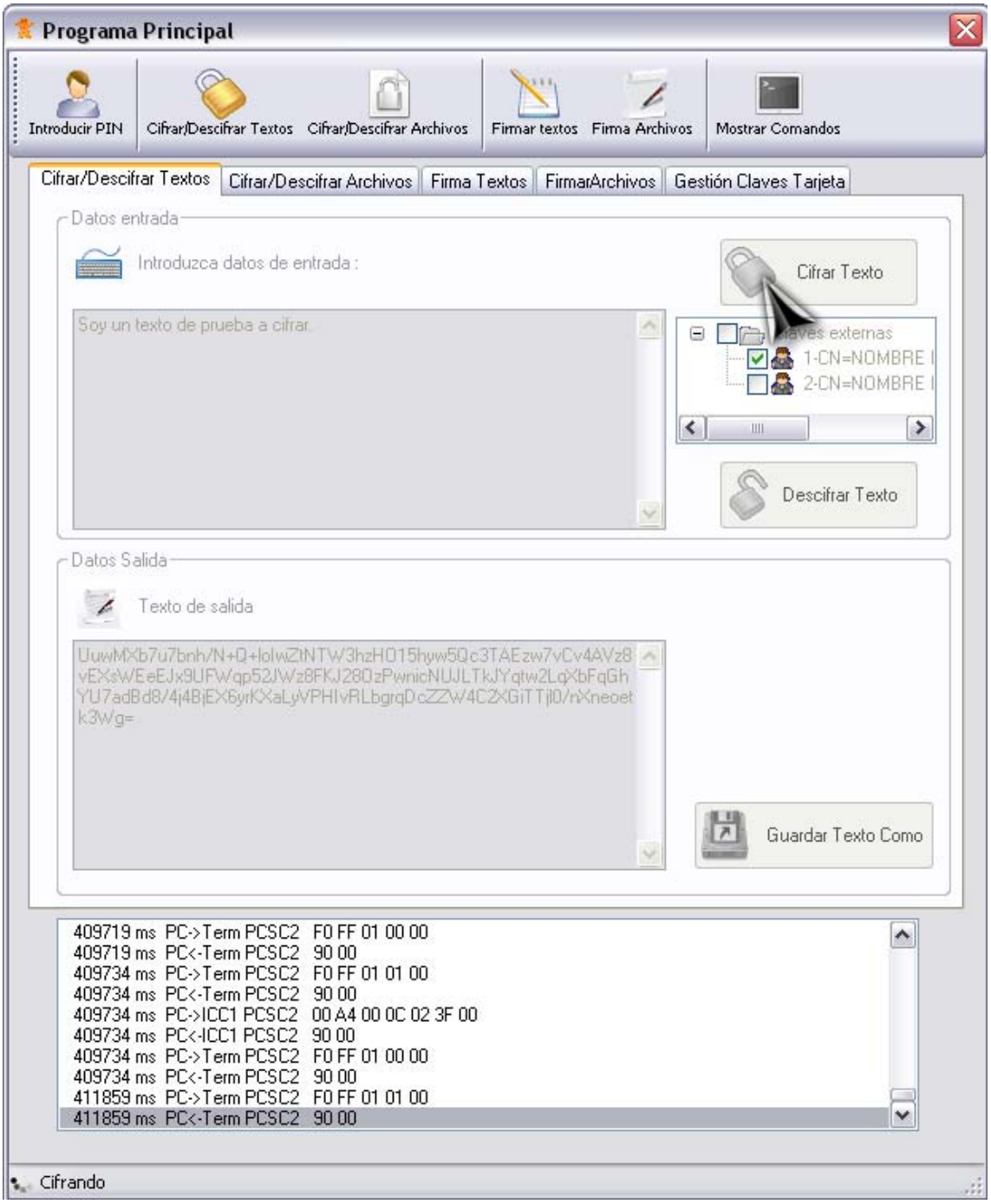

FIGURA 51. PESTAÑA CIFRAR/DESCIFRAR TEXTOS EN EJECUCIÓN.

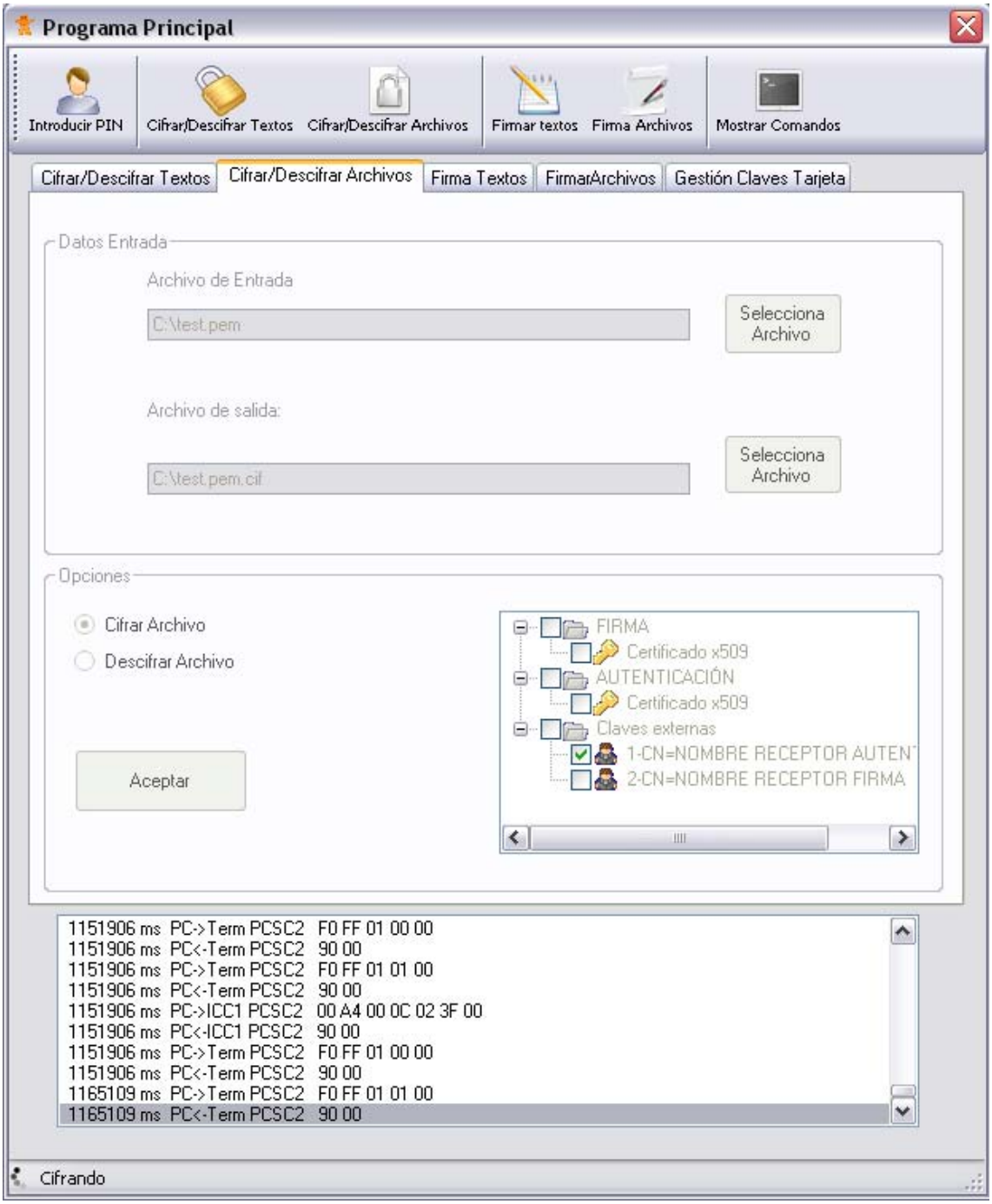

FIGURA 52. PESTAÑA CIFRAR DESCIFRAR ARCHIVOS.

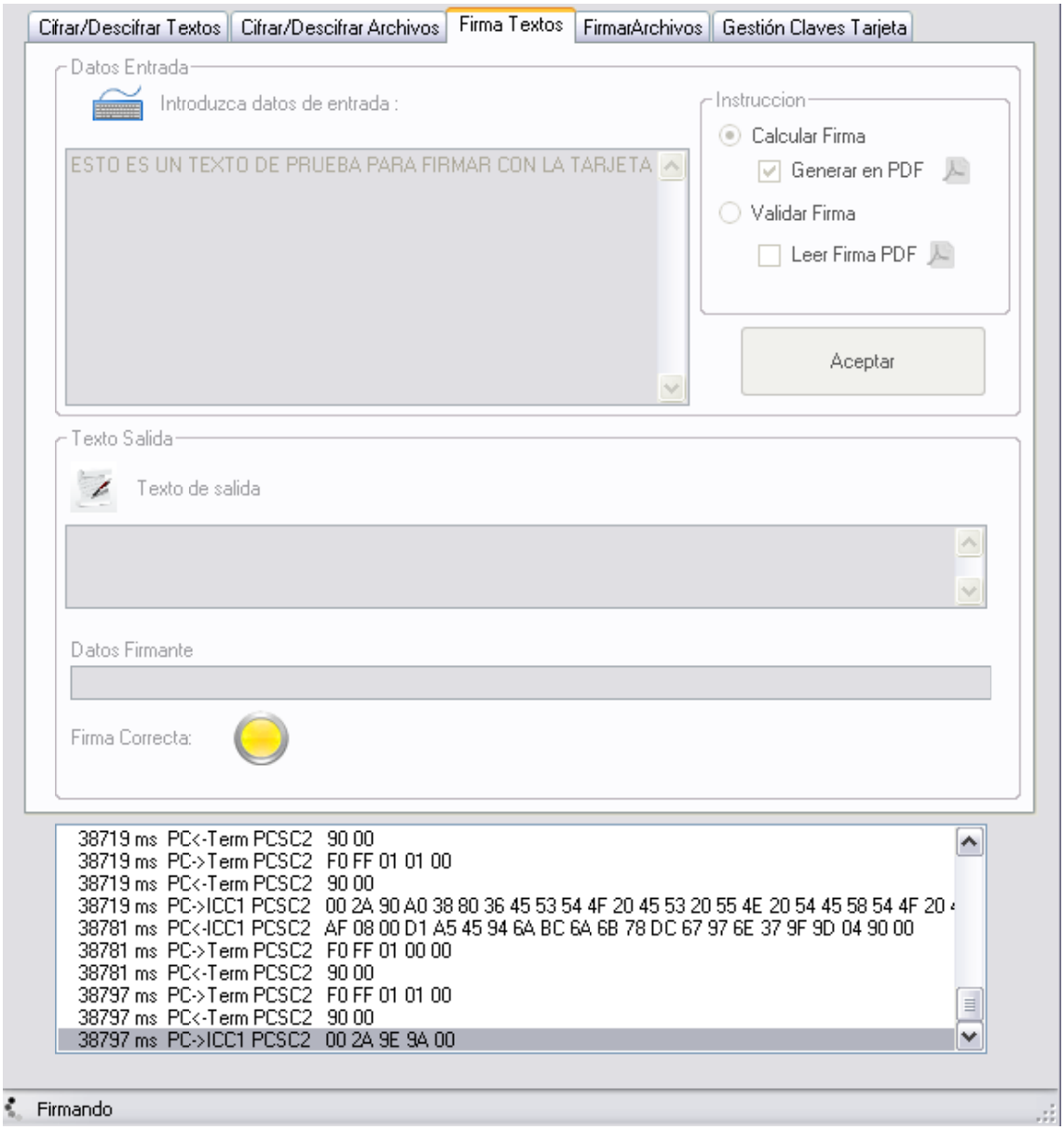

FIGURA 53. PESTAÑA FIRMA DE TEXTOS.

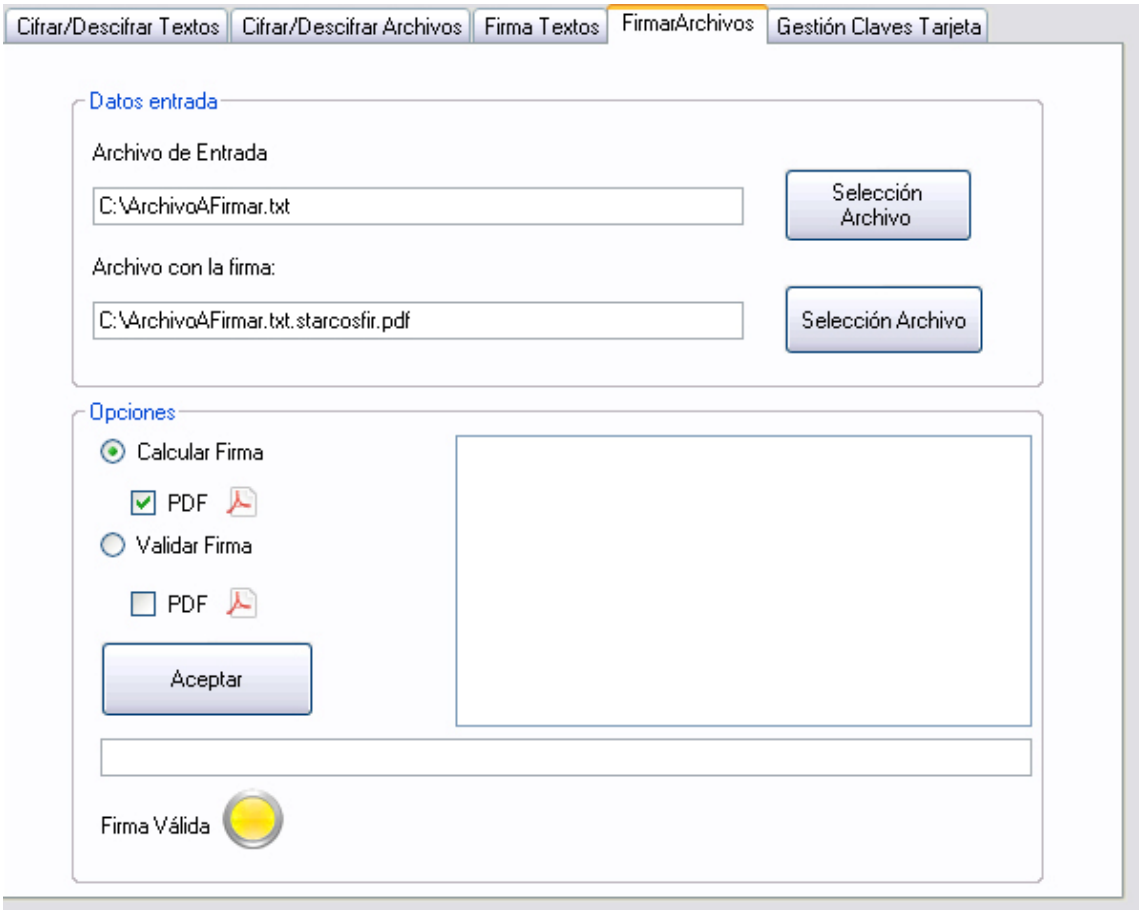

FIGURA 54. PESTAÑA FIRMAR ARCHIVOS

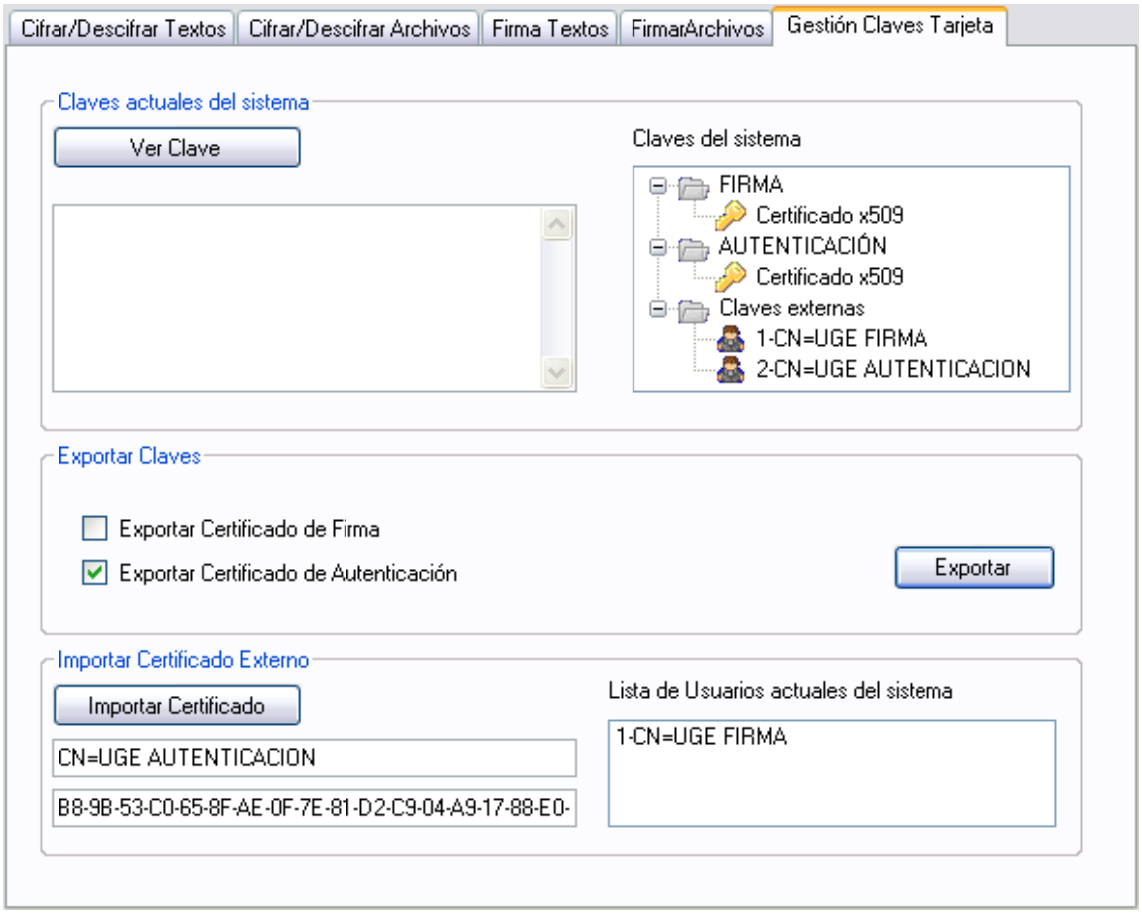

FIGURA 55. PESTAÑA GESTIÓN CLAVES TI.

8.2 Generación PDF

Usuario :
CN=FIRMA_USER_1
Fecha :
17/08/2010 18:12:28
Tipo Tarjeta :
Tarjeta Starcos SPK2.4

----BEGIN TEXT----
ESTO ES UN TEXTO DE PRUEBA PARA FIRMAR CON LA TARJETA
----END TEXT----
----BEGIN SIGNATURE----
O9YtHwXAvrHQVNpGgSQYK833nMrrK5oU3A4/TTvSjgQCQJoO0Ejmx9+YDbv3FAxqGc1LAIsFl9ZZLgY4CHwAtdSbZO
DxWvo4VIOUQTkuVTR1C362+wXsmTm5veahPRq2aRcfIkvc/h7+tWn904MfYQSVVhY7BwckszlMCyKE6pE=
----END SIGNATURE----

FIGURA 56. ARCHIVO PDF GENERADO.