UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

ESTUDIO Y REALIZACIÓN DE MECANISMOS DE SEGURIDAD PARA LAS TARJETAS INTELIGENTES (*Smart Cards*), BASADOS EN EL ALGORITMO AES (*Advanced Encryption Standard*)

T E S I S

QUE PARA OBTENER EL GRADO DE:
MAESTRO EN INGENIERÍA (ELÉCTRICA)
PRESENTA:
MANUEL ALEXANDRO MORENO LIY

Director de Tesis: Dr. Franciso J. García Ugalde

CIUDAD UNIVERSITARIA Agosto de 2006

AGRADECIMIENTOS

A mi esposa Olga,
por el apoyo y sus continuos llamados a no claudicar en la realización de este trabajo.

A mi Familia,
por todo el apoyo recibido para mi formación humana y académica.

A la Facultad de Ingeniería UNAM,
por inculcarme los más altos valores éticos del ser humano.

Al Posgrado de Ingeniería de la UNAM,
por ser una casa abierta para el aprendizaje e investigación.

Al Dr. Francisco García Ugalde,
por su enorme comprensión, tiempo compartido, y sus acertados consejos en la dirección de este trabajo.

A mis sinodales,
por el tiempo y sus valiosas sugerencias
para enriquecer este trabajo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Es ampliamente reconocido que la seguridad en los datos está jugando un papel importante en el diseño de sistemas de tecnología de información, lo que implica la adopción de mecanismos de protección que proporcionen los servicios de control de acceso, confidencialidad, integridad, disponibilidad y no repudio, y que al mismo tiempo son ya un factor de competitividad en el mercado.

La Criptografía es una respuesta ante estos retos de seguridad, sin embargo su aplicación en Tarjetas Inteligentes representa varios problemas que hay que solventar como el rendimiento y las restricciones de cómputo de la misma arquitectura (memoria y procesamiento). Las Tarjetas Inteligentes son de los más recientes avances en lo que se refiere a la tecnología de información. Su tamaño se asemeja al de una tarjeta de crédito y tiene empotrado un *chip* que le permite almacenar datos y comunicarse a través de un lector con una computadora.

En este trabajo de tesis se realiza un estudio y realización de los mecanismos de seguridad para Tarjetas Inteligentes con base en el algoritmo de cifrado simétrico por bloques AES (*Advanced Encryption Standard*). El AES sustituye a la norma oficial de cifrado de Estados Unidos que se conoce como el DES (*Data Encryption Standard*) debido a que ofrece un mayor nivel de seguridad en cuanto al tamaño de la llave y su flexibilidad para implementarse en cualquier plataforma disponible actualmente.

El algoritmo *Rijndael* resultó el ganador del proceso de selección del AES, y cuenta con una estructura iterativa que maneja una longitud variable del bloque de datos y de la llave. En este trabajo se desarrollaron las rutinas de cifrado directo y cifrado inverso del algoritmo para un tamaño de 128 *bits* tanto para el bloque de datos como para la llave.

Se utilizó la tarjeta *ATMega163* que fue adquirida previamente por la Universidad Nacional Autónoma de México. Esta tarjeta pertenece a la familia de tarjetas *Funcard* y cuenta con un microprocesador *AVR AT90S8515* de 8 *bits* tipo *RISC* (*Reduced Instruction Set Computer*) de la empresa ATMEL.

La optimización del código buscó en todo momento su implantación real en la tarjeta *ATMega163* con resultados aceptables en el uso de los recursos disponibles. La primera decisión importante para lograr los objetivos de eficiencia y rendimiento de este trabajo fue la de realizar la implementación del algoritmo *Rijndael* en lenguaje ensamblador para contar con mayor control sobre la lectura y escritura de datos en memoria y sobre la correspondiente manipulación de los datos dentro de cada una de las operaciones del algoritmo. El ambiente de desarrollo *AVR Studio 4*, el cual es proporcionado de manera gratuita por la misma empresa ATMEL, fue una pieza muy importante que permitió realizar la simulación del código producido.

El estudio del conjunto de instrucciones del microprocesador *AVR AT90S8515* permitió implementar, casi en su totalidad, las operaciones del algoritmo con instrucciones directas sobre los datos de los registros, disminuyendo los ciclos de reloj empleados por las operaciones de lectura y escritura de resultados temporales a la memoria *SRAM*.

Por otra parte, se analizaron las estrategias de optimización del algoritmo *Rijndael* a fin de seleccionar aquellas que permitieran los mejores beneficios en cuanto a rapidez de

ejecución y uso de recursos de la tarjeta *ATMega163*. Como resultado, se crearon dos versiones del cifrado directo: la primera utiliza una tabla de búsqueda, y la segunda utiliza el desplazamiento lógico hacia la izquierda y hace el manejo del posible *bit* de acarreo. Cada una de estas versiones tiene ventajas y desventajas en cuanto a la seguridad y a la eficiencia en el uso de recursos.

Para el cifrado directo se obtuvieron tiempos de ejecución más bajos que los reportados en otros trabajos similares. Un logro importante en este trabajo de tesis fue la creación de la rutina de cifrado inverso (descifrado). Ambas rutinas fueron incorporadas a nivel de comandos en el sistema operativo *SOSSE* de dominio público, logrando así su integración en la tarjeta *ATMega163*.

Introducción

En la década de los ochenta la computadora personal significó un cambio muy importante en la economía al proporcionar el procesamiento de grandes cantidades de datos para la automatización de tareas. En la actualidad las redes juegan un papel muy importante en la mayoría de las actividades del ser humano ya que hacen posible que la información se encuentre disponible en todo lugar y en todo momento. Es ampliamente reconocido que la seguridad en los datos está jugando un papel importante en el diseño de sistemas de tecnología de información, esto implica la adopción de mecanismos de protección que proporcionen los servicios de control de acceso, confidencialidad, integridad, disponibilidad y no repudio, y que al mismo tiempo son ya un factor de competitividad en el mercado.

Actualmente se tiene la opción de trabajar con sistemas empotrados en los cuales se ejecutan una gran cantidad de aplicaciones. Además, el crecimiento explosivo de las comunicaciones digitales ha traído consigo nuevos retos para la seguridad. Muchas de estas aplicaciones se sustentan invariablemente en mecanismos de seguridad. Entre las más importantes podemos mencionar seguridad para teléfonos inalámbricos, esquemas de protección de copias para productos de audio y video, entre otros.

En muchas aplicaciones las Tarjetas Inteligentes, como un tipo de sistema empotrado, son utilizadas como dispositivos seguros portátiles. Para su seguridad, las aplicaciones hacen uso de generadores y verificadores MAC *(Message Authentication Code)* y del cifrado y descifrado de datos mediante el uso de un algoritmo de cifrado por bloques.

La Criptografía es una respuesta ante estos retos de seguridad, sin embargo su aplicación en Tarjetas Inteligentes representa varios problemas que hay que solventar. En primer término tenemos el factor de rendimiento. En este caso, un rendimiento aceptable o adecuado para las aplicaciones está en función de los algoritmos de cifrado eficientes. En segundo término tenemos el factor de costo. En las Tarjetas Inteligentes existen restricciones en el tamaño y capacidad de los recursos de cómputo (procesamiento, memoria, entre otros). Entre más recursos de cómputo tenga un sistema empotrado más caro es, aunque obviamente más versátil.

Por lo tanto la implantación de algoritmos de cifrado en Tarjetas Inteligentes debe ser capaz de explotar al máximo los recursos de cómputo disponibles.

Resumen

Es ampliamente reconocido que la seguridad en los datos está jugando un papel importante en el diseño de sistemas de tecnología de información, lo que implica la adopción de mecanismos de protección que proporcionen los servicios de control de acceso, confidencialidad, integridad, disponibilidad y no repudio, y que al mismo tiempo son ya un factor de competitividad en el mercado.

La Criptografía es una respuesta ante estos retos de seguridad, sin embargo su aplicación en Tarjetas Inteligentes representa varios problemas que hay que solventar como el rendimiento y las restricciones de cómputo de la misma arquitectura (memoria y procesamiento). Las Tarjetas Inteligentes son de los más recientes avances en lo que se refiere a la tecnología de información. Su tamaño se asemeja al de una tarjeta de crédito y tiene empotrado un *chip* que le permite almacenar datos y comunicarse a través de un lector con una computadora.

En este trabajo de tesis se realiza un estudio y realización de los mecanismos de seguridad para Tarjetas Inteligentes con base en el algoritmo de cifrado simétrico por bloques AES (*Advanced Encryption Standard*). El AES sustituye a la norma oficial de cifrado de Estados Unidos que se conoce como el DES (*Data Encryption Standard*) debido a que ofrece un mayor nivel de seguridad en cuanto al tamaño de la llave y su flexibilidad para implementarse en cualquier plataforma disponible actualmente.

El algoritmo *Rijndael* resultó el ganador del proceso de selección del AES, y cuenta con una estructura iterativa que maneja una longitud variable del bloque de datos y de la llave. En este trabajo se desarrollaron las rutinas de cifrado directo y cifrado inverso del algoritmo para un tamaño de 128 *bits* tanto para el bloque de datos como para la llave.

Se utilizó la tarjeta *ATMega163* que fue adquirida previamente por la Universidad Nacional Autónoma de México. Esta tarjeta pertenece a la familia de tarjetas *Funcard* y cuenta con un microprocesador *AVR AT90S8515* de 8 *bits* tipo *RISC* (*Reduced Instruction Set Computer*) de la empresa ATMEL.

La optimización del código buscó en todo momento su implantación real en la tarjeta *ATMega163* con resultados aceptables en el uso de los recursos disponibles. La primera decisión importante para lograr los objetivos de eficiencia y rendimiento de este trabajo fue la de realizar la implementación del algoritmo *Rijndael* en lenguaje ensamblador para contar con mayor control sobre la lectura y escritura de datos en memoria y sobre la correspondiente manipulación de los datos dentro de cada una de las operaciones del algoritmo. El ambiente de desarrollo *AVR Studio 4*, el cual es proporcionado de manera gratuita por la misma empresa ATMEL, fue una pieza muy importante que permitió realizar la simulación del código producido.

El estudio del conjunto de instrucciones del microprocesador *AVR AT90S8515* permitió implementar, casi en su totalidad, las operaciones del algoritmo con instrucciones directas sobre los datos de los registros, disminuyendo los ciclos de reloj empleados por las operaciones de lectura y escritura de resultados temporales a la memoria *SRAM*.

Por otra parte, se analizaron las estrategias de optimización del algoritmo *Rijndael* a fin

de seleccionar aquellas que permitieran los mejores beneficios en cuanto a rapidez de ejecución y uso de recursos de la tarjeta *ATMega163*. Como resultado, se crearon dos versiones del cifrado directo: la primera utiliza una tabla de búsqueda, y la segunda utiliza el desplazamiento lógico hacia la izquierda y hace el manejo del posible *bit* de acarreo. Cada una de estas versiones tiene ventajas y desventajas en cuanto a la seguridad y a la eficiencia en el uso de recursos.

Para el cifrado directo se obtuvieron tiempos de ejecución más bajos que los reportados en otros trabajos similares. Un logro importante en este trabajo de tesis fue la creación de la rutina de cifrado inverso (descifrado). Ambas rutinas fueron incorporadas a nivel de comandos en el sistema operativo *SOSSE* de dominio público, logrando así su integración en la tarjeta *ATMega163*.

Hipótesis

Los siguientes puntos forman la hipótesis de este trabajo de tesis:

- Es posible implantar las rutinas de cifrado directo y de cifrado inverso del algoritmo *Rijndael* en una tarjeta inteligente.
- Es posible mejorar el tiempo de ejecución reportado en [10] para el cifrado directo.
- La implantación completa el algoritmo *Rijndael* deberá usar eficientemente los recursos disponibles en la Tarjeta Inteligente de tal forma que se acople directamente a su sistema operativo.
- El cifrado directo y el cifrado inverso del algoritmo *Rijndael* podrán ser usados como primitivas criptográficas para uso de aplicaciones diversas.

Objetivos

Objetivo Principal

Implantar las rutinas de cifrado directo y de cifrado inverso de manera eficiente en una arquitectura específica de Tarjeta Inteligente.

Objetivos Específicos

- Estudiar la tecnología de las Tarjetas Inteligentes.
- Estudiar la teoría de los algoritmos de cifrado simétrico por bloques.
- Estudiar el algoritmo de cifrado *Rijndael*, su aplicación y criterios de optimización en Tarjetas Inteligentes.
- Estudiar la arquitectura de la Tarjeta Inteligente *ATMega163* que será usada en la implantación real del algoritmo *Rijndael*.
- Estudiar el sistema operativo de domino público *SOSSE* para Tarjetas Inteligentes.
- Desarrollar las rutinas de cifrado directo y de cifrado inverso del algoritmo *Rijndael* de una forma eficiente para la Tarjeta Inteligente *ATMega163*.
- Mejorar el tiempo de ejecución de la rutina de cifrado directo del algoritmo *Rijndael*, comparándola con trabajos similares.

Justificación

Los beneficios del uso de Tarjetas Inteligentes dependen de la aplicación, por ejemplo telefonía pública y telefonía móvil, banca y transacciones electrónicas, acceso a servicios públicos entre otras.

En general, las aplicaciones soportadas benefician a los usuarios cuando sus estilos de vida tienen un punto en común con tecnologías de acceso a la información y de

procesamiento de pagos. Estos beneficios incluyen la capacidad de administrar o controlar gastos de manera efectiva, reducir el fraude, la papelería y los trámites. En la actualidad, los recursos de una Tarjeta Inteligente brindan la capacidad de ofrecer múltiples servicios haciendo uso de la misma infraestructura por lo que los costos iniciales de despligue y administración de las mismas tarjetas se ven reducidos.

Por estas razones es interesante implantar el algoritmo de cifrado *Rijndael* sobre una arquitectura específica de Tarjeta Inteligente, en este caso la *ATMega163*. Se ha escogido este algoritmo ya que precisamente ha sido el ganador del AES *(Advanced Encryption Standard)* que es el nombre bajo el cual el NIST *(National Institute of Standards and Technology)* de Estados Unidos realizó un proceso de selección abierto para escoger un algoritmo de cifrado por bloques superior en seguridad, rapidez y flexibilidad a los algoritmos DES *(Data Encryption Standard)* y Triple DES o 3DES.

El reto aquí es el desarrollo completo del algoritmo buscando siempre la optimización de los recursos de la Tarjeta Inteligente. Incluso los mismos autores del algoritmo[1] han desestimado la implantación del cifrado inverso en arquitecturas de 8 bits, típicas de una Tarjeta Inteligente, debido a que el cifrado directo tiene mayor diversidad de aplicaciones. Sin embargo, el cifrado inverso es importante ya que permite realizar los procesos de autenticación tanto de la misma tarjeta como del sistema anfitrión de la aplicación.

Método

Para el desarrollo de este trabajo de tesis se tendrá una componente muy importante de investigación en los temas de Tarjetas Inteligentes, antecedentes de Criptografía con enfoque hacia el cifrado simétrico por bloques, el algoritmo *Rijndael*, la arquitectura de la tarjeta *ATMega163*, el sistema operativo *SOSSE*, herramientas de desarrollo, y herramientas para la inserción del código producido en la tarjeta.

El resultado de esta investigación será la implantación real del algoritmo *Rijndael* en la Tarjeta Inteligente buscando cumplir con el objetivo principal y objetivos específicos planteados con anterioridad.

Contenido

El trabajo de tesis se compone de 6 capítulos. En el capítulo 1 se trata el tema de Tarjetas Inteligentes en el que se presenta su historia, clasificación, descripción de componentes, protocolos, seguridad y aplicaciones. En el capítulo 2 se aborda el tema de Criptografía con especial enfásis en el cifrado simétrico o de llave privada. El capítulo 3 está dedicado por completo al algoritmo *Rijndael* en el cual se describe su estructura y operación. Estos tres primeros capítulos tienen la intención de brindar los antecedentes suficientes al lector para facilitar la comprensión del resto de los capítulos.

En el capítulo 4 se describe en detalle la implantación del algoritmo *Rijndael* en la

[1] *Vincent Rijmen y Joan Daemen.*

tarjeta *ATMega163*. Aquí se discuten los factores que se tomaron en cuenta para su implantación y optimización, descripción de la arquitectura, las herramientas de desarrollo empleadas, y la integración a nivel sistema operativo de la tarjeta.
Los resultados y conclusiones de este trabajo están contenidos en los capítulos 5 y 6 respectivamente.

Capítulo 1: Tarjetas Inteligentes

Introducción

Las Tarjetas Inteligentes o *Smart Cards*, son de los más recientes avances en lo que se refiere a la tecnología de información. Una Tarjeta Inteligente es un dispositivo del tamaño de una tarjeta de crédito que contiene uno o más circuitos integrados y que también puede emplear una o más de las siguientes tecnologías: banda magnética, código de barras (en una, o en dos dimensiones), transmisores de radiofrecuencia sin contacto, biometría, criptografía, autenticación, o identificación por fotografía.

Los circuitos integrados se encuentran en un *chip*, el cual está empotrado en la tarjeta. Este *chip* actúa como un microcontrolador y tiene la capacidad tanto de almacenar datos, como el mismo sistema operativo de la tarjeta, conocido como *COS* (*Card Operating System*).

Una Tarjeta Inteligente por sí sola no tiene sentido; ésta necesita de elementos externos para que pueda operar y ser de utilidad. Es necesario un lector para que se establezca la comunicación entre la tarjeta y el sistema anfitrión (que normalmente es una computadora personal) en el cual reside la aplicación. La tarjeta y la aplicación, en conjunto, pueden proporcionar mejores niveles de seguridad, la capacidad de registrar, almacenar y actualizar datos, y proporcionar interoperabilidad entre diversos organismos.

La Tarjeta Inteligente puede ofrecer una solución al problema de administración de identidades y también proporciona los medios para realizar la reingeniería de procesos ineficientes con un alto retorno de inversión. Los beneficios por el uso de esta tecnología dependen de la aplicación. En general, las aplicaciones soportadas benefician a la gente en la medida en que su estilo de vida tiene un punto en común con tecnologías de acceso a la información y el procesamiento de pagos. Estos beneficios incluyen la capacidad de administrar o controlar gastos de manera efectiva, reducir el fraude, la papelería, y los trámites. La Tarjeta Inteligente también proporciona la conveniencia de tener una sola tarjeta para acceder a múltiples servicios.

En la actualidad, las áreas de aplicación más importantes de las Tarjetas Inteligentes son la telefonía y las telecomunicaciones, las finanzas, el acceso a diversos servicios, y el control de acceso. Estas aplicaciones serán discutidas con mayor detalle más adelante.

Historia

Las Tarjetas Inteligentes fueron creadas y patentadas en la década de los setenta. Las primeras tarjetas solamente almacenaban datos, dejando gran parte de la responsabilidad de la seguridad en el lector. Los primeros usos en forma masiva ocurrieron en la década de los ochenta en Francia, y se dieron en el sistema público telefónico por medio de tarjetas prepagadas (*Télécarte*) y en el uso de tarjetas de débito con microchip (*Carte Bleue*).

En la década de los noventa surgió en Europa la aplicación de monedero electrónico en

la que la Tarjeta Inteligente guardaba y manipulaba valores. Algunos casos fueron los del Reino Unido con *Mondex*, Alemania con *Geldkarte* y Francia con *Moneo*. Ninguno de estos programas atrajo de manera notable el interés público por lo que su uso no fue importante.

Con la aparición de la norma ISO 7816 en el año de 1986, se estableció el primer paso para regular la operación de las Tarjetas Inteligentes. En 1996 se formó el grupo de trabajo *PC/SC* (*Personal Computer / Smart Card*) por las empresas *Sun Microsystems*, *Toshiba*, *Verifone*, *Hewlett-Packard*, *IBM*, *Microsoft*, *Schlumberger*, *Bull*, *Siemens* y *Gemplus* con la finalidad de facilitar la interacción entre Tarjetas Inteligentes, lectores, y computadoras personales.

Ya con la aparición del *microchip*, se inició en Europa a finales de la década de los noventa el uso amplio de la Tarjeta Inteligente como parte de la telefonía móvil basada en la norma *GSM*, siendo hasta la actualidad la aplicación más popular de las Tarjetas Inteligentes.

Componentes

A la Tarjeta Inteligente se le puede ver como una pequeña computadora con puertos de entrada y salida, sistema operativo, y un lugar para el almacenamiento de datos, tal y como se ilustra en la Figura 1.1.

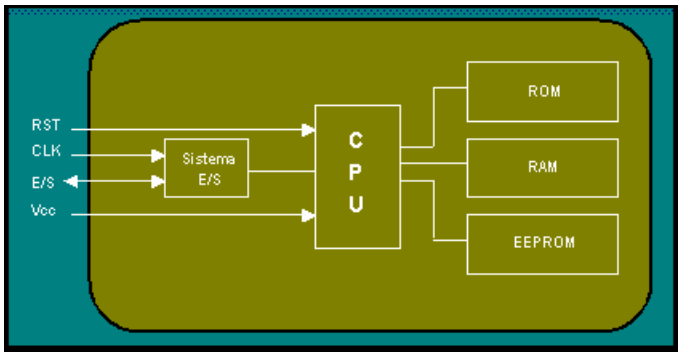

Figura 1.1. Componentes de una Tarjeta Inteligente.

El microprocesador ejecuta al sistema operativo (*Java Virtual Machine, Windows for Smart Card OS, MULTOS*), normalmente son de 8 *bits* (bus de datos) (8051, 6805, HC05, *AVR* entre otros). Aunque para aplicaciones más demandantes se están usando de 16 y 32 *bits*.

Una señal de reloj es usada para manejar la lógica de la tarjeta y como referencia para el enlace de comunicación serial. Comúnmente se usan dos frecuencias de reloj: 3.57 *MHz* y 4.92 *MHz*. La razón de estas frecuencias se debe a la disponibilidad de cristales de bajo costo para formar los circuitos de oscilación de reloj; ambas frecuencias son usadas en los sistemas de televisión (*NTSC* y *PAL*) para la señal portadora de color.

Como en toda computadora, la memoria *RAM* (*Random Access Memory*) es el lugar donde el sistema operativo y las aplicaciones almacenan datos temporales tales como el

estado de las tarjetas, registros de transacción y resultados de los cálculos. La capacidad de la memoria *RAM* va desde los 256 *bytes* a los 2 *Kbytes*, aunque aplicaciones con alta demanda de cómputo requieren hasta 4 *Kbytes*.

La memoria *ROM* (*Read Only Memory*) contiene el sistema operativo, puede almacenar una máquina virtual y algunas aplicaciones en código nativo que no pueden ser modificadas una vez que se ha liberado la tarjeta en producción. Su capacidad va de 8 *Kbytes* a 64 *Kbytes,* o incluso un poco más de acuerdo a la demanda de algunas aplicaciones más sofisticadas. Recientemente se ha introducido la tecnología *Flash* (*Flash EEPROM*) que puede ser añadida, o incluso reemplazar a la memoria *ROM,* para almacenar código de programa. Esta tecnología ofrece una mayor flexibilidad a los desarrolladores de sistemas operativos, ya que el sistema operativo puede ser cargado varias veces en la tarjeta.

La memoria *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*), es una memoria no volátil que puede ser borrada con condiciones especiales de acceso a lectura y escritura. Esta memoria es usada para almacenar datos y partes del sistema operativo que posiblemente necesiten modificarse durante la vida del *chip*, y almacenar datos de programa de usuario como un *applet* de java. La capacidad de una *EEPROM* va de 8 *Kbytes* a 32 *Kbytes*. La tecnología *Flash* está reemplazando rápidamente a las EEPROM debido a su mayor velocidad para las operaciones de escritura y su durabilidad para la retención de datos. Las memorias actuales garantizan un tiempo de retención de datos de al menos 10 años, o de al menos 100,000 ciclos de lectura/escritura.

Para la Interfaz de Entrada/Salida, los datos son enviados o recibidos de manera alterna a través de una sola línea. La transferencia de datos usa un protocolo específico (el cual se presentará posteriormente), y la tasa de transmisión depende de la frecuencia del reloj, actualmente es de 9600 *bps*. El voltaje para la operación de la tarjeta se encuentra entre 3 y 5 *volts*.

Opcionalmente algunas Tarjetas Inteligentes contienen un criptoprocesador dedicado a tareas criptográficas específicas especialmente aquellas relacionadas con cálculos exponenciales y modulares.

Dos son las características que hacen a las tarjetas la mejor opción para aplicaciones en las cuales se requiere la protección de datos. Primero, en una tarjeta la información puede ser procesada sin divulgar los datos de la misma. Segundo, los usuarios pueden llevar datos en la tarjeta sin tener que utilizar otro medio de almacenamiento.

Una tarjeta puede restringir el uso de información hacia una persona autorizada con una contraseña. Sin embargo, si esta información debe ser transmitida por radio frecuencia, o líneas telefónicas, se recomienda cifrar los datos. Algunas Tarjetas Inteligentes son capaces de realizar operaciones de cifrado y descifrado.

Clasificación

En la literatura se pueden encontrar diversos criterios para clasificar a las Tarjetas Inteligentes como por ejemplo: con base en el tipo de interfaz con que se comunican con el lector, de acuerdo al tamaño de la tarjeta, por el tipo de sistema operativo, o por

su tecnología. En la siguiente figura se muestra una clasificación que reúne las principales características de una Tarjeta Inteligente.

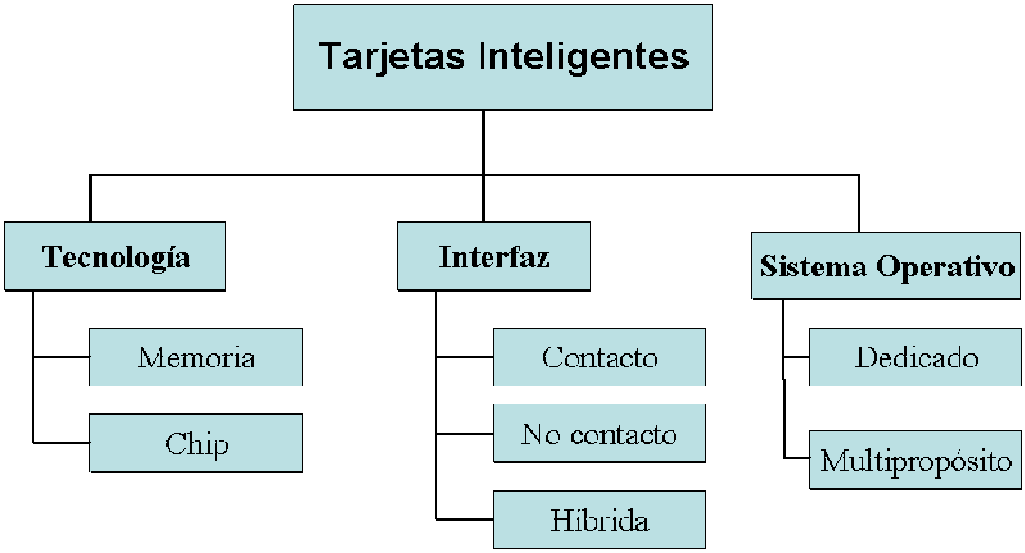

Figura 1.2. Clasificación resumida de las Tarjetas Inteligentes.

Las tarjetas de memoria tienen como funcionalidad principal el almacenamiento de datos en la memoria *EEPROM*. Las primeras tarjetas de este tipo no usaban mecanismos de protección para el acceso a los datos por lo que solamente era necesario enviar los comandos correspondientes al protocolo de comunicación usado. Posteriormente, se añadieron mecanismos de protección como la autenticación del usuario mediante un *PIN* (*Personal Identification Number*), y la implantación de un esquema de lógica segura en la que la tarjeta controla la forma en que se acceden los datos; un ejemplo de esto es el uso de tarjetas prepagadas de telefonía pública en donde solamente se pueden decrementar los valores almacenados.

El uso de un microcontrolador hace que las tarjetas ganen el calificativo de inteligentes, ya que cuentan con la capacidad de procesamiento de datos. El acceso a los datos se controla por el sistema operativo de la tarjeta y por las aplicaciones.

A fin de que la Tarjeta Inteligente pueda comunicarse con el lector e intercambiar datos, ésta posee una interfaz que puede ser de contacto, sin contacto, o híbrida.

La norma ISO 7816-2 define las características eléctricas de la interfaz de contacto. Por esta interfaz se realiza la transferencia de datos y se le proporciona energía a la tarjeta. La Figura 1.3 muestra la disposición de los contactos de acuerdo con esta norma.

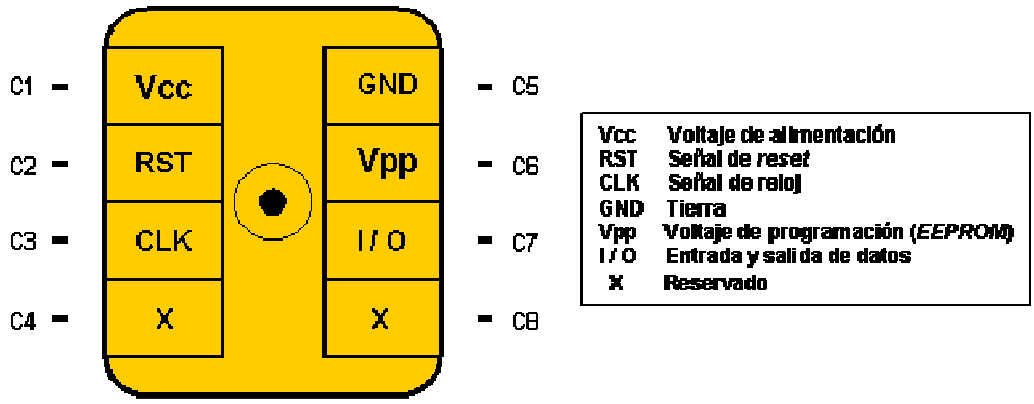

Figura 1.3. Disposición de los contactos de acuerdo con la norma ISO 7816-2.

Las tarjetas con interfaz sin contacto operan por señales electromagnéticas (radiofrecuencia) por medio de las cuales reciben la frecuencia a la que deben operar, y la energía para funcionar en algunos casos. La norma ISO 14443 define las especificaciones para este tipo de tarjetas; sin embargo, existen por lo menos más de diez tipos de tecnología en el mercado que no son compatibles entre sí y que por lo tanto impide el uso amplio de estas tarjetas en la actualidad. Las aplicaciones en las que se requiere la lectura de datos del usuario en un período corto de tiempo, como es el caso de pagos de casetas para el transporte, representan las áreas de oportunidad para el uso de las tarjetas con interfaz sin contacto.

En la actualidad, las tarjetas con interfaz de contacto y las tarjetas con interfaz sin contacto usan distintos protocolos de comunicación. Las tarjetas con interfaz de contacto tienen un mayor nivel de seguridad en el momento de intercambiar datos ya que se requiere que la tarjeta sea insertada en lector, mientras que las tarjetas con interfaz sin contacto proporcionan mayor comodidad de uso. Con la finalidad de conjugar estas ventajas, se implementó la alternativa de empotrar dos *chips* independientes en la misma tarjeta, cada uno manejando su propia interfaz y sin comunicación interna entre ellas, a esta versión se le conoce como tarjeta con interfaz híbrida. Recientemente, ya se cuenta con las dos interfaces en un solo *chip* a fin de reducir costos de producción, lo que se conoce como tarjeta tipo *combi*.

El sistema operativo cumple un papel fundamental en una Tarjeta Inteligente ya que proporciona una capa de abstracción entre el *hardware* y las aplicaciones. En un inicio, el *COS* era monolítico, es decir, se implantaba al momento de crear la tarjeta por lo que resultaba imposible modificarlo posteriormente, y solamente era útil para ejecutar una aplicación específica. Posteriormente, el sistema operativo fue desarrollado con una estructura modular lo que permitió que se pudieran ejecutar diversas aplicaciones para un solo *COS*, o tener un *COS* con comandos especializados para una aplicación en particular como lo es el manejo de imágenes en el sector salud.

Protocolos de Comunicación

En la actualidad existen dos protocolos de comunicación para las Tarjetas Inteligentes:

- Protocolo T = 0 (transmisión asíncrona *half duplex* orientada a caracter)

- Protocolo T = 1 (transmisión asíncrona *half duplex* orientada a bloques de caracteres)

El protocolo T=0 fue el único especificado en la norma ISO 7816-3. Pero en 1992 la ISO incluyó también al protocolo T = 1 como un anexo del primero. Es necesario que la tarjeta y el lector operen con un protocolo común para llevar a cabo una configuración óptima. En principio, esto debe realizarse mediante el uso de un comando especial denominado *PTS* (*Protocol Type Selection*), el cual es enviado desde el lector hacia la tarjeta después de la Respuesta a un *Reset* (*ATR*).

A fin de mantener la compatibilidad con los sistemas que solamente pueden manejar el protocolo T=0, se introdujo un nuevo concepto que identifica dos modos de operación: el modo de negociación y el modo específico. Una tarjeta que opera en el modo de negociación puede ser indicada para que trabaje con el protocolo T=0, o con el protocolo T=1, mediante el uso del comando *PTS*, mientras que aquella que opera en el modo específico no reconoce al comando *PTS*. Aunque puede cambiarse al modo de negociación mediante el comando *Reset*.

El protocolo de comunicación T=0

El lector siempre inicia el comando para el protocolo T=0. La interacción entre el lector y la tarjeta da como resultado una serie de comandos y respuestas sucesivas. Para este protocolo, los datos fluyen solamente en una dirección a la vez en un formato comando – respuesta. La dirección del flujo de datos está implícita en la definición del comando y por lo tanto el lector y la tarjeta necesitan tener el conocimiento necesario a priori. Cuando se requiere transferir datos en ambas direcciones para un comando en particular entonces se utiliza un comando llamado *Get Response* el cual es ejecutado por el lector.

El protocolo T=0 sirve para transmitir datos tanto a nivel aplicación como a nivel enlace del modelo *OSI*. Para este protocolo, se realiza la transmisión de cada *byte* de manera independiente y en un formato llamado *APDU* (*Application Data Unit*); el mensaje de comando consiste de una cabecera de cinco caracteres que es enviado por el lector hacia la tarjeta. La tarjeta contesta con un *byte* de procedimiento después del cual los datos son enviados desde, o hacia la tarjeta. El flujo de mensajes para el protocolo T=0 se muestra en la Figura 1.4. La cabecera de comando consiste de los siguientes cinco *bytes*:

CLA - Clase de instrucción
INS - El código de instrucción (por ejemplo leer memoria)
P1 - Calificador de instrucción (por ejemplo dirección de memoria)
P2 - Calificador de instrucción adicional
P3 - Longitud del bloque de datos

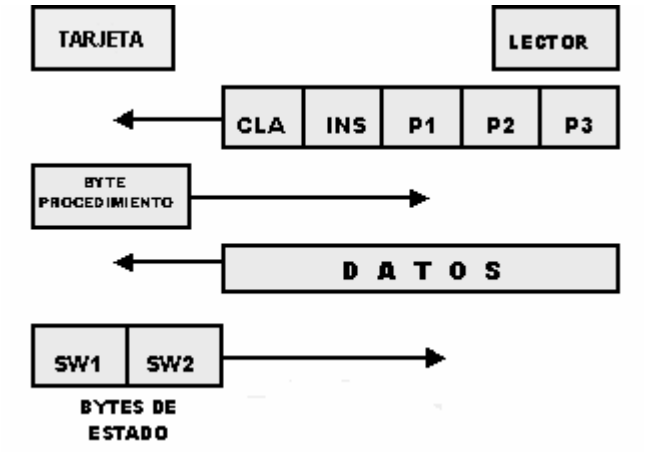

Figura 1.4. Operación del protocolo T=0 del lector hacia la Tarjeta Inteligente y viceversa.

Existen dos *bytes* de status *SW1* y *SW2*. Estos *bytes* son enviados desde la tarjeta hacia el lector como complemento del comando para indicar el estado actual de la tarjeta. La respuesta normal es la siguiente:

$SW1, SW2 = 90_{hex}, 00_{hex}$

Cuando *SW1* tiene el valor $6X_{hex}$, o el valor $9X_{hex}$, se reportan varias condiciones de error por la tarjeta. La norma ISO 7816-3 define 5 condiciones de error:

$SW1=6E_{hex}$ - La tarjeta no soporta la clase de instrucción
$SW1=6D_{hex}$ - Código inválido de instrucción
$SW1=6B_{hex}$ - Referencia incorrecta
$SW1=67_{hex}$ - Longitud incorrecta
$SW1=6F_{hex}$ - Sin diagnóstico en particular

El protocolo T=0 incluye un mecanismo de control de paridad muy sencillo para detección y corrección de errores. Este mecanismo consiste en realizar la operación lógica *XOR* de los *bits* recibidos y cuyo resultado es verificado con la paridad esperada. Si hubo un error, entonces se asume que existió un número impar de errores y la línea de Entrada/Salida se pone a un nivel lógico bajo dentro del primer tiempo de guardia. El lector observa esta condición y retransmite el caracter.

El protocolo de comunicación T=1

En esencia lo que hace el protocolo T=1 es empaquetar un bloque de caracteres, lo cual permite las funciones de control de flujo, encadenamiento de bloques, y corrección de errores.

La elección del protocolo de comunicación para la tarjeta no es algo sencillo ya que se tienen que considerar las ventajas que el protocolo T=1 puede ofrecer y después examinar el precio que se tiene que pagar.

La ventaja más obvia del protocolo T=1 es la habilidad para manejar el flujo de datos en ambas direcciones (aunque no al mismo tiempo). En el protocolo T=0 se mostró que

para un comando en particular, ese es el dato que es enviado, o recibido, por la tarjeta. Esta limitante se debe al uso de un solo *byte* para definir la longitud del dato relacionado con el comando.

El protocolo T=1 no presenta la restricción del protocolo T=0 en el que se tiene una relación maestro-esclavo, la cual obliga a que el lector siempre inicie un comando al cual la tarjeta debe responder. Para el protocolo T=1 un comando puede ser iniciado por cualquier parte respetando las restricciones del mismo protocolo.

Otra ventaja del protocolo T=1 es la capacidad de encadenar los bloques de datos de tal manera que un bloque de datos de una longitud arbitraria pueda ser transferido como resultado de un solo comando por la transmisión del número apropiado de tramas encadenadas en secuencia.

También el protocolo T=1 cuenta con un sistema de manejo de errores más sofisticado. Este permite el uso de un código de detección de errores por bloque (*EDC*) y la capacidad para retransmitir bloques que son sujetos a alguna condición de error. La detección de errores se lleva a cabo por medio del algoritmo *LRC* (*Longitudinal Redundancy Check*), o por el algoritmo *CRC* (*Cyclic Redundancy Check*). A manera de comparación, el protocolo T=0 tiene un esquema de detección y corrección de errores más primitivo.

Claramente existe un precio a ser pagado por este protocolo. Aparte de la mayor complejidad en el *software* requerido por la tarjeta y el lector, el protocolo T=1 demanda una mayor cantidad de memoria *RAM* de la tarjeta, ya que ésta necesita mantener el último bloque enviado en caso de una retransmisión debida a una condición de error.

En general el protocolo T=1 ofrece ventajas en donde la aplicación maneja un número considerable de bloques de datos, particularmente cuando se requiere transferir datos en ambas direcciones como parte de un comando en específico. La eficiencia del protocolo solamente es visible cuando se transmiten grandes cantidades de datos, ya que a nivel de la capa física todavía se opera en modo carácter tal y como lo realiza el protocolo T=0.

La trama del protocolo T=1 consiste de tres campos tal y como se ilustra en la Figura 1.5.

Prólogo			Información	Epílogo
NAD	*PCB*	*LEN*	*APDU*	*EDC*
1 byte	*1 byte*	*1 byte*	*0 – 254 bytes*	*1 – 2 bytes*

Figura 1.5. Campos de la trama del protocolo T=1.

El campo de prólogo se forma por tres *bytes*: el primero corresponde a la dirección del

nodo (*Node Address*) y que sirve para identificar flujos múltiples de comunicación, el segundo es el *byte* de control de protocolo (*Protocol Control Byte*) y sirve para especificar el tipo de bloque que es enviado, así como algunos otros parámetros de comunicación, y finalmente el tercero corresponde a la cantidad de datos que son enviados (0 a 254 *bytes*). El campo de información es donde se envían los datos de la aplicación, y el campo de epílogo corresponde al código de detección de errores.

A nivel aplicación el Protocolo T=1 hace uso de una *APDU* (*Application Data Unit*) similar a la descrita en el Protocolo T=0.

Sistema de archivos

El sistema de archivos es uno de los componentes más importantes en las Tarjetas Inteligentes para el almacenamiento de datos. La norma ISO 7816-4 especifica un sistema de archivos de tipo jerárquico cuya estructura se muestra a continuación.

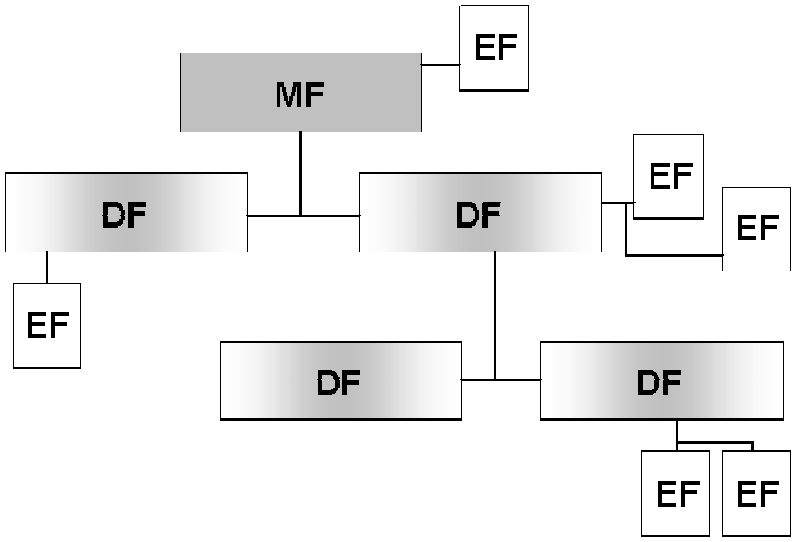

Figura 1.6. Esquema del sistema de archivos bajo la norma ISO 7816-4.

Como en cualquier sistema de archivos jerárquico se tiene un directorio raíz llamado Archivo Maestro (*Master File*), el cuál es el único de su clase dentro de la tarjeta, a fin de que las aplicaciones tengan un solo punto de referencia para acceder a los demás archivos. Del Archivo Maestro se desprenden los directorios conocidos como Archivos Dedicados (*Dedicated Files*), y los archivos normales, o Archivos Elementales (*Elementary Files*).

Los Archivos Dedicados agrupan a los Archivos Elementales pertenecientes a una aplicación en particular. Los Archivos Elementales se dividen en:

- Archivos Transparentes, los cuales son tratados como una secuencia de *bytes*; éstos son parecidos a los archivos regulares del sistema operativo *UNIX* o *Windows* y su tamaño mínimo es de un *byte*.
- Archivos Lineales Fijos, los cuales son tratados como una secuencia de registros de longitud fija. Estos archivos son útiles cuando se accede a tamaños de bloque

específicos de información de manera frecuente. El tamaño de cada registro está en el rango de 1 a 254 *bytes*, pero todos los registros tienen el mismo tamaño.

- Archivos Lineales Variables, los cuales son tratados como una secuencia de registros de longitud variable. Estos archivos son útiles cuando no se conoce de antemano el tamaño de los registros que se utilizarán.
- Archivos Cíclicos, son una especie de anillo, los cuales son tratados como una secuencia sin fin de registros de longitud variable. Estos archivos son útiles cuando se mantiene el registro de ciertas acciones como transacciones y accesos.

Los Archivos Elementales también pueden contener datos del sistema operativo, llaves para el cifrado, y código de programa. Todos los archivos se componen de dos partes: la cabecera y el cuerpo. La cabecera contiene información de sistema tal como el identificador de dos *bytes* de longitud para hacer referencia al archivo mismo, el tipo de archivo, la estructura del archivo (por ejemplo si está compuesto de tres registros de longitud de 10 *bytes*), las condiciones de acceso al archivo, y la ubicación del archivo en la estructura jerárquica. El cuerpo está formado por los *bytes*, o registros, que contienen los datos.

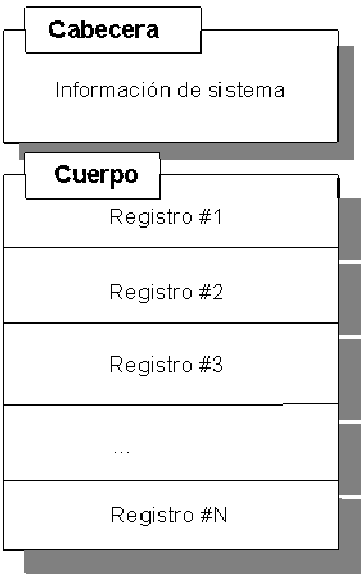

Figura 1.7. Estructura de un archivo en la norma ISO 7816-4.

Aplicaciones

Las Tarjetas Inteligentes han tenido una rápida expansión debido al Internet y al comercio electrónico. Hoy en día, las tarjetas tienen una mayor aceptación en aplicaciones que necesitan el almacenamiento seguro de información sensible como contraseñas, certificados digitales y dinero electrónico.

A continuación se describirán las principales áreas de aplicación de las Tarjetas Inteligentes en la actualidad, las cuales se dividen en: pagos electrónicos, seguridad y autenticación, transporte, telefonía y telecomunicaciones, programas de lealtad, y servicios al sector salud.

Pagos electrónicos

Dentro de las aplicaciones de pagos electrónicos destaca la del monedero electrónico. Los fondos son cargados en la tarjeta para ser usados como dinero en efectivo. De esta forma la tarjeta puede ser empleada para hacer compras sin que necesariamente se utilice un *PIN*, lo cual hace que la transacción sea mucho más rápida a costa del riesgo por robo de la misma tarjeta. Por esto último es que en el monedero electrónico solamente se usan cantidades pequeñas de dinero a fin de reducir el impacto por pérdida.
Las tarjetas Proton de Banksys, VisaCash de Visa, y Mondex de Mastercard son esfuerzos para extender el uso del monedero electrónico a fin de reducir el costo de operación de los bancos y establecimientos.

Frecuentemente los programas de lealtad son incorporados en tarjetas expedidas por líneas aéreas y tiendas departamentales, y acompañan a las aplicaciones de monedero electrónico. Con los programas de lealtad, los comerciantes otorgan los llamados "puntos" por las compras realizadas, los cuales pueden ser utilizados para adquirir otros bienes y servicios del mismo comerciante.

Otro ejemplo de uso de las tarjetas en el rubro de pagos electrónicos es el boleto, o *token*, electrónico. En esta aplicación, se almacena el boleto electrónico en la memoria de la tarjeta, el cual puede ser recargado dependiendo del tipo de memoria. También es posible almacenar distintos boletos para diferentes servicios a fin de sacar el máximo provecho de la tarjeta.
De esta manera la tarjeta puede ser utilizada para realizar pagos en gasolineras, en los parquímetros, o viajes en el sistema urbano de transporte. Esto reduce en gran medida la operación por el manejo de monedas así como también se reduce el riesgo por robo.

Seguridad y autenticación

Los fraudes realizados mediante el uso de tarjetas, o lectores falsos, han obligado a buscar mecanismos para autenticar a estos dispositivos. Los algoritmos de cifrado han brindado una solución a este problema, y de igual manera han obligado a que las Tarjetas Inteligentes tengan los suficientes recursos de cómputo para implementarlos.

Cuando solamente se requiere determinar si la tarjeta es válida, se tiene un esquema de autenticación unilateral que se basa en el concepto de reto – respuesta. El lector genera un número aleatorio (reto) el cual es enviado hacia la tarjeta. La tarjeta cifra este número y lo envía de regreso hacia el lector (respuesta); el lector descifra lo que ha recibido y lo compara con el número aleatorio generado a fin de determinar si la tarjeta es válida o no. La siguiente figura ayudará a comprender mejor este concepto.

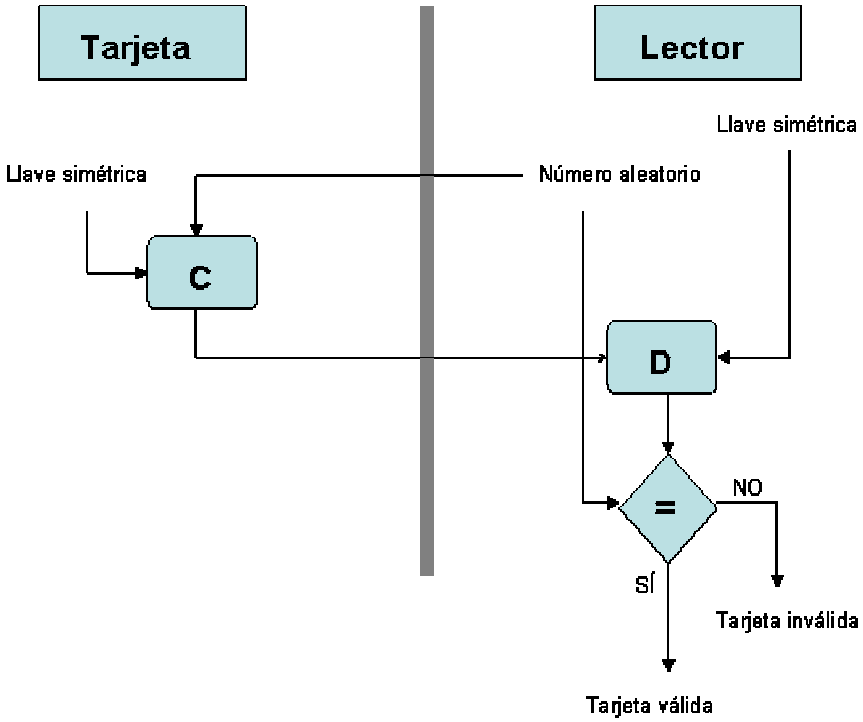

Figura 1.8. Autenticación de la tarjeta.

Cuando se requiere realizar la autenticación de ambas partes, entonces cada una genera su propio número aleatorio y realiza un proceso similar al que se ilustra en la Figura 1.8.

En la actualidad la identificación de un individuo es uno de los procesos más complejos y cotidianos en el terreno de la Tecnología de Información. La combinación de la capacidad de la Tarjeta Inteligente para almacenar información de la identidad de una persona y el equipo adecuado de verificación, ha permitido mejorar el proceso de control de acceso a lugares físicos, o el acceso a redes de computadoras, y aplicaciones. Relacionado a esto, la tarjeta también ayuda al proceso de autorización ya que es posible almacenar en ella los perfiles y privilegios del usuario, por ejemplo el acceso a áreas restringidas de un edificio, o el acceso a menús de una aplicación.

Transporte

La Tarjeta Inteligente puede ser utilizada en el sector de transporte como dinero electrónico para los conductores que necesitan pagar una cuota para utilizar una autopista. El balance en la tarjeta puede ser incrementado en estaciones fijas de pago o mediante un proceso de prepago. Este uso de la tarjeta se ha extendido a diversos servicios de transporte en algunas ciudades de Europa; en ellas se cuenta con una infraestructura sólida para reconocer a la tarjeta ya sea en una estación de autobús, o de metro. Una situación más compleja se presenta si el cargo que debe hacerse a la tarjeta es variable dependiendo del viaje, pero esto se resuelve insertando la tarjeta tanto en la entrada como en la salida de la estación para calcular el monto correcto.

Telefonía y Telecomunicaciones

La tecnología *GSM* (*Global System for Mobile Communications*) es una de las más usadas para la telefonía móvil. Dentro de cada teléfono celular que usa *GSM*, se encuentra una pequeña tarjeta inteligente que se conoce como *SIM* (*Subscriber Identity Module*). El *SIM* tiene una llave secreta la cual se utiliza en esquema de reto – respuesta para autenticar el mismo *SIM* a la red telefónica. Este tipo de autenticación basado en una llave secreta, guardada en un medio a prueba de alteraciones físicas, es muy efectivo contra la clonación de teléfonos, la cual es una situación que se presentaba de manera frecuente con los teléfonos analógicos.
Desde 1996, el *ETSI* (*European Telecommunications Standard Institute*) creó la norma *SIM Toolkit* (*GSM 11.14*) para brindar servicios de valor agregado y comercio electrónico mediante el uso de teléfonos *GSM*.
Las aplicaciones basadas en el *SIM Toolkit* consisten en una serie de procedimientos y comandos que extienden las funciones entre el teléfono *GSM* y la tarjeta *SIM*. Esto permite programar nuevos servicios con independencia de los fabricantes de tarjetas y teléfonos. De esta manera el operador puede personalizar los servicios de cada abonado a través del teléfono móvil. Es posible programar aplicaciones en la tarjeta *SIM* para visualizar una serie de menús, o automatizar procedimientos en el teléfono. Siguiendo las instrucciones desplegadas en la pantalla del teléfono se podrá, por ejemplo, ordenar operaciones bancarias sin necesidad de realizar una llamada, o solicitar mediante un mensaje las últimas noticias sin tener que recordar el código que corresponde de este servicio.

Sector salud

Debido a la seguridad de almacenamiento que proporciona la Tarjeta Inteligente se ha aplicado su uso para distintos servicios del sector salud. La información como los datos personales, el seguro de gastos médicos, y el perfil médico del paciente pueden ser manejados por aplicaciones que administran el nivel de acceso a éstos de una manera similar a un sistema de control de flujos. Diversos hospitales en Francia, Alemania, y algunos más en Asia están implementado la tarjeta para el sector salud. En el caso de Estados Unidos se piensa utilizar la tarjeta como un pasaporte médico, con lo cual diversos programas de salud puedan compartir información del paciente, con autorización previa de éste mismo, con la finalidad de otorgar mayores beneficios de los tratamientos y al mismo tiempo minimizar los costos administrativos.

Componentes de un sistema basado en las Tarjetas Inteligentes

La configuración de un sistema basado en las Tarjetas Inteligentes depende de los procedimientos de administración, personalización, y entrega de la tarjeta, de las aplicaciones y de las capacidades o recursos de la tarjeta, y de los factores técnicos externos seleccionados para el sistema. Sin embargo, es posible identificar factores comunes que siempre estarán presentes en cualquier sistema de esta naturaleza los cuales se enlistan a continuación:

- Tarjetas Inteligentes.
- Lector de tarjetas.
- Sistema central de administración de las tarjetas. Este componente mantiene la base de datos de usuario a fin de realizar la captura, almacenamiento, y

recuperación de datos necesarios para la administración del ciclo de vida de las tarjetas.

- Aplicaciones.
- Infraestructura de cómputo y de comunicaciones.

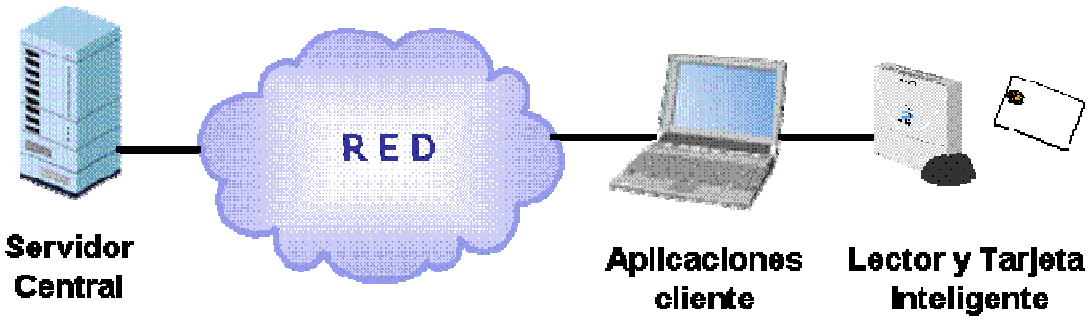

Figura 1.9. Componentes típicos de un sistema basado en Tarjetas Inteligentes.

Capítulo 2: Antecedentes de Criptografía

Historia

De acuerdo a su etimología, la Criptografía es el arte de saber escribir en caracteres secretos; su origen proviene del griego *kriptos* que significa oculto y del griego *grafos* que significa escritura. En la actualidad, la Criptografía es considerada como una ciencia.

La necesidad de la Criptografía surge por el hecho de que el hombre necesita que aquello que escribe no pueda ser leído por otros, o al menos sin su consentimiento. Esto ha sido, es, y será el principal objetivo de la Criptografía a lo largo de la historia.

Desde la época de los hebreos, se encuentra el uso de la Criptografía: en la Biblia hay muestras del sistema conocido como *Tabas*, el cual consiste en invertir el alfabeto, es decir, cambiar la A por la Z, la B por la Y, y así sucesivamente. Los griegos usaron la Criptografía en la guerra entre Atenas y Esparta. En este caso, el sistema se basaba en la adición de caracteres al mensaje, dificultando así su lectura. El receptor poseía un rodillo con el que se eliminaban dichos caracteres al enrollar el mensaje en él.

El historiador romano Polibio (200 a.C. – 118 a.C.), utilizó un método de sustitución de caracteres, por medio de una tabla, con el propósito de transmitir información de manera más eficaz. El resultado de esta sustitución implicaba la escritura de un mensaje usando solamente 5 posibles caracteres o letras. Este método fue la base de muchos sistemas criptográficos posteriores. El emperador romano Julio César (101 a.C. – 43 a.C.), usó un esquema que consistía en mover el carácter a representar *x* número de posiciones dentro del alfabeto. Por ejemplo, se sustituía la A por la D, la B por la E, y así sucesivamente.

A partir del Renacimiento la Criptografía adquirió cierta importancia y complejidad. Durante esta época, el humanista Leon Battista Alberti escribió, en 1446, el libro sobre Criptografía más antiguo que se conoce en el mundo occidental. Su trabajo abarcó los temas de cifrado monoalfabético y polialfabético[1].

Durante el siglo XIX se difunde ampliamente el método de reordenación de los símbolos del mensaje. Este método y el de sustitución conforman la llamada Criptografía Clásica. Fue durante las dos guerras mundiales cuando más se desarrolló la Criptografía, dada la necesidad de mantener comunicaciones militares confidenciales.

Tras la publicación en 1949 de la *"Teoría de las Comunicaciones Secretas*" y la *"Teoría Matemática de la Comunicación"* (Claude Elwood Shannon, 1916) la Criptografía deja de ser considerada un arte y a partir de aquí se le considera una ciencia, dadas sus complejas relaciones con otras ciencias como la Estadística, la Teoría de Números, la Teoría de la Información, entre otras.

---

[1] En el cifrado monoalfabético se sustituye cada letra por otra que ocupa la misma posición en un alfabeto desordenado y esta correspondencia se mantiene a lo largo de todo el mensaje. En el cifrado polialfabético la sustitución aplicada a cada carácter varía en función de la posición que ocupe éste dentro del texto original. En realidad corresponde a la aplicación cíclica de *n* cifrados monoalfabéticos, es decir, de varios alfabetos desordenados.

La aparición de nuevas tecnologías para el procesamiento de datos y los avances en las telecomunicaciones permiten una mayor difusión de la información, tanto pública como privada. Es por ello que es necesario tomar precauciones para evitar que terceros puedan leer, o incluso manipular, un mensaje que se desea que sólo sea leído por cierta(s) persona(s). Debido a esta situación, en la década de los setenta, el Departamento de Normas y Estándares de Estados Unidos publica el primer diseño lógico de un algoritmo criptográfico, el *DES* (*Data Encryption Standard*), el cual estaría llamado a ser el principal método para mantener la confidencialidad de los datos a finales del siglo XX. Además, en esas mismas fechas se comenzaba a originar lo que sería, hasta ahora, uno de los adelantos más significativos de la Criptografía: los sistemas de cifrado asimétricos.

Hoy en día la Criptografía se considera una ciencia englobada dentro de la Criptología, que a su vez también incluye el Criptoanálisis[2].

Conceptos

Antes de continuar, es necesario mencionar que todo texto que pueda ser leído sin necesidad de ser modificado se le conoce como texto claro. Al proceso de conversión del texto claro para ocultar su mensaje se llama Cifrado, y el resultado se conoce como texto cifrado o criptograma. Finalmente, al proceso inverso, es decir, el que convierte el criptograma en texto claro se le conoce como Descifrado o Cifrado Inverso.

Un mensaje ha de ser protegido contra dos tipos de amenazas: pasiva, que es el acceso no deseado a la información y, activa, que es la alteración de la información.

En el siglo XIX Auguste Kerchoffs escribió su obra "*La Criptografía Militar*", en la que estableció unas reglas que todo algoritmo Criptográfico debería cumplir. Estas reglas siguen siendo importantes en la actualidad y son las siguientes:

- El algoritmo criptográfico debe ser inquebrantable desde el punto de vista práctico.
- Todo sistema criptográfico debe estar compuesto por dos tipos de información:
    - Pública: como lo es su diseño.
    - Privada: como lo es la llave[3].
- La llave escogida debe ser fácil de recordar y modificar.
- El criptograma debe ser transmitido usando los medios de comunicación habituales (por ejemplo el telégrafo en aquel entonces).
- La complejidad del proceso de recuperación del mensaje debe corresponder con el beneficio obtenido.

Ataques

La Criptografía y el Criptoanálisis persiguen objetivos totalmente opuestos, pero la evolución de ambas está estrechamente relacionada debido a que los criptoanalistas tratan de romper los algoritmos de cifrado actuales, y a su vez los criptógrafos buscan la manera de diseñar mejores algoritmos que una vez que son difundidos serán objeto de

[2] Ciencia que estudia los métodos para conocer el diseño y operación de un esquema de cifrado

[3] La llave puede considerarse como una contraseña; su función es alterar de manera única el resultado del cifrado.

estudio de los criptoanalistas y así sucesivamente.

El objetivo real de los criptógrafos es proteger el mensaje por un período de tiempo suficientemente razonable como para que cuando se consiga romper el algoritmo, la información contenida ya no sea útil. El objetivo de los criptoanalistas es, por el contrario, reducir ese tiempo y obtener partes del texto original que permitan reconstruirlo por completo.

Los distintos ataques a algoritmos de cifrado con la finalidad de romperlos, se clasifican en función de la información conocida por el criptoanalista, los cuales se mencionan a continuación.

- **Ataque con texto cifrado solamente.** El ataque se basa en obtener una cantidad considerable de criptogramas y realizar un análisis estadístico a fin de encontrar una estructura predefinida e idéntica que pueda ayudar a inferir el algoritmo de cifrado empleado.

- **Ataque con texto en claro conocido.** El criptoanalista conoce, o puede predecir, el texto correspondiente a ciertas partes del criptograma. A partir de esto el criptoanalista ha de conseguir descifrar el resto del criptograma, bien sea deduciendo la llave empleada al momento de aplicar el cifrado aprovechando las palabras de las que se tiene una correspondencia. Este ataque es especialmente efectivo contra sistemas de cifrado simétrico por bloques, que se mencionarán más adelante, ya que en éstos cada palabra conserva su longitud inicial en el criptograma.

- **Ataque con texto en claro escogido.** El criptoanalista puede obtener el texto cifrado correspondiente a un texto claro escogido por él, pero sin conocer la llave empleada en el proceso de cifrado. Este ataque permite comparar la transformación de diversos textos en claro a fin de obtener la llave utilizada.

- **Ataque con texto cifrado escogido.** En este caso el criptoanalista puede obtener el texto claro correspondiente a un criptograma de su elección. Al igual que en el caso anterior, tampoco se conoce la llave empleada.

- **Ataque con intermediario.** Este ataque se basa simplemente en la intrusión del criptoanalista en el momento en el que las dos partes que quieren intercambiar la información están intercambiándose las llaves. El criptoanalista ha de aprovechar este momento para hacer llegar a ambas partes su propia llave. Así el éste recibe los mensajes primero, y luego los envía al destinatario legítimo, haciendo creer a ambas partes que el proceso de comunicación es seguro. Este método pone de manifiesto uno de los grandes inconvenientes del sistema simétrico, el intercambio de llaves, que ha de ser realizado a través de un medio seguro, de lo contrario el enemigo puede hacerse fácilmente con éstas, pudiendo así cifrar y descifrar sin problema con absoluta transparencia tanto para emisor como para receptor.

- **Ataque de prueba y ensayo.** Este ataque consiste en probar cada una de las posibles llaves, hasta dar con la que permite descifrar el criptograma. Este ataque también se le conoce como Ataque de Fuerza Bruta.

Cifrado simétrico o de llave privada

En el cifrado simétrico se utiliza una misma llave tanto para el proceso de cifrado como para el proceso de descifrado. Este es el método más sencillo, a nivel matemático el menos complejo, y ha sido empleado durante mucho tiempo. La seguridad proporcionada por los algoritmos simétricos depende, en gran medida, de la longitud de la llave y de su no divulgación a terceras partes. El cifrado simétrico puede realizarse de dos formas: por bloque y por flujo.

En el cifrado por bloques, el mensaje se divide en bloques de *bytes* de igual tamaño. Después, a cada bloque se le aplica el algoritmo de cifrado y esto da como resultado un bloque de la misma longitud que el original. En 1949 Claude Elwood Shannon sugirió la combinación de los métodos de confusión y difusión[4] en el desarrollo de nuevos algoritmos de cifrado por bloque a fin de evitar la redundancia de la fuente y que las estadísticas del criptograma estuvieran influidas por el texto original.

A fin de proteger la confidencialidad y la integridad de la información, existen modos de operación para el uso de un algoritmo de cifrado por bloques; esto quiere decir que un modo de operación define la forma en que se utilizará el algoritmo junto con el uso de registros lógicos y operaciones lógicas elementales.

El modo de operación más sencillo es el *ECB* (*Electronic Codebook*), y que consiste en subdividir el texto original que se quiere codificar en bloques de tamaño fijo los cuales se cifran usando la misma llave. Las ventajas de este modo de operación son que permite codificar los bloques de manera independiente de su orden, lo cual es adecuado para codificar bases de datos o archivos en los que se requiere un acceso aleatorio. Además, el modo *ECB* es resistente a errores, ya que si uno de los bloques sufre una alteración, el resto quedaría intacto. Una desventaja del modo *ECB* se presenta cuando el texto en claro contiene patrones repetitivos ya que el criptograma también los presentará lo cual lo hace susceptible a ataques con texto cifrado solamente. Otro riesgo importante que presenta el modo *ECB* es el de la sustitución de bloques, el cual consiste en que el atacante puede cambiar un bloque sin mayor dificultad, y alterar los mensajes incluso desconociendo la llave y el algoritmo empleados.

El modo *CBC* (*Cipher Book Chaining Mode*) incorpora un mecanismo de retroalimentación en el cifrado por bloques. Esto significa que la codificación de bloques anteriores condiciona la codificación del bloque actual, por lo que será más difícil sustituir un bloque individual en el mensaje cifrado. Esto se consigue mediante la operación lógica *XOR* entre el bloque de *bytes* que se desea cifrar y el último criptograma obtenido. La primera vez que se cifra un bloque, no existe un criptograma previo, así que se utiliza un vector de inicialización. A pesar de que el criptograma del bloque actual depende de todos los criptogramas anteriores, esto no implica que un error en un bloque se propague; por lo tanto, para descifrar un bloque solamente se necesita el criptograma actual y el anterior.

---

[4] Los algoritmos de cifrado simétricos se apoyan en los conceptos de confusión (tratar de ocultar la relación que existe entre el texto en claro, el texto cifrado y la clave, es decir, realizar sustituciones simples) y difusión (trata de repartir la influencia de cada *bit* del mensaje original lo más posible entre el mensaje cifrado, es decir, realizar permutaciones) que se combinan para dar lugar a los denominados cifrados de producto.

El modo de operación *CFB* (*Cipher Feedback Mode*) permite codificar la información en unidades inferiores al tamaño del bloque, lo cual es útil en aplicaciones multimedia en las que la información debe ser transmitida tan pronto se encuentre disponible. El funcionamiento del modo *CFB* consiste en generar una secuencia de *bits* basada en el criptograma anterior y en el algoritmo de cifrado. A esta secuencia se le aplica la operación lógica *XOR* con el texto en claro. Al igual que en el modo *CBC*, se necesita un vector de inicialización al inicio del cifrado.

El modo de operación *OFB* (*Output FeedBack Mode*) es similar al modo *CFB*, con la diferencia que se utiliza un generador de números pseudoaleatorios que depende de un vector de inicialización. El modo de operación *CTR* (*Counter Mode*) es similar al modo *OFB*, pero la secuencia de números pseudoaleatorios no depende de valores previos de la misma si no de un contador.

El cifrado por flujo funciona mediante la combinación de caracteres del texto en claro y del flujo aleatorio de caracteres, o llave en este caso. Este cifrado fue propuesto por Gilbert S. Vernam en 1917 cuando realizaba investigación en el área de telegrafía. Vernam propuso utilizar el código Baudot[5] para la representación de cada carácter al cual se le realizaba la operación lógica *XOR* con un carácter correspondiente a una llave secreta. Para recuperar el mensaje se volvía a aplicar la misma operación dado que $XOR^{-1} = XOR$.

En la actualidad, el cifrado por flujo se divide en dos grupos: síncrono y asíncrono. El cifrado síncrono es aquel en el que la pérdida de un carácter durante la transmisión entre emisor y receptor supone la pérdida de sincronización entre emisor y receptor, y por lo tanto se ha de sincronizar la transmisión para que el receptor pueda descifrar correctamente el criptograma. Su ventaja se encuentra en que la alteración de un carácter durante la transmisión sólo supone un carácter mal traducido, mientras que el resto, anteriores o posteriores, quedará intacto. El cifrado asíncrono se caracteriza porque cada símbolo del texto cifrado depende de un número fijo de símbolos del texto original. Con esto se elimina la necesidad de sincronización entre emisor y receptor, pero un error en la transmisión del mensaje supone un error en el resto de proceso de descifrado.

La principal ventaja del cifrado por flujo es su velocidad, ya que ésta es superior a la del cifrado por bloque. Debido a que los cifradores por flujo cifran un *bit* a la vez, éstos son más adecuados para implementaciones por *hardware*. La distinción entre cifrado por flujo o cifrado por bloque corresponde simplemente al principio de funcionamiento, ya que hoy en día la mayoría de los nuevos algoritmos de cifrado simétrico permiten cifrar indistintamente en flujo o en bloque y con llave de tamaño variable, en función de las necesidades, a través de variantes del mismo algoritmo.

Cifrado asimétrico o de llave pública

En 1976 Whitfield Diffie y Martin Hellman, Ingenieros en Electrónica de la Universidad de Stanford, publicaron el articulo *"New Directions in Cryptography"* que supuso una auténtica revolución dentro del mundo de la Criptografía. En este trabajo, los autores proponían emplear distintas llaves para los procesos de cifrado y descifrado.

[5] En el código Baudot cada caracter está representado por cinco unidades o pulsos.

El objetivo planteado fue que cualquier persona pudiera cifrar empleando una llave pública, pero solo el verdadero destinatario pudiera descifrar el mensaje, al aplicar una llave privada. Esto evitaba el problema de la distribución de llaves del cifrado simétrico, y a la vez permitía otras funciones como la autenticación, la firma digital, y la no repudiación.

Diffie y Hellman proponían el uso de problemas sin solución desde el punto de vista computacional aunque realmente sí existe una solución. Su idea se basaba en que el tiempo necesario para obtener el resultado usando computadoras fuese tan elevado, que una vez revelada la información su utilidad fuese totalmente nula. En el artículo citado, se planteaba aprovechar la particularidad de ciertas operaciones matemáticas, muy sencillas de realizar en un sentido, pero con una inversa extremadamente costosa, principalmente a nivel de tiempo. Ejemplos claros de esta situación son la potenciación y los logaritmos.

Las siguientes propiedades de complejidad computacional que debía cumplir todo algoritmo asimétrico, según Diffie y Hellman, se enlistan a continuación:

- El usuario debe ser capaz de calcular sus llaves, pública y privada, en tiempo polinomial[6].
- El proceso de cifrado, aplicando la llave pública, debe ser en tiempo polinomial.
- El proceso de descifrado, aplicando la llave privada, debe ser en tiempo polinomial.
- El proceso de hallar la llave privada, mediante la llave pública, a pesar de tener solución a nivel teórico, debe ser inviable.
- El proceso de descifrado del criptograma aplicando la llave pública debe ser inviable.

Los algoritmos de cifrado asimétrico se clasifican en función de la complejidad del problema en que basan su seguridad:

- **Problema de la Factorización Entera.** Su principio es simple: no existe ningún método eficiente para factorizar el resultado del producto de dos números grandes primos. Un ejemplo es el algoritmo RSA, creado en 1977 por Ronald Rivest, Adi Shamir y Leonard Adleman.
- **Problema del Logaritmo Discreto.** El problema del logaritmo discreto consiste en calcular un número **x**, conocidos los números **h** y **g,** para que se cumpla la siguiente expresión: $\mathbf{h}^{\mathbf{x}} = \mathbf{g}$ (generalmente **g** es un número en aritmética modular, por ejemplo: **g** = **p** MOD (**q**)). A pesar de que el problema del logaritmo discreto está presente cualquier conjunto de números, en Criptografía se suele emplear el conjunto de los números enteros (**Z**). Entre los sistemas más representativos de esta clasificación se encuentran Diffie-Hellman de intercambio de llaves, el DSS (*Digital Signature Standard*) y el algoritmo de ElGammal.
- **Problema del Logaritmo Discreto Elíptico.** Se conoce también como Criptografía de Curva Elíptica. Estos sistemas basan su seguridad en el

[6] En términos prácticos, tiempo polinomial puede entenderse como un tiempo razonablemente corto.

problema del Logaritmo Discreto, pero la diferencia consiste en utilizar puntos (par de coordenadas cartesianas) como los símbolos del mensaje a cifrar. Para poder trabajar con ellos, se definen las propiedades características de todo grupo conmutativo: conmutación, asociación, elemento neutro y elemento simétrico, y las operaciones necesarias para trabajar como la suma, la multiplicación, y el producto. En la actualidad no se ha encontrado ningún sistema que permita romper los algoritmos de cifrado de curva elíptica en tiempo polinomial, ni tan siquiera para llaves cortas. Por ello, estos sistemas, pese a usar llaves mucho más reducidas, consiguen niveles de seguridad superiores a los de otros algoritmos. En esta clasificación se encuentran el Diffie-Hellman elíptico, y la versión elíptica del ElGamal.

Métodos híbridos de cifrado

El uso de un algoritmo de cifrado simétrico y uno asimétrico dentro de una aplicación tiene el objetivo de proporcionar mayores niveles de seguridad en un tiempo razonable. En este caso, el algoritmo simétrico se utiliza para el cifrado de grandes cantidades de datos, mientras que el algoritmo asimétrico es usado para crear las llaves para automatizar la distribución de la llave para el cifrado simétrico.

La llave pública del receptor es usada para cifrar la llave para el algoritmo simétrico y el mensaje cifrado con esta llave. El receptor usa su llave privada para obtener la llave del algoritmo simétrico a fin usar ésta para descifrar el criptograma.

Una llave de sesión es una llave para el cifrado simétrico que solamente se usa durante una comunicación, o sesión, entre dos personas. La llave solamente es válida para dicha sesión.

Aplicaciones

Firma Digital

La firma autógrafa es, en los documentos en papel, la marca que certifica que un texto ha sido autorizado por una persona concreta. Con el desarrollo de las telecomunicaciones, se planteó el problema de firmar documentos digitales o electrónicos. Contra lo que pueda parecer, la firma digital no es la digitalización de la firma autógrafa mediante el empleo de medios ópticos, como por ejemplo un escáner.

El problema de contar con un reemplazo para las firmas manuscritas es difícil. Básicamente, lo que se requiere es un sistema mediante el cual el emisor pueda enviar un mensaje al receptor de manera que:

1. El receptor pueda verificar la identidad del emisor.

2. El emisor no pueda repudiar (negar) el contenido del mensaje.

3. El receptor no haya podido alterar el mensaje.

Tomando como ejemplo un caso en el que un cliente solicita a un banco la compra de un paquete de acciones, se tiene que el primer requisito es necesario cuando el banco necesita asegurarse de que el cliente que da la orden de compra es realmente quién dice ser; el segundo requisito es necesario para proteger al banco contra fraudes, por ejemplo, cuando el cliente niega haber solicitado la compra de las acciones; y el tercer requisito es necesario para proteger al cliente en el caso de que el banco argumente que se le solicitó un número distinto de acciones para su compra.

En las situaciones en que se requiere la certeza de la validez de un mensaje, esto es, conocer que no ha sido modificado durante su transmisión, se utiliza una función de dispersión unidireccional[7] que toma una parte arbitrariamente grande de texto y a partir de ella calcula una cadena de *bits* de longitud menor y fija, es decir, se genera un resumen del documento o compendio del mensaje (*Message Digest*). El concepto de integridad es el mismo que se presenta con un *CRC*[8], el cual se añade al final de una trama, de tal forma que el receptor es capaz de comprobar la integridad de las tramas recibidas.

La función de una sola vía debe tener las siguientes propiedades:

1. Dado un texto P, es fácil calcular su compendio de mensaje *MD*(P).

2. Dado un compendio de mensaje *MD*(P), es imposible encontrar el mensaje original. Esta propiedad se conoce como Resistencia a Preimágenes.

3. Es imposible generar dos mensajes que tengan el mismo compendio de mensaje. Esta propiedad se conoce como Resistencia a Colisiones.

---

[7] Conocidas también como funciones de una sola vía o funciones *Hash*.

[8] *Cyclic Redundanc Check*

Los compendios de mensaje funcionan tanto en llave privada como en llave pública, siendo los de mayor uso los algoritmos MD5 y SHA (*Secure Hash Algorithm*).

La firma digital consiste en cifrar el compendio de un mensaje o *hash* con la llave privada. Con la llave pública se puede realizar la comparación entre el *hash* descifrado del mensaje y el *hash* calculado por el receptor. Si existe una coincidencia entonces se considera que el mensaje se encuentra firmado.

Diffie y Hellman establecieron en su artículo *"New Directions in Cryptography"* las principales condiciones que debía cumplir la firma digital:

- Unicidad. Es imposible obtener una firma digital sin su correspondiente llave privada.
- Infalsificable. El intento de falsificar una firma digital debe dar con un problema computacionalmente irresoluble.
- Irrevocabilidad. El autor de una firma digital, tras haber sido reconocida como suya por un tercero (autoridad, notario, etc.) no puede negar su autoría.

La firma digital sirve también para la autenticación, ya que si el mensaje es manipulado, la firma digital, al ser dependiente de éste, ya no se corresponde con él.

En principio cualquier algoritmo de llave pública puede usarse para firmas digitales. El estándar de facto de la industria es el algoritmo RSA y muchos productos de seguridad lo usan.

Capítulo 3: El algoritmo *Rijndael*

Historia

En Marzo de 1975 el gobierno de los Estados Unidos designó al *DES* (*Data Encryption Standard*) como su norma oficial para el cifrado de datos sensibles pero no clasificados. El *DES* posteriormente fue adoptado a nivel mundial.

Desde su creación, el *DES* fue criticado debido a su tamaño de llave (56 *bits*), que con el poder de cómputo actual lo hace vulnerable a ataques de fuerza bruta[9]. Así lo muestra el primer ataque público de fuerza bruta exitoso del *DES* en 1997 el cual tomó aproximadamente cuatro meses en llevarse a cabo. Aún así, el *DES* permaneció en uso oficial hasta que en 1998 la *Electronic Frontier Foundation* demostró que un ataque de esta naturaleza se puede realizar en 56 horas.

A fin de encontrar un algoritmo de cifrado más robusto, fue propuesto un proceso que en términos generales consiste en la aplicación del algoritmo que utiliza el *DES* tres veces, lo que se conoce como Triple *DES*. El Triple *DES* es considerado seguro, sin embargo es demasiado lento para su implantación en hardware y software, y solamente trabaja con bloques de datos de 64 *bits*.

De manera paralela y a partir de 1997, el *NIST* (*National Institute for Standards and Technology*) de Estados Unidos inició el llamado a un proceso de selección para sustitución del *DES*: el *AES* (*Advanced Encryption Standard*). En la ronda final estuvieron compitiendo 5 algoritmos: *MARS*, *RC6*, *Rijndael*, *SERPENT* y *TWOFISH*. Finalmente el *NIST* anunció al algoritmo *Rijndael* como el ganador del *AES* en Octubre del 2000. Este proceso se efectuó con la participación de la comunidad criptográfica mundial lo cual convierte al *Rijndael* en un algoritmo robusto y digno de la confianza de todos. El nombre *Rijndael* es la combinación de los apellidos de sus autores Joan Daemen y Vincent Rijmen originarios de Bélgica.

Especificaciones

El *AES* especifica un algoritmo de cifrado simétrico por bloques con las características principales siguientes:

- Longitud de la llave con los suficientes *bits* para tener un espacio de posibles llaves mucho mayor al del *DES*[10] y resistente a ataques de fuerza bruta.
- Rapidez y flexibilidad para ejecutarse en diversas plataformas.
- Manejo de distintos tamaños de bloques de datos.

El algoritmo *Rijndael* tiene una longitud variable de llave y de bloque de datos que puede ser de 128, 192 ó 256 *bits*. Esto da como resultado que se tengan 3.4×10^{38} posibles llaves de 128 *bits*, 6.2×10^{57} posibles llaves de 192 *bits* y 1.1×10^{77} posibles

[9] Un ataque de fuerza bruta es aquel en que se prueban todas las posibles combinaciones de una llave de acuerdo a su longitud en *bits*.
[10] En el *DES* se tienen 7.2×10^{16} posibles llaves de 56 *bits*.

llaves de 256 *bits.* De manera oficial, el *AES* solamente reconoce un tamaño de llave de 128 *bits* y un tamaño variable de bloque de datos que puede ser de *128*, *19*2 y 256 *bits*.

Durante la fase de selección, el algoritmo *Rijndael* demostró una gran flexibilidad para adaptarse a diversas plataformas como las arquitecturas de 8 *bits* de las Tarjetas Inteligentes y las arquitecturas de 32 *bits* de las computadoras personales. Además, sus autores han demostrado que es posible combinar ciertas operaciones del propio algoritmo a fin de hacer mejor uso de recursos como la memoria y el procesador. Esta versatilidad hace que el algoritmo *Rijndael* sea superior en cuanto a rapidez de ejecución con respecto al Triple *DES*, por lo que el uso de este último disminuirá rápidamente.

Descripción del algoritmo

El algoritmo *Rijndael* es iterativo y se basa en aplicar un número determinado de rondas a un valor intermedio que se denomina estado. La entrada y salida del algoritmo es un bloque de datos de 128 bits lo que se representa por el parámetro N_b=4 que refleja el número de palabras de 32 *bits*. La longitud de la llave se representa por el parámetro N_k que también refleja el número de palabras de 32 bits.

Puesto que el algoritmo permite emplear diferentes longitudes de bloque y de llave, el número de rondas o iteraciones requerido en cada caso es variable. En la tabla siguiente se indica cuantas rondas (N_r) son necesarias en función de los parámetros N_b y N_k. El primer renglón representa la especificación oficial del *AES*.

	N_b=4 (128 *bits*)	N_b=6 (192 *bits*)	N_b=8 (256 *bits*)
N_k=4 (128 *bits*)	N_r=10	N_r=12	N_r=14
N_k=6 (192 *bits*)	N_r=12	N_r=12	N_r=14
N_k=8 (256 *bits*)	N_r=14	N_r=14	N_r=14

Tabla 3.1. Número de rondas del algoritmo *Rijndael* en función de los parámetros N_b y N_k.

El estado se representa mediante una matriz rectangular de *bytes* que posee cuatro filas, y N_b columnas. Para el presente trabajo se tiene que $N_b = N_k = 4$. La representación de la llave tiene una estructura similar a la del estado, y consiste de una matriz de llave de ronda con cuatro filas y N_k columnas.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Figura 3.1. Ejemplo de matriz de estado y matriz de llave de ronda con $N_b = N_k = 4$ (128 *bits*).

El bloque que se pretende cifrar o descifrar se traslada directamente *byte* a *byte* sobre la matriz de estado por columnas. La llave original también es copiada de forma similar en su matriz correspondiente; por cada iteración del algoritmo se va calculando una nueva llave, o llave de ronda, en un proceso llamado Planeación de Llaves.

Cifrado directo

El cifrado directo del algoritmo *Rijndael* se realiza mediante la ejecución de rondas de transformación las cuales están formadas por cuatro distintas transformaciones que operan a nivel de *byte*, las cuales son:

- ***Substitute Bytes***. Esta es una operación no lineal que consiste en la sustitución de un *byte* de la matriz de estado mediante el uso de una tabla de búsqueda (*S-Box*).
- ***Shift Rows.*** Esta es una operación de permutación que consiste en realizar un desplazamiento lógico hacia la izquierda sobre un *byte*. La cantidad de lugares a desplazar se hace de una manera predefinida de acuerdo al número de renglón de la matriz de estado.
- ***Mix Columns.*** Cada *byte* de la matriz de estado es reemplazado por una combinación lineal de los *bytes* en la misma columna.
- ***Add Round Key***. Consiste en realizar la operación lógica *XOR* entre la matriz de estado y la matriz de llave de ronda.

Cada ronda de transformación está parametrizada por una llave de ronda que es obtenida mediante el proceso de Planeación de Llaves. Este proceso toma como entrada la llave original y mediante un algoritmo que se explicará más adelante, se obtiene una llave para cada ronda con la misma longitud que la llave original. La siguiente figura muestra de manera esquemática la estructura del algoritmo *Rijndael* para el cifrado directo y la estructura de cada ronda de transformación.

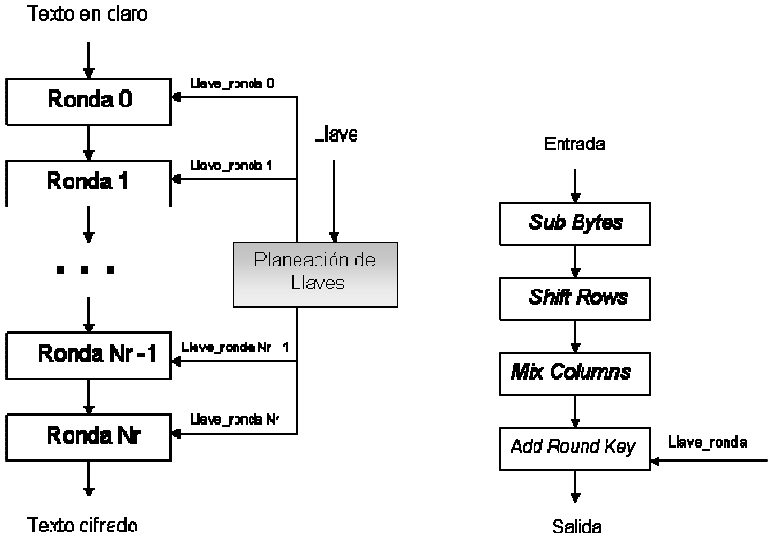

Figura 3.2. Estructura del algoritmo *Rijndael*.

Es importante mencionar que la ronda inicial consiste únicamente en realizar la función *Add Round Key* entre la matriz de estado y la llave original. Para la última ronda (N_r) se omite únicamente la transformación *Mix Columns*.

La transformación *Substitute Bytes* consiste en la sustitución de un *byte* por otro de acuerdo a la siguiente relación: primero se usa una transformación en la que cada *byte* del estado se representa de manera polinomial sobre el campo $GF(2)$ y posteriormente se calcula su inverso multiplicativo en el campo finito $GF(2^8)$ (el elemento nulo o cero se mapea a sí mismo); después se aplica una función *affine*[11] definida por la siguiente relación:

$$\begin{bmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \\ s_4' \\ s_5' \\ s_6' \\ s_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Figura 3.3. Matriz empleada por la transformación *Substitute Bytes*.

La matriz de *8x8* de la función *affine* anterior es invertible y cada uno de sus coeficientes pertenece a un elemento del campo $GF(2)$. La transformación *Substitute Bytes* puede ser expresada en su totalidad mediante la siguiente tabla de sustitución (*S-Box*).

[11] Una función *affine* tiene la siguiente estructura: $y=ax+b$ en las que a y b son constantes.

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	Ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	C3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1ª	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	Ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	Fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	Cd	0c	13	Ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	Dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	Ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabla 3.2. Tabla de sustitución *S-Box* para la transformación *Substitute Bytes*.

Si un *byte* tiene el valor {01010011}, entonces se puede separar por dos grupos de cuatro *bits* y a cada uno de ellos se le representa de forma hexadecimal entonces se tiene un valor de {53}. El resultado de la sustitución de este valor en la tabla de búsqueda se determina por la intersección del renglón marcado con 5 y la columna marcada con 3 y que corresponde a {ed} o {11011110}.

La siguiente figura muestra el efecto de la transformación *Substitute Bytes* en la matriz de estado.

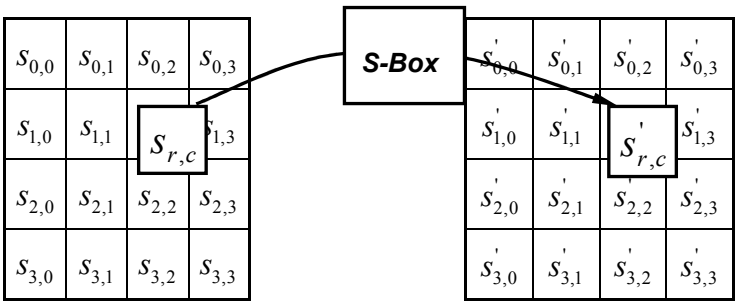

Figura 3.4. Efecto de la transformación *Substitute Bytes*.

La transformación *Shift Rows* consiste en desplazar a la izquierda las filas de la matriz de estado de manera cíclica. El desplazamiento de cada fila está en función de la longitud del bloque de datos N_b y del número de fila en la matriz de estado. En la primer fila (ó fila cero de acuerdo a la notación matricial) no existe desplazamiento.

N_b	Número de posiciones a desplazar en la fila 1	Número de posiciones a desplazar en la fila 2	Número de posiciones a desplazar en la fila 3
4	1	2	3
6	1	2	3
8	1	3	4

Tabla 3.3: Número de posiciones a desplazar según el tamaño de bloque N_b.

La siguiente figura es un ejemplo del resultado de la transformación *Mix Columns* para N_b =4.

S

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

S'

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

Figura 3.5. Efecto de la transformación *Shift Rows* para N_b = 4.

La transformación *Mix Columns* opera sobre cada columna de la matriz de estado. Cada columna se representa como un polinomio en el campo finito $GF(2^8)$, éste último es multiplicado por un polinomio fijo *c(x)* módulo el polinomio $x^4 + 1$. El polinomio *c(x)* es el siguiente:

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} .$$

La transformación *Mix Columns* puede escribirse en forma matricial de la siguiente manera:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad \text{Para } 0 \le c < N_b$$

Figura 3.6. Representación matricial de la transformación *Mix Columns*.

Como resultado de esta transformación los cuatro *bytes* de cada columna son reemplazados por los siguientes valores:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

En la transformación *Add Round Key* se realiza la operación lógica *XOR* entre la matriz de estado y la llave de ronda.

En el proceso de Planeación de Llaves, las llaves de ronda se derivan de la llave original mediante el uso de una función de expansión y otra función de selección. La función de expansión produce N_r+1 llaves de ronda, cada una de igual tamaño que la llave original lo que produce un flujo de *bytes* de tamaño $4*(N_r+1)*N_b$. La función de selección toma de manera consecutiva bloques del mismo tamaño que la matriz de estado de la secuencia obtenida, y los va asignando a cada llave de ronda.

La función de expansión tiene dos versiones según el valor de N_k. Sea *K(i)* un vector de *bytes* de tamaño $4*N_k$, que contiene la llave original, y sea *W(i)* un vector de $N_b*(N_r+1)$ registros de cuatro *bytes*, entonces:

Para $N_k \leq 6$:

1) Para *i* desde 0 hasta N_k - 1 hacer:
   W(i) ← (*K*(4 * *i*), *K*(4 * *i* + 1), *K*(4 * *i* + 2), *K*(4 * *i* + 3))
2) Para *i* desde N_k hasta N_b * (N_r + 1) hacer:
   temp ← *W*(*i* - 1)
   Si (*i* mod N_k) = 0
      temp ← **Sub**(**Rot**(temp)) ⊕ RC(*i* / N_k)
   W(i) ← *W*(*i* - N_k) ⊕ temp

Para $N_k > 6$:

1) Para *i* desde 0 hasta N_k - 1 hacer:
   W(i) ← (*K*(4 * *i*), *K*(4 * *i* + 1), *K*(4 * *i* + 2), *K*(4 * *i* + 3))
2) Para i desde N_k hasta N_b * (N_r + 1) hacer:
   temp ← *W*(*i* - 1)
   Si (*i* mod N_k) = 0
   temp ←**Sub**(**Rot**(temp)) ⊕ RC(*i* / N_k)
   Si (*i* mod N_k) = 4
      temp ← **Sub**(temp)
   W(i) ← *W*(*i* - N_k) ⊕ temp

La función **Sub** realiza la sustitución por medio de la tabla *S-Box* de cada uno de los *bytes* del registro de cuatro que se le proporciona como parámetro. La función **Rot** realiza el desplazamiento lógico hacia la izquierda en una posición de los *bytes* del registro, por lo que si esta función tiene como parámetro (W_0, W_1, W_2, W_3) el resultado será (W_1, W_2, W_3 W_0). Finalmente, RC(*i*) es una constante que pertenece al campo finito $GF(2^8)$ la cual se obtiene de la siguiente forma:

$$RC(i)=x^{i-1} \quad \text{para } i=1 \ldots N_r$$

La siguiente tabla muestra las constantes de ronda para el caso de una longitud de 128 *bits* tanto del bloque de datos como de la llave.

RC(1)	RC(2)	RC(3)	RC(4)	RC(5)	RC(6)	RC(7)	RC(8)	RC(9)	RC(10)
0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1B	0x36

Tabla 3.4. Constantes de ronda para $N_b = N_k = 4$.

Cifrado inverso

Las transformaciones usadas para el cifrado directo del algoritmo *Rijndael* son invertibles por lo que se les hará referencia como *Inv Substitute Bytes*, *Inv Shift Rows*, *Inv Mix Columns* e *Inv Add Round Key*. El cifrado inverso consiste básicamente en la aplicación de estas transformaciones inversas y las llaves de ronda en el orden contrario. El tipo de operaciones involucradas en el algoritmo *Rijndael* permite que el orden de ejecución de las transformaciones se altere a fin de que se tenga una estructura similar a la del cifrado directo, a lo que se conoce como cifrador inverso equivalente. Esto es útil para arquitecturas de 32 *bits* pero en arquitecturas de 8 *bits* no se tienen ventajas significativas.

Para la transformación *Inv Shift Rows* se realizan los desplazamientos lógicos pero hacia la derecha; se sigue la misma estructura utilizada para el cifrado directo en cuanto al número de desplazamientos de acuerdo a la fila y al parámetro N_b.

La transformación *Inv Substitute Bytes* se obtiene mediante la aplicación de la inversa de la función *affine* definida para el cifrado directo y por el cálculo del inverso multiplicativo en el campo finito de $GF(2^8)$; el elemento nulo o cero también se mapea a sí mismo. Esta transformación también puede realizarse mediante el uso de una tabla de búsqueda o de sustitución.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	A1	66	28	d9	24	B2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	F6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	A8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	B0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabla 3.5. Tabla de sustitución *S-Box* para la transformación *Inv Substitute Bytes*.

La transformación *Inv Mix Columns* opera sobre cada columna de la matriz de estado;

cada columna es tratada como un polinomio sobre el campo finito $GF(2^8)$, el cual es multiplicado por un polinomio fijo $c^{-1}(x)$ módulo el polinomio $x^4 + 1$.
El polinomio $c^{-1}(x)$ es el siguiente:

$$c^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

La transformación *Inv Mix Columns* puede escribirse en forma matricial de la siguiente manera:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad \text{Para } 0 \le c < N_b$$

Figura 3.7. Representación matricial de la transformación *Inv Mix Columns*.

Como resultado de esta transformación los cuatro *bytes* de cada columna son reemplazados por los siguientes valores:

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$
$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$
$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Los coeficientes de la matriz hacen que la transformación *Inv Mix Columns* sea más costosa computacionalmente. Sin embargo en el Capítulo 4 se mostrará que es posible factorizar el polinomio $c^{-1}(x)$ a fin de reducir la complejidad de la implantación.

Con respecto a la transformación *Add Round Key*, ésta es su propia inversa ya que involucra solamente la aplicación de la operación lógica *XOR*. Para el proceso del cálculo de llaves de ronda se tiene la opción de que durante el cifrado directo se guarde la última llave de ronda en memoria, la cual serviría para calcular todas las demás llaves de ronda en el sentido inverso. Sin embargo, se tendría el problema de que si se presenta alguna condición que interrumpa el suministro de energía a la tarjeta, entonces la última clave de ronda en la memoria se perdería. En el Capítulo 4 se analizará la opción más adecuada para el proceso de Planificación de Llaves de acuerdo a los objetivos a alcanzar en este trabajo.

Capítulo 3: El algoritmo *Rijndael*

Historia

En Marzo de 1975 el gobierno de los Estados Unidos designó al *DES* (*Data Encryption Standard*) como su norma oficial para el cifrado de datos sensibles pero no clasificados. El *DES* posteriormente fue adoptado a nivel mundial.

Desde su creación, el *DES* fue criticado debido a su tamaño de llave (56 *bits*), que con el poder de cómputo actual lo hace vulnerable a ataques de fuerza bruta[1]. Así lo muestra el primer ataque público de fuerza bruta exitoso del *DES* en 1997 el cual tomó aproximadamente cuatro meses en llevarse a cabo. Aún así, el *DES* permaneció en uso oficial hasta que en 1998 la *Electronic Frontier Foundation* demostró que un ataque de esta naturaleza se puede realizar en 56 horas.

A fin de encontrar un algoritmo de cifrado más robusto, fue propuesto un proceso que en términos generales consiste en la aplicación del algoritmo que utiliza el *DES* tres veces, lo que se conoce como Triple *DES*. El Triple *DES* es considerado seguro, sin embargo es demasiado lento para su implantación en hardware y software, y solamente trabaja con bloques de datos de 64 *bits*.

De manera paralela y a partir de 1997, el *NIST* (*National Institute for Standards and Technology*) de Estados Unidos inició el llamado a un proceso de selección para sustitución del *DES*: el *AES* (*Advanced Encryption Standard*). En la ronda final estuvieron compitiendo 5 algoritmos: *MARS*, *RC6*, *Rijndael*, *SERPENT* y *TWOFISH*. Finalmente el *NIST* anunció al algoritmo *Rijndael* como el ganador del *AES* en Octubre del 2000. Este proceso se efectuó con la participación de la comunidad criptográfica mundial lo cual convierte al *Rijndael* en un algoritmo robusto y digno de la confianza de todos. El nombre *Rijndael* es la combinación de los apellidos de sus autores Joan Daemen y Vincent Rijmen originarios de Bélgica.

Especificaciones

El *AES* especifica un algoritmo de cifrado simétrico por bloques con las características principales siguientes:

- Longitud de la llave con los suficientes *bits* para tener un espacio de posibles llaves mucho mayor al del *DES*[2] y resistente a ataques de fuerza bruta.
- Rapidez y flexibilidad para ejecutarse en diversas plataformas.
- Manejo de distintos tamaños de bloques de datos.

El algoritmo *Rijndael* tiene una longitud variable de llave y de bloque de datos que puede ser de 128, 192 ó 256 *bits*. Esto da como resultado que se tengan 3.4×10^{38} posibles llaves de 128 *bits*, 6.2×10^{57} posibles llaves de 192 *bits* y 1.1×10^{77} posibles

[1] Un ataque de fuerza bruta es aquel en que se prueban todas las posibles combinaciones de una llave de acuerdo a su longitud en *bits*.
[2] En el *DES* se tienen 7.2×10^{16} posibles llaves de 56 *bits*.

llaves de 256 *bits.* De manera oficial, el *AES* solamente reconoce un tamaño de llave de 128 *bits* y un tamaño variable de bloque de datos que puede ser de *128*, *19*2 y 256 *bits*.

Durante la fase de selección, el algoritmo *Rijndael* demostró una gran flexibilidad para adaptarse a diversas plataformas como las arquitecturas de 8 *bits* de las Tarjetas Inteligentes y las arquitecturas de 32 *bits* de las computadoras personales. Además, sus autores han demostrado que es posible combinar ciertas operaciones del propio algoritmo a fin de hacer mejor uso de recursos como la memoria y el procesador. Esta versatilidad hace que el algoritmo *Rijndael* sea superior en cuanto a rapidez de ejecución con respecto al Triple *DES*, por lo que el uso de este último disminuirá rápidamente.

Descripción del algoritmo

El algoritmo *Rijndael* es iterativo y se basa en aplicar un número determinado de rondas a un valor intermedio que se denomina estado. La entrada y salida del algoritmo es un bloque de datos de 128 bits lo que se representa por el parámetro N_b=4 que refleja el número de palabras de 32 *bits*. La longitud de la llave se representa por el parámetro N_k que también refleja el número de palabras de 32 bits.

Puesto que el algoritmo permite emplear diferentes longitudes de bloque y de llave, el número de rondas o iteraciones requerido en cada caso es variable. En la tabla siguiente se indica cuantas rondas (N_r) son necesarias en función de los parámetros N_b y N_k. El primer renglón representa la especificación oficial del *AES*.

	N_b=4 (128 *bits*)	N_b=6 (192 *bits*)	N_b=8 (256 *bits*)
N_k=4 (128 *bits*)	N_r=10	N_r=12	N_r=14
N_k=6 (192 *bits*)	N_r=12	N_r=12	N_r=14
N_k=8 (256 *bits*)	N_r=14	N_r=14	N_r=14

Tabla 3.1. Número de rondas del algoritmo *Rijndael* en función de los parámetros N_b y N_k.

El estado se representa mediante una matriz rectangular de *bytes* que posee cuatro filas, y N_b columnas. Para el presente trabajo se tiene que $N_b = N_k = 4$. La representación de la llave tiene una estructura similar a la del estado, y consiste de una matriz de llave de ronda con cuatro filas y N_k columnas.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Figura 3.1. Ejemplo de matriz de estado y matriz de llave de ronda con $N_b = N_k = 4$ (128 *bits*).

El bloque que se pretende cifrar o descifrar se traslada directamente *byte* a *byte* sobre la matriz de estado por columnas. La llave original también es copiada de forma similar en su matriz correspondiente; por cada iteración del algoritmo se va calculando una nueva llave, o llave de ronda, en un proceso llamado Planeación de Llaves.

Cifrado directo

El cifrado directo del algoritmo *Rijndael* se realiza mediante la ejecución de rondas de transformación las cuales están formadas por cuatro distintas transformaciones que operan a nivel de *byte*, las cuales son:

- ***Substitute Bytes***. Esta es una operación no lineal que consiste en la sustitución de un *byte* de la matriz de estado mediante el uso de una tabla de búsqueda (*S-Box*).
- ***Shift Rows.*** Esta es una operación de permutación que consiste en realizar un desplazamiento lógico hacia la izquierda sobre un *byte*. La cantidad de lugares a desplazar se hace de una manera predefinida de acuerdo al número de renglón de la matriz de estado.
- ***Mix Columns.*** Cada *byte* de la matriz de estado es reemplazado por una combinación lineal de los *bytes* en la misma columna.
- ***Add Round Key***. Consiste en realizar la operación lógica *XOR* entre la matriz de estado y la matriz de llave de ronda.

Cada ronda de transformación está parametrizada por una llave de ronda que es obtenida mediante el proceso de Planeación de Llaves. Este proceso toma como entrada la llave original y mediante un algoritmo que se explicará más adelante, se obtiene una llave para cada ronda con la misma longitud que la llave original. La siguiente figura muestra de manera esquemática la estructura del algoritmo *Rijndael* para el cifrado directo y la estructura de cada ronda de transformación.

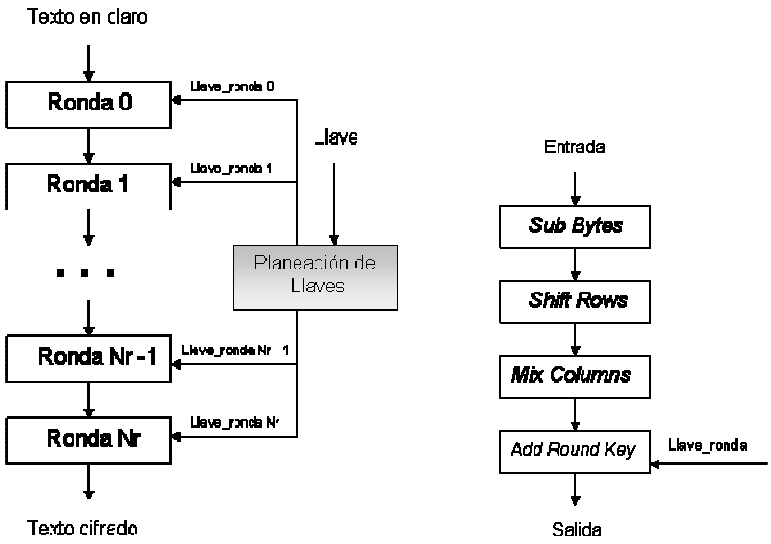

Figura 3.2. Estructura del algoritmo *Rijndael*.

Es importante mencionar que la ronda inicial consiste únicamente en realizar la función *Add Round Key* entre la matriz de estado y la llave original. Para la última ronda (N_r) se omite únicamente la transformación *Mix Columns*.

La transformación *Substitute Bytes* consiste en la sustitución de un *byte* por otro de acuerdo a la siguiente relación: primero se usa una transformación en la que cada *byte* del estado se representa de manera polinomial sobre el campo $GF(2)$ y posteriormente se calcula su inverso multiplicativo en el campo finito $GF(2^8)$ (el elemento nulo o cero se mapea a sí mismo); después se aplica una función *affine*[3] definida por la siguiente relación:

$$\begin{bmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \\ s_4' \\ s_5' \\ s_6' \\ s_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Figura 3.3. Matriz empleada por la transformación *Substitute Bytes*.

La matriz de *8x8* de la función *affine* anterior es invertible y cada uno de sus coeficientes pertenece a un elemento del campo $GF(2)$. La transformación *Substitute Bytes* puede ser expresada en su totalidad mediante la siguiente tabla de sustitución (*S-Box*).

[3] Una función *affine* tiene la siguiente estructura: $y=ax+b$ en las que a y b son constantes.

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	Ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	C3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1ª	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	Ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	Fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	Cd	0c	13	Ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	Dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	Ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabla 3.2. Tabla de sustitución *S-Box* para la transformación *Substitute Bytes*.

Si un *byte* tiene el valor {01010011}, entonces se puede separar por dos grupos de cuatro *bits* y a cada uno de ellos se le representa de forma hexadecimal entonces se tiene un valor de {53}. El resultado de la sustitución de este valor en la tabla de búsqueda se determina por la intersección del renglón marcado con 5 y la columna marcada con 3 y que corresponde a {ed} o {11011110}.

La siguiente figura muestra el efecto de la transformación *Substitute Bytes* en la matriz de estado.

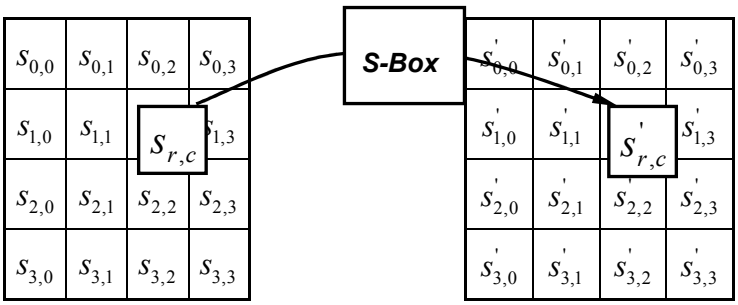

Figura 3.4. Efecto de la transformación *Substitute Bytes*.

La transformación *Shift Rows* consiste en desplazar a la izquierda las filas de la matriz de estado de manera cíclica. El desplazamiento de cada fila está en función de la longitud del bloque de datos N_b y del número de fila en la matriz de estado. En la primer fila (ó fila cero de acuerdo a la notación matricial) no existe desplazamiento.

N_b	Número de posiciones a desplazar en la fila 1	Número de posiciones a desplazar en la fila 2	Número de posiciones a desplazar en la fila 3
4	1	2	3
6	1	2	3
8	1	3	4

Tabla 3.3: Número de posiciones a desplazar según el tamaño de bloque N_b.

La siguiente figura es un ejemplo del resultado de la transformación *Mix Columns* para N_b =4.

S

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

S'

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

Figura 3.5. Efecto de la transformación *Shift Rows* para N_b = 4.

La transformación *Mix Columns* opera sobre cada columna de la matriz de estado. Cada columna se representa como un polinomio en el campo finito $GF(2^8)$, éste último es multiplicado por un polinomio fijo *c(x)* módulo el polinomio $x^4 + 1$. El polinomio *c(x)* es el siguiente:

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} .$$

La transformación *Mix Columns* puede escribirse en forma matricial de la siguiente manera:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad \text{Para } 0 \le c < N_b$$

Figura 3.6. Representación matricial de la transformación *Mix Columns*.

Como resultado de esta transformación los cuatro *bytes* de cada columna son reemplazados por los siguientes valores:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

En la transformación *Add Round Key* se realiza la operación lógica *XOR* entre la matriz de estado y la llave de ronda.

En el proceso de Planeación de Llaves, las llaves de ronda se derivan de la llave original mediante el uso de una función de expansión y otra función de selección. La función de expansión produce N_r+1 llaves de ronda, cada una de igual tamaño que la llave original lo que produce un flujo de *bytes* de tamaño $4*(N_r+1)*N_b$. La función de selección toma de manera consecutiva bloques del mismo tamaño que la matriz de estado de la secuencia obtenida, y los va asignando a cada llave de ronda.

La función de expansión tiene dos versiones según el valor de N_k. Sea *K(i)* un vector de *bytes* de tamaño $4*N_k$, que contiene la llave original, y sea *W(i)* un vector de $N_b*(N_r+1)$ registros de cuatro *bytes*, entonces:

Para $N_k \leq 6$:

1) Para *i* desde 0 hasta N_k - 1 hacer:
   W(i) ← (*K*(4 * *i*), *K*(4 * *i* + 1), *K*(4 * *i* + 2), *K*(4 * *i* + 3))
2) Para *i* desde N_k hasta N_b * (N_r + 1) hacer:
   temp ← *W*(*i* - 1)
   Si (*i* mod N_k) = 0
      temp ← **Sub**(**Rot**(temp)) ⊕ RC(*i* / N_k)
   W(i) ← *W*(*i* - N_k) ⊕ temp

Para $N_k > 6$:

1) Para *i* desde 0 hasta N_k - 1 hacer:
   W(i) ← (*K*(4 * *i*), *K*(4 * *i* + 1), *K*(4 * *i* + 2), *K*(4 * *i* + 3))
2) Para i desde N_k hasta N_b * (N_r + 1) hacer:
   temp ← *W*(*i* - 1)
   Si (*i* mod N_k) = 0
   temp ←**Sub**(**Rot**(temp)) ⊕ RC(*i* / N_k)
   Si (*i* mod N_k) = 4
      temp ← **Sub**(temp)
   W(i) ← *W*(*i* - N_k) ⊕ temp

La función **Sub** realiza la sustitución por medio de la tabla *S-Box* de cada uno de los *bytes* del registro de cuatro que se le proporciona como parámetro. La función **Rot** realiza el desplazamiento lógico hacia la izquierda en una posición de los *bytes* del registro, por lo que si esta función tiene como parámetro (W_0, W_1, W_2, W_3) el resultado será (W_1, W_2, W_3 W_0). Finalmente, RC(*i*) es una constante que pertenece al campo finito $GF(2^8)$ la cual se obtiene de la siguiente forma:

$$RC(i)=x^{i-1} \quad \text{para } i=1 \ldots N_r$$

La siguiente tabla muestra las constantes de ronda para el caso de una longitud de 128 *bits* tanto del bloque de datos como de la llave.

RC(1)	RC(2)	RC(3)	RC(4)	RC(5)	RC(6)	RC(7)	RC(8)	RC(9)	RC(10)
0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1B	0x36

Tabla 3.4. Constantes de ronda para $N_b = N_k = 4$.

Cifrado inverso

Las transformaciones usadas para el cifrado directo del algoritmo *Rijndael* son invertibles por lo que se les hará referencia como *Inv Substitute Bytes*, *Inv Shift Rows*, *Inv Mix Columns* e *Inv Add Round Key*. El cifrado inverso consiste básicamente en la aplicación de estas transformaciones inversas y las llaves de ronda en el orden contrario. El tipo de operaciones involucradas en el algoritmo *Rijndael* permite que el orden de ejecución de las transformaciones se altere a fin de que se tenga una estructura similar a la del cifrado directo, a lo que se conoce como cifrador inverso equivalente. Esto es útil para arquitecturas de 32 *bits* pero en arquitecturas de 8 *bits* no se tienen ventajas significativas.

Para la transformación *Inv Shift Rows* se realizan los desplazamientos lógicos pero hacia la derecha; se sigue la misma estructura utilizada para el cifrado directo en cuanto al número de desplazamientos de acuerdo a la fila y al parámetro N_b.

La transformación *Inv Substitute Bytes* se obtiene mediante la aplicación de la inversa de la función *affine* definida para el cifrado directo y por el cálculo del inverso multiplicativo en el campo finito de $GF(2^8)$; el elemento nulo o cero también se mapea a sí mismo. Esta transformación también puede realizarse mediante el uso de una tabla de búsqueda o de sustitución.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	A1	66	28	d9	24	B2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	F6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	A8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	B0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabla 3.5. Tabla de sustitución *S-Box* para la transformación *Inv Substitute Bytes*.

La transformación *Inv Mix Columns* opera sobre cada columna de la matriz de estado;

cada columna es tratada como un polinomio sobre el campo finito $GF(2^8)$, el cual es multiplicado por un polinomio fijo $c^{-1}(x)$ módulo el polinomio $x^4 + 1$.
El polinomio $c^{-1}(x)$ es el siguiente:

$$c^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

La transformación *Inv Mix Columns* puede escribirse en forma matricial de la siguiente manera:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad \text{Para } 0 \le c < N_b$$

Figura 3.7. Representación matricial de la transformación *Inv Mix Columns*.

Como resultado de esta transformación los cuatro *bytes* de cada columna son reemplazados por los siguientes valores:

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$
$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$
$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Los coeficientes de la matriz hacen que la transformación *Inv Mix Columns* sea más costosa computacionalmente. Sin embargo en el Capítulo 4 se mostrará que es posible factorizar el polinomio $c^{-1}(x)$ a fin de reducir la complejidad de la implantación.

Con respecto a la transformación *Add Round Key*, ésta es su propia inversa ya que involucra solamente la aplicación de la operación lógica *XOR*. Para el proceso del cálculo de llaves de ronda se tiene la opción de que durante el cifrado directo se guarde la última llave de ronda en memoria, la cual serviría para calcular todas las demás llaves de ronda en el sentido inverso. Sin embargo, se tendría el problema de que si se presenta alguna condición que interrumpa el suministro de energía a la tarjeta, entonces la última clave de ronda en la memoria se perdería. En el Capítulo 4 se analizará la opción más adecuada para el proceso de Planificación de Llaves de acuerdo a los objetivos a alcanzar en este trabajo.

Capítulo 4: Implantación del algoritmo *Rijndael* en Tarjetas Inteligentes

Consideraciones Generales para la Implantación

En el diseño y la implantación de algoritmos de cifrado simétrico en tarjetas inteligentes se debe tener siempre presente el uso eficiente de los recursos disponibles debido a las restricciones existentes en las memorias *RAM*, la *ROM*, y en el procesamiento.

En el capítulo 3 se presentó la estructura general del algoritmo *Rijndael*, el cual es un algoritmo iterativo que trabaja con bloques de datos. Por lo tanto sus operaciones internas pueden ser reducidas a operaciones de 8 *bits*, lo que representa una ventaja muy importante en arquitecturas como con la que cuenta la tarjeta *ATMega163*.

En [1], los autores del algoritmo *Rijndael* demuestran que es factible la combinación de operaciones a fin de lograr una disminución del tiempo de ejecución. Por ejemplo, las operaciones *Shift Rows* y *Substitute Bytes* pueden realizarse en un solo paso, y junto con la operación *Add Round Key* pueden ser desarrolladas con instrucciones directas que operen sobre *bytes* en microprocesadores de 8 *bits*.

Por otra parte, en el proceso de planificación de llaves se recomienda hacer un desarrollo que haga uso de un *buffer* cíclico con un tamaño no mayor al tamaño de la llave original. Esto permite que no se mantenga en uso una gran cantidad de memoria *RAM* para almacenar todas las llaves de ronda.

Finalmente, la característica de confusión en el algoritmo *Rijndael* recae en la operación *Substitute Bytes* que consiste en la sustitución de datos mediante tablas de búsqueda, o tablas de *lookup*. Esta operación requiere que se invierta una porción considerable de ciclos de ejecución debido a los procesos de búsqueda de datos en memoria. Sin embargo, se puede buscar una reducción del tiempo de ejecución mediante el conocimiento detallado del conjunto de instrucciones disponibles del microprocesador.

La plataforma ATmega163

La tarjeta *ATMega163* fue adquirida por la Universidad Nacional Autónoma de México antes de iniciar este trabajo de tesis y por lo tanto la premisa fue hacer uso de ella. La *ATMega163* es relativamente barata y además tiene la ventaja de que no es necesario firmar un contrato de confidencialidad para su adquisición. Esta tarjeta pertenece a la familia de tarjetas *Funcard* y cuenta con un microprocesador *AVR AT90S8515* de 8 *bits* tipo *RISC* (*Reduced Instruction Set Computer*) de la empresa ATMEL, con las siguientes características:

- Treinta y dos registros de propósito general.
- Arquitectura tipo *Harvard*: el espacio de memoria de datos y el espacio de memoria de programa están separados.
- Memoria de programa con tecnología *Flash ROM* de 8192 palabras (cada palabra de 2 *bytes*).
- Memoria de datos *SRAM* de 1024 *bytes*.
- Memoria interna *EEPROM* de 512 *bytes*.

- Memoria externa *EEPROM* de 256*K bytes*.
- *Bus* de datos de 8 *bits* y *bus* de direcciones de 16 *bits*.

Con los treinta y dos registros de propósito general se puede hacer el modo de direccionamiento directo a la memoria de datos, lo cual implica que se utilicen solamente dos ciclos de reloj. Además, estos registros tienen conexión directa con la Unidad Aritmética Lógica del microprocesador lo que significa que la mayoría de las instrucciones se realicen en un solo ciclo de reloj.

Los registros de propósito general se nombran del R0 al R31. Los registros X (formado por los registros R26 y R27), Y (formado por los registros R28 y R29) y Z (formado por los registros R30 y R31) son de 16 *bits*, y son utilizados como apuntadores para el direccionamiento de datos. Para acceder a la memoria de programa se usa el modo de direccionamiento indirecto en el que se usan tres ciclos de reloj. La Unidad Aritmética Lógica, de la arquitectura *ATMega163,* no tiene acceso directo a la memoria de programa tal y como se ilustra en la figura 4.1.

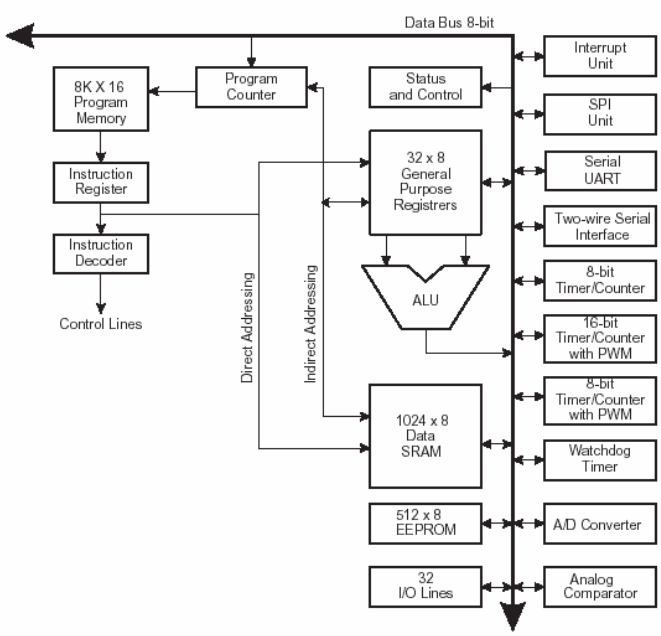

Figura 4.1. Arquitectura ATMega163.

Como en la gran mayoría de las tarjetas inteligentes, la *ATMega163* recibe la frecuencia de operación de 3.57 *MHz* del lector de tarjetas lo cual permite contar con un rendimiento de la tarjeta cercano a los 3.57 millones de instrucciones por segundo.

Herramientas de simulación, desarrollo e implantación.

Para el desarrollo de las rutinas de cifrado directo y cifrado inverso del algoritmo *Rijndael* se usa el conjunto de instrucciones disponible para la plataforma *ATMega163*, es decir, se programa a bajo nivel, o ensamblador. Para tal propósito se tiene el ambiente de desarrollo *AVR Studio 4*, el cual es proporcionado de manera gratuita por la empresa ATMEL. *AVR Studio 4* ofrece una plataforma para la simulación del código producido, sobre sistemas operativos *Windows*, para todos los microprocesadores de la familia *AVR*.

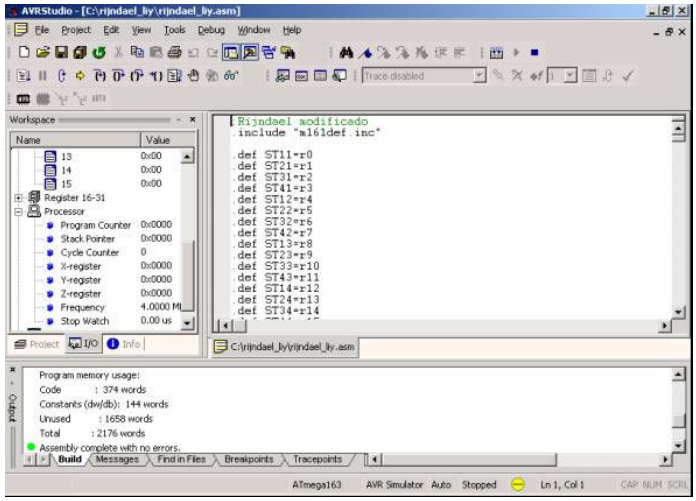

Figura 4.2. La herramienta *AVR Studio 4*.

Con *AVR Studio 4* se pueden obtener las mediciones que son de interés como son: el tiempo de ejecución, el número de ciclos de reloj usados, el tamaño del código y la cantidad de memoria utilizada. Este ambiente posee bastante flexibilidad ya que tiene la opción de establecer la frecuencia a la que debe trabajar el microprocesador de la tarjeta inteligente, lo cual permite tener una simulación apegada a la implantación real. Para comprobar este punto, se realizaron pruebas con el código producido en [10] y cuyo método de medición fue distinto al que se propone en este trabajo de tesis. Los resultados obtenidos con *AVR Studio 4* fueron los mismos que están reportados en dicha fuente.

Sin embargo *AVR Studio 4* no cuenta con un compilador propio y requiere apoyarse en uno externo. Además, el código producido debe estar incorporado en el sistema operativo de la tarjeta inteligente, el cual para este trabajo es el sistema operativo *SOSSE*.

Por tal motivo, el compilador *avr-gcc* resulta ser la opción ideal. Este compilador es de dominio público y su distribución incluye el paquete *AVR Libc*[1] que es un subconjunto de la librería de C estándar para los microprocesadores de la familia *AVR* de ATMEL.

La herramienta *AVR Studio 4* y el compilador *avr-gcc* utilizan diferentes directivas a nivel ensamblador, por lo que se deben tener en cuenta las consideraciones indicadas en la tabla 4.1 para realizar el cambio de versiones de código fuente.

Directiva en *AVR Studio 4*	**Directiva en *avr-gcc***
.asm	.S
.def	.set
.db	.byte
.org	.balign hi
hi	.hi8
Lo	.lo8

Tabla 4.1. Equivalencias de directivas en ensamblador entre *AVR Studio 4* y *avr-gcc*.

[1] *http://savannah.nongnu.org/projects/avr-libc*

Junto con la tarjeta *ATMega163* fue adquirido también por la Universidad Nacional Autónoma de México el programador por *hardware* de tarjetas. Este programador es el *MasterCRD2* el cual soporta varios modos de operación.

Figura 4.3. El programador *MasterCRD2*.

Los modos que son de interés para este trabajo son el Modo 4, que permite la programación de la tarjeta con microprocesador *AVR* de ATMEL a fin de insertar el sistema operativo *SOSSE* y las rutinas de cifrado y descifrado del algoritmo *Rijndael*, y el Modo 1 que permite enviar comandos (*APDU´s*) hacia la tarjeta mediante el protocolo T=0 a una frecuencia de 3.57 *MHz*.

El *software* para cargar el código en la tarjeta es el *Master Burner*; este programa permite seleccionar el tipo de programador, el tipo de tarjeta, el código de programa (*SOSSE* más el algoritmo *Rijndael*), y los datos que residirán en la memoria *EEPROM*.

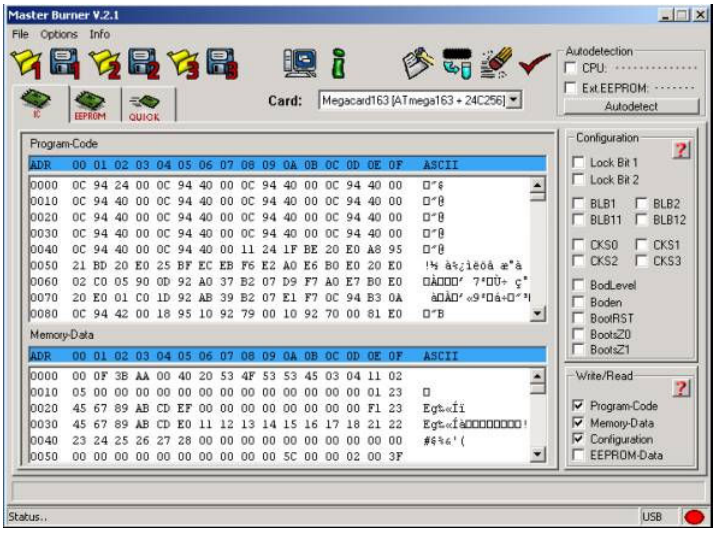

Figura 4.4. Software *Master Burner* para la programación de tarjetas.

Criterios de optimización y desarrollo del algoritmo Rijndael.

Generales.

La primera decisión importante para lograr los objetivos de eficiencia y rendimiento de este trabajo fue la de realizar la implementación del algoritmo *Rijndael* en lenguaje ensamblador. Esto permitirá que se tenga un mayor control sobre la lectura y escritura de datos en memoria y sobre la correspondiente manipulación de los datos dentro de

cada una de las operaciones del algoritmo.

Después de analizar el conjunto de instrucciones de la *ATMega163* se optó por mantener el estado del algoritmo de manera permanente en los registros de propósito general. Esta estrategia reducirá el tamaño del código y el tiempo de ejecución del programa ya que las operaciones *Add Round Key, Shift Rows*, *Substitute Bytes, Mix Columns* y sus correspondientes operaciones inversas pueden ser implementadas en su mayor parte con instrucciones directas sobre los datos de los registros. Por lo tanto se disminuirá el número de operaciones de lectura y escritura de resultados temporales a la memoria *SRAM*.

Cifrado directo.

La ejecución de las operaciones *Shift Rows* y *Substitute Bytes* se realizará en un solo paso. Esto es factible ya que el orden en que son realizadas estas operaciones no afecta el resultado final y por lo tanto se logra la reducción del número de ciclos de ejecución.

La tabla de búsqueda usada por la operación *Substitute Bytes* se guardará en la memoria *Flash ROM*. Esta decisión tiene dos objetivos: el primero es preservar la memoria *SRAM* y el segundo es disminuir el tiempo de ejecución ya que el proceso de lectura de un dato en la memoria *EEPROM* ocupa 3 ciclos de reloj adicionales respecto al proceso de lectura de un dato en la memoria *Flash ROM*.

En la operación *Mix Columns* se requiere implementar una multiplicación de matrices sobre el campo de *Galois* GF(2^8); la multiplicación del coeficiente {02} (polinomio x) y el *byte* de estado se puede realizar de dos formas. En la primera opción, la multiplicación se resuelve a nivel código mediante un simple desplazamiento lógico a la izquierda y con el tratamiento adecuado del posible acarreo, sin embargo esta postura tiene la desventaja de no ofrecer un tiempo constante de ejecución ya que éste depende del valor del *byte* actual que se esté analizando y por lo mismo la implementación podría estar sujeta con mayor facilidad a ataques del tipo que exploten fugas de información.[2] La segunda opción consiste en implantar la multiplicación por medio de una tabla de búsqueda, lo cual tiene la ventaja de ofrecer un tiempo constante de ejecución a costa de ocupar más espacio en memoria. Ambas opciones serán exploradas en este trabajo de tesis.

Con el propósito de mantener ocupada la menor cantidad de memoria *SRAM* se usará la variante que consiste en el cálculo de cada llave de ronda sobre demanda, y que implica el uso permanente de sólo 16 *bytes* durante toda la ejecución de esta rutina. La llave original residirá también en la memoria *Flash ROM* y se trasladará a la memoria *SRAM* para el proceso de planificación de llaves de ronda.

La siguiente figura muestra el diagrama de flujo de la rutina de cifrado directo a implementar con los bloques principales de las operaciones del algoritmo *Rijndael*.

[2] Ataques como *SPA (Simple Power Analysis) y DPA (Differential Power Analysis)*.

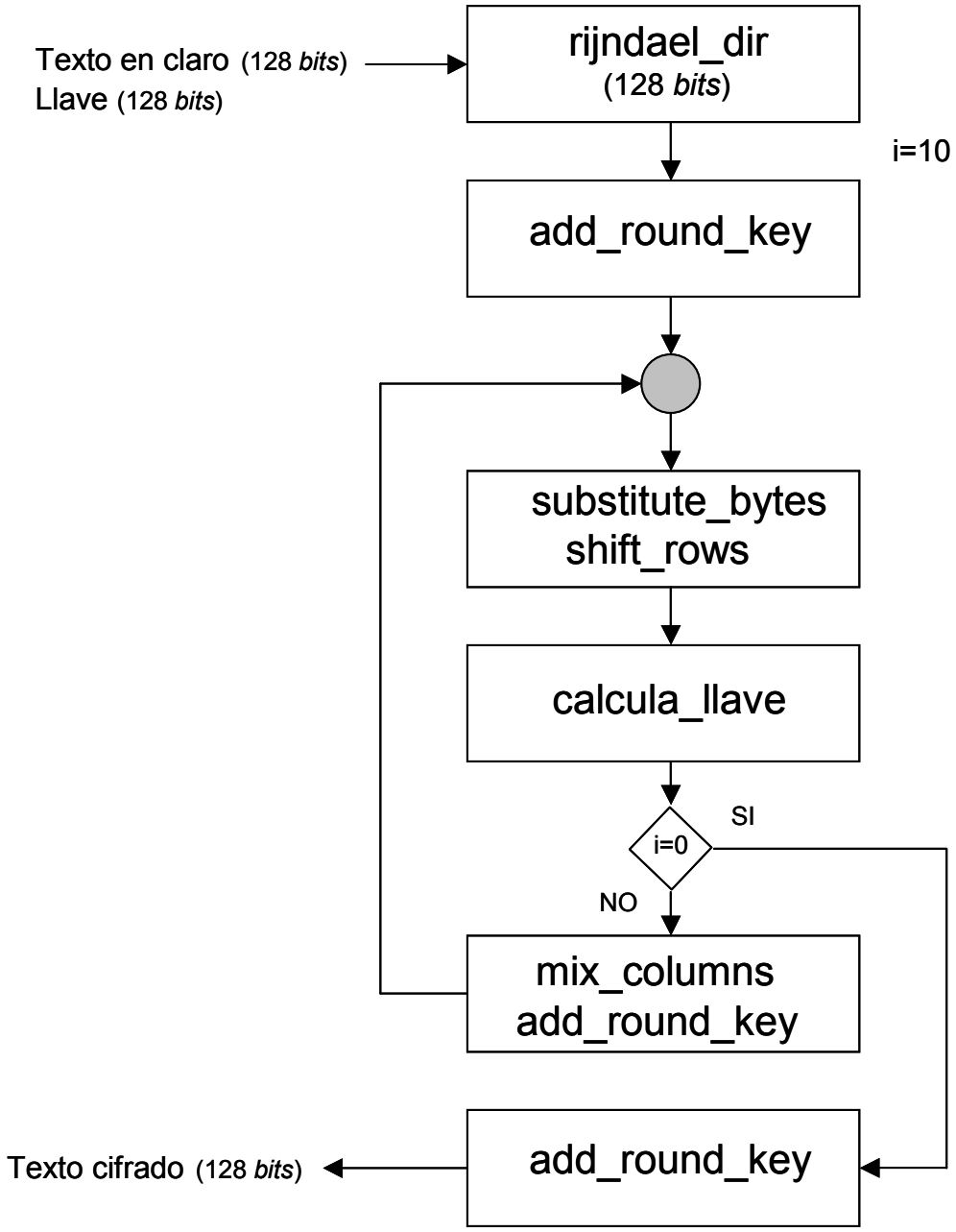

Figura 4.5. Diagrama de flujo de la rutina de cifrado directo.

Cifrado inverso.

Las operaciones *Inv_Shift Rows* e *Inv_Substitute Bytes* tienen una lógica de implementación bastante similar a sus correspondientes en el cifrado directo por lo que se mantiene su ejecución en un solo paso. Sin embargo *Inv_Substitute Bytes* requiere de una tabla de búsqueda distinta a la usada por *Substitute Bytes*; esta tabla también se encontrará en la memoria *Flash ROM*.

El cifrado inverso demanda el uso de las llaves de ronda en el sentido inverso, es decir, se usa al principio la última llave de ronda y así sucesivamente. Para resolver la implementación se tienen dos opciones:

- Calcular todas las llaves con la misma rutina empleada en el cifrado directo, pero con la variante de almacenarlas en memoria *SRAM* para su posterior uso.
- Calcular todas las llaves con la misma rutina empleada en el cifrado directo y almacenar únicamente la última llave. A partir de este punto, calcular las llaves en el sentido inverso sobre demanda.

La primera opción implica que se ocupen 176 *bytes* de la memoria *SRAM* durante toda la rutina. Esto se aleja del objetivo de contar con una implementación que haga uso eficiente de los recursos de la tarjeta, por lo que esta solución solamente se va a implementar para reportar el número de ciclos y el tiempo de ejecución utilizados.

Con la segunda opción se mantiene el uso de 16 *bytes* solamente de la memoria *SRAM* pero con la desventaja de realizar un doble procesamiento en la planificación de llaves. Obviamente el número de ciclos de reloj y el tiempo de procesamiento aumentarán, sin embargo esta penalización no será demasiada debido a la optimización del código en otros aspectos, lo que compensará el resultado final. Por lo tanto esta alternativa es la que tiene más apego a los objetivos planteados en este trabajo.

Para la operación *Inv_Mix Columns* se tiene que la matriz correspondiente está compuesta por los coeficientes {09}, {0E}, {0B} y {0D}. En esta condición, la implementación de la multiplicación correspondiente sobre el campo de *Galois* GF(2^8) resulta lenta para arquitecturas de 8 bits. Sin embargo en [1] se presenta la relación existente entre el polinomio *c(x)* usado en *Mix Columns* y el polinomio *d(x)* usado en *Inv_Mix Columns* y que en forma matricial se expresa de la siguiente manera:

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{pmatrix}$$

Figura 4.6. Relación existente entre los polinomios *c(x)* y *d(x)*.

Esta relación permite que la operación *Inv_Mix Columns* se implemente por medio de una etapa de preprocesamiento, seguida de la operación *Mix Columns,* lo cual permitirá que se aproveche una parte del código creado para el cifrado directo.

Cabe mencionar que solamente se implantará la operación *Inv_Mix Columns* por medio de una tabla de búsqueda, ya que si se toma la opción de usar solamente las instrucciones del microprocesador, se tendría un código demasiado grande debido a la gran cantidad de condiciones y saltos empleados para el manejo de los acarreos. La tabla es la misma usada en la operación *Mix Columns*.

La siguiente figura muestra el diagrama de flujo de la rutina de cifrado inverso a implementar con los bloques principales de las operaciones del algoritmo *Rijndael*.

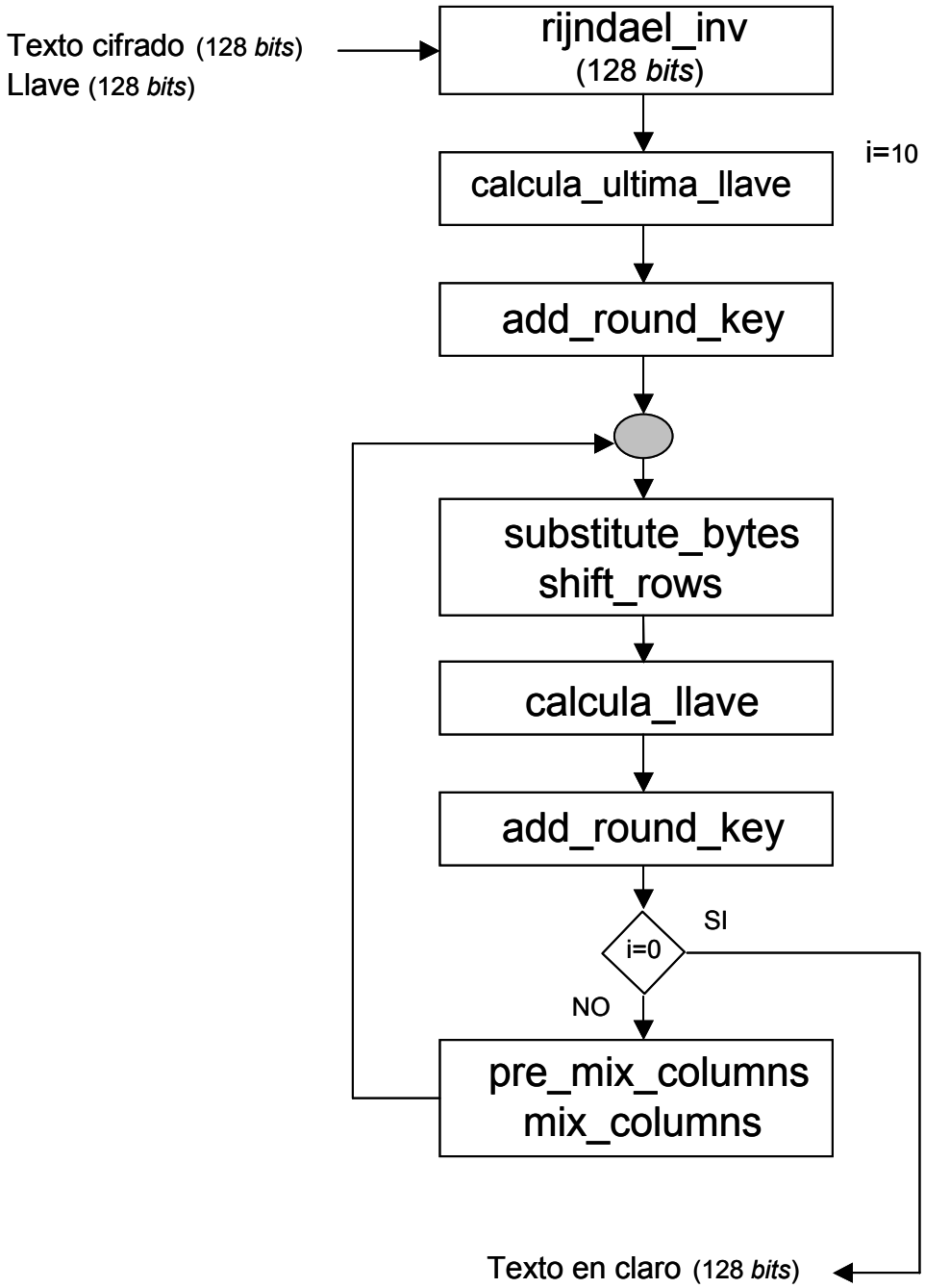

Figura 4.7. Diagrama de flujo de la rutina de cifrado inverso.

Integración del algoritmo en el sistema operativo SOSSE.

Descripción y estructura.

El sistema operativo *SOSSE* (*Simple Operating System for Smartcard Education*)[3] es de dominio público[4] y fue desarrollado primordialmente para propósitos de educación e investigación. El soporte de *SOSSE* solamente tiene cobertura para todos los

[3] *Desarrollado por Matthias Brüestle*
[4] *Disponible en http://www.franken.de/users/mbsks/sos/*

microprocesadores de la familia *AVR*.

Con *SOSSE* se tiene acceso a su código fuente lo que permite hacer las modificaciones necesarias para integrar las rutinas desarrolladas. Este sistema operativo se encarga de toda la funcionalidad de la tarjeta y está escrito en su mayoría, en lenguaje C; la excepción son algunas rutinas que se encargan de los aspectos de bajo nivel y que forman una capa de abstracción *(Hardware Abstraction Layer)* que facilita la incorporación de nuevos comandos.

La estructura general de *SOSSE* es la siguiente:

- **Sistema de archivos.** El sistema operativo *SOSSE* cuenta con un sistema de archivos jerárquico.
- **Capa de abstracción de *hardware*.** Está formada por el conjunto de funciones para transmisión y recepción de datos mediante el protocolo T=0.
- **Módulo de comandos.** Contiene todos los comandos responsables de ejecutar las instrucciones recibidas del lector. Estas funciones realizan tareas específicas y sus resultados establecen el estado de la tarjeta y envían la *APDU* de respuesta hacia el lector.
- **Módulo de autenticación.** Consta de funciones para el control de acceso hacia los archivos y para el uso de un *PIN (Personal Identification Number)* y de un proceso de reto y respuesta *(Challenge-Response)*.
- **Módulo de control.** Funciona como un ciclo continuo en el que se está en espera de los comandos a ejecutar.

Inserción del algoritmo *Rijndael* en *SOSSE*.

Una vez que ya se tiene el código del algoritmo *Rijndael* con las directivas en ensamblador de *avr-gcc*, se tiene la posibilidad de integrarlo en el sistema operativo *SOSSE*. El siguiente diagrama sirve de referencia para entender la forma en que se tendrá la interacción entre el sistema anfitrión, el lector y la Tarjeta Inteligente.

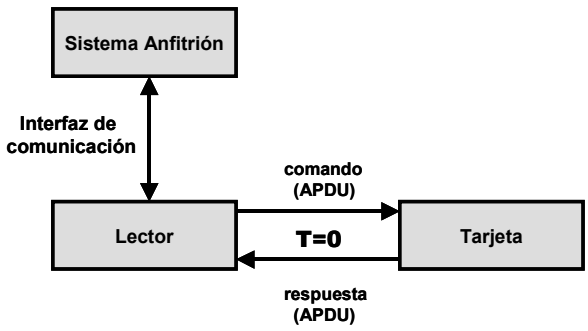

Figura 4.8. Interacción entre sistema anfitrión, lector y tarjeta inteligente.

La interfaz de comunicación entre el sistema anfitrión y el lector consiste de un manejador tipo *PC/SC (Personal Computer/Smart Card)* lo que permite la interacción transparente entre computadoras personales y tarjetas. Para este trabajo se utilizó el

manejador *PC/SC* genérico *Dumbmouse driver*[5].

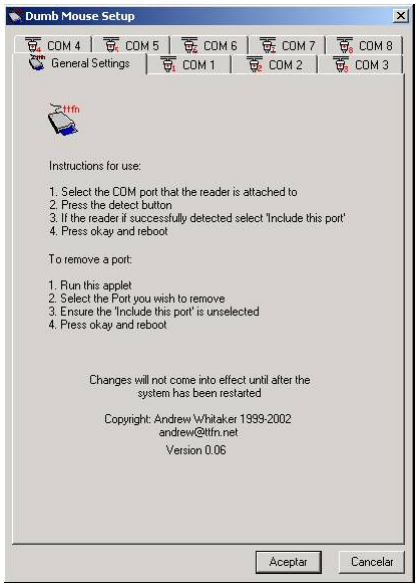

Figura 4.9. El manejador *PC/SC*.

El programa *Kobil Smart Card Terminal*[6] se usó como aplicación en la cual el usuario ingresa los comandos (*APDU´s*) para indicar el cifrado o descifrado de datos. El dispositivo *MasterCRD2* funciona como el lector de tarjetas mediante el modo de programación 1. El intercambio de comandos y respuestas entre el lector y la tarjeta *ATMega163* se realiza por medio del protocolo T=0.

En este proceso de inserción hubo que tomar en cuenta que el código desarrollado está en ensamblador, el cual debe ser llamado por rutinas en lenguaje C en el que está escrito *SOSSE*. Por lo tanto se realizaron las siguientes adecuaciones en el código:

- Apego a la convención de llamadas a funciones, utilizada en *Avr-libc,* que consiste en que los argumentos son manejados por los registros R25 al R8. Por lo tanto se modificó el código para que recibiera el texto en claro, o el texto cifrado, en la dirección contenida en los registros R24 y R25, y para que el resultado también se guarde a partir de la misma dirección indicada por estos registros.
- Manejo de un solo archivo para el código en ensamblador, al cual se le nombró

[5] *http://www.ttfn.net/techno/dm.html*
[6] *Componente de las soluciones del fabricante Kobil Systems (www.kobil.com)*

rijndael.S, y que contiene las rutinas de cifrado directo y cifrado inverso, que pueden ser llamadas desde el lenguaje C por medio de las directivas:

- `.global rijndael_dir`
- `.type rijndael_dir, @function`
- `.global rijndael_inv`
- `.type rijndael_inv, @function`

Donde `rijndael_dir` es el nombre de la rutina de cifrado directo en ensamblador y `rijndael_inv` es el nombre de la rutina de cifrado inverso también en ensamblador.

Se creó el archivo *rijndael.h* con las declaraciones en lenguaje C que sirven para invocar a las rutinas desarrolladas en ensamblador del archivo *rijndael.S*. Las declaraciones son las siguientes:

- Para el cifrado directo: `void rijndael_dir (iu8 *textcl)`
- Para el cifrado inverso: `void rijndael_inv (iu8 *textci)`

Donde `iu8` corresponde a la definición de dato de un *byte* sin signo, que es usado en *SOSSE* (archivo *types.h*), y `textcl` y `textci`, son apuntadores al texto en claro y al texto cifrado, respectivamente.

Por medio de comandos o *APDU´s,* se envía a la tarjeta el texto en claro, o el texto cifrado, para que se realice la operación correspondiente. Se modificó el archivo *commands.h* de *SOSSE* para indicar el número de instrucción correspondiente; las líneas agregadas son:

- `#define INS_rijndael_dir  0x60`
- `#define INS_rijndael_inv  0x62`

CLA	INS	P1	P2	P3	Datos
80	60	00	00	10	Texto Claro (16 Bytes)

Figura 4.10. Estructura de la *APDU* para el cifrado.

CLA	INS	P1	P2	P3	Datos
80	62	00	00	10	Texto Cifrado (16 Bytes)

Figura 4.11. Estructura de la *APDU* para el descifrado.

En el archivo *commands.c* se definieron las funciones en lenguaje C de los comandos de cifrado directo y de cifrado inverso. Estas funciones se forman por el comando de recepción de bloque de datos por medio del protocolo T=0 y por el comando para el

cifrado directo, o para el cifrado inverso, del algoritmo *Rijndael*.

En el archivo *main.c*, el cual se encarga del ciclo continuo de atención para la ejecución de comandos de *SOSSE*, se hace el llamado de los comandos de cifrado directo, y de cifrado inverso, de acuerdo con el número de instrucción definido en el archivo *commands.h*.

El resultado, el texto cifrado, o el texto descifrado, se obtiene mediante el comando *Get response* que ya está implementado en *SOSSE*. Este comando se envía hacia la tarjeta y el resultado es devuelto junto con los *bytes* de estado (90 y 00 si no hubo ningún problema).

CLA	INS	P1	P2	P3
80	C0	00	00	10

....	SW1	SW2
Texto cifrado o descifrado (16 bytes)	90	00

Figura 4.12. Estructura del comando *Get Response*.

Capítulo 5: Resultados

Resumen de la Implantación

Con la finalidad de obtener mayor control de las operaciones con los datos así como en el manejo de los recursos disponibles de la tarjeta *ATMega163,* se utilizó el lenguaje ensamblador para el desarrollo de las rutinas de cifrado directo y de cifrado inverso del algoritmo *Rijndael* para un tamaño de llave de 128 *bits*.

Por otra parte, ciertas estrategias de optimización del algoritmo *Rijndael* fueron seleccionadas con el objetivo de balancear los criterios de rapidez de ejecución y el uso de recursos disponibles de la tarjeta inteligente. Estas estrategias se enlistan a continuación.

- Mantenimiento del estado del algoritmo de manera permanente en los registros de propósito general.
- Ejecución de las operaciones *Shift Rows* y *Substitute Bytes* en un solo paso.
- Ejecución de las operaciones *Mix Columns* y *Add_Round_Key* en un solo paso.
- Ejecución de las operaciones *Inv_Shift Rows* e *Inv_Substitute Bytes* en un solo paso.
- Cálculo de cada llave de ronda sobre demanda.
- Uso de la memoria *Flash ROM* para el almacenamiento de las tablas de búsqueda.

Además, se realizaron algunas variantes en el diseño y creación de las rutinas. Por ejemplo, en el cifrado directo se realizaron dos versiones para la implementación de la operación *Mix Columns*: en la primera versión se usa una tabla de búsqueda, y en la segunda versión se utiliza el desplazamiento lógico hacia la izquierda y se hace el manejo del posible *bit* de acarreo. Para el cifrado inverso también se crearon dos versiones: en la primera versión se calcula la última llave de ronda y a partir de ésta se calculan todas las demás, y en la segunda versión se generan todas las llaves de ronda y se almacenan en memoria *SRAM* para su posterior uso. En la siguiente sección se muestran los resultados obtenidos para cada una de las versiones creadas del algoritmo *Rijndael*.

Métricas obtenidas

Con la herramienta *AVR Studio 4* se realizó la simulación del código producido, con la cual se obtuvieron: el tiempo de ejecución, el número de ciclos de reloj usados, el tamaño del código y la cantidad de memoria utilizada.

Esta información resultó muy importante, ya que fue la pauta para conocer si las estrategias escogidas para la optimización del algoritmo *Rijndael* eran correctas, y también para hacer los ajustes necesarios en el código mediante el conjunto de instrucciones disponibles en lenguaje ensamblador.

Como se mencionó en el primer capítulo, un objetivo específico de este trabajo de tesis

fue la creación de una implementación del cifrado directo más rápida y que use recursos equiparables con la versión reportada en [10]. Este hecho sirvió también como un criterio para decidir si la rutina de cifrado directo iba por buen camino.

Para el caso del cifrado inverso, se buscó que la implementación privilegiará el uso óptimo de memoria ya que ésta es un recurso limitado en una tarjeta inteligente, y que al mismo tiempo, se logrará que conviviera con la rutina de cifrado directo y el sistema operativo *SOSSE*. Las siguientes tablas contienen la información relevante para cada versión desarrollada del algoritmo *Rijndael*.

Versión (cifrado directo)	**Ciclos de reloj**	**Tamaño del Código (bytes)**	**Memoria RAM (bytes)**	**Tiempo de ejecución (ms)**
rijndael_dir_v1	3899	1252	32	0.97475

Características Principales.

1) Legibilidad del código.
- Solamente se usaron *loops* para el control de las iteraciones de las rondas.

2) Uso óptimo de memoria *SRAM*.
- La llave de ronda se calcula conforme se va necesitando.

3) El texto o estado siempre se mantiene en los registros R0 a R15.
- Disminución en el número de ciclos ya que la manipulación de los datos en los registros se realiza mediante el direccionamiento directo.

4) Uso de tabla de búsqueda en la función *Mix_Columns* para la función *xtime*.
- Tiempo constante en la ejecución del código.
- Aumento del tamaño del código en 256 *bytes*.

6) Se realizó la *serialización de operaciones* pero en partes:
- Las funciones de *Sub_Bytes* y *Shift-Rows* fueron realizadas en un solo paso y las funciones *Mix_Columns* y *Add_Round_Key* también fueron realizadas en un solo paso.

Tabla 5.1. Métricas obtenidas para la versión rijndael_dir_v1

Versión	**Ciclos de reloj**	**Tamaño del Código (bytes)**	**Memoria RAM (bytes)**	**Tiempo de ejecución (ms)**
rijndael_dir_v2	3755	1018	32	0.93875

Características Principales.

1) No se usó la tabla de búsqueda en la función *Mix_Columns* para la función *xtime*.
   - Se resolvió a nivel código: se utilizó el comando *lsl* y una condición para realizar la operación *xor* con la constante 1B (hex).
   - El tiempo de ejecución de la función *Mix_Columns* no es constante debido a que depende del estado del algoritmo.

2) Se tienen las otras mismas características de la versión rijndael_dir_v1.

Tabla 5.2. Métricas obtenidas para la versión rijndael_dir_v2

Versión	Ciclos de reloj	Tamaño del Código (bytes)	Memoria RAM (bytes)	Tiempo de ejecución (ms)
rijndael_inv_v1	5138	1470	32	1.2845

Características Principales.

1) Legibilidad del código.
   - Solamente se usaron *loops* para el control de las iteraciones de las rondas.

2) Uso óptimo de memoria *SRAM*.
   - La llave de ronda se calcula conforme se va necesitando.

3) Reprocesamiento en la etapa de derivación de llaves de ronda.
   - Primero se calcula la última llave de ronda, lo que implica calcular todas las llaves de ronda. A partir de la última llave de ronda se van calculando las llaves de ronda en orden inverso y se van escribiendo en la misma dirección base de la memoria *SRAM*.

4) El texto cifrado o estado siempre se mantiene en los registros R0 a R15.
   - Disminución en el número de ciclos ya que la manipulación de los datos en los registros se realiza mediante el direccionamiento directo

5) Uso de etapa de preprocesamiento en la función *Inv_Mix_Columns*:
   - Se implementa la etapa de preprocesamiento sin la ayuda de una tabla de búsqueda, y se utiliza el código creado para la función *Mix_Columns* del cifrado directo de la versión rijndael_dir_v1.
   - El tiempo de ejecución de la función *Inv_Mix_Columns* no es constante debido a que depende del estado del algoritmo.

6) Se realizó la *serialización de operaciones* pero en partes:
   - Las funciones de *Inv_Sub_Bytes* e *Inv_Shift-Rows* fueron realizadas en un solo paso y las funciones *Inv_Mix_Columns* y *Add_Round_Key* también fueron realizadas en un solo paso.

Tabla 5.3. Métricas obtenidas para la versión rijndael_inv_v1

Versión	Ciclos de reloj	Tamaño del Código (bytes)	Memoria RAM (bytes)	Tiempo de ejecución (ms)
rijndael_inv_v2	4228	1334	192	1.057
Características Principales. 1) No existe el preprocesamiento en la etapa de derivación de llaves de ronda pero se usa una mayor cantidad de memoria *SRAM*. • Se calculan las llaves de ronda y se van almacenando en memoria *SRAM*. El procesamiento se hace usando los registros (direccionamiento directo). Una vez que se tienen calculadas las llaves se van utilizando en orden inverso. 2) Se tienen las otras mismas características de la versión rijndael_inv_v1.				

Tabla 5.4. Métricas obtenidas para la versión rijndael_inv_v2

Comparación con otros trabajos realizados

En el trabajo realizado por [10] se realizaron dos versiones del algoritmo *Rijndael* para el cifrado directo: el *AES* Rápido y el *AES* Pequeño. Es importante mencionar que la premisa primordial de ese trabajo fue la de lograr implantaciones que convivieran junto con el sistema operativo *SOSSE* en la tarjeta inteligente *AT90S8515*, la cual solamente posee 8 *Kbytes* de memoria *Flash ROM*. Además, se reporta que la versión rápida no fue optimizada en su velocidad y que la versión pequeña solamente reduce el tamaño del código cuando la velocidad no se ve disminuida de manera notable. A continuación se enlistan las métricas reportadas para las dos versiones creadas en [10]:

Versión	Ciclos de reloj	Tamaño del Código (bytes)	Memoria RAM (bytes)	Tiempo de ejecución (ms)

AES Rápido	4911	772	32	1.29
AES Chico	6553	598	32	1.65

Tabla 5.5. Métricas obtenidas para las versiones de cifrado directo del algoritmo *Rijndael* reportadas en [10]

Al tamaño del código reportado para ambas versiones se deben agregar los 256 *bytes* de la tabla de búsqueda, ya que ésta fue definida de manera externa al programa principal y por lo tanto esta cantidad de *bytes* no fue contabilizada.

Las siguientes gráficas muestran los resultados obtenidos en este trabajo comparados con las versiones creadas en la referencia [10].

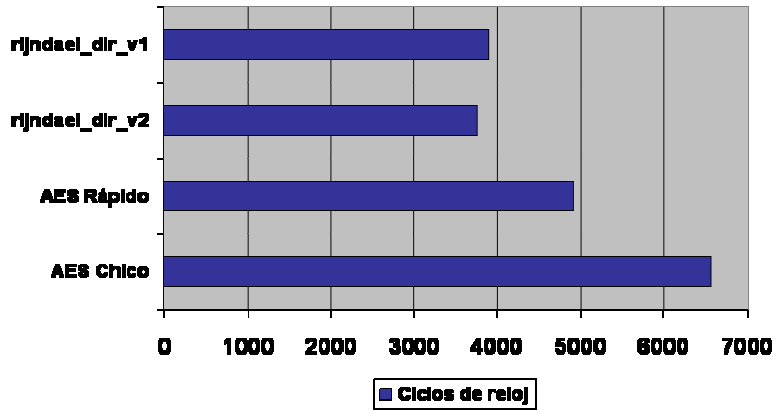

Figura 5.1. Ciclos de reloj de cada versión creada del algoritmo *Rijndael*.

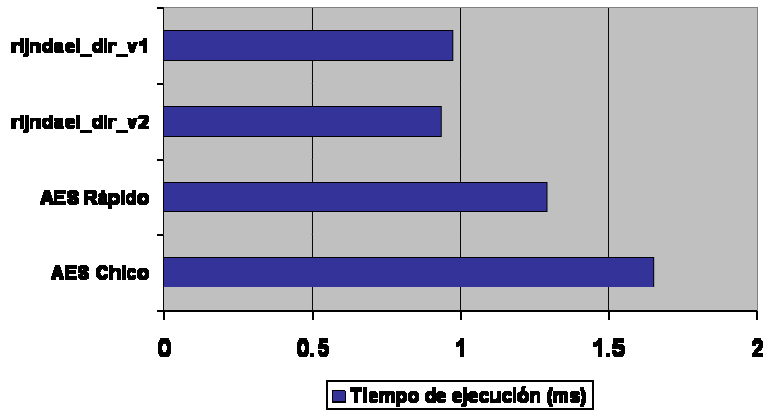

Figura 5.2. Tiempo de ejecución de cada versión creada del algoritmo *Rijndael*.

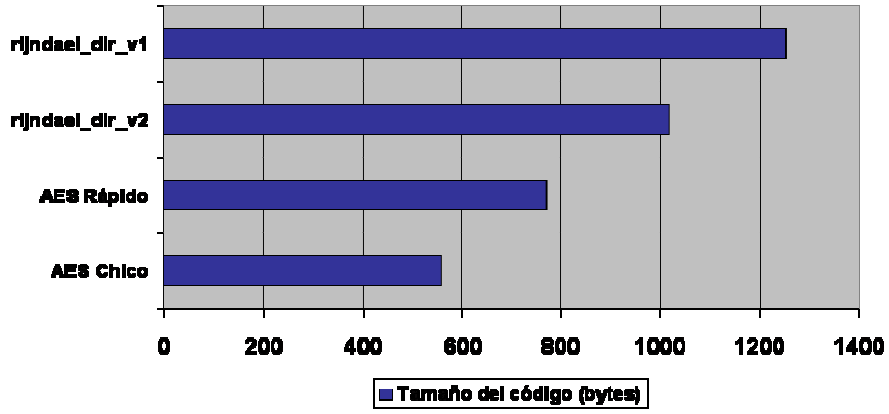

Figura 5.3. Tamaño del código de cada versión creada del algoritmo *Rijndael*.

Como se puede apreciar en las gráficas, la implementación rijndael_dir_v2 fue la que resultó más rápida, tanto en ciclos de reloj como en tiempo de ejecución, aunque este último parámetro no es constante debido a la forma en que la operación *Mix_Columns* fue implementada. La implementación rijndael_dir_v1 ocupó más *bytes* de código que todas las demás, pero al mismo tiempo fue la única versión que ofreció un tiempo de ejecución constante debido al uso de tablas de búsqueda, lo cual elimina la dependencia en el valor del estado del algoritmo.

Para el caso del cifrado inverso, la versión rijndael_inv_v1 logró convivir con las dos versiones del cifrado directo creadas en este trabajo y con *SOSSE*, y por lo tanto se alcanzó el objetivo de incorporar el algoritmo *Rijndael* como una primitiva criptográfica a nivel sistema operativo de una tarjeta inteligente.

Capítulo 6: Conclusiones

Para obtener una implantación eficiente del algoritmo *Rijndael* fue necesario desarrollarlo en lenguaje ensamblador a fin de tener mayor facilidad para acceder a los recursos de la tarjeta *ATMega163*. De igual manera, el aprendizaje del conjunto de instrucciones del microcontrolador de la tarjeta resultó imprescindible para eficientar las operaciones de lectura y escritura de datos de la memoria. Estos dos aspectos permitieron que se obtuviera un conocimiento mayor sobre la forma en que cada una de las transformaciones del algoritmo afecta al estado.

Por otra parte, se estudiaron las estrategias de optimización del algoritmo *Rijndael* a fin de seleccionar aquellas que permitieran los mejores beneficios en cuanto a rapidez de ejecución y uso de recursos de la tarjeta *ATMega163*. Durante esta etapa se observó la concurrencia constante entre seguridad y rendimiento, por ejemplo, para el cifrado directo se contó con dos opciones para la implementación de la operación *Mix Columns*: la primera utiliza una tabla de búsqueda, y la segunda utiliza el desplazamiento lógico hacia la izquierda y hace el manejo del posible *bit* de acarreo. Con la tabla de búsqueda se ocupa más memoria pero se ofrece un tiempo constante de ejecución del algoritmo ya que el acceso a un dato no depende de su valor, mientras que con el uso de instrucciones del microcontrolador se ahorra espacio en memoria pero la implantación del algoritmo está sujeta a ataques de medición de tiempo ya que la ejecución de *Mix Columns* depende del valor *byte* a transformar. Obtener el adecuado balance entre la seguridad y el rendimiento es un factor crítico para toda aplicación que pretenda proteger la confidencialidad e integridad de la información, lo cual representa un verdadero reto para los encargados del desarrollo e implantación.

Para el cifrado directo se obtuvieron tiempos de ejecución más bajos que los reportados en otros trabajos tal y como se mostró en el Capítulo 5. Un logro importante en este trabajo de tesis fue la creación de la rutina de cifrado inverso (descifrado), de la cual se crearon dos versiones que giran en torno a la forma en que se generan las llaves de ronda. Para estas dos versiones, no se encontraron referencias para comparar resultados, sin embargo el hecho de que fuese posible su implantación en la tarjeta *ATMega163* fue un logro muy significativo.

El ambiente de desarrollo *AVR Studio 4*, resultó una herramienta muy poderosa para la simulación del código producido. Con *AVR Studio 4* fue posible detectar las partes del código que tenían que ser depuradas y optimizadas a fin de alcanzar mejores tiempos de ejecución sin descuidar el uso de la memoria disponible. Las estrategias seleccionadas para la optimización del algoritmo fueron probadas con esta herramienta, y resultó de mucha ayuda, la capacidad de visualizar de manera permanente el estado de cada una de las transformaciones en los registros de propósito general del microcontrolador.

Debido a que el sistema operativo *SOSSE* es de dominio público se tiene acceso a su código fuente. Esto permitió realizar las modificaciones necesarias en su estructura para integrar las rutinas de cifrado directo y cifrado inverso del algoritmo *Rijndael*. Como resultado, se logró la inserción de estas rutinas como un comando del sistema operativo *SOSSE*, lo cual servirá para que se utilicen como bloques básicos en la construcción de comandos más complejos.

Más allá de los retos impuestos por la misma plataforma para crear una implantación factible y eficiente del algoritmo, la mayor enseñanza personal fue la forma en que diversas áreas como la Criptografía, la arquitectura de microcontroladores, la tecnología de las Tarjetas Inteligentes, y la misma programación y los ambientes de desarrollo, se complementan y entrelazan para tener una aplicación práctica del cifrado simétrico de llave privada a través del algoritmo *Rijndael.*

Con la finalidad de dar un seguimiento posterior a este trabajo, es recomendable la creación de comandos más complejos que se basen en las rutinas ya creadas. Por ejemplo, la autenticación de la tarjeta mediante un esquema de reto – respuesta (*challenge - response*) es posible ya que se tiene la capacidad de realizar el descifrado. El esquema de trabajo aquí planteado permitirá que se apliquen conceptos y propuestas sobre diversas áreas de la Criptografía en una plataforma abierta.

Bibliografía

[1] Daemen Joan, Rijmen Vincent, “Efficient Block Ciphers for Smartcards”, USENIX Workshop on Smartcard Technology, Mayo 1999.
[2] Electronic Frontier Foundation, “EFF DES Cracker”, http://www.eff.org/descracker.html
[3] AES NIST Webpage, http://www.nist.gov/aes
[4] Lu Chi-Feng, Kao Yan-Shun, Chiang Hsia-Ling, Yang Chung-Huang, “Fast implementation of AES Cryptographic Algorithms in Smart Cards”, http://crypto.nknu.edu.tw/publications/2003ICCST_aes.pdf
[5] Thiagarajan Eashwar, Gourishetty Madhuri, “Study of AES and its efficient software implementation”, http://islab.oregonstate.edu/koc/ece679/project/2003/thiagarajan-gourishetty.pdf
[6] Wollinger Thomas, Guajardo Jorge, Paar Christof, “Cryptography in embedded systems: An overview”, Proceedings of the Embedded World 2003 Exhibition and Conference, Febrero 2003.
[7] The International Engineering Consortium, “Smart Cards in Wireless Services”, http://www.iec.org
[8] Fiskiran Murat A., Lee B. Ruby, “Performance Impact of Addressing Modes on Encryption Algorithms”, http://www.iccd-conference.org/proceedings/2001/12000542.pdf
[9] U.S. General Services Administration Office of Government, “Government Smart Card Handbook”, http://www.smartcardalliance.org/pdf/industry_info/smartcardhandbook.pdf
[10] Escalante Bañuelos Alberto Nicolás, “El algoritmo de cifrado AES: su implantación y optimización de bajo nivel para mejorar los mecanismos de seguridad de una tarjeta inteligente”, Julio 2003, Facultad de Ingeniería, UNAM.
[11] Harris Shaon “CISSP Certification”, segunda ed., McGrawHill,2003.
[12] Bruce Schneier, *Applied Cryptography*, segunda ed., John Wiley and Sons, Inc, 1996.
[13] ATMEL, *8-bit AVR Microcontroller with 8k Bytes In-System Programmable Flash. AT90S8515*, http://www.atmel.com/atmel/acrobat/doc0841.pdf
[14] ATMEL, *AVR Instruction Set*, http://www.atmel.com/dyn/resources/prod_documents/DOC0856.PDF.
[15] Matthias Brüstle, *SOSSE*, http://www.mbsks.franken.de/sosse/
[16] CardLogix, “Smart Cards and Security Basics”, http://www.cardlogix.com
[17] Kobil Systems, “Kobil Smart Card Terminals”, http://www.kobil.com
[18] Gemplus, “Smart Cards Operating Systems”, http://www.gemplus.com
[19] WeetHet, “WeetHet-SmartCard-MasterBurner”, http://www.weethet.nl/english/smartcard_masterburner.php
[20] Guthery Scott B., Jurgensen Timothy M., *Smart Cards: A Developer´s Toolkit*, primera ed., Prentice Hall, 2002.
[21] Diffie W., Hellman M. E., New Directions in Cryptography, IEEE Transactions on Information Theory, Vol. IT-22, No. 6, Noviembre 1976.